A Hierarchical Framework for Constructing Computationally Efficient Algorithms for Distributed Inference Problems

by

Maurice Kyojin Chu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2003

© Maurice Kyojin Chu, MMIII. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author .....
Department of Electrical Engineering and Computer Science
January 3, 2003

Certified by .....
Sanjoy K. Mitter
Professor of Electrical Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

A Hierarchical Framework for Constructing Computationally Efficient Algorithms for Distributed Inference Problems

by

Maurice Kyojin Chu

Submitted to the Department of Electrical Engineering and Computer Science
on January 3, 2003, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

In this thesis, we present a methodology for designing computationally efficient algorithms for large-scale inference systems on system architectures with distributed autonomous agents. The principle of *information-based computation* is the underlying idea driving elements of this methodology. The methodology consists of a *layered information processing architecture*, which organizes specific inference tasks into layers so that the information extracted from these tasks can be exploited to design an efficient inference system, and the *hierarchy*, which is a representation combining elements of circuits and branching programs from which efficient distributed local algorithms implementing global tasks on general system architectures can be generated.

The general issues addressed by this methodology are: (1) how to efficiently collect distributed information for estimation when collecting information is expensive, (2) how to distribute a global algorithm among multiple agents, and (3) how to take advantage of prior computations to aid in subsequent computations.

To demonstrate that the methodology is a good one, we present an application of sensor networks to battlefield awareness. The problem is to gain intelligence about targets in the environment from the information collected by a set of sensor nodes which are distributed over a field and have real physical constraints on energy and communication. In particular, solutions to the problem of information collection for target localization and the problem of multiple target tracking and classification are formulated and demonstrated using the proposed methodology. Difficult problems such as the data association problem in multiple target tracking are handled when possible by the careful design of interactions between layers of an appropriately constructed layered information processing architecture.

Thesis Supervisor: Sanjoy K. Mitter
Title: Professor of Electrical Engineering

Acknowledgments

My career at MIT has spanned just over a decade. Through it all, Prof. Sanjoy Mitter has been my advisor, and I am deeply grateful for his deep insights and insistence on understanding the fundamentals of a problem which have inspired me to do the same. I'd like to thank the other members of my thesis committee, Prof. John Tsitsiklis, Prof. Tommi Jaakkola, and Dr. Feng Zhao. In particular, I'd like to thank Dr. Feng Zhao for giving me the opportunity to work with him since it is under him where my work on sensor networks began.

I'd like to thank those colleagues in my lab who have made my academic life at MIT enjoyable. In particular, I'd like to thank Vijay, Constantine, Ramesh, and Julius. Talking with you guys about research and the other issues in life has been the opening of a door which I did not know was closed.

And then there are those outside of academia who have given me fond memories and support that I will always cherish. People come in and out of our lives at various times, and I'd like to list a few of them I will remember from this time: Edwin, Steve, Sam, Jack, Danny, Ray, Stan, John, Shao, Eric, Jean, Helen, Kevin, Dave, Rich, MJ, Amy, Julie, Marcia, and Nancy. Thanks guys for the great times and the times to come.

Finally, I thank my brother Alex, my mother Soo, and my father Kwang for their belief in me through the years. I would not have been able to accomplish this without your love and support.

This research was partially supported by (MURI Sensorweb) "Data Fusion in Large Arrays of Microsensors (Sensor Web)" ARO Grant DAAD19-00-1-0466, (NSF-KDI) "KDI: Learning, Adaptation, and Layered Intelligent Systems" NSF Grant ECS-9873451, (Washington University) "A Research Center in the Scientific Foundations of Image Representation and Analysis" Washington University, (Brown University) "Vision Strategies and ATR Performance: A Mathematical / Statistical Formulation and Critique" Subcontract #654-21256, (ARO-ASSERT) "Hierarchy Representation of Images for Coding Compression and Data Fusion - ASSERT" ARO Grant DAAG55-97-1-0304, and (Johns Hopkins University) "Center for Imaging Science" Johns Hopkins University. They are very much appreciated.

Contents

1 Introduction **21**
1.1 Motivation 21
1.2 Problems/Issues and Basic Functions 24
1.3 Canonical Scenarios for Battlefield Awareness 29
1.3.1 Intelligence Updating 30
1.3.2 Data Association of Crossing Tracks 31
1.3.3 Scenario Summary 37
1.4 Overview of Proposed Methodology 37
1.5 Layered Information Processing Architecture 42
1.5.1 Top-down and Bottom-up Processing 44
1.5.2 Novel Aspect: Complementary Processes 47
1.6 Distributed Autonomous Agents 50
1.7 Hierarchy Representation 52
1.8 Contributions of Thesis 55
1.8.1 Overall Architecture 55
1.8.2 Computation and Communication Structure 57
1.8.3 Battlefield Awareness with Sensor Networks 58
1.8.4 Focus 58
1.9 Related Work 59
1.10 Organization of Thesis 62

2 Formulation of Inference Tasks **65**
2.1 System Architecture of Sensor Networks 65
2.1.1 Basic Functionality and Issues 65
2.1.2 Physical Characteristics 66
2.1.3 Resource Constraints 68
2.1.4 Nature of Information 70
2.2 Sensor Measurement Models 71
2.2.1 Acoustic Measurements 72
2.2.2 Magnetometer Measurements 82
2.2.3 Discussion 82
2.3 Localization Formulation 83

2.3.1 Distributed Estimation in Sensor Networks 84
2.3.2 Problem Formulation 86
2.4 Tracking Formulation 97
2.5 Classification Formulation 101
2.5.1 Classifying Frequency Signatures 102
2.5.2 Classifying Metallic Targets 103
2.6 Layered Information Processing Architecture 104
2.6.1 Overall Architecture 104
2.6.2 Localization 107
2.6.3 Tracking 108
2.6.4 Classification and Beyond 109
2.6.5 Conclusions 109

3 Tracking and Classification 111
3.1 Background 113
3.1.1 Single Target Tracking 114
3.1.2 Multiple Target Tracking 117
3.2 One Way Processes on the Layered Architecture 120
3.2.1 Purely Bottom-up Approach 120
3.2.2 Purely Top-down Approach 121
3.3 Tracking Overview 121
3.3.1 The Abstract Idea 122
3.3.2 Multiple Target to Multiple Sensor Model 123
3.3.3 Inference on the Model 124
3.3.4 Approximate Inference on the Model 125
3.4 Individual Information Processing Tasks 127
3.4.1 Track Initiation 130
3.4.2 Track Updating 136
3.4.3 Localization for a Specific Target 142
3.4.4 Classification of Frequency Signatures 151
3.4.5 Classification of Metallic Targets 154
3.5 Integration of Individual Tasks with Triggers 156
3.5.1 Track Initiation Triggers 157
3.5.2 Track Updating 157
3.5.3 Localization for a Specific Track 158
3.5.4 Classification of Frequency Signatures 159
3.5.5 Classification of Metallic Targets 160
3.6 Distributed Implementation 161
3.6.1 Track Initiation 162
3.6.2 Track Updating 163
3.6.3 Localization for a Specific Target 165
3.6.4 Classification of Frequency Signatures 165

3.6.5 Classification of Metallic Targets 167
3.7 Simulations 168
3.7.1 Nominal Measurement Model Parameters 168
3.7.2 Demonstrations of Nominal Behavior of the Inference System . 171
3.7.3 Demonstrations of Limits of Proposed Inference System 187
3.8 Limitations and Extensions 197

4 Target Localization 203
4.1 Encoding of Information Collection in Hierarchies 204
4.1.1 Question 1: Set of Possible Data Collection Mechanisms . . . 206
4.1.2 Question 2: Incorporating Partial Processing 217
4.1.3 Question 3: Measuring Computational Resources 228
4.1.4 Conclusions 233
4.2 Inference Task to Set of Computations 234
4.2.1 Deriving a Set of Computations from a Given Discretization . 238
4.2.2 Issues on Choosing a Discretization 247
4.3 Toy Example 252
4.3.1 Description of Simple Network 253
4.3.2 Step 1: Set of Computations 258
4.3.3 Step 2: Graph Topology Construction 261
4.3.4 Step 3: Agent Assignments, Uplinks, Downlinks, and Trigger Sets 268
4.3.5 Step 4: Converting to Communication Algorithms 275
4.3.6 Limitations of the Approach 281
4.3.7 Summary 284
4.4 Heuristics 285
4.5 A Variation on Hierarchy Construction 288
4.5.1 Hierarchy Representation of the IDSQ algorithm 288
4.5.2 Generalization of the IDSQ Criterion and an Alternative Hierarchy Construction 293
4.6 Distributed Construction of DCA's 296
4.7 Simulations 298

5 Hierarchy Representation 311
5.1 Relationship to Circuits and Branching Programs 314
5.1.1 As a Circuit with a Selection Mechanism 314
5.1.2 As a Branching Program with Parallel Processes 316
5.1.3 Preview of the Hierarchy 318
5.2 Application to Object Recognition 321
5.2.1 Background 322
5.2.2 Layers of Representation 323
5.3 Motivation 324

5.4 Mathematical Formalization 325
5.4.1 Hierarchy Definition 326
5.4.2 Layers 331
5.5 Graph Topology Construction 335
5.5.1 Theory 336
5.5.2 Heuristical Construction 340
5.5.3 Discussion 344
5.6 Evaluation Algorithms 348
5.6.1 Typical Evaluations 349
5.6.2 Canonical Message Passing Algorithm 352
5.6.3 Evaluation Resources 360
5.7 Conversion to Rules 361
5.8 Summary 365

6 Conclusions and Future Work 367

A Information-Driven Sensor Querying (IDSQ) 371
A.1 Basic IDSQ Algorithm 371
A.1.1 Overview 371
A.1.2 Estimation 374
A.1.3 Data Collection 375
A.1.4 Extensions 376
A.1.5 Conclusions 377
A.2 IDSQ Tracking Algorithm 377

B Information Measures 381
B.1 Spread Type Measures 381
B.2 Entropy Type Measures 383
B.3 Discussion 384

C Particle Filters 385

List of Figures

1-1 The square labelled with an M signifies a magnetometer. The dotted circle around the magnetometer is the effective range of the magnetometer. Assuming that the target can be tracked, we wish to update information as to whether the target is metallic or not when the target is within range of the magnetometer. 30

1-2 The problem of data association when two targets cross shown with the two possibilities that can occur after the targets split. In general with n targets crossing, there are $n!$ possibilities after all targets have split. 32

1-3 A possible solution for resolving the data association problem when there exists discriminatory information about the two targets (depicted as white and gray filled circles) before ambiguity in data association arises. The idea is to detect when there is the possibility of ambiguity in data association, use a discriminant to resolve ambiguities while the targets are near each other, and discard the use of the discriminant when targets are significantly far apart when there is no ambiguity in data association. 34

1-4 Two targets (depicted as white and gray filled circles) crossing where there exists discriminatory information between the two tracks but the information is unavailable until after the targets have split apart. Target 1 is non-metallic, and target 2 is metallic. The proposed solution for this scenario is to create a hypothesis of possible data associations and resolve the hypothesis when the discriminatory information becomes available. The figure shows the case when the magnetometer in the top right of the field determines that the target in range is metallic. The resolution of the data association hypothesis is that target 2 travelled from the bottom left to the top right of the field and that target 1 travelled from the top left to the bottom right of the field. 36

1-5 The inference tasks of localization, tracking, and classification arranged into layers to begin the construction of the layered information processing architecture. The data, indicated by the rounded gray box labelled with "Sensor Measurements", is placed at the bottom. 38

1-6 Decomposition of each inference task into a set of information processes. Interaction of information extracted from each process are indicated by black arrows. This is the *layered information processing architecture*. 39

1-7 Distributed algorithms for each individual process are implemented for system architectures with distributed autonomous agents. Communications may be necessary between agents when generating distributed algorithms for each information process as indicated by the horizontal, dashed arrows. Sensor measurements observed by each agent is local. 40
1-8 Methodology for designing computationally efficient algorithms for inference systems. 41
1-9 A layered information processing architecture. Arrows indicate the direction of the possible flow of messages interfacing the abstraction layers. 43
1-10 The proposed interaction between layers to support the complementation of top-down and bottom-up processing. The arrows going outwards indicate how computations of this layer affect computations of other layers. Arrows coming inwards to this layer indicate how computations of adjacent layers affect computations of this layer. 48
1-11 The proposed interaction between layers results in loops of computation between layers which proceed from generic to specific. The loops between other layers are processed concurrently. 49
1-12 A graphical representation of a complex computation as a network of simpler computations. 53

2-1 The architecture of a node of the sensor network. Each node is equipped with a general purpose processor, a set of sensors, and a wireless communication device. Data and information from the outside world come in through the sensors and wireless receiver. Messages can be sent to other sensor nodes by the wireless transmitter. 67
2-2 Sensors only provide significant information about targets (black circles with gray dot) if they are "near" or local to them. Two instances of the target (circles), located on the far left side and on the far right, are shown along with the set of sensor nodes which are affected by the presence of the target ($\times$ in square). 71
2-3 The intensity scaling factor as a function of the attenuated distance d between target location and sensor position. 74
2-4 The intensity likelihood function of the target position x given various sensor measurement values z with parameters $A = 10$, $\sigma_i = .5$, and $\alpha = 2$. The sensor is depicted by the square. Dark regions signify regions of target positions with high likelihood, and light regions signify regions of target positions with low likelihood. 75
2-5 The TDOA likelihood function as a function of target position x for various values of time difference z. The dipole of omni-directional microphones is situated horizontally as shown by the horizontal line drawn over the square which represents the TDOA sensor. The first microphone of the dipole is on the right, and the second microphone of the dipole is on the left. . . . 81

2-6 Usual parameter estimation process. 84
2-7 Estimation process in a sensor network. 85
2-8 Estimation process in a sensor network with data collection. 85
2-9 The proposed layered information processing architecture for the inference tasks of localization, tracking, and classification on sensor networks. Each layer consists of computations which range from generic to specific. The arrows indicate what information is passed between layers, and the italic text indicates either the spawning or priming effect of computations in the respective layer based on the information passed. 106

3-1 The Markov random field representation of the random variables defined in the model from target positions X^j to sensor measurements Z_i. We label edges of the Markov random field with the densities that define the relationship between the random variables. 124
3-2 The individual information processing tasks in each layer of the layered information processing architecture and the information dependencies between them. Black block arrows indicate information being exchanged between the different processes. Thin black arrows indicate possible top-down priming and bottom-up spawning interactions. The gray boxes indicate that there is a trigger for initiating the corresponding processing task. The large rounded gray boxes labelled by "Sensor Measurements" and "Tracks" refer to the data measured by the sensor network and the representation of tracks constructed from the data respectively by this inference system. 128
3-3 Example sensor network of 67 sensor nodes. Numbers next to sensor nodes indicate the ID number of the sensor node. 171
3-4 The trajectories of the 6 targets to demonstrate the distributed multiple target tracking. Initial target positions are shown with an asterisk, and the lines indicate the target position over time at constant velocity. 173
3-5 Snapshots of running the distributed multiple target tracking algorithm. . 174
3-6 Snapshots of running the distributed multiple target tracking algorithm on a sensor network consisting of intensity sensors only. 175
3-7 Sensor network to demonstrate the canonical scenario of intelligence updating. Numbers next to sensor nodes indicate the ID number of the sensor node. 177
3-8 Snapshots of demonstrating classification occurring when information is available. 178
3-9 Snapshots of demonstrating handling of data association problem when no discriminatory information is available. 180
3-10 Snapshots of demonstrating handling of data association problem when discriminatory information is available. Targets travel at constant velocity and cross. 182

3-11 Snapshots of demonstrating handling of data association problem when discriminatory information is available. Targets travel at constant velocity, converge, and split apart. 183
3-12 Snapshots of demonstrating handling of data association problem when discriminatory information is not available until after tracks have diverged. Targets travel at constant velocity and cross. 185
3-13 Snapshots of demonstrating handling of data association problem when discriminatory information is not available until after tracks have diverged. Targets travel at constant velocity, converge, and split apart. 186
3-14 Mean track error plotted as a function of α, the scaling on the noise standard deviation of the simulated measurements. Plots show the individual mean track errors over twenty runs as well as the average of those twenty runs for 1, 2, 3, 4, and 5 tracks moving in parallel. 190
3-15 Track length plotted as a function of α, the scaling on the noise standard deviation of the simulated measurements. Plots show the individual track lengths over twenty runs as well as the average track length over those twenty runs for 1, 2, 3, 4, and 5 tracks moving in parallel. 191
3-16 Comparison of the average mean track error and average track length for simulations with 1, 2, 3, 4, and 5 tracks moving in parallel. 192
3-17 Mean track error, track length, and misclassification error over values of p. 194
3-18 Mean track error and track length plotted as a function of intensity attenuation coefficient α generating the simulated measurements. 196
3-19 Mean track error and track length plotted as a function of the distance d between the 4 targets moving in parallel. 197

4-1 A system architecture with three distributed autonomous agents shown as triangles labelled by an integer from $\{1, 2, 3\}$. Initially, each agent can observe the value of its respective variable indicated underneath each triangle. 204
4-2 One possible mechanism for collecting data. (a) Agents 1 and 2 both send the value of their respective variable to agent 3. (b) Agent 3 is now aware of all variable values which is indicated by the the three variables listed underneath it. 206
4-3 Another possible mechanism for collecting data. (a) Agents 1 sends x_1 to agent 2. (b) Then, agent 2 sends x_1 and x_2 to agent 3. (c) Finally, agent 3 is aware of all variable values which is again indicated by the variables listed under the agent. Note that a residual effect of this collection mechanism is that agent 2 is aware of both x_1 and x_2. 207
4-4 Graph representing the three variables to be collected by some agent. . . 209
4-5 Graph of Figure 4-4 with all black nodes of down degree greater than 2 converted into a subgraph of several black nodes with down degree 2. The three graphs (a), (b), and (c) show the resulting graphs. 210

4-6 The 12 different graphs with unique agent assignments denoted by an agent label from $\{1, 2, 3\}$ next to each node. The four graphs in each column correspond to the respective unassigned graphs of Figure 4-5. The bold edges correspond to real information transfers encoded in the graph. . . . 211

4-7 Graphs for data collection mechanism A with edges labelled by uplinks and downlinks. (a) Agents 2 and 3 send to agent 1. (b) Agent 2 sends to agent 1. Agent 1 queries agent 3. (c) Agent 1 queries agent 2. Agent 3 sends to agent 1. (d) Agent 1 queries agent 2. Then, agent 1 queries agent 3. . . . 214

4-8 (a) Graph B from Figure 4-6 with all bold edges chosen to be uplinks. (b) Graph A from Figure 4-6 with all bold edges chosen to be downlinks. . . 220

4-9 The hierarchy for the graph of Figure 4-8a showing the partial computations and trigger sets which initiate communications. 222

4-10 The hierarchy for the graph of Figure 4-8b showing the partial computations and trigger sets which initiate communications. 225

4-11 Simple sensor network containing 8 sensor nodes shown as squares. The $\times$'s represent the sampled target locations to serve as the parameter space. . . 254

4-12 The initial graph representing the dependencies of the triggers $\{T_i\}$ on the measurement tokens $\{a_k^v\}_{v \in \mathcal{Z}_k, k \in \{1, \ldots, n\}}$. 264

4-13 The black nodes, labelled b_1 to b_{14} from left to right, are the set of enumerated nodes from the enumeration step which are candidates for reducing the down degree of some node T_i. The number of dashed edges emanating above each enumerated node (black node) indicates the commonality heuristic. We have omitted those nodes a_k^v which were not connected to any node T_i. 265

4-14 The updated hierarchy after adding enumerated node b_2 into the hierarchy. The gray nodes indicate nodes of down degree greater than 2 which we must replace with a subgraph of nodes. 268

4-15 The progression of the rest of the iterations to form the hierarchy. The top two figures show the hierarchy after two more iterations. In the resulting graph topology of the hierarchy shown at the bottom of the figure, each black node is associated with a logical AND function. These AND functions are the partial computations for the triggers. 269

4-16 The agent assignment $\hat{\mathcal{A}}_1$, denoted as bold italic numbers next to each hierarchy node, minimizes the total communication distance when $k_{query} = 1$. All edges corresponding to a real communication are chosen to be uplinks. 274

4-17 The agent assignment $\hat{\mathcal{A}}_2$, denoted as bold italic numbers next to each hierarchy node, minimizes the total communication distance when $k_{query} = 0$. Edges corresponding to a real communication are chosen to be both uplinks and downlinks. 275

4-18 The known variable values of each node of the sensor network are shown underneath the respective node. (a) Before communications occur, each sensor node i only knows its own sensor measurement z_i. (b) Then, the communication algorithm in sensor node 3 sends z_3 to sensor node 5 as indicated by the black arrow. (c) The communication algorithm in sensor node 5 queries for the local knowledge state of sensor node 2 as shown by the dotted arrow. Then, sensor node 2 responds with the value of measurement z_2. (d) Sensor node 5 has enough data to satisfy its stopping condition $STOP_5$ as indicated by the $\times$ in sensor node 5. 279

4-19 The known variable values of each node of the sensor network are shown underneath the respective node. (a) Before communications occur, each sensor node i only knows its own sensor measurement z_i. (b) Then, the communication algorithm in sensor node 1 sends z_1 to sensor node 4 and the communication algorithm in sensor node 8 sends z_8 to sensor node 4 as indicated by the black arrows. (c) Sensor node 4 has enough data to satisfy its stopping condition $STOP_4$ as indicated by the $\times$ in sensor node 4. Although there exist communication rules in C_4 which could be triggered by the local knowledge state of sensor node 4, the stopping condition takes precedence so that the communications encoded by these rules do not take place. 282

4-20 The hierarchy associated with the IDSQ algorithm. All nodes above the input layer are assigned to the leader node l and are associated with a two-input logical AND. Each node represents some local knowledge state, and downlinks represent a query from a remote node. 292

4-21 A small sensor network consisting 10 sensor nodes placed randomly within a square area of side length 100 centered at the origin. The outline box indicates the chosen field of this sensor network. 299

4-22 A large sensor network consisting of 64 sensor nodes placed approximately on a square grid. The outline box indicates the chosen field of this sensor network. 300

4-23 Individual plots of energy drain (AVE), maximal distance (MD) between target and a sensor node which transmits, and the number of handoffs (HANDOFF) before a sensor node declares that the global localization task has been computed. 305

4-24 Plots comparing number of handoffs before a sensor node declares that the global localization task has been computed and the average energy drain. . 306

4-25 Individual plots of energy drain (AVE), maximal distance (MD) between the target and a sensor node which transmits, and the number of handoffs (HANDOFF) before a sensor node declares that the global localization task has been computed. 308

4-26 Plots comparing number of handoffs before a sensor node declares that the global localization task has been computed and the average energy drain. . 309

5-1 Steps in using the hierarchy for constructing architecture specific algorithms. The hierarchy bridges the gap between the abstract computations and the implementation of the computation by an algorithm. 313
5-2 A simple circuit. 315
5-3 A simple branching program. 317
5-4 The *graph topology* of a hierarchy is a circuit. In the picture, nodes have been arranged so that all directed edges point upwards. 319
5-5 The choice of uplinks, downlinks, and trigger sets associated with uplinks, pictured as dashed arrows and subsets associated with dashed arrows, encompass the *evaluation algorithm* part of the hierarchy. 319
5-6 Each hierarchy node is associated with a function with two sets of directed edges: information edges and control edges. Each node is a computational unit that runs the *canonical message passing algorithm* which is a message passing scheme that determines when this node should evaluate its associated function and when this node should initiate the evaluation of functions of other nodes. 320
5-7 An example of an agent assignment. 321
5-8 The steps of using the hierarchy representation for constructing computationally efficient algorithms. The process of graph topology construction constructs the N, F, and E parts of the hierarchy while the process of choosing an evaluation algorithm constructs the U, D, and Υ parts of the hierarchy. 330
5-9 An example of the graph topology of a hierarchy with each node denoted by the set of layers that the node lies in. All edges are information edges directed upwards. 332
5-10 Graphical picture of the evaluation requirement. If it happened to be that $\tilde{P}$ is as shown in the figure, then for the given ρ, the evaluation requirement requires that we explicit evaluate $\tilde{P}(i)$ for those i which are highlighted on the x-axis of the graph. Note that $\tilde{P}(i)$ is different for different values of the input $\underline{v} \in V^n$. Thus, the subset of I for which we explicitly evaluate $\tilde{P}$ is different for different $\underline{v}$. 351
5-11 The three sets of links incident on a node a. Uplinks and downlinks pointing towards a are the edges in L_a. Uplinks connecting a to nodes above a are the edges in U_a. Downlinks connecting a to nodes below a are the edges in D_a. 354
5-12 The six groups of neighbor nodes of any node a in the hierarchy. Uplinks are denoted by arrows that point upwards along an edge, and downlinks are arrows that point downwards along an edge. 358

A-1 The block diagram of the IDSQ algorithm, which is run in the leader node only, shows the feedback loop between the estimation process and the data collection process. The boxes labelled with "D" are delay elements. 372

List of Tables

4.1 The data collection mechanism corresponding to each graph of Figure 4-6. 212
4.2 Sensor positions for sensor network of toy example given in units of tens of meters. 253
4.3 Trigger evaluations for each of the eight joint measurements in $\mathcal{Z}_{likely}$. . . 261
4.4 Heuristic values for each enumerated node shown ranked according to larger **common** followed by smaller **cost** if there are ties. Rank numbers are repeated if there is no way to break a tie. 267
4.5 The set of local rules generated by the uplinks and downlinks of the hierarchy of Figure 4-17. 276

Chapter 1

Introduction

1.1 Motivation

Current technology enables us to engineer sensor rich systems capable of collecting massive amounts of data with the potential to extract information about the system state and their environment to an extent which has been unimaginable or prohibitively expensive in the past. With recent advances in sensing, embedded computation, and wireless communications, it is possible to construct networks consisting of a large number of heterogeneous elements, each with the ability to sense its environment in various modalities, process data and extract information, and communicate the results of these computations with each other. These *sensor networks* offer a new kind of system concept: the coordinated collaboration of a network of distributed autonomous agents working together to extract the desired information from the data collected locally by each of the agents.

We imagine deploying thousands of sensor nodes in the environment or embedding these sensor nodes within an engineered system. With better awareness, construction of engineered systems with complex, intelligent, and robust behavior becomes ever more possible, and the potential applications of sensor networks include intelligent transportation grids, surveillance and security systems in civilian domains, monitoring and diagnosis of engineered systems, and battlefield situational awareness, which is the application we focus on which serves to demonstrate the ideas in this thesis. However, we emphasize that these sensor networks only have the "potential to extract information" because although these networks have the ability to collect massive amounts of data at the numerous sensor nodes, without efficient algorithms which can extract the desired information from the raw data, the desired information contained in the data is effectively unattainable and, hence, unusable. Thus, to realize the potential information to be gained through a sensor network, a set of algorithms must be designed which extract the desired information from the data collected by the sensors of the sensor network.

We refer to the problems of extracting information from data as inference prob-

lems which we can pose in terms of standard probabilistic estimation and detection theory. However, there are two important characteristics about sensor networks which standard estimation and detection theory is unsuitable to handle.

1. Computational resources are limited. By computational resources, we refer to any resources which can limit or enhance the desired operation of a system. These include the standard resources of time and memory due to real-time and finite memory constraints as well as other physical resources such as limited energy sources and the availability of multiple processors. Of particular interest will be limited energy sources. Every action a sensor node performs, whether it be sensing, processing, or communicating, consumes a certain amount of energy. Thus, for sensor networks where nodes have limited energy sources (as in the application to battlefield awareness), it is necessary that the algorithms which perform inference are efficient in order to prolong the functioning lifetime of the sensor network.

2. Information to be extracted is physically distributed across several sensor nodes. For example, our ability to extract depth information about objects of unknown size from our eyes is due to the fact that there exist predictable differences between the images of the object on the retina of each eye from which we can infer the depth of the object. Thus, information about the depth of the object resides in neither eye alone but is physically distributed between our two eyes. In order to extract information from data which is physically distributed, it is necessary for the appropriate information containing nodes in a sensor network to collaborate and communicate with each other. The significance of this unavoidable need for communication is due to the first characteristic of limited computational resources: since communication is a consumer of energy, minimizing the communications occurring is an important consideration when designing efficient inference algorithms in sensor networks.

In standard estimation theory, the cost associated with the computation of an estimate and the cost associated with collecting physically distributed data needed for estimation is not explicitly considered. Thus, these characteristics lead to the following important difference between standard estimation problems and estimation on sensor networks.

With limited computational resources, the goal of estimation is no longer to produce optimal estimates but to compute estimates of reasonable quality which minimize the amount of computational resources expended. This goal implies the need for (1) specifying a notion of "reasonable quality" of an estimate, (2) characterizing the set of all possible algorithms which implement the computation of this estimate, and (3) the ability to measure the computational resource requirements for all possible implementations so that the minimizing (most efficient) implementation can be chosen

commensurate with the performance requirements.

The key conceptual idea we adopt in order to design efficient algorithms for inference tasks is the following principle which we refer to as the *principle of information-based computation.*

Principle of Information-Based Computation
Based on the information gained from previous computations, determine what computations to perform next in an intelligent fashion.

The main idea of this principle is to exploit all information that is currently known in order to extract further information. Thus, knowledge of certain information triggers the extraction for other information. When different information is known, extraction of different information is triggered. This simple idea is intuitively rather obvious and is embodied in decision trees, branching programs, and active querying techniques in the sense that a choice is made to incorporate certain observations or queries depending on what information is currently known. For sensor networks, applying the information-based computation principle becomes more complicated due to two characteristics.

1. The massive amount of data and the large quantity of diverse information we hope to extract from this data requires an organizational tool if we are to design an efficient general inference system which embodies the information-based computation principle.

2. Due to the possibility of parallel computation since there is a processor in each node of the sensor network, we are faced with the challenge of applying the information-based computation principle in a parallel manner. Each sensor node should make local decisions of what information to extract or communicate so that, in a global sense, the sensor network is extracting information based on what global information is currently known.

We take a holistic view for designing efficient large-scale inference systems meaning that the different inference tasks are to be designed all together rather than in isolation. This holistic view will allow us to determine what information is extracted from the various inference tasks and what further information should be further extracted.

To handle the massive amounts of data and large quantity of information to be extracted, we organize the set of inference tasks into layers and design interactions between these tasks in order to exploit the information gained from these tasks. The resulting structure is referred to as the *layered information processing architecture* which we introduce in Section 1.5 and is used for designing a specific inference system on sensor networks in Chapter 3.

In order to exploit the possibility of parallel computation, we need a computational model which can allow parallel local decision making and which embodies the

information-based computation principle. Note that the existing computational models like decision trees and branching programs are inherently sequential. We introduce the *hierarchy representation* which serves as a meta-representation from which distributed algorithms with desired global behavior can be generated. The hierarchy can be viewed as a computational model which both embodies the information-based computation principle and allows parallel local decision making. The hierarchy is introduced in Section 1.7, applied to a specific inference task for sensor networks in Chapter 4, and, finally, presented in its most general form in Chapter 5.

The information-based computation principle is pervasive throughout this entire thesis. The layered information processing architecture and the hierarchy are elements of a methodology introduced in this thesis which embodies this principle for the construction of efficient algorithms for distributed inference problems. To begin, let us discuss the general problems and issues and the basic functions to be performed in sensor networks, which serves as the exemplary system architecture for our proposed methodology, in Section 1.2. Then, Section 1.3 presents some canonical scenarios encountered by a sensor network used for battlefield awareness and the behavior we desire from the inference system. Finally, Section 1.4 and subsequent sections describe the proposed methodology for constructing efficient algorithms which implement the inference tasks.

1.2 General Problems/Issues and Basic Functions in Sensor Networks

There are three primary issues when designing inference algorithms on sensor networks.

1. **Limited Computational Resources** In particular, we will focus on the limited availability of battery power to operate the nodes of the sensor network. There are other resources like limited processing capabilities and limited memory which can potentially limit the possible algorithms running on these sensor nodes (e.g. the difference between designing an algorithm to run on an 8-bit processor with 1 MB of memory versus a Pentium 4 processor with 1 GB of memory is tremendous); however, the main constraint limiting the miniaturization of the physical construction of sensor nodes is not the processor or memory chips but the battery power source. As mentioned in [20], the trend is that sensor nodes can be equipped with powerful processors with enough memory so that the primary consideration in the design of algorithms is the power consumed during the implementation of the algorithm.

2. **Dynamic Network Topology** With the possibility of mobile sensor nodes and individual node failures during the lifetime of a sensor network, inference

algorithms must be able to handle the possibility of a dynamically evolving network topology. The consequences of this dynamic network are that inference algorithms should not depend on any specific assumptions about the configuration of sensor nodes and should periodically detect for faulty nodes. For example, the GEAR algorithm in [28] computes a route for data dissemination dynamically so that the "holes" generated by node failures do not prevent the routing of data.

3. **Scalability** The sheer number of sensor nodes in the sensor network can be a limiting factor if the inference algorithms scale poorly with the number of nodes. Inference problems like localizing multiple targets and tracking multiple targets within range of the sensor network are *global* inference problems in the sense that the information extracted about targets refers to targets within the entire sensor network. A centralized approach where data from all sensor nodes are sent to some chosen "central" sensor node which then processes all the data to perform inference works well for networks with a small number of sensor nodes; however, when the number of sensor nodes is in the thousands, the amount of data that must be communicated not only drains battery power – especially if multi-hop communication is implemented – but is a significant bottleneck which can degrade the performance of the inference algorithms. For example, long latencies in data transfer can severely degrade the performance of inference algorithms that operate under real-time constraints. Thus, dealing with the issue of scalability argues for distributed solutions where relevant information is extracted before communications occur. The idea is that sensor nodes collaborate within local neighborhoods. Then, the information extracted within these local neighborhoods can be fused into larger and larger neighborhoods until the global inference problem can be solved.

In light of the three main issues above and our focus on designing efficient algorithms for inference on sensor networks, we will categorize the set of basic functions that must be implemented in order for sensor networks to become a reality. The following categories do not necessarily imply a separation of algorithms which implement each of these functions alone. In fact, we will argue that designing efficient inference algorithms on sensor networks *requires* interaction among these basic functions.

1. **Self-configuration** The function of self-configuration involves computing information about basic characteristics of the sensor network which are parameters needed in order for sensor nodes to collaborate effectively to perform inference about targets in the sensor network. Typically, sensor nodes will be deployed in an ad hoc fashion so that there must be some method of determining the relative position of the sensor nodes, determining the set of neighbors with which each node can communicate, setting up a communication protocol so that nodes and their neighbors can communicate reliably, and learning other specific characteristics of neighboring nodes like the types of sensors attached to them and the

measurement models generating the data. Essentially, self-configuration sets up the basic operations of the sensor network, and the information gained about the current state of the sensor network will be needed for performing inference tasks. Self-configuration is needed because of the ad hoc nature of the sensor network during deployment and the possibility of a dynamically changing network topology.

2. **Information Transfer** By information transfer, we refer to the problem of efficiently transferring and routing information between nodes of the sensor network. We categorize the set of information transfer problems into three subcategories: (1) dissemination, (2) retrieval, and (3) information collection for inference. The three subcategories differ in the intended reason for transferring information and where the decision is made for transferring information.

   In dissemination, the problem is to transfer information either from a remote broadcast station or some node within the network to a subset or all of the nodes of the sensor network. The purpose of dissemination is to make several nodes aware of some information. The decision of where information is to be ultimately transferred is made at the information source in dissemination. In retrieval, the problem is to route information from nodes of the sensor network either to a particular node of the network or to some remote receiver station. Gathering the information which is potentially distributed throughout nodes of the sensor network is the purpose of retrieval. The decision of where information is to be ultimately transferred is made at the information sink in retrieval. Finally, information collection for inference refers to the task of collecting information in order to perform specific inference tasks. This is an information transfer task for the purpose collecting enough information so that a particular inference task can be computed. The decision of what and where information is to be transferred is made locally by each sensor node depending on what information has been collected by the sensor node. For example, when partial estimates are computed, it is possible to determine what data can provide the most expected information so that data is collected on a need-to-know basis as in the IDSQ algorithm [7] (See Appendix A).

   Information transfer is an important problem which arises from the distributed nature of information in the sensor network. Because wireless communication is the biggest consumer of power, minimizing the number of bits of information and the distance that these bits must be communicated is critical for prolonging the lifetime of the sensor network.

3. **Network Monitoring** To handle the possibility of node failures, there must be routines for detecting faulty or soon to become faulty nodes. Monitoring parameters like the current power level of sensor nodes in the network will be important for adjusting the algorithms for information transfer and inference for

the purpose of prolonging the lifetime of the sensor network. Network monitoring is needed in order to make the most efficient use of a dynamically evolving sensor network.

4. **Inference** Inference is the primary function of the sensor network which, to be successfully implemented, needs the functions of self-calibration, data transfer, and network monitoring due to the ad hoc, distributed, and dynamic nature of the network. We wish to design algorithms on sensor networks which can perform a variety of inference tasks which, in the battlefield awareness application, include localizing targets, tracking the motion of targets, classifying the types of targets, and inferring other characteristics of targets and groups of targets. Thus, the general problem we face is how to implement global inference tasks by local distributed algorithms running on nodes of the sensor network. Distributing global computation into a set of local computations addresses the issue of scalable algorithms on sensor networks.

   Aside from the added complexity of having to run these inference algorithms on a sensor network, these inference tasks are non-trivial tasks in themselves. From the observation that the information extracted by different inference tasks can usually aid each other, we will concentrate on taking advantage of these relationships and design the appropriate information exchange between inference tasks so that the resulting inference algorithms are efficient. This idea to trigger computations based on evidence of the potential information to be gained by a computation is an instance of the principle of information-based computation. We will present specific scenarios of how information from different inference tasks can aid each other for the application of battlefield awareness in Section 1.3.

In this thesis, we will focus on two specific problems:

1. designing efficient information collection algorithms for specific inference tasks (from the information transfer category) and

2. the design of efficient inference algorithms on sensor networks which take advantage of the relationship among different inference tasks (from the inference category).

Both problems are examples where efficient algorithms are designed following the general principle of information-based computation. Information-based computation for information collection is concerned with determining what information to communicate based on what information is currently known. Information-based computation for designing inference algorithms is concerned with determining what inference algorithms to initiate based on what information has been currently extracted from other inference tasks.

We refer to the problems of inference on sensor networks as *distributed inference problems* because the information needed to perform the inference tasks are distributed throughout nodes of the sensor network and algorithms for collecting the information must be distributed. The elements that must be considered for designing efficient, distributed algorithms for information collection and inference in sensor networks are

(E1) a set of approximations from the optimal estimate (when posing inference as an estimation problem) which trade off the deviation from the optimal estimate with the computational resource requirements of an algorithm implementing the estimate,

(E2) a characterization of the set of all algorithms, including *distributed* algorithms, which implement a particular approximation to the optimal estimate,

(E3) a measure of the computational resources expended for each possible algorithm implementing an approximation to the optimal estimate, and

(E4) the incorporation of interactions between various inference tasks.

The contributions of this thesis are twofold. The first set of contributions include specific algorithms which implement information collection and inference for gaining intelligence about targets in the application of sensor networks to battlefield awareness. Although we could focus on designing specific inference algorithms which run on a specific type of sensor network with specific kinds of sensors and specific wireless communication devices, what is really needed to make such inference systems a reality is a general methodology for the design of computationally efficient algorithms which implement inference tasks on any proposed system architecture. Thus, the second, and more general, set of contributions is a general methodology for designing computationally efficient algorithms for distributed inference problems. The methodology consists of three components which we only briefly mention now.

1. The *system architecture* is the physical hardware architecture, basic operations of the system, and other physical constraints of the system.

2. The *hierarchy* is a representation from which efficient distributed algorithms running on the system architecture for information collection and inference can be generated. This representation addresses elements E2 and E3 above.

3. The *layered information processing architecture* is a software architecture with which we can systematically design interactions between various inference tasks. This layered architecture addresses element E4.

Approximations to the optimal estimate in element E1 depend on the specific estimation problem, an appropriate choice of the space of possible approximations, and

a measure of similarity between the approximate estimate and the optimal estimate. Thus, being application-dependent, the general methodology does not address E1 directly.

We will describe the three components of the methodology mentioned above in more detail in Section 1.4 after we have presented the kinds of problems which motivated the methodology. Let us consider some particular scenarios and the types of inference tasks we wish a sensor network to perform for the application to battlefield situational awareness.

1.3 Canonical Scenarios for Battlefield Awareness

The purpose of deploying a sensor network for the application of battlefield awareness is to gain intelligence about friendly and hostile targets in a specified region, which we will refer to generically as the *field*. These targets include vehicles like humvees, tanks, and jets, as well as human troops and other objects of interest. Typical inference tasks to be performed on these sensor networks regarding targets in the field include

- localization of the position of targets,
- tracking the movement of targets over time,
- predicting possible maneuvers of these targets,
- classification of the type of target, and
- other inference tasks identifying characteristics of targets and groups of targets.

In this section, we present some canonical scenarios of targets moving through the field, the kinds of information we wish to extract about the targets, and the potential difficulties that must be dealt with by the sensor network. The presentation of this section is high-level, and we reserve the formal formulation of the inference tasks to be implemented on the sensor network for battlefield awareness for Chapter 2.

These inference tasks must be accomplished through the collaboration of sensor nodes which are randomly deployed in the field based on measurements from the equipped heterogeneous sensors on these sensor nodes which can include acoustic, seismic, magnetic, thermal, infrared, and visual sensors. These sensors vary in effective range of detection for a target. For example, magnetic sensors are generally short-range sensors which can detect metallic targets in a relatively small region. On the other hand, acoustic, infrared, and visual sensors can detect targets at longer range.

Among the typical inference tasks to be implemented on a sensor network, we will focus on the task of target tracking. We note, however, that constructing efficient algorithms for target tracking requires constructing efficient algorithms for localization, classification and possibly other inference tasks since these other inference tasks have

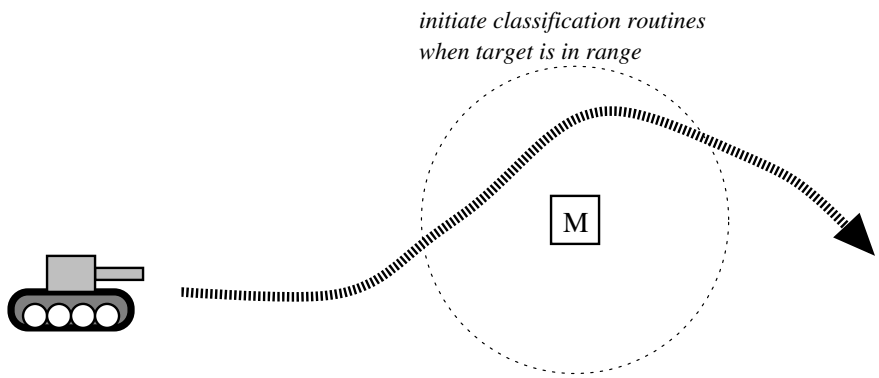

Figure 1-1: The square labelled with an M signifies a magnetometer. The dotted circle around the magnetometer is the effective range of the magnetometer. Assuming that the target can be tracked, we wish to update information as to whether the target is metallic or not when the target is within range of the magnetometer.

a role to play in target tracking and vice versa. For example, localization provides information about possible target locations which are used for computing tracks associated with targets in the field. Knowing the current tracks, the classification task can determine the class of each target individually. Conversely, the information provided by classification routines can play a critical role in handling the *data association* problem inherent in multiple target tracking. We will provide a high-level description of the problem of data association in Section 1.3.2 and leave the detailed description of the data association problem for Section 3.1.2.

The canonical scenarios we present will be focussed on the inference task of target tracking in particular to those where the data association problem arises. Along with the description of the scenarios, we suggest the possible information exchanges that can occur between different inference tasks which can facilitate the general goal of gathering intelligence on the targets of interest. In particular, we will suggest how to handle the problem of data association in multiple target tracking depending on the kinds of discriminatory information available.

1.3.1 Intelligence Updating

Depending on the kinds of sensors near the location of the target, it would be useful to gather as much information about the target as possible when the information becomes available. For example, due to the short range of magnetic sensors, the ability to determine whether a target is metallic is highly dependent on whether the target happens to be in range of a magnetic sensor. Figure 1-1 shows a possible scenario. Once the target is known to be within range of the magnetometer, information should be extracted from the magnetometer to update whether the target is metallic or not.

This is an example of how localization information triggers a classification routine. By triggering this classification routine only when targets are within range of the magnetometer, we save computational resources in terms of collecting data from the magnetometer.

1.3.2 Data Association of Crossing Tracks

We present a set of scenarios in this section where the problem of data association arises when tracks cross in the field. Let us take a moment to give a high level description of the data association problem. In Figure 1-2, we show two targets, a tank and an airplane, which move towards each other and diverge. If the only information we are able to gather about these two targets are their sequences of positions over time, there is an underlying ambiguity as to which direction the two targets have travelled as indicated by the two question marks in the boxed area of the figure. Thus, there are two possibilities that could have occurred.

(Hypothesis 1) The two target have crossed each other so that the tank ends up at the top right of the field and the airplane ends up at the bottom right of the field as indicated in the bottom left portion of Figure 1-2.

(Hypothesis 2) The two targets came near each other and then diverged so that the tank ends up at the bottom right of the field and the airplane ends up at the top right of the field as indicated in the bottom right portion of Figure 1-2.

The problem of resolving this ambiguity is referred to as the data association problem. Clearly, with no additional information about targets other than their position, there is no way to resolve this ambiguity. However, when discriminating information is available, there is the possibility of resolving the ambiguity and thus solve data association problem.

Scenario 1: Discriminating Information Immediately Available

Two examples of discriminating information which is available before the targets begin to cross and can be used to resolve the ambiguity in data association are the following.

1. Given a motion model of the targets (like limits on the acceleration of targets), the ambiguity can be resolved based on physical limitations of the possible movements of the targets. For example, in Figure 1-2, hypothesis 2 may be impossible since the jet, which we assume is travelling much faster than the tank, may not be able to turn so sharply. Using the motion model to resolve the data association of the two targets is an example of using higher level knowledge about possible motions of the target in order to perform data association.

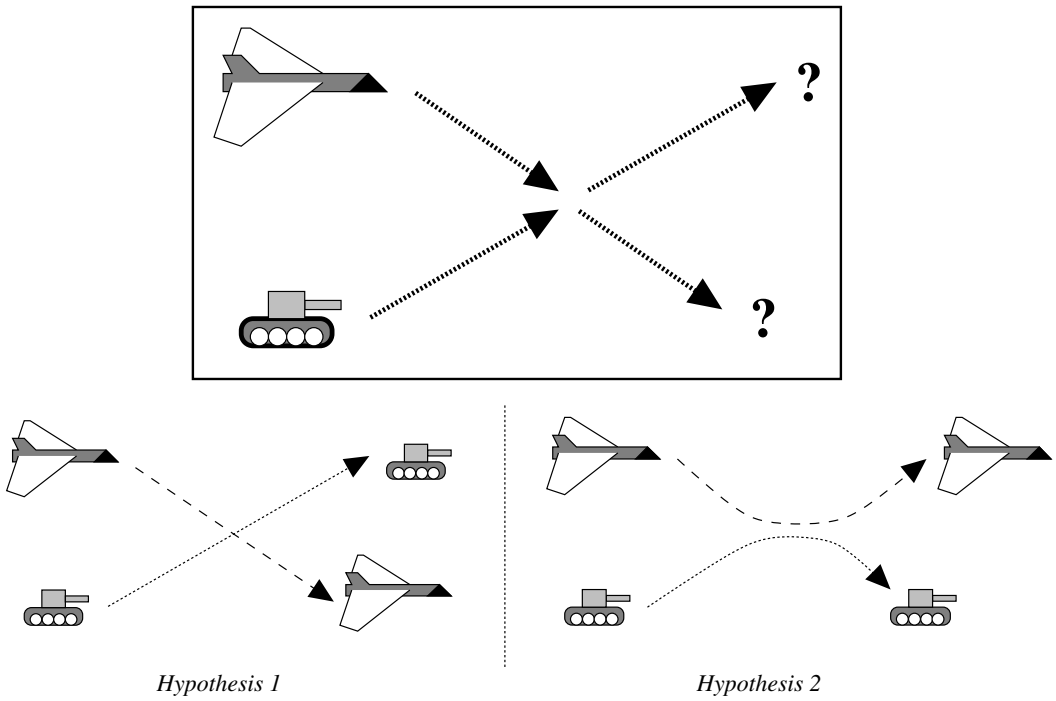

Figure 1-2: The problem of data association when two targets cross shown with the two possibilities that can occur after the targets split. In general with n targets crossing, there are $n!$ possibilities after all targets have split.

2. Another scenario is when the crossing targets have been classified and are known to emit different frequency signatures which can be measured by acoustic sensors. Assuming it is possible to determine the frequency content of the sound at any position in the field with some reasonable degree of accuracy, data association can be performed by classifying the acoustic signals from the predicted positions of the targets so that there is essentially no ambiguity when the targets cross. Using this classification information is an example of how information from a classification task can aid the task of multiple target tracking.

The above two scenarios are cases when there exists enough information about the targets before they cross so that there is essentially no ambiguity in the data association. Although the targets may have crossed in position, the state space of the targets is actually a higher-dimensional space, where

1. there exist velocity components as well as position components when using information from motion models and

2. there exist components which identify the class of the targets when using classification information.

Thus, in these higher-dimensional abstract spaces, the state of the two targets do not cross, and hence, there is no data association problem. Abstractly, using extra information means to work in the space which is the Cartesian product of the space of target positions and the space of the extra information. By working in this higher-dimensional space, the notion of "crossing" targets is different than targets crossing in position alone. Certainly if targets cross in the higher-dimensional abstract space, then they must cross in position, but the reverse is not necessarily true. Therefore, it is possible to resolve the data association problem when the extra information about the targets is discriminatory.

Using extra information to resolve the data association problem motivates the following solution for multiple target tracking when the extra information can discriminate amongst targets. This is pictured in Figure 1-3. When two targets are observed to be near each other (e.g. if targets are closer than some specified distance from each other), then we initiate a routine which attempts to find some discriminant between the two targets. While the targets are near each other, the extra information provided by the discriminant allows us to resolve any ambiguities in the data association. Then, after the targets are significantly far apart, we can resume tracking as usual. The assumption is that computing the discriminant expends extra computational resources, so if there is no need to compute it, then we should not.

Scenario 2: Delayed Discriminating Information

Let us consider a scenario when information which can discriminate the crossing targets is unavailable while the targets are near each other but can be obtained later

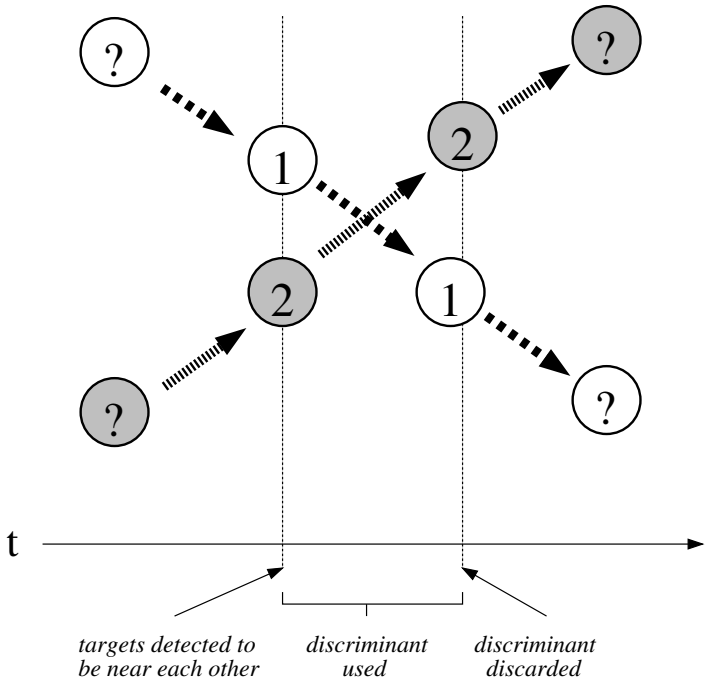

Figure 1-3: A possible solution for resolving the data association problem when there exists discriminatory information about the two targets (depicted as white and gray filled circles) before ambiguity in data association arises. The idea is to detect when there is the possibility of ambiguity in data association, use a discriminant to resolve ambiguities while the targets are near each other, and discard the use of the discriminant when targets are significantly far apart when there is no ambiguity in data association.

after the targets are significantly far apart again. Figure 1-4 shows one such scenario. There are two targets moving through the field, and before they cross, classification routines have determined that target 1 is non-metallic and target 2 is metallic based on readings from two magnetometers which target 1 and target 2 have passed within range. Assuming that the only discriminating information between target 1 and target 2 is whether the target is metallic or not, there is no discriminating information available to resolve the ambiguity in data association when the two targets cross because there is no magnetometer in range of the region where the two targets cross. Thus, the best we can do at this point is to create a data association hypothesis that of the two targets which split away from this region of ambiguity, one of them is a metallic target and its previous motion history originated from the bottom left of the field, and the other is a non-metallic target with previous motion history originating from the top left of the field. At a later time, if discriminating information becomes available as is the case when one of the targets travels in range of the magnetometer at the top right of the field, we can resolve the data association hypothesis created when the two targets crossed. In particular, if the magnetometer at the top right of the field determines that the target in range is metallic, then the resolution of the hypothesis is that target 2 originated from the bottom left of the field and proceeded diagonally to the top right of the field. Furthermore, target 1 originated from the top left of the field and proceeded diagonally to the bottom right of the field. On the other hand, if the magnetometer determines that the target in range is non-metallic, we have the other possibility. Target 1 proceeded from the top left of the field towards the middle and then turned upwards to the top right whereas target 2 proceeded from the bottom left of the field towards the middle and then turned downwards to the bottom right of the field.

When discriminatory information is delayed, the inference algorithm must perform the bookkeeping to keep track of all possible data association hypotheses. After discriminatory information is obtained, data association hypotheses can be resolved where possible. A potential problem with this bookkeeping is that the set of possible data association hypotheses grows exponentially with the number of targets that cross. The exponential growth of hypotheses is further compounded if target crossings occur again before previous data association hypotheses are resolved. Thus, it will be important to have an efficient representation of data association hypotheses when faced with limited computational resources such as memory.

Scenario 3: Discriminating Information Unavailable

The final scenario for crossing targets is the case when discriminating information is unavailable. The absolute best that we can do when there is no way to resolve the ambiguity in data association is to merely keep track of all data association hypotheses. As mentioned in Scenario 2, the set of possible data association hypotheses tends to grow exponentially, and since hypotheses where no discriminating information is

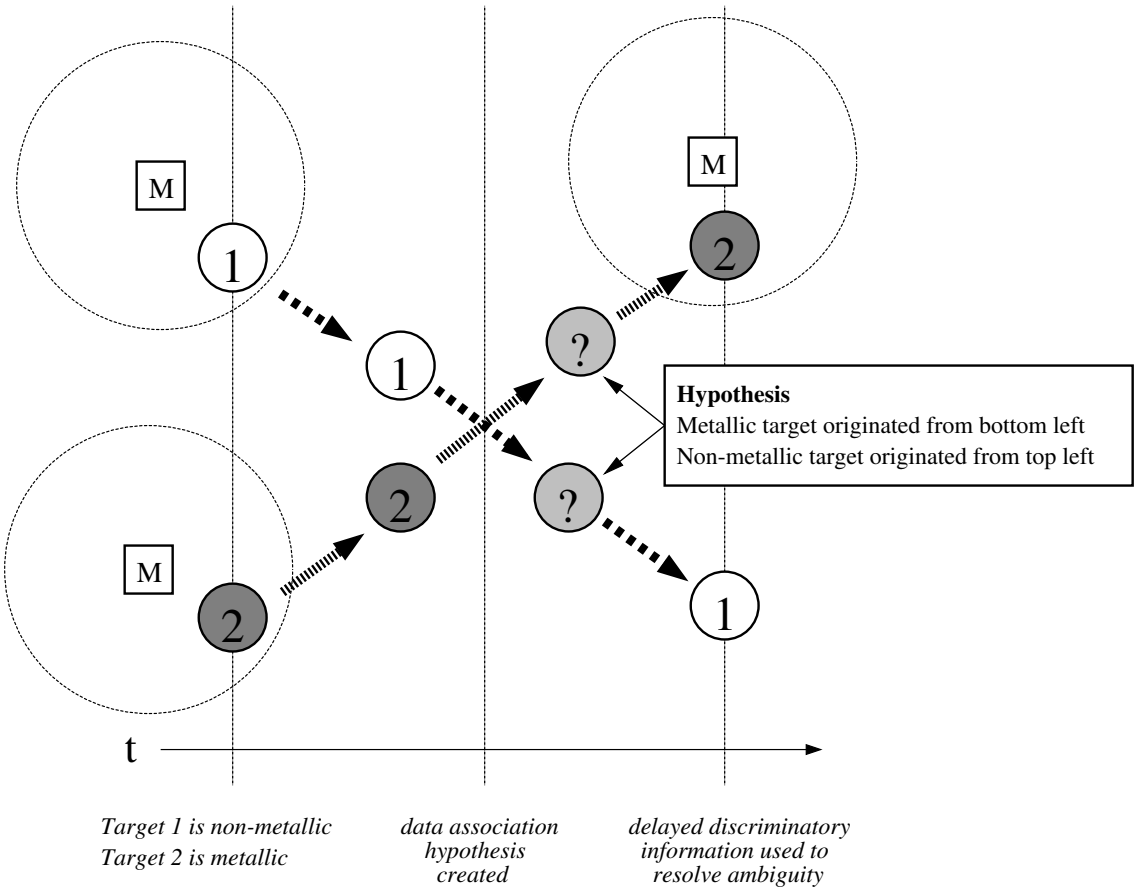

Figure 1-4: Two targets (depicted as white and gray filled circles) crossing where there exists discriminatory information between the two tracks but the information is unavailable until after the targets have split apart. Target 1 is non-metallic, and target 2 is metallic. The proposed solution for this scenario is to create a hypothesis of possible data associations and resolve the hypothesis when the discriminatory information becomes available. The figure shows the case when the magnetometer in the top right of the field determines that the target in range is metallic. The resolution of the data association hypothesis is that target 2 travelled from the bottom left to the top right of the field and that target 1 travelled from the top left to the bottom right of the field.

ever available will never be resolved, it is especially important to have an efficient representation of the data association hypotheses when faced with limited memory.

1.3.3 Scenario Summary

The scenarios presented in this section showed the kinds of information we wish to extract about targets in the field, the potential problems we face when extracting the appropriate information, and some high-level solutions for dealing with the potential problems. We have motivated that in order to achieve computational efficiency, it is desirable to initiate computations only when there is enough evidence to do so. This is the principle of information-based computation, and when we get into the details of designing localization, tracking, and classification algorithms on sensor networks, we will see that this principle can be widely applied to design computational efficiency into inference algorithms.

1.4 Overview of Proposed Methodology

In this section, we give an overview of the proposed methodology which embodies the principle of information-based computation. As mentioned before, there are three components which comprise this methodology:

1. the system architecture,
2. the layered information processing architecture, and
3. the hierarchy representation of information processing tasks.

Let us elaborate on these three components.

The *system architecture* refers to the physical characteristics of the system including the physical configuration and placement of processors and sensors, the possible communication pathways between processors, and a set of basic operations which can be performed by the system. Ultimately, the application of the methodology results in algorithms composed of basic operations of the system architecture so that the efficiency of the algorithms is determined by the computational resource requirements of the basic operations.

In particular, we will focus on system architectures with *distributed autonomous agents*. By an *agent*, we mean a physical unit with basic operations which can

1. sense the environment and collect data through sensors attached to itself,
2. process and extract information from the data collected, and
3. communicate data and information to neighboring agents.

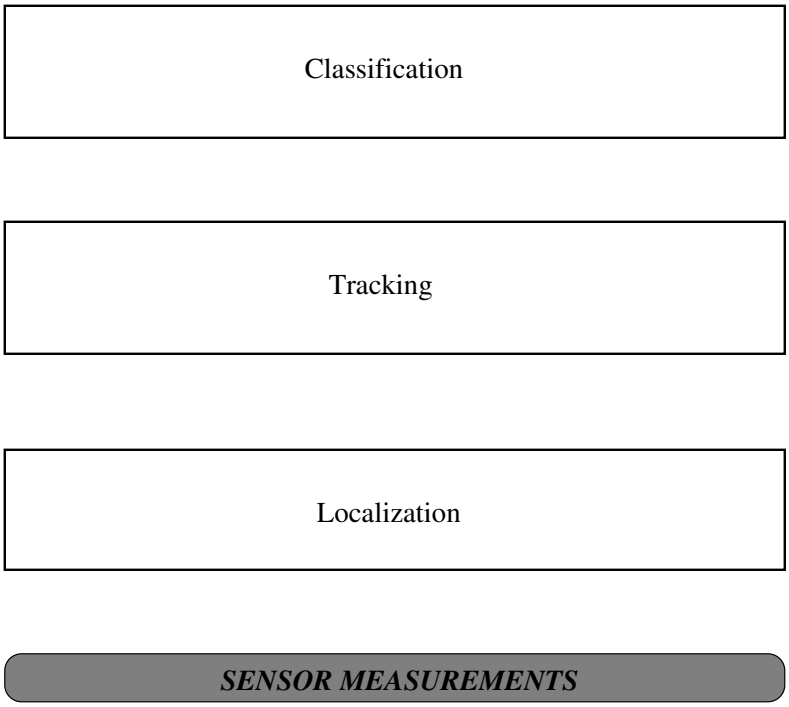

Figure 1-5: The inference tasks of localization, tracking, and classification arranged into layers to begin the construction of the layered information processing architecture. The data, indicated by the rounded gray box labelled with "Sensor Measurements", is placed at the bottom.

To say that the agents are *autonomous* means that the basic operations of one agent can be independent of the basic operations of other agents. Autonomy of agents allows the possibility of concurrent sensing, processing, and communication which can be beneficial for inference tasks with real-time constraints. To say that the agents are *distributed* means that the data collected by the sensors of an agent and the information resulting from the processing of data in this agent is immediately accessible to this agent only. The only way to share this data and extracted information with other agents is by communication so that distributedness is a constraint on what information and data is immediately accessible to each agent. Section 1.6 will elaborate on the issues presented by a system architecture with distributed autonomous agents.

Whereas the system architecture is a hardware architecture, the second component of the methodology, referred to as the *layered information processing architecture*, is a software architecture which organizes the global inference tasks into layers. The first step of arranging inference tasks into layers is shown in Figure 1-5. We say "global" because the layered information processing architecture is a global view of the the inference system without regard to the fact that the system architecture consists of

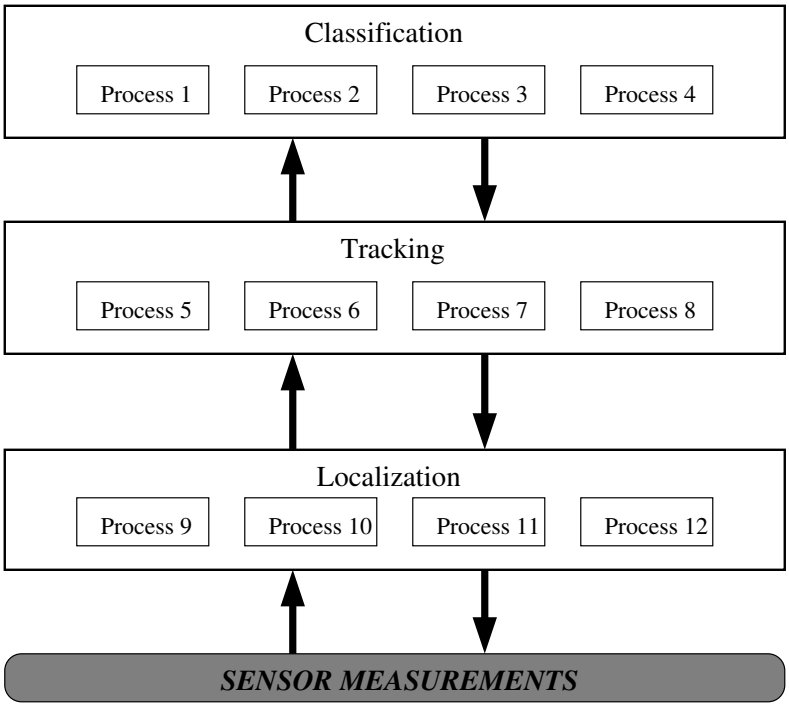

Figure 1-6: Decomposition of each inference task into a set of information processes. Interaction of information extracted from each process are indicated by black arrows. This is the *layered information processing architecture.*

distributed autonomous agents. The raw data observed by the system is placed below the inference tasks and is the observation of the entire system without regard (for now) to the constraints of a system architecture with distributed autonomous agents.

We decompose each global inference task into a set of global subtasks, which we refer to as *information processes* as shown in Figure 1-6. By specifying the interaction between information processes within layers and between layers, the implementation details of each individual information process can be abstracted. This abstraction allows us to design multi-threaded algorithms on a system architecture where the information processes from several layers can be running concurrently. Furthermore, we are making the individual information processes modular so that each of them can be implemented separately as long as the interactions between them are preserved. We will be particularly interested in the design of inter-layer interactions so that the principle of information-based computation can be exercised to result in the design of an efficient inference system. Figure 1-6 is the layered information processing architecture, and Section 1.5 will elaborate on this component of the methodology.

We make a distinction between "algorithms which implement information processes" and "algorithms which run on the system architecture". Algorithms which

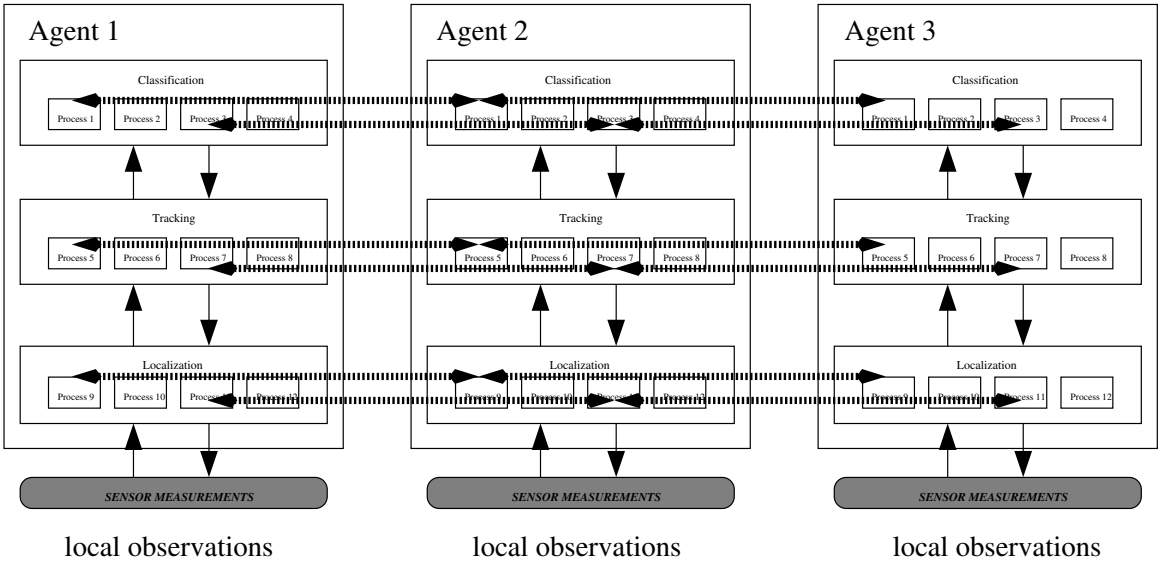

Figure 1-7: Distributed algorithms for each individual process are implemented for system architectures with distributed autonomous agents. Communications may be necessary between agents when generating distributed algorithms for each information process as indicated by the horizontal, dashed arrows. Sensor measurements observed by each agent is local.

implement information processes are the abstract computations to be performed to implement the information processes in the layered information processing architecture. On the other hand, an algorithm which runs on the system architecture refers to the sequence of basic operations to be performed on a system architecture. The relationship between the two is that every information process on the layered information processing architecture must be translated into an algorithm on the system architecture. The analogy in terms of computer programming is that the algorithms which implement an information process are to *source code* as algorithms which run on the system architecture are to *machine code*. The bridge between source code and machine code is the *compiler*. For this methodology, the bridge between the information process and the algorithm on the system architecture is the final component of the methodology, the *hierarchy representation*.

The hierarchy representation is a meta-representation which allows us to translate algorithms which implement information processes of the layered information processing architecture into algorithms running on the system architecture. Figure 1-7 shows the case of generating distributed algorithms for information processes on a system architecture with distributed autonomous agents. Since there exist several implementations of any information process, and thus, several possible hierarchies for an information process, the problem of constructing an efficient algorithm on a

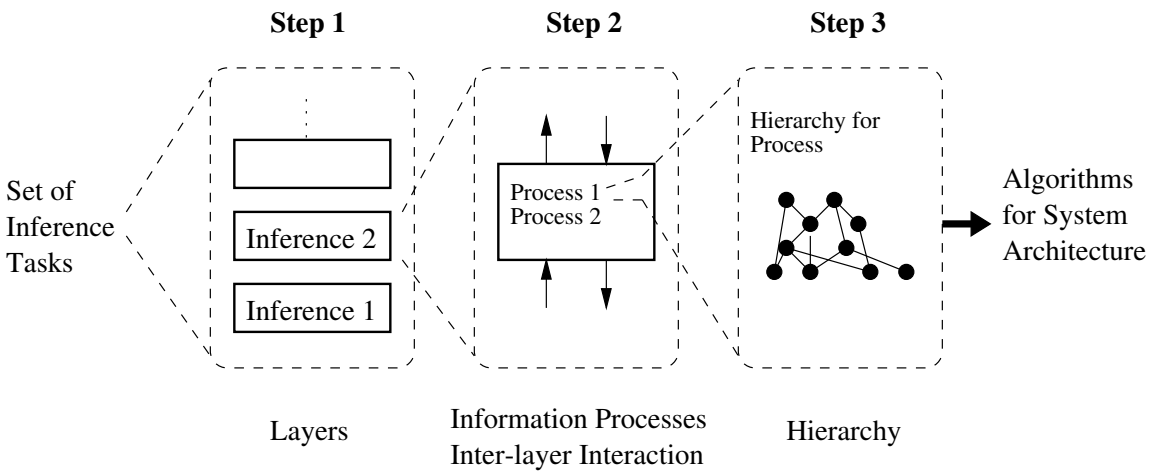

Figure 1-8: Methodology for designing computationally efficient algorithms for inference systems.

system architecture which computes a particular information process becomes one of constructing a good hierarchy representation for the information process. Therefore, it is in this part of the methodology where the computational resource requirements of an algorithm running on the system architecture which implements an information process for some inference task is optimized and distributed algorithms for system architectures with distributed autonomous agents are generated.

A hierarchy representation is not necessarily needed to translate all information processes into an algorithm on the system architecture if it is obvious how such a translation can be accomplished. The interesting cases of when a hierarchy representation is useful for this translation is when an information process must be translated into a set of distributed, local algorithms running on multiple, distributed autonomous agents. Thus, the case when the hierarchy representation is most useful is when we must construct *distributed* algorithms.

Summarizing, the methodology for constructing efficient inference algorithms for a specific system architecture proceeds as shown in Figure 1-8.

1. Given the set of inference tasks to be performed, arrange the tasks into layers to begin building the layered information processing architecture.

2. Decompose the inference tasks of each layer into a set of information processes, and design the inter-layer interactions of processes between layers. The design of the inter-layer interactions has ramifications on the computational efficiency of the resulting inference system which we will discuss in Section 1.5. Note that the layered information processing architecture is a holistic view of all information processes and the interactions among them.

3. Translate all information processes into algorithms on the system architecture. If needed, use the hierarchy as a tool for translating an information process into an efficient algorithm on the system architecture. The computational requirements of the basic operations of the system architecture are taken into account in this step so that the optimization of computational resource usage of an algorithm running on the system architecture implementing an information process is performed here.

Although we have introduced the hierarchy as a meta-representation for constructing a computationally efficient algorithm running on a system architecture which implements an information process, the more abstract notion of the hierarchy is an encoding of a complex computation by a network of simple computations. In this more general sense, we can consider the layered information processing architecture as a hierarchy representation of the "complex computation" of general inference in terms of a network (in this case, organized into layers) of the "simpler computations" of specific inference tasks. Thus, we speculate that the hierarchy can be used to decompose inference into inference tasks and inference tasks into information processes from which layers of a layered information processing architecture can emerge. However, due to the computational intractability of encoding complex computations by a network of simpler computations, we will consider the first two steps of the methodology to be handcrafted by the human engineer and only automate the use of the hierarchy in the last step. Due to the potential generality for using the hierarchy representation at both the macro-level of designing a layered information processing architecture and at the micro-level of translating information processes into algorithms, we emphasize the hierarchy and will devote an entire chapter (Chapter 5) on it in this thesis.

The rest of this chapter is organized as follows. Section 1.5 will propose how to organize the inference tasks into layers for the layered information processing architecture, how to design the information processes within a layer, and how to design inter-layer interactions. Section 1.6 then describes the issues involved when the system architecture consists of distributed autonomous agents. Section 1.7 highlights the important aspects of the hierarchy representation which will be the meta-representation we use for translating information processes into computationally efficient algorithms running on a system architecture with distributed autonomous agents. The main contributions are listed in Section 1.8, followed by a description of chapter content and list of experiments performed to demonstrate the claims of this thesis in Section 1.10.

1.5 Layered Information Processing Architecture

We consider inference problems involving the extraction of many kinds of information from data collected by a large number of sensors. Since the kinds of information we wish to extract are numerous and different kinds of information are usually related, we

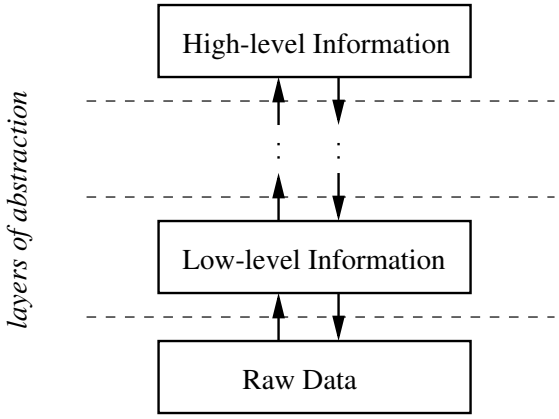

Figure 1-9: A layered information processing architecture. Arrows indicate the direction of the possible flow of messages interfacing the abstraction layers.

propose a layered information processing architecture for the extraction of information and the management of the relationships between different kinds of information. We will describe the general ideas of the layered information processing architecture in this section. Section 2.6 will present the specific layered information processing architecture demonstrating these ideas for the application of battlefield awareness.

Layers are stacked one on top of the other, and each layer is responsible for a specific inference task. We position inference tasks in this layered architecture according to how much computational effort is required to extract the desired information from the raw data. Those inference tasks which require less computational effort are positioned in lower layers, and those inference tasks which require more computational effort are positioned in higher layers. Concretely, we can measure computational effort according to how much of the computational resources of time, memory, communication, and energy is needed to compute the information. Intuitively, the kinds of information which require less computational effort correspond to low-level concrete information about the raw data whereas the kinds of information which require more computational effort correspond to high-level abstract information as shown in Figure 1-9. For example, consider raw data from a microphone which is essentially a sampled sound pressure signal which originated from somebody speaking in the vicinity. Low-level information we can extract from the signal include the intensity and frequency content of the signal. These can be computed by computing the mean square and the fast Fourier transform (FFT) of the signal respectively. Higher-level information could be extracting phonemes, words, and sentences from the acoustic signal. Extracting this kind of information requires classification routines utilizing background knowledge like the grammar of the language.

Each layer consists of a set of *information processes*, which are functions of the raw

data and information extracted by other layers. To say that an information process is evaluated or computed means that the associated function has been evaluated. The result of the function evaluation is referred to as the information extracted from the information process. After information processes are computed, the resulting information resides in memory in the layer.

To design efficiency into the system, we wish to implement the general principle of information-based computation which will be embodied in this layered information processing architecture by a message passing scheme between layers. These inter-layer interactions result in a selection mechanism which selects which information processes to compute based on what information has already been extracted. The possible messages passed between layers are of two types.

1. **(Information Message)** An information message passes some information residing in a layer to another layer.

2. **(Query Message)** A query message passes a request to another layer for certain information. The effect of this message is that the other layer will compute the appropriate information process if necessary and respond with an information message.

The effect of passing a message to another layer is the possibility of initiating computations in that layer. This effect is clear for a query message; however, it is also possible that the availability of new information in a layer can initiate computations in that layer if there are mechanisms which trigger computations based on the information known in the layer. Thus, the important effect of passing messages is to initiate extraction of certain information based on information already known.

With this kind of software architecture, layers of abstraction emerge in that the particular implementation details of computing an information process can be considered a black box to processes in other layers since the only effect of a message from another layer is to initiate a computation. These layers of abstraction allow us to design the implementation of information processes of each layer independently as long as the specified initiations of certain computations are satisfied. There are no restrictions on the number of information processes that may be computed concurrently so that this layered information processing architecture supports the design of multi-threaded algorithms running on a system architecture.

1.5.1 Top-down and Bottom-up Processing

The inter-layer interactions on these layered architectures can be grouped into two main categories: top-down processing and bottom-up processing. The two processes are essentially complements of each other in that in top-down processing the flow of computations proceeds from top to bottom and in bottom-up processing the flow of computations proceeds from bottom to top. The two processes may seem symmetrical, but because the information extracted at higher layers tend to be abstract

and the information extracted at lower layers are concrete, there is a philosophical difference between the two processes. To illustrate this philosophical difference, let us consider the problem of object recognition in computer vision. We choose the domain of computer vision to illustrate this philosophical difference in order to show the pervasiveness of categorizing algorithms as top-down or bottom-up. However, we stress that this categorization is not an absolute one because what algorithms we consider top-down and bottom-up depends on what information we consider abstract and concrete.

Consider the problem of recognizing all faces that may be present in black and white photographs. Let us assume that we have 3-dimensional parametric models of faces which include parameters for orientation and position, lighting conditions, and viewing angle. Furthermore, we have a model of the image generated on the photograph in the presence of a given set of faces with given parameters. The raw data is the photograph, and the abstract information we wish to extract are the parameters of the faces that are present in the photograph.

A top-down approach to extracting information about faces in a photograph would be to generate the images of every face model with every possible parameter value. Then, with some appropriately chosen matching function, we can compare these generated images with the photograph and determine whether a match is found. The information extracted from this straightforward computation will be a set of faces and their associated parameters which are present in the photograph.

A bottom-up approach for extracting information about faces in a photograph would be to compute a set of features which contain information about the presence of faces which are easily computable in various parts of the photograph. From the results of this feature extraction, further processing can be performed which aggregates these features and computes other high-level features until there is enough evidence to declare a face is present in the image. In the face recognition literature specifically for recognition of faces which are facing the camera, the popular features extracted are the detection of two dark spots for the eyes, a light area corresponding to the forehead, and a light area for the nose. Then, aggregation is performed to determine whether these features are in the correct relative positions from each another. If there is a match, then the algorithm declares that a face has been found.

In both of these approaches, one can imagine a set of layers of representation. The top layer consists of the space of faces since we consider faces to be the "abstract" information. The bottom layer consists of the space of photographs since we consider photographs to be the "raw data". Although in the top-down approach, we did not necessarily make use of any intermediate layers, the primary characteristic of the top-down approach is that computation begins with the abstract objects, which in this case are the 3-dimensional parametric face models, and proceeds down to the raw data. In the bottom-up approach, we begin computation with the raw data, compute features which we consider to be some intermediate layer representation, aggregate those features which we consider to be some higher intermediate layer representation,

and ultimately, recognize faces which are the representations of the top layer. The characteristic of the bottom-up approach is that computation begins at the bottom-layer with the raw data and proceeds upwards until the top layer representations are computed.

In both top-down and bottom-up processing, we manipulate the representations we begin with and extract information to help us convert the current representation into the representation at the next layer. In this sense, we can consider the general processing of extracting information to be the same for both top-down and bottom-up processing with the only difference being the starting point of the computations. If we consider information extraction as a search for matches in some information space, the mechanism for exploring the neighborhood around the current point in the information space is the same for both top-down and bottom-up processing; however, the initial starting point for this search is different.

On the other hand, the interpretations of what those computations are is different. We consider top-down processing to be a "pull" type of information extraction approach. The idea is that we know what we wish to know, so let us query for information about it. Conversely, we consider bottom-up processing to be a "push" type of information extraction approach. The idea is we know what the data is, so let us see what information can be easily extracted out of data which may be useful. Some examples of top-down processing approaches in computer vision are automatic target recognition (ATR) problems [21] using templates and other so-called model-based approaches. Examples of bottom-up processing approaches in computer vision include edge detection, segmentation, and what is more generally known as feature extraction.

With limited computational resources, top-down and bottom-up processing have some significant differences in performance. Because top-down processing begins with the abstract models and the space of parameters for these models is too large to search through exhaustively, we must limit computation to a subregion of the search space. The primary drawback with top-down processing is that it is computationally intractable to search the entire space by adopting solely a query type of information extraction; however, for the set of model parameters for which information is queried, the error in recognition is small. Conversely, bottom-up processing suffers from the difficulty in determining what features to compute from the data. Thus, bottom-up processing tends to result in performance much worse than in top-down processing because the features tend to be error prone. On the other hand, bottom-up processing can search a larger region of the search space than top-down processing albeit at the expense of degraded performance.

To get the best of both worlds, it has long been conjectured that top-down and bottom-up processes should complement each other in some fashion. In [25], Ullman presents one possible way to combine top-down and bottom-up processing in a data structure he calls counterstreams. The idea is simple and can be stated in terms of a search through an abstract information space. The top-down process is a search where

the initial starting point corresponds to the top layer representations. The bottom-up process is a search where the initial starting point corresponds to the given data. Both processes search through the information space, possibly in a random fashion, while leaving behind a primed trail of the set of points it has searched. Then, if the two processes end up meeting at some point, a match is declared. Because a primed trail was left behind, it is possible to determine the path from object to data which led to the match. By strengthening the probability of following this path in the future for both processes, the idea is that on future occurrences of the same image, the match between object and data will be more efficient. To be more concrete, if the representations of the top layer are 3-dimensional models and the representations of the bottom layer are images, the search space is the set of sequences of transformations from 3-dimensional models to images. If a sequence of transformations is found between a particular 3-dimensional model with specified parameters and the given input image, then a match is declared between the two representations. The motivating reason for starting the search for a match from both the top layer and the bottom layer is because some transformations are easier to compute from the top layer representation and some transformations are easier to compute from the bottom layer representation. For example, the effect of a 3-d rotation to a 3-d model of an object is a linear transformation on the 3-d model whereas the effect of the 3-d rotation on the images of an object is a non-linear transformation on the image. Operations like scaling, brightness normalization, skew, and edge detection are easy operations to perform on an image so that the effect of these transformations should originate from the image.

Ullman has proposed a way of combining top-down and bottom-up processes. We will present a method for designing an interactive flow of computations going up and down a layered information processing architecture. Our idea is to design specific interaction mechanisms between layers so that a *selection mechanism* emerges that selects when to initiate a computation. This selection mechanism is an instance of applying the information-based computation principle, and the idea is that by choosing which computations to explicitly evaluate in an intelligent manner, the overall system will extract information more efficiently.

1.5.2 Novel Aspect: Complementary Processes

In top-down processes, the information to be extracted from the higher layers initiated computations in the lower layers, whereas in bottom-up processes, the information extracted from the lower layers initiated computations in the higher layers. It is clear that in order to combine top-down and bottom-up processes, messages arriving from *both* higher and lower layers should be able to initiate computations. However, aside from allowing initiations from both higher and lower layers, we will further propose a particular characteristic of the set of information processes within each layer.

We propose designing a layered information processing architecture with the fol-

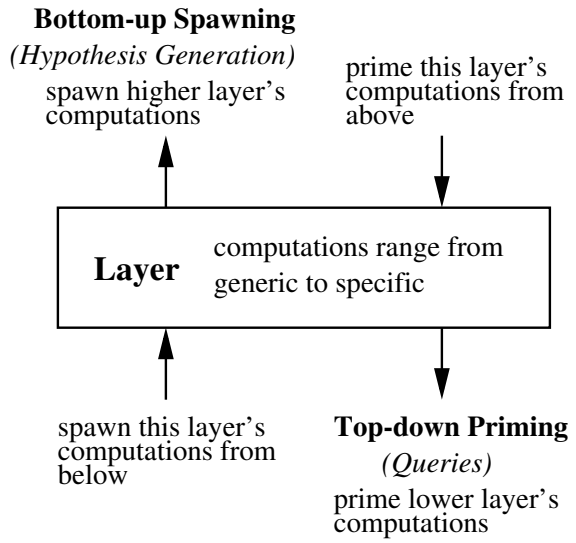

Figure 1-10: The proposed interaction between layers to support the complementation of top-down and bottom-up processing. The arrows going outwards indicate how computations of this layer affect computations of other layers. Arrows coming inwards to this layer indicate how computations of adjacent layers affect computations of this layer.

lowing characteristics.

Intra-layer The information processes within a layer can extract information which ranges from generic to specific. The idea is that the computation of generic information is cheap and is always somewhat informative whereas the computation for specific information is expensive and may either be highly informative or not informative at all. Thus, we desire the information extracted in each layer to proceed from generic to specific.

Top-down Priming *(Queries)* Higher layers can prime what information should be extracted at lower layers. The idea is that as a higher layer incrementally extracts more information, it can initiate the more specific information processes in lower layers.

Bottom-up Spawning *(Hypothesis Generation)* Depending on the information passed up from a lower layer, we can hypothesize what information should be extracted at the higher layer. The idea is to compute for the most generic information which incrementally adds to the information known at this layer.

Figure 1-10 shows the above characteristics for each layer of the layered architecture. Note that priming and spawning have essentially the same kind of effect to the adja-

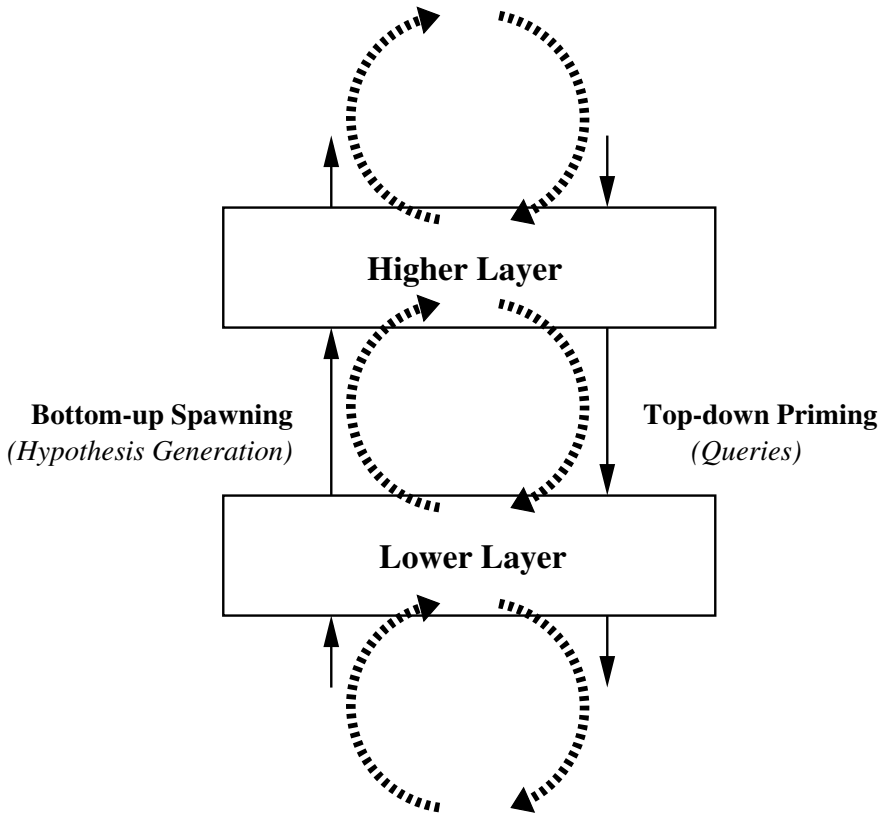

Figure 1-11: The proposed interaction between layers results in loops of computation between layers which proceed from generic to specific. The loops between other layers are processed concurrently.

cent layers except that one heads down the layered architecture and the other heads up the layered architecture respectively.

The inter-layer interaction of the layers proposed above results in a dynamic process of computation in loops between layers which causes the computations within a layer to proceed from generic information extraction to specific information extraction. Furthermore, the loops between each layer can initiate computations concurrently. Figure 1-11 shows the loops of computation occurring between layers. Let us elaborate on the progression from extracting generic information to specific information within a layer. Initially, the lower layer will extract some generic information. This information is passed up to the higher layer which spawns some computations in the higher layer. The information extracted from the computations in the higher layer will prime the lower layer to compute for more specific information. Note that in this one loop, the information being extracted at the lower layer has become incrementally more specific. As this process is iterated, the information to be extracted

at both the higher and lower layers can become more specific. Furthermore, since computations are spawned at the higher layer, the loops of processing between layers higher up will begin to emerge and computations will begin to proceed from generic to specific at all layers. The emerging selection mechanism is to spawn certain specific computations only when there has been enough generic information to warrant their evaluation.

The proposed intra-layer computation characteristics and inter-layer interaction characteristics were motivated by the principle of information-based computation. The idea is to initiate computations at each layer when there is good evidence that the result of the computation can be useful. The result is a concurrent progression from generic computations to specific computations within each layer.

The above discussion has been highly abstract, but we will demonstrate the range of generic to specific computations within layers, top-down priming interactions, and bottom-up spawning interactions in the application of battlefield awareness in Section 2.6. Furthermore, Chapter 3 will demonstrate how to use this proposed layered information processing architecture to construct algorithms to be run on a sensor network.

1.6 Distributed Autonomous Agents

The particular system architecture we are interested in is one consisting of distributed autonomous agents. Clearly, the advantage of the availability of multiple agents is the possibility of concurrent sensing, processing, and communication; however, this advantage comes with the added complexity of coordinating the individual agents to act together in a way which achieves some global behavior. Thus, the primary problem with a system architecture consisting of distributed autonomous agents is how to distribute a global information process into a set of local algorithms.

The problem of distributing computation falls under the subject of the relationship between local processes and emergent global behavior. There are two problems in this local vs. global relationship. We will refer to the generation of local algorithms which exhibit some desired global behavior as the problem of *synthesis*. The hierarchy representation will be the key tool we use in order to generate these local algorithms which compute some global information process. Furthermore, we will see that a whole family of local algorithms which exhibit the same global behavior can be generated with the hierarchy representation. The importance of this characteristic is that optimizing computational efficiency of local algorithms using the hierarchy is possible. The inverse to the problem of synthesis is the problem of *analysis*. This problem is to determine what global behavior emerges from the concurrent operation of local algorithms. Again, the hierarchy representation seems to be a possible tool for studying emergent global behavior from local processes since a hierarchy can be constructed from the set of local algorithms; however, we will not pursue how the

hierarchy can be used for the analysis of emergent global behavior and concentrate solely on the problem of synthesis and the role the hierarchy has to play in that problem.

Incorporating *cooperation* into the behavior of autonomous agents will be important in constructing efficient local algorithms for each agent. In order for autonomous agents to act efficiently together, communications between agents is a necessity so that what is communicated and how these communications are minimized is of primary importance. Let us consider an analogy to illustrate the importance of efficient communications. Say we have a committee of experts in different fields, and they are assigned the global task of understanding some unexplained phenomenon recently observed. Each committee member is an autonomous agent with a different knowledge base and has the ability to ask for information from other experts and provide information to other experts. The following four characteristics will be important for this committee to accomplish their task efficiently.

1. **Independent Filtering** Each committee member should look at the phenomenon, and determine a set of possible explanations to the best of his knowledge. Furthermore, for each possible explanation, he should determine what added information would refine his set of possible explanations.

2. **Knowledge of Other's Expertise** Each committee member should have some high-level knowledge about the expertise of each other. In this way, a committee member will be able to determine who could benefit from his knowledge and who could add to his own knowledge. This knowledge is primarily used for determining which committee members should collaborate.

3. **Informative Communications** For those committee members which collaborate, the information shared should add to the understanding. If the level of understanding is not increased, then the communications that occurred were useless and may as well not have occurred at all.

4. **Cooperation** For our application, it is only necessary that only one committee member understand the phenomenon. Other applications can have requirements where every committee member must understand the phenomenon. These different requirements certainly have an effect on how much collaboration is necessary. Thus, for the case when only one committee member must understand the phenomenon, in order to minimize the collaborations taking place, the committee member with the most potential understanding should take a leader role while others concede and take a follower role to avoid a lot of unnecessary communication. It is possible to assign leaders and followers a priori; however, a better option would be for leaders and followers to emerge depending on the phenomenon observed in order to avoid situations where a lot of communication cost is incurred so that the least knowledgeable committee member can obtain enough information to understand the phenomenon.

The analogy above will help guide our intuition on how to generate distributed local algorithms which implement some global information process. In particular, we refer to the design of good heuristics to construct hierarchies which is discussed in Section 4.4.

1.7 Hierarchy Representation

A major part of this thesis is the presentation of the hierarchy representation which will be specifically used for translating an information process in the layered information processing architecture into an algorithm on the system architecture. In particular, the hierarchy representation will be used to systematically convert global information processes into computationally efficient local algorithms on system architectures with distributed autonomous agents. Chapter 4 will present the details of this representation applied to target localization, and a general presentation of the hierarchy will be given in Chapter 5 so that we will only outline the main ideas behind the representation in this section.

1. **All computations are functions.** The first idea is to consider all computations of any information process as a function from the space of possible data, say Z, to some abstract information space V. To compute an information process means to evaluate a function

$$f : Z \to V$$

given some data $z \in Z$. Thus, the hierarchy representation is about representing a function f in some manner. For the most part, we will assume that V is the reals $\mathbb{R}$, and that Z is n-dimensional Euclidean space $\mathbb{R}^n$.

2. **A hierarchy represents functions as compositions of other functions.** We have described the hierarchy representation as an encoding of a complex computation in terms of a network of simpler computations. By this, we mean to represent a function of many variables, say

$$f(z_1, z_2, z_3, z_4) \ ,$$

by compositions of functions of less variables, say as

$$f(z_1, z_2, z_3, z_4) = g(h(z_1, z_2), l(z_3, z_4))$$

where g, h, and l are all functions of two variables. This is a network of simple computations because we envision a feedforward network type of representation as shown in Figure 1-12. We refer to this network as an implementation of the computation of f because it specifies a set of simpler computations that

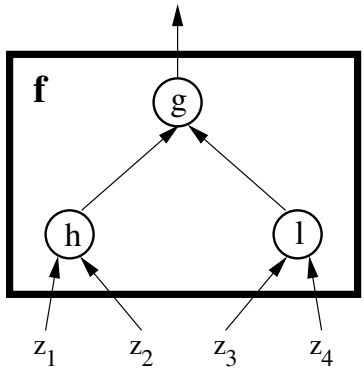

Figure 1-12: A graphical representation of a complex computation as a network of simpler computations.

must be performed and how the results of these simpler computations are to be used in order to compute f. For instance, after evaluating h and l, the results of these evaluations are fed into the computation of g to result in the overall evaluation of f. Note that there can exist several networks of simpler computations which represent the same complex computation, so that there are several possible implementations for the same computation f.

If a cost is assigned to each computation, g, h, and l, then we can measure the cost of this particular implementation of f. There are also further refinements which we wish to take advantage of. For instance, if it turns out that the result from the evaluation of h for given z_1 and z_2 means that the evaluation of g is independent of the result from the evaluation of l, then there is no need to evaluate l on z_3 and z_4. This can save us the cost of having to evaluate l as well as having to gather the data z_3 and z_4.

3. **The hierarchy is to be used as a meta-representation for bridging the gap between the function associated with a computation and the particular algorithm implementing it on a system architecture.** In order to translate an information process into an algorithm, the idea is to represent a complex computation as a network of simpler computations which correspond with the basic operations of the system architecture. Then, the resulting hierarchy can be considered a meta-representation of an algorithm for the system architecture.

   In particular, for the problem of distributing a computation over several processors, the idea is that once the computation is decomposed into a network of simpler computations, the simpler computations can be assigned to different processors. Thus, we separate the problem of distributing a computation into two parts. First, we decompose a complex computation into a network of

simpler computations. Then, with a given decomposition, we assign the computations in the hierarchy to different processors. A side effect of these processor assignments is the necessity of communications since certain computations rely on the results of other computations. The decomposition of complex computations into simpler ones allow us to distribute computations.

The choice of a decomposition certainly affects the computational resource requirements of the resulting algorithm on a system architecture, so that the process of decomposing a computation cannot be oblivious to the system architecture.

4. **All computations encoded in a hierarchy need not be evaluated all the time.** The idea here is that, although the hierarchy seems like nothing more than a feedforward network, there is no reason why we must evaluate the computations in a hierarchy as in a feedforward network. What is desired is to evaluate those computations for which there is enough evidence to warrant their evaluation. Thus, given some *evaluation requirement*, we will derive an *evaluation algorithm* on the hierarchy which results in a partial evaluation of the computations of the hierarchy. The set of evaluation algorithms reduces down to labelling edges in the hierarchy as uplinks or downlinks. The key relationship for choosing uplinks and downlinks will be the input-output relationship of the functions encoded in the hierarchy.

   These evaluation algorithms on a hierarchy is the formalism behind the selection mechanism which was discussed in Section 1.5 regarding the interaction between layers in the layered information processing architecture. Furthermore, the notions of complementing top-down and bottom-up processes is captured by labelling the edges in the hierarchy as uplinks or downlinks.

Parts 1 and 2 above are not novel ideas since computations have long been been considered abstractly as functions and, furthermore, feedforward networks, like neural networks, are representations of functions in terms of compositions of other functions. However, parts 3 and 4 are novel. By considering the hierarchy as a meta-representation, we have a representation from which we will be able to enumerate, in theory, the set of all possible algorithms which implement any computation. The construction of a particular decomposition is referred to as a *graph topology construction* for that computation. Finally, since there is no reason to evaluate the computations encoded in a hierarchy as in a feedforward network, we will define a mechanism to select computations to be evaluated based on the results of previous evaluations. This partial evaluation of computations in the hierarchy will be important for scenarios of limited computational resources since it will not be possible to evaluate all computations all the time. Note that this idea for partial evaluation is an instance of the principle of information-based computation. Once the graph topology is combined with an evaluation algorithm, the result is what we refer to as a hierarchy.

Although we will only make explicit use of the hierarchy representation for the translation of information processes to algorithms, the conceptual ideas behind choosing evaluation algorithms on a hierarchy formalize the choice of a message passing scheme in a layered information processing architecture from which the sought selection mechanism emerges. In particular, the information processes of each layer will be the "simpler" computations which are the elements of a macro-hierarchy representation of the "complex" computation of general inference in the entire layered information processing architecture. The message passing scheme in the layered information processing architecture could then be encoded by uplinks and downlinks in the appropriately constructed macro-hierarchy. Any choice of uplinks and downlinks results in an evaluation algorithm which only evaluates those information processes that need to be evaluated. Thus, a selection mechanism has emerged.

This thesis demonstrates the use of the hierarchy representation to construct computationally efficient algorithms in system architectures with distributed autonomous agents. Furthermore, the conceptual ideas behind the hierarchy representation play a large part in handcrafting the layered information processing architecture and the associated message passing scheme.

1.8 Contributions of Thesis

We categorize the contributions of this thesis into three parts:

1. the *overall architecture* for designing a general inference system,
2. the *computation and communication structure* for systematically distributing global algorithms into efficient local algorithms, and
3. distributed algorithms for the specific application of sensor networks to battlefield awareness.

1.8.1 Overall Architecture

We present an overall architecture in this thesis which organizes all inference tasks we wish the system to perform and the dependencies between them. The organizational structure employed is the *layered information processing architecture* which we introduced in Section 1.5 and will discuss in detail for the battlefield awareness application in Chapter 3. The layered information processing architecture is a culmination of several conceptual ideas in order to design computational efficiency into the inference system.

Holistic View

We take a holistic view of the design methodology for constructing efficient distributed algorithms on system architectures with distributed autonomous agents. That is, we

first consider the entire system without concern for the distributed nature of the system architecture. As shown in Figure 1-8, global inference tasks are decomposed into individual information processes. These individual information processes are still global in the sense that the constraints imposed by a system architecture with distributed autonomous agents have still not been considered. The important aspect of decomposing a global inference task into a set of global individual information processes is to determine what information is gained from these processes and how this information can be useful for other processes. It is these dependencies of information which we wish to exploit. Then, once these dependencies have been organized in the layered information processing architecture, we will finally consider how to distribute the implementation of each of these individual information processes. The *hierarchy* can be employed to generate a distributed algorithm for each of these individual information processes.

Exploiting Information of Other Tasks

When we take a holistic view of the system and decompose the inference tasks into individual information processes, our purpose in decomposing these tasks is to examine how information from one information process can be exploited to implement another information process. By assuming that information from other information processes are available, the solution to some information processes are significantly simplified. Without assuming the availability of this extra information, a solution to the information processes is more difficult. Chapter 3 will demonstrate how exploiting the information extracted from different information processes for the inference tasks of localization, tracking, and classification can significantly simplify the solution to the individual information processes.

Certainly, there must be an order in which information is extracted by the inference system. For example, if information process A assumes information is available which is extracted from information process B and information process B assumes information is available which is extracted from information process A, a chicken-and-egg problem emerges since neither A nor B can extract information unless the other process has extracted information. Such an ordering of which information processes should extract information first must be built into the inference system so that the chicken-and-egg problem does not emerge which is another reason why an organizational structure like the layered information processing architecture is needed.

Algorithms Initiated On-Demand

The algorithms for implementing each individual information process are distributed algorithms which, in general, will require communications to occur between agents of the system architecture. Thus, it is important that these distributed algorithms are initiated only when there exists enough information that the algorithms will extract significant information so that the inference system is computationally efficient. The

conditions which initiate algorithms will be referred to as *triggers*. Note that these triggers are the embodiment of the information-based computation principle which initiates certain algorithms to be run when there is good evidence that the information extracted from them are useful.

Estimates of Reasonable Accuracy

As mentioned in the beginning of this chapter, we require that estimates for the various inference tasks only be of reasonable accuracy depending on performance specifications of the system. The general tradeoff in mind is that highly accurate estimates require more computational resources, and since we are in the realm of limited computational resources, we only wish to compute estimates which are considered to be of reasonable accuracy so that the computational resource requirements can be minimized.

Integration of Top-down and Bottom-up Processing

In Section 1.5.1, we discussed top-down and bottom-up processing and the motivation behind allowing information to travel both up and down the layered information processing architecture. By allowing this two-way flow of information, we have more choices for exploiting the information between information processes and, thus, more opportunities for significantly simplifying the implementation of them.

1.8.2 Computation and Communication Structure

The second major contribution of this thesis is a computation and communication structure we call the *hierarchy*. It is a meta-representation for the automated generation of efficient local algorithms implementing global tasks with communications between agents. In the holistic view of the methodology, the hierarchy is used in the last step of generating distributed algorithms for each of the individual processes in the layered information processing architecture. The hierarchy was introduced in Section 1.7, will be applied to the task of target localization in Chapter 4, and will be presented in its most general form in Chapter 5.

As in the layered information processing architecture, the hierarchy is a structure which is a culmination of several concepts. In particular, the concepts for computation efficiency mentioned in the previous section are incorporated by the hierarchy. Furthermore, two more conceptual ideas are present in the hierarchy in order to generate distributed algorithms for global processes.

Compositional View

As in the holistic view of the layered information processing architecture, the hierarchy incorporates a compositional view of an information process. The idea is to abstractly

consider any information process as a function evaluation. Then, we represent the function by compositions of primitive functions. By representing functions in this way, we can then begin to consider how to distribute the information process over multiple agents.

Agent Assignments

The final conceptual idea to distribute an information process is that the different parts of the composition of functions can be assigned to be evaluated by different agents. Since the evaluation of a function requires the availability of inputs to the function, any agent assignment requires communications to take place to ensure the availability of inputs. Thus, we can view the generation of efficient distributed algorithms to be an optimization of an agent assignment over a functional decomposition of a function corresponding to an information process.

1.8.3 Battlefield Awareness with Sensor Networks

By applying the layered information processing architecture and hierarchy which embody the conceptual ideas discussed in the previous contribution sections, we have the following contributions to the specific application of sensor networks to battlefield awareness:

1. detection and appropriate resolution of ambiguity in data association for multiple target tracking by incorporating higher-level information,

2. distributed algorithms for localization, tracking, and classification for battlefield awareness on sensor networks, and

3. the automated generation of local communication algorithms for each sensor node for the task of target localization adapted to the sensor network topology.

1.8.4 Focus

We mention here that the contributions of this thesis are not on the most optimal implementation of the individual processes of the layered information processing architecture. The layered information processing architecture is a modular architecture in the sense that the particular implementation of the individual processes can be implemented separately as long as the information assumed to be available to each process and the information extracted by each process is specified.

The focus of this thesis is on how to incorporate interactions of information between processes in the layered information processing architecture so that the result is an efficient general inference system. In particular, we focus on the following aspects of the layered information processing architecture.

1. The assumptions of what information is available for implementing an information process must be designed to be made available in the general inference system.

2. The information extracted by the general inference system is made on a need-to-know basis so that computational resources are not expended to extract redundant and extraneous information.

3. Processes are to be triggered on-demand, so that determining what conditions should initiate a process is an important part of the implementation of an information process.

Furthermore, we focus on how to implement the individual information processes by a distributed algorithm on the system architecture with distributed autonomous agents. Thus, we will discuss the distributed implementation of all information processes we propose in Chapter 3. In particular, distributing the implementation of the target localization process using the hierarchy is discussed in Chapter 4.

1.9 Related Work

This section describes work related to this thesis.

Sensor Network Architecture
The work in this thesis provides a methodology for designing efficient algorithms for distributed inference problems on sensor networks. Sensor networks are a new kind of system architecture which require a new theory of estimation that considers the costs of sensing, processing, and communication and the physical constraints of the system architecture. The models for sensing, processing, and communication are based on the cited characteristics of Wireless Integrated Network Sensors (WINS) in [20]. An alternative project building sensor networks is the Smart Dust project as described in [14]. Whereas WINS uses RF communications across short distances, the Smart Dust project originally focussed on using optical communications, although RF communications are now being implemented in the Smart Dust project. The point is that differences in the system architecture must be accounted for when designing algorithms for inference.

Tracking
The inference task we focus on when demonstrating the layered information processing architecture is the problem of tracking. Implementation of the filter for updating track states will be based on the particle filtering algorithm, also known as the condensation algorithm [13]. We refer to the results in [18] when claiming that more specific motion models require less particles in order to track a target. We give a presentation of particle filters in Appendix C. Of particular interest will be the multiple

target tracking problem and the data association problem. There exist two major approaches for handling the data association problem: the multiple hypothesis tracker (MHT) in [22] and the joint probabilistic data association filter (JPDAF) in [1] and [2]. We will discuss these approaches in more detail in Chapter 3. Furthermore, there exist extensions of particle filters such as [11] for implementing a filter which can track multiple targets although this approach is not conducive to a distributed implementation on a sensor network. We are faced with the problem of implementing a multiple target tracker in a distributed fashion since it will not be possible to send all sensor data to a central processor for tracking multiple targets. This limitation is not handled in the standard MHT and JPDAF algorithms for multiple target tracking so that we are faced with the problem of constructing a distributed multiple target tracking algorithm. Our approach to multiple target tracking will be to allow interactions between the tracking layer and localization layer so that information about the current tracks can prime the localization layer to localize for a specific track. The effect of this interaction between tracking and localization is to reduce the problem of multiple target tracking into several single target tracking problems in certain cases which can be implemented in a distributed way. In the cases when we cannot separate the localization for multiple tracks and, hence, encounter a data association problem, we will use discriminatory information provided by other layers of the layered information processing architecture when available. In [17], an approach which considers tracking and classification tasks together is presented to simultaneously track and classify moving targets using airborne radars. Similarly, this thesis is focussed on exploiting information from different inference tasks as in [17] as well as considering how to implement these interactions in a distributed fashion on sensor networks.

Information Transfer

Of particular interest in this thesis is the design of an information collection mechanism for the problem target localization. Information collection falls under the basic function of information transfer. An example of a communication protocol for retrieval is the LEACH algorithm in [9] where a randomized hierarchical clustering protocol is employed to broadcast information to a receiver. By randomly rotating the local cluster heads, the energy load is evenly distributed among nodes of the sensor network to prolong the lifetime of the sensor network. An example of a communication protocol for dissemination is the GEAR algorithm in [28]. Motivated by the fact that the destination of information may be based on geographical location rather than on specific sensor nodes, the idea in this algorithm is for routing to proceed according to local decisions made by sensor nodes along the multi-hop route from source to destination. These local decisions are energy-aware, and hence, lead to an energy-efficient routing protocol. A similar routing protocol based on local decisions weighing the information gain and cost of communication is the constrained anisotropic diffusion routing (CADR) algorithm in [7]. The above communication protocols are not aware of the information that is being routed through the network. A step towards being

aware of the information being routed is the directed diffusion paradigm in [12]. The idea is to set up *interests* for certain information which sets up gradients for information flow. As information begins to flow through the network, certain routes are *reinforced* depending on their efficiency so that the network adaptively constructs efficient routes for information. Furthermore, in-network data aggregation and caching are set up so that applications will have easy access to the data they require. Our work on information collection takes the control of communications occurring up to the application level. The information-driven sensor querying (IDSQ) algorithm [7] is most related to the work in this thesis and was the initial work which led to applying the hierarchy representation to construct distributed communication algorithms. The IDSQ algorithm is an active querying technique which chooses what sensor data to collect based on the expected information content of the data. It is essentially a sequential decision process which we have generalized using the hierarchy to allow parallel decisions to take place. We describe the IDSQ algorithm in detail in Appendix A.

Active Querying and Testing

The information-based computation principle is based on the underlying concept in active querying techniques: determine what to query based on what is known. This technique is embodied in the IDSQ algorithm for target localization as well as in other application domains like computer vision. In [8], tracking roads from satellite images is accomplished through a series of tests corresponding to matched filters for short road segments which are chosen one at a time in order to minimize the uncertainty about the possible true location of the road given the results of previous tests. This kind of active testing technique is compared to the "divide-and-conquer" strategy of a "Twenty Questions" game from which the measure of uncertainty is chosen to be Shannon entropy. Our work is in line with the information-based computation principle underlying these techniques which has resulted in taking a holistic view of the inference system we wish to construct and utilizing a layered information processing architecture to design interactions between different information processes of different layers. Furthermore, the hierarchy representation is intended to be a tool which extends these active querying techniques where there is implicitly only a single decision maker to the case when there are several decision makers which should cooperate together to gather the appropriate information.

Top-down and Bottom-up Processing

As discussed in Section 1.5.1, the layered information processing architecture and the hierarchy representation have been motivated by the speculations in the computer vision literature for complementing top-down and bottom-up processing. Ullman has suggested the counterstreams structure in [25], which was described on page 46, as a possible way for complementing the two processes. The key idea in motivating the two-sided search for a match in the counterstreams structure is that certain deformations between images and objects are more easily performed on the object models

themselves while other deformations are more easily performed on the images. The idea we have taken away is that the intuition of a bottom-up process is different than a top-down process. In the layered information processing architecture, the bottom-up interactions are referred to as spawning actions because these are to initiate generic computations in layers above. On the other hand, the top-down interactions are referred to as priming actions because these are to initiate more specific computations in the layers below. Regarding the hierarchy representation, we have two kinds of links: uplinks and downlinks. Uplinks are interpreted as spawning the computation of nodes above while downlinks are considered to be queries for specific information from below.

Compositionality

We take a compositional view for constructing large-scale systems where we can decompose a general inference system into a set of inference tasks, which are further decomposed into a set of information processes, and so on. This compositional view of wholes being composed of parts which are further composed of subparts is present in [3], [19], [4], [5], and [6]. The essential idea in compositionality is to incrementally aggregate subparts into parts and parts into larger parts until we aggregate parts into wholes. By performing these incremental aggregation steps, what was once an intractable problem mapping the subparts to wholes has become a tractable step by step procedure. We take this compositional view when we represent complex functions by compositions of simpler functions in the hierarchy. The notion of a step by step procedure to aggregate parts into larger parts is comparable to our idea of explicitly evaluating only a subset of the simple functions composing the complex function.

Computational Models

Finally, the hierarchy representation can be considered to be a computational model which combines elements of a circuit and a branching program. It generalizes a circuit by adding a selection mechanism to determine which gates of the circuit to explicitly evaluate. It generalizes a branching program by allowing the parallel traversal of branches. This generalization is important if we are to construct distributed algorithms on the sensor network. We will leave the detailed discussion of the relationship of the hierarchy to circuits and branching programs for Chapter 5.

1.10 Organization of Thesis

This chapter has motivated and introduced the typical inference problems on sensor networks for the application to battlefield awareness and proposed a general methodology for the construction of efficient algorithms for these distributed inference systems.

Chapter 2 will formulate the inference tasks of localization, tracking, and classifi-

cation for battlefield awareness and propose a specific layered information processing architecture for the interaction between these inference tasks.

Chapter 3 presents the solution to the target tracking and classification problems using the layered information processing architecture. The individual processes of the layered information processing architecture and their interactions are defined and implemented to result in a distributed algorithm on the sensor network. The solution handles the canonical scenarios presented in Section 1.3.

Chapter 4 then focusses on the problem of target localization. How the hierarchy is to be used as a meta-representation for constructing efficient information collection algorithms which solve the problem of target localization on the sensor network is explained in full detail.

Chapter 5 formalizes the general hierarchy representation and discusses its facility in constructing computationally efficient algorithms. The conceptual ideas of this chapter encompass the general principle of information-based computation.

Finally, Chapter 6 will provide a summary, give conclusions, and suggest future work.

Chapter 2

Formulation of Inference Tasks for Battlefield Awareness

In this chapter, we formulate the various inference tasks which are needed to gain intelligence about targets in the scenarios presented in Section 1.3. In order to formulate the inference tasks, we first describe the system architecture of sensor networks. Then, to gain some intuition on what kinds of information sensor measurements may provide, we first describe the measurement models of the sensors we expect to use in sensor networks for battlefield awareness. Then, we formulate the inference problems and finally outline the solutions based on our proposed methodology for constructing computationally efficient algorithms for these distributed inference problems.

2.1 System Architecture of Sensor Networks

2.1.1 Basic Functionality and Issues

An ad hoc heterogeneous sensor network consists of several thousands of autonomous nodes with a priori unknown positions and possibly other parameters each with the ability to sense the environment, process the data collected, and communicate with each other. The function of this sensor network is to gain intelligence about targets in its environment. The specific intelligence tasks we refer to are the inference tasks of localization, tracking, and classification. We choose to apply the methodology for designing efficient inference algorithms for this application because of the diversity in the kinds of inference tasks to be performed and because sensor networks are an example of a system architecture with distributed autonomous agents.

The kinds of information about targets to be extracted from the inference tasks based on data collected by the sensor network include the following.

1. **Localization** This inference task detects targets present in the field and estimates the target positions.

2. **Tracking** This inference task tracks the movement of targets in the field.

3. **Classification** This inference task identifies what kinds of targets are in the field.

4. **Other Inference Tasks** Other inference tasks can include estimating characteristics about groups of targets. For example, in military applications, analyzing the strengths and weaknesses of the groups of targets can be useful for determining defensive or offensive strategies against enemy targets.

Each of these inference tasks fall into the field of estimation and detection theory but with a significant difference relative to classical estimation and detection theory. The system architecture is one with distributed autonomous agents. Thus, some major questions for this problem include the following.

(I1) With data distributed throughout individual nodes of the network, how can we pass data or partially processed data around so that each of the inference tasks can be successfully accomplished?

(I2) How can we design algorithms which implement these global inference tasks in a distributed fashion? For example, in the the presence of multiple targets, how can we design a multiple target tracker which can manage the initiation, update, and deletion of multiple tracks in a distributed way?

(I3) Since the information extracted from one inference task can aid the inference of another, how can we utilize the information that has already been extracted to make the further extraction of information efficient? For example, how can we make use of classification information to aid tracking and localization of targets?

The above issues are concerned with three general problems: **(I1)** how to deal with distributed data, **(I2)** how to distribute an algorithm amongst many agents, and **(I3)** how to take advantage of prior computations in subsequent computations. Furthermore, we are also concerned with the computational efficiency and *scalability* of the resulting algorithms. Scalability can be an important issue if the performance of the inference algorithms degrades with the number of nodes in the sensor network. We will see that the performance of the inference algorithms for localization, tracking, and classification presented in the latter half of this thesis will not depend on the number of nodes in the sensor network so that these algorithms do not have problems of scalability.

2.1.2 Physical Characteristics

Each sensor node can be equipped with a variety of sensors including GPS, acoustic, seismic, magnetic, thermal and visual sensors. With a processor, memory, and

wireless communication devices, each node has the ability to receive data from neighboring nodes, process the data into information, and transmit this information to neighboring nodes. We assume that the processor in each node is powerful enough that implementing algorithms in the nodes will not be a problem. In the current state of technology, the processors in these sensor nodes are powerful enough to run a Linux operating system. Figure 2-1 shows the architecture of a node. Each node is

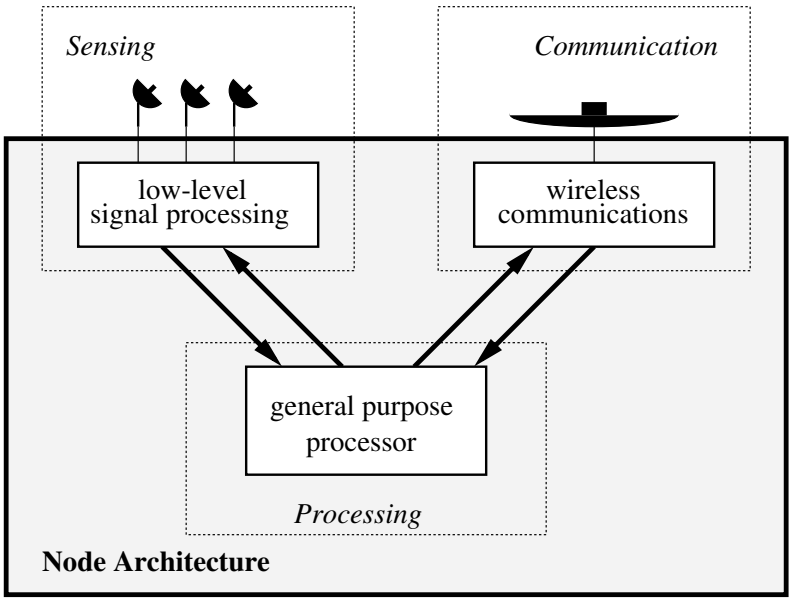

Figure 2-1: The architecture of a node of the sensor network. Each node is equipped with a general purpose processor, a set of sensors, and a wireless communication device. Data and information from the outside world come in through the sensors and wireless receiver. Messages can be sent to other sensor nodes by the wireless transmitter.

equipped with a set of sensors, a wireless communication device, and a general purpose processor. This allows each node to collect data, transmit and receive messages from other sensor nodes, and run algorithms on the processor. The basic operations of the node aside from the usual computations a computer can perform include the control of the signal processing routines to be performed on the raw data collected by the sensors, and the transmission of messages to other sensor nodes. We assume that the results of the low-level signal processing routines and any messages received by the wireless communication device are stored in memory which is accessible by the general purpose processor.

For the specific inference tasks we consider for the application of sensor networks to battlefield awareness, we assume that sensor nodes are stationary, so that once they are deployed they will forever remain in the same location. The ad hoc nature of the sensor network is that the initial positions of the sensor nodes are not known

a priori. Thus, the algorithms to be designed for the inference tasks on this sensor network must be able to handle arbitrary configurations of sensor nodes in a field. Furthermore, the types of sensors we consider will primarily be focussed on acoustic sensors, although the methods presented for constructing inference algorithms can generalize to any type of sensor. We will describe the specific measurements we extract from the sensors and their measurement models in Section 2.2.

Let us describe the particular phases of operation we envision for how this sensor network is to proceed. Recall the four basic functions described in Section 1.2: (1) self-configuration, (2) data transfer, (3) network monitoring, (4) inference.

Deployment Several thousands of sensor nodes will be deployed by aircraft onto a field so that the position of sensor nodes are random. Since this field will generally be in hostile territory, efficient use of energy resources by each sensor node is critical because maintenance of these sensor nodes is not possible.

Configuration A priori, individual nodes have no knowledge of the other nodes, so there must be a configuration phase where nodes will exchange basic knowledge about themselves with their neighbors. This basic knowledge consists of knowledge about the models generating data at each sensor node due to the possibility of different modalities of sensors being attached to different nodes. If these models are parametric, then basic knowledge includes the specific parameter values for each node, which can include quantities like the position of the node, the types of sensors attached to the node, and local environmental conditions. This is the basic function of self-configuration.

Operation Once the configuration phase is complete, various intelligence gathering tasks can begin. The basic functions performed in this phase are network monitoring, data transfer, and inference. In particular, we are interested in devising algorithms for inference and data transfer assuming that the current state of the network, obtained from the network monitoring function, is known. Thus, the algorithms presented assume that self-configuration and network monitoring are successful so that basic knowledge about the state of the neighboring sensor nodes are known by each node.

The next section will discuss the resources, which algorithms running during the **Operation** phase, should conserve in particular to the application to battlefield awareness although the resource considerations discussed will apply generally to other applications as well.

2.1.3 Resource Constraints

A large network of these nodes has the potential to be a powerful computational system. However, just as a committee of great minds may have the potential to solve difficult problems, without a good way of organizing the processes and interactions

between members, a good deal of effort can be lost to useless chatter. In the context of inference in sensor networks, wasted effort translates to inefficient use of computational resources which can have devastating effects like rendering the entire sensor network useless.

It is desirable for the sensor network to remain in operation for as long as possible, and thus, it is important for these energy limited sensor nodes to conserve energy. Minimizing energy consumption will be the primary factor measuring efficiency of algorithms running on a sensor network. Certainly, other factors like time latency in estimation will affect the usefulness of the sensor network since many of the inference problems are to be run in real-time.

Let us first concentrate on energy resources. Each sensor node is powered by a battery with limited energy, and energy is consumed any time the sensor node

1. collects data from its sensors,
2. processes data and information, and
3. communicates information to other sensor nodes.

There exists a range of energy requirements for the types of sensors that can be attached to a sensor node. In order to conserve energy, it is desirable to use those sensors which have low energy requirements often and only use those sensors with high energy requirements when needed. Thus, controlling when sensors should be turned on and off can be an important consideration for designing efficient algorithms.

The energy consumed by processing information and data either from a general purpose microprocessor or a signal processing chip depends on how often the transistors are switching. Thus, designing algorithms which minimize the amount of processing required is certainly a factor for making the algorithms efficient. However, the current trend in technology is reducing the energy consumption by the processors to a point that processing is not a significant consumer of energy. Thus, we will not be too concerned with how much energy is consumed during processing.

The sensor nodes are equipped with wireless communication devices capable of transmitting and receiving bits. Information theory tells us that there is a fundamental lower bound on how much energy we must use in order to obtain nearly error-free communication. Thus, the lower bound on the energy needed for transmission and reception cannot be reduced by better technology so that the design of efficient inference algorithms will be primarily concerned with the minimization of the amount of communication that must take place between sensor nodes. A model of the energy required for transmitting k bits a distance of d meters is

$$E_{Tx}(k,d) = E_{elec}k + \epsilon_{amp}kd^2 \ , \tag{2.1}$$

and a model of the energy required for receiving k bits is

$$E_{Rx}(k) = E_{elec}k \tag{2.2}$$

where $E_{elec} = 50\text{nJ/bit}$ and $\epsilon_{amp} = 100\text{pJ/bit/m}^2$. This model for the energy consumption by communication was taken from [9]. Note that the energy needed for transmission is linear with the number of bits k to be transmitted and quadratic with the distance d of the transmission. Furthermore, the energy needed for receiving bits E_{Rx} is significant, and keeping the receiver circuitry on will drain a significant amount of energy. Thus, in order to design efficient algorithms for data transfer, it is desirable to keep the wireless communication device off as much as possible and turn it on only when necessary. One such possible scheme is to design a low power sleep mode for the sensor node during which it can detect when another sensor node is trying to activate it. If such a low power sleep mode is realizable and the energy to transmit a "wake up" message to sleeping nodes is not too large, then we can concentrate solely on minimizing the transmissions needed since we have significantly reduced the cost of keeping the receiver on idle.

As for the resources of time and memory, since the inference processes are to run in real-time and data is collected at a relatively fast rate, it is necessary that the inference algorithms extract the relevant information in the data quickly. Thus, recursive algorithms for filtering which will be used in tracking will be employed to minimize the amount of memory and processing needed.

The next section will elaborate on the general characteristics of the data collected by each sensor node. In particular, the relationship between the information to be extracted from data versus the data itself is discussed.

2.1.4 Nature of Information

In this section, we will describe the intuition of how information contained in the data is collected by nodes of the sensor network. The point of this discussion is to argue that collaboration between sensors is unavoidable if we are to perform inference on this sensor network. We characterize the nature of information for any given inference task in the sensor network with the following three terms. Figure 2-2 serves as a picture to illustrate the three terms below.

Local Each node of the sensor network is capable of collecting only incomplete information about targets in a local region of the environment. For example, if sensor nodes are equipped with acoustic sensors, these sensors only provide useful data if the target is fairly close to the sensor node due to the attenuation of the sound signal as it propagates through the atmosphere. This locality characteristic holds in general for other types of sensors like seismic, magnetic, thermal, and, in a lesser degree, for visual sensors. In Figure 2-2, the sensors (squares) with $\times$ in them are the sensors which are in range of the targets.

Distributed A single node's measurements alone do not suffice to perform inference of reasonable quality. Hence, it is necessary for sensor nodes to collaborate and communicate information to each other. For example, if only sound intensity is

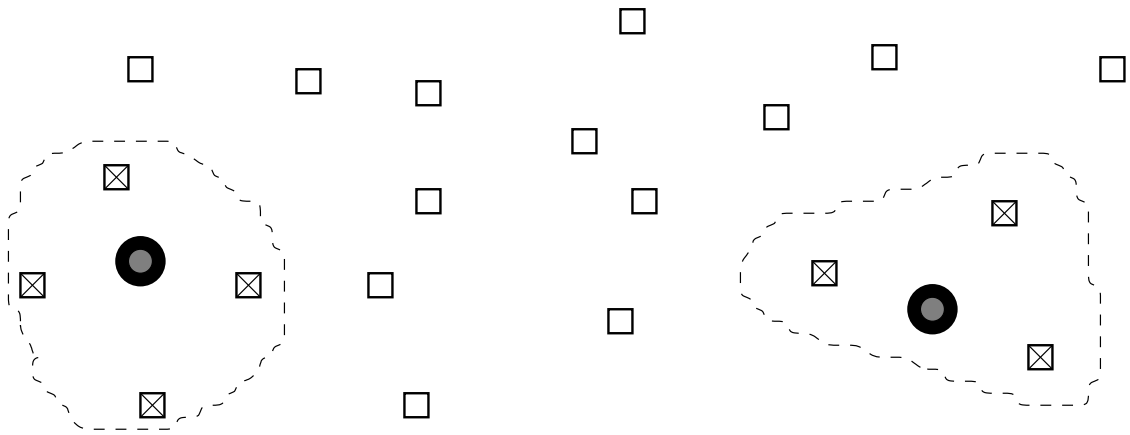

Figure 2-2: Sensors only provide significant information about targets (black circles with gray dot) if they are "near" or local to them. Two instances of the target (circles), located on the far left side and on the far right, are shown along with the set of sensor nodes which are affected by the presence of the target ($\times$ in square).

extracted from acoustic signals, the information we obtain about target location is that the target lies in a ring at a certain distance from the sensor node. Thus, to localize better than a ring around some particular sensor node, collaboration is necessary. In Figure 2-2, the sensors which are circled around each target must collaborate the data they have collected in order to localize the target.

Redundant Not all nodes of the sensor network contain information for a particular estimation task. For example, if we have sound intensity measurements from nodes that are both far and near the target, the measurements from nodes that happen to be far away from the target contribute very little added information about the target location because the measurements from nodes that are near the target already contribute a lot of information. In Figure 2-2, the sensor nodes without a $\times$ are sensors whose information add little to the information gained by the sensors marked with a $\times$.

The notion of information and how information is extracted from data will be discussed in Section 4.1.2. For the purposes of the presentation of the sensor measurement models in Section 2.2 however, we hope to build the intuition of how information resides in the data in terms of the qualities: "local", "distributed", and "redundant".

2.2 Sensor Measurement Models

We will consider four types of sensor measurements in sensor networks for the application of battlefield awareness. All sensors are considered to be stationary. Three of

these sensor measurements are computed from the signals of omni-directional microphones, the intensity, frequency bandlimited intensity, and time difference of arrival (TDOA) measurements, and the third sensor measurement type is computed from the signal of a magnetometer.

2.2.1 Acoustic Measurements

We assume that all targets emit a sound as they are idling or moving through the field around a sensor network which we represent as a signal

$$s(t) \ , t = 0, 1, \ldots$$

which corresponds to the sound pressure at the location of the target. We represent the position of the target over time as a time series of points on the Euclidean plane,

$$x(t) \in \mathbb{R}^2 \ , t = 0, 1, \ldots \ .$$

For an omni-directional microphone, which we label by i, located at $x_i \in \mathbb{R}^2$, the raw sound pressure signal $r_i(t)$ measured by this microphone is a time series of discrete time samples derived from the source sound pressure signal $s(t)$ appropriately delayed and attenuated as it propagates through the atmosphere. The amount of attenuation and delay is variable over the frequency content of the signal $s(t)$ and is further complicated by the effect of reflections from the ground and other obstacles during the propagation of the sound wave from target location to microphone.

All measurements extracted from acoustic sensors will be based on some low-level signal processing on the discrete-time received sound pressure signal $r_i(t)$ for $t = 0, 1, \ldots$. More information on the physics of sound can be found in the references [10], [15], and [23].

2.2.1.1 Intensity Measurements

The first simple measurement we can extract from the received sound pressure signal $r_i(t)$ is an average energy measurement over a short time window, which we will simply refer to as the *intensity*, since sound intensity is proportional to the square of the sound pressure. That is, the intensity measurement $Z_i(t)$ at time t is related to the signal $r_i(t)$ according to

$$Z_i(t) = \frac{1}{\Delta} \sum_{\tau=t-\Delta+1}^{t} r_i^2(\tau)$$

where Δ is a chosen small interval of time. Let us present a model for this intensity measurement.

We assume that all targets emit sounds with a known intensity $A \in \mathbb{R}$, so that all intensity sensors measure an attenuated intensity depending on the distance between

the sensor location and the actual target location. In free space, the sound intensity drops off according to the inverse square of the distance from the sound source. However, due to the effect of reflections from the ground and other obstacles and other atmospheric effects, the sound intensity attenuation can drop off effectively as a different power of the inverse distance. In particular, if the location of the sensor is $x_i \in \mathbb{R}^2$ and the location of the target at time t is $x(t) \in \mathbb{R}^2$, then we model the intensity measurement of the sensor by

$$Z_i(t) = A\ f(d(x_i, x(t))^\alpha) + N_i(t) \tag{2.3}$$

where $d(\cdot,\cdot)$ is the Euclidean distance, α is the atmospheric attenuation coefficient, $N_i(t)$ is zero mean white Gaussian noise with variance σ_i^2, and

$$f : \mathbb{R}^+ \to \mathbb{R}$$

is a monotonically decreasing function on the nonnegative reals with $f(0) = 1$ which is the scale factor of the nominal intensity measured by the sensor. In particular, we choose

$$f(d) = \begin{cases} \frac{d^3}{2.5\times10^8} - \frac{d^2}{2.5\times10^5} + 1 & , d \leq 500 \\ \frac{250}{d} & , d > 500 \end{cases} \tag{2.4}$$

which we show in Figure 2-3. The scaling factor f has been chosen so that if the target is right on top of the sensor, then the sensor measures the full intensity of the target. At an attenuated distance of $d^\alpha = 500$ units away, the intensity measured by the sensor is $\frac{1}{2}$ that of the full intensity A of the target, and after that, the intensity measured by the sensor drops off inversely according to the attenuated distance d^α. The attenuation coefficient α models the attenuation effects of the atmosphere. For 3-dimensional free space where sound propagates as an expanding sphere around the source, α is equal to 2; however, on the earth's surface due to reflections from the surface and reflective and absorbent obstacles, the effective α generally ranges from 1.2 to 3.

Dissipation by the atmosphere is frequency sensitive so that different frequency components of the acoustic signal attenuate differently. There is also the possibility of phase effects due to interference by reflections from the ground or other obstacles. Furthermore, there is a delay due to the finite propagation speed of the sound signal. These effects are not explicitly modelled and are lumped into the additive noise term $N_i(t)$ in our intensity model which serves as a simple first order approximation of the real intensity computed by the low-level signal processing.

An intensity measurement essentially provides range information about the target location from the sensor. To illustrate, the intensity measurement model defines a likelihood function

$$p_{intensity}(z_i|x)$$

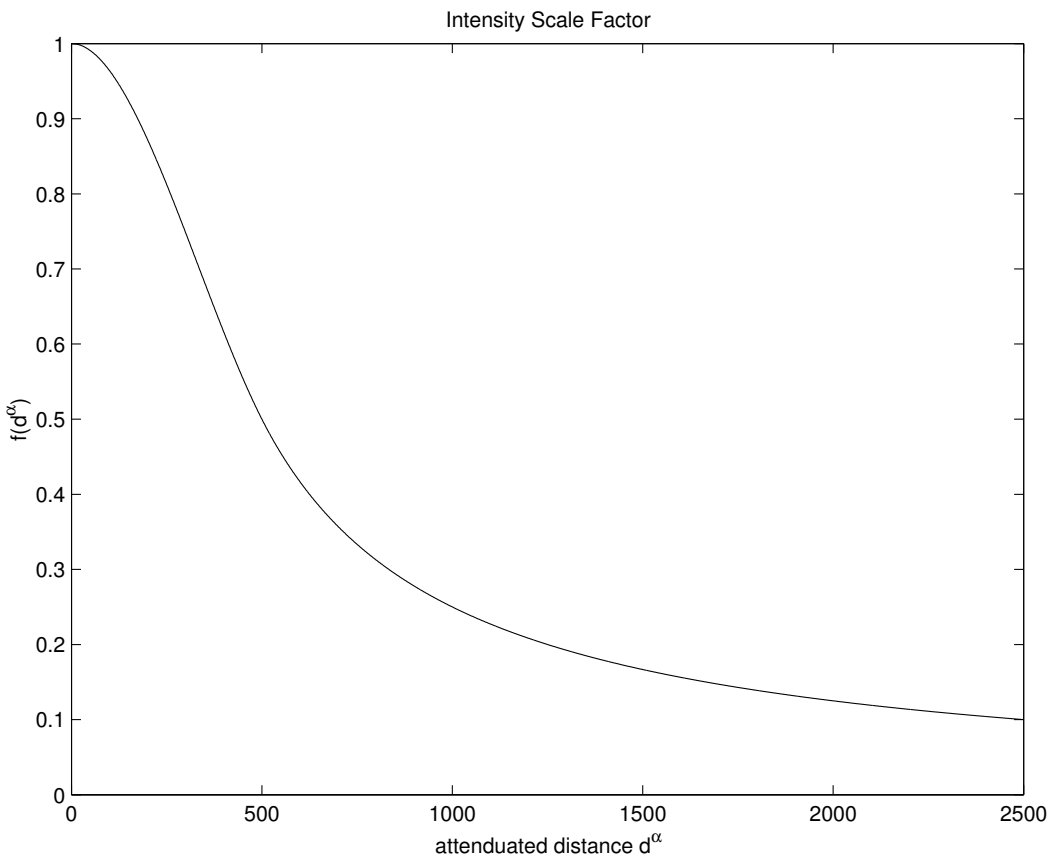

Figure 2-3: The intensity scaling factor as a function of the attenuated distance d between target location and sensor position.

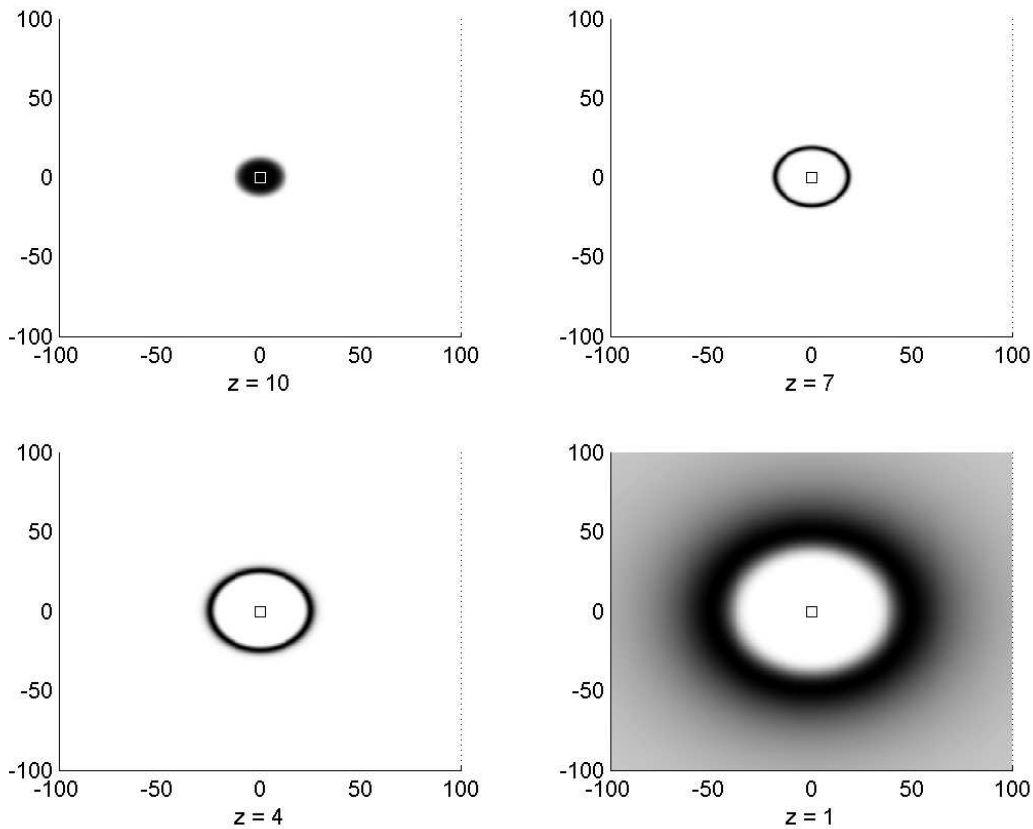

Figure 2-4: The intensity likelihood function of the target position x given various sensor measurement values z with parameters $A = 10$, $\sigma_i = .5$, and $\alpha = 2$. The sensor is depicted by the square. Dark regions signify regions of target positions with high likelihood, and light regions signify regions of target positions with low likelihood.

where we have dropped the time term t to simplify notation and because we assume stationarity. That is, the likelihood of $Z_i(t) = z_i$ given the target located at $x(t) = x$ is the same for all t. From Eqn. 2.3, the only source of randomness in the model is the noise term N_i so that the likelihood function is

$$p_{intensity}(z_i|x) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{1}{2\sigma_i^2}\left(z_i - A\ f(d(x_i, x(t))^{\alpha})\right)^2\right] .$$

Figure 2-4 shows the intensity likelihood function viewed as a function of target position x for various values of z_i. These figures show that an intensity measurement provides information about the distance of the target from the sensor.

Although there is some skepticism in using this model due to not knowing the intensity A of the target in general and the fact that the attenuation coefficient α is sensitive to atmospheric variabilities and obstacles in the field, we will use this

model for its simplicity of calculation from the raw sound pressure signal from the microphone while capturing information about range of the target.

We have presented the intensity measurement model above when there is exactly one target in the field. For the case of multiple targets under the assumption that the sounds emitted by the targets are uncorrelated, the received intensity by the sensor is the sum of the intensity contributions received from each target. That is, when there are m targets in the field, the intensity measurement of the sensor is modelled by

$$Z_i(t) = \sum_{j=1}^{m} \left[A^j f(d(x_i, x^j(t))^\alpha) + N_{i,j}(t)\right]$$

where $\{A^j\}_{j=1}^m$ are the known intensities of each target, $x^j(t)$ is the position of target $j \in \{1, \ldots, m\}$ at time t, $N_{i,j}(t)$ is zero mean, white Gaussian noise with variance $\sigma_{i,j}^2$, and the other elements of the model are the same as before. By assuming that sounds from different targets are uncorrelated, we assume that interference effects between sounds pressure signals are negligible. Such interference effects are lumped into the additive noise term $N_{i,j}(t)$ so that the model serves only as a first order model when multiple targets are present.

2.2.1.2 Frequency Bandlimited Intensity Measurements

The second simple measurement we can extract from the received signal $r_i(t)$ is a refinement of the intensity measurement where we take into account the frequency content of the signal. From a short window of length Δ of the discrete-time received signal

$$\{r_i(\tau)\}_{\tau=t-\Delta+1}^{t},$$

we can compute the discrete Fourier transform (DFT),

$$R_i(k) = \sum_{n=0}^{\Delta-1} r_i(t - \Delta + 1 + n) e^{-j(2\pi/\Delta)kn}$$

for $k = 0, \ldots, \Delta - 1$. The DFT gives us a representation of a short window of the received signal $r_i(\tau)$ as a weighted combination of complex sinusoids weighted by $R_i(k)$. Essentially, $R_i(k)$ gives the amplitude of each frequency component which composes the signal $\{r_i(\tau)\}_{\tau=t-\Delta+1}^{t}$.

The idea for frequency bandlimited intensity measurements is to compute the intensity of the part of the signal which is in a given frequency band. From the DFT, the set of frequency components corresponds to each element of $R_i(k)$ for $k = 0, \ldots, \Delta-1$. We group these frequency components into b mutually exclusive and collectively exhaustive bands, $\{B_\phi\}_{\phi=1}^{b}$, where $B_\phi \subset \{0, \ldots, \Delta-1\}$ and $B_\phi \cap B_{\phi'} = \emptyset$ for $\phi, \phi' \in \{1, \ldots, b\}$ and $\phi \neq \phi'$. The intensity of the set of frequencies in a band B_ϕ

is computed by

$$Z_i(t,\phi) = \frac{1}{\Delta^2} \sum_{k \in B_\phi} R_i^2(k)$$

where $R_i(k)$ are the DFT coefficients calculated from a Δ-window of $r_i(t)$. The relationship between this frequency bandlimited intensity measurement and the intensity measurement of the previous section is given by

$$Z_i(t) = \sum_{\phi=1}^{b} Z_i(t,\phi)$$

which can be derived by using Parseval's relation.

The model for frequency bandlimited intensity measurements given a set of frequency bands $\{B_\phi\}_{\phi=1}^b$ is given by

$$Z_i(t,\phi) = A_\phi \; f_\phi(d(x_i, x(t))^{\alpha_\phi}) + N_i(t,\phi) \tag{2.5}$$

where $\{A_\phi\}_{\phi=1}^b$ is the known frequency intensity signature of the target corresponding to the frequency bands $\{B_\phi\}_{\phi=1}^b$, $d(\cdot,\cdot)$ is the Euclidean distance, x_i is the location of this sensor, $x(t)$ is the position of the target at time t, α_ϕ is the atmospheric attenuation coefficient which can depend on the frequency band, $f_\phi(\cdot)$ is the intensity scaling factor which can depend on the frequency band, and $N_i(t,\phi)$ is zero mean, white Gaussian noise with variance $\sigma_{i,\phi}^2$. At each time t, this sensor can compute the intensity received by the microphone for each frequency band B_ϕ.

For these frequency bandlimited intensity measurements, we can model more explicitly the variation of attenuation based on frequency content of the signal. However, we still do not explicitly model the effect of delays from the finite propagation speed of sound or possible phase effects due to reflections from the ground or other obstacles. These effects are lumped into the additive noise term $N_i(t,\phi)$ which serves as a first order approximation of the bandlimited intensity computed by low-level signal processing.

Each of the frequency bandlimited intensities $\{Z_i(t,\phi)\}_{\phi=1}^b$ at time t gives range information about the target given the nominal intensity A_ϕ of the target in each frequency band ϕ. In particular, the likelihood function $p_{intensity,\phi}(z_{i,\phi}|x)$ is given by

$$p_{intensity,\phi}(z_{i,\phi}|x) = \frac{1}{\sqrt{2\pi}\sigma_{i,\phi}} \exp\left[-\frac{1}{2\sigma_{i,\phi}^2}(z_{i,\phi} - A_\phi f_\phi(d(x_i, x(t))^{\alpha_\phi}))^2 \right] .$$

Since we modelled the noise $N_i(t,\phi)$ as independent for each ϕ, the likelihood of the target position is the product of the likelihoods from each frequency band so that

$$p_{freq}(\{z_{i,\phi}\}_{\phi=1}^b|x) = \prod_{\phi=1}^{b} p_{intensity,\phi}(z_{i,\phi}|x) \ .$$

Note that frequency bandlimited intensity measurements $Z_i(t, \phi)$ are a refinement of the intensity measurements $Z_i(t)$ from before where the relationship between the two is given by

$$Z_i(t) = \sum_{\phi=1}^{b} Z_i(t, \phi)$$

for all time t. We introduce this refinement of intensity measurements so that we can make use of classification routines which classify targets based on the frequency content of the emitted acoustic signals to extract extra information for our inference tasks when needed. We refer specifically to using the frequency content of acoustic signals in order to solve the problem of data association when targets with different frequency content cross in the field.

We have presented the frequency bandlimited intensity measurement model above when there is exactly one target in the field. As with the intensity measurement model for multiple targets, we will model the frequency bandlimited intensity measured by a sensor to be the sum of the frequency bandlimited contributions from each target. That is, when there are m targets in the field, the frequency bandlimited intensity measurement of the sensor is modelled by

$$Z_i(t, \phi) = \sum_{j=1}^{b} \left[A_\phi^j f_\phi(d(x_i, x^j(t))^{\alpha_\phi}) + N_{i,j}(t, \phi) \right]$$

where $\{A_\phi^j\}_{j=1}^b$ are the known intensities in frequency band ϕ of each target, $x^j(t)$ is the position of target $j \in \{1, \ldots, m\}$ at time t, $N_{i,j}(t, \phi)$ is zero mean, white Gaussian noise with variance $\sigma_{i,j}^2$, and the other elements of the model are the same as before. As mentioned in the intensity measurement model, this model does not model the possible interference effects among the acoustic signals from all targets. Such interference effects are lumped into the additive noise term $N_{i,j}(t, \phi)$ so that the model serves only as first order model when multiple targets are present.

2.2.1.3 Time Difference of Arrival (TDOA) Measurements

The final measurement we extract from acoustic signals from omni-directional microphones is from a dipole arrangement of two omni-directional microphones with known displacement in position. Assuming that the two microphones are located at $x_1 \in \mathbb{R}^2$ and $x_2 \in \mathbb{R}^2$ so that they are spaced a small distance $d(x_1, x_2)$ apart, the received signal of each microphone, say $r_1(t)$ and $r_2(t)$ corresponding to the microphones located at x_1 and x_2 respectively, are essentially time-delayed versions of each other when there is a target emitting sound within range of them. By computing the cross correlation over a small window over the two signals,

$$R_{12}^t(\tau) = \sum_{\delta=-\Delta+1}^{0} r_1(t+\delta) r_2(t-\tau+\delta) \ ,$$

the local maxima of $R_{12}^t(\tau)$ at each time t correspond to possible time difference of arrival of the signals

$$Z_i(t) = \{\text{set of local maxima of } R_{12}^t(\tau)\} \ .$$

The range of the time difference τ between the two microphones depends on the distance between the microphones and is in the interval $[-\frac{d(x_1,x_2)}{c}, \frac{d(x_1,x_2)}{c}]$ where c is the speed of sound. The number of local maxima that occur in the cross correlation depends on the frequency content of the signal. When the signal is wide band noise, the local maxima in the cross correlation tend to correspond with actual targets, whereas if the signal is a single frequency tone and the wavelength of the tone is smaller than the distance $d(x_1, x_2)$ of the dipole, then several redundant local maxima can occur. The time difference of arrival (TDOA) measurements extracted from the cross correlation $R_{12}^t(\tau)$ give information about the bearing of a target in relation to the dipole of microphones. We refer to this dipole of microphones as a TDOA sensor.

To model this measurement, we idealize the signal to a pulse and model local maxima of the cross correlation by the time difference of arrival of the pulse between the dipole of microphones. When there is a single target at location $x(t)$ at time t, we model the measured time difference by

$$Z(t) = \begin{cases} T_1(t) - T_2(t) & , \max_{i=1,2} d(x(t), x_i) \leq \gamma \\ \emptyset & , \text{otherwise} \end{cases} \tag{2.6}$$

where

$$T_i(t) = \frac{d(x(t), x_i)}{c} + \left(\frac{d(x(t), x_i)}{c}\right)^{\frac{\beta}{2}} N_i(t)$$

for $i = 1, 2$ correspond to the absolute time of arrival of the pulse by each microphone in the dipole, γ is a scalar which represents the range of the dipole arrangement of microphones, $d(\cdot, \cdot)$ is Euclidean distance, x_i is the position of microphone i of the dipole arrangement, c is the speed of sound, β is the coefficient governing how much noise perturbs the absolute time of arrival as a function of distance, and $N_i(t)$ is zero mean, white Gaussian noise with variance σ_i^2. We model the absolute time of arrival of the pulse to be the noise-free propagation time plus a noise term whose standard deviation grows as the $\frac{\beta}{2}^{th}$ power with the propagation time of the pulse.

Let us explain the γ threshold of the distance between target and sensor. When targets are far away, the cross correlation function results in small valued local maxima due to the intensity attenuation of the signal with distance. Usually, when computing the local maxima of the cross correlation $R_{12}(\tau)$ we set a threshold and return only those local maxima which are greater than this threshold. We model this by setting a threshold γ for the range of a target from the TDOA sensor so that $Z(t)$ results in no time difference measurement when the targets is farther than γ units away. Note that our model for TDOA measurements does not take into account the possible redundant local maxima occurring in the cross correlation when the target emits a

single frequency tone; however, real targets tend to emit bands of frequencies so as long as the spacing between the dipole of microphones is smaller than the wavelength of the smallest significant frequency, this model should be satisfactory.

The information gained by a TDOA measurement is a bearing estimate of the target relative to the TDOA sensor. To illustrate this, let us consider the likelihood function associated with the TDOA measurement model (Eqn. 2.6). Assuming the target is within range of the TDOA sensor, given the target location x, the likelihood function of the TDOA measurement is given by

$$p_{tdoa}(z|x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{z-\mu}{\sigma}\right)^2\right]$$

where

$$\mu = \frac{d(x(t), x_1) - d(x(t), x_2)}{c}$$

and

$$\sigma^2 = \left(\frac{d(x(t), x_1)}{c}\right)^\beta \sigma_1^2 + \left(\frac{d(x(t), x_2)}{c}\right)^\beta \sigma_2^2 \ .$$

In Figure 2-5, we show the likelihood $p_{tdoa}(z|x)$ as a function of the target position for various values of z with parameters, $d(x_1, x_2) = \frac{1}{6}$, $\sigma_1 = \sigma_2 = 5 \times 10^{-5}$, $\beta = 1$, and $c = 1100$. The effect of the growing noise term is that the likely positions of the target start to bow out as we consider those target positions farther away from the TDOA sensor. Furthermore, there is an inherent ambiguity in the dipole arrangement of microphones which makes the likelihood function symmetric about the axis of the dipole. The reason is that targets at mirror images about the axis of the dipole generate the same time difference of arrival.

For multiple targets in range of the TDOA sensor, $Z(t)$ is a collection of TDOA measurements where each TDOA measurement corresponds to each target within range of the TDOA sensor. If the measurements generated by m targets within range of the TDOA sensor are $Z(t) = \{z_1, \ldots, z_m\}$, then we will model the likelihood of the target position x by maximum of the individual likelihoods. That is,

$$p_{multiple,tdoa}(z_1, \ldots, z_m|x) = \max_{j \in \{1,\ldots,m\}} p_{tdoa}(z_j|x) \ .$$

For example, in Figure 2-5, if the TDOA measurement at time t was observed to be

$$z(t) = \{-1.5152 \times 10^{-4}, 1.5152 \times 10^{-4}\} \ ,$$

then the likelihood would be the combined dark regions of the top-left picture and the bottom-right picture.

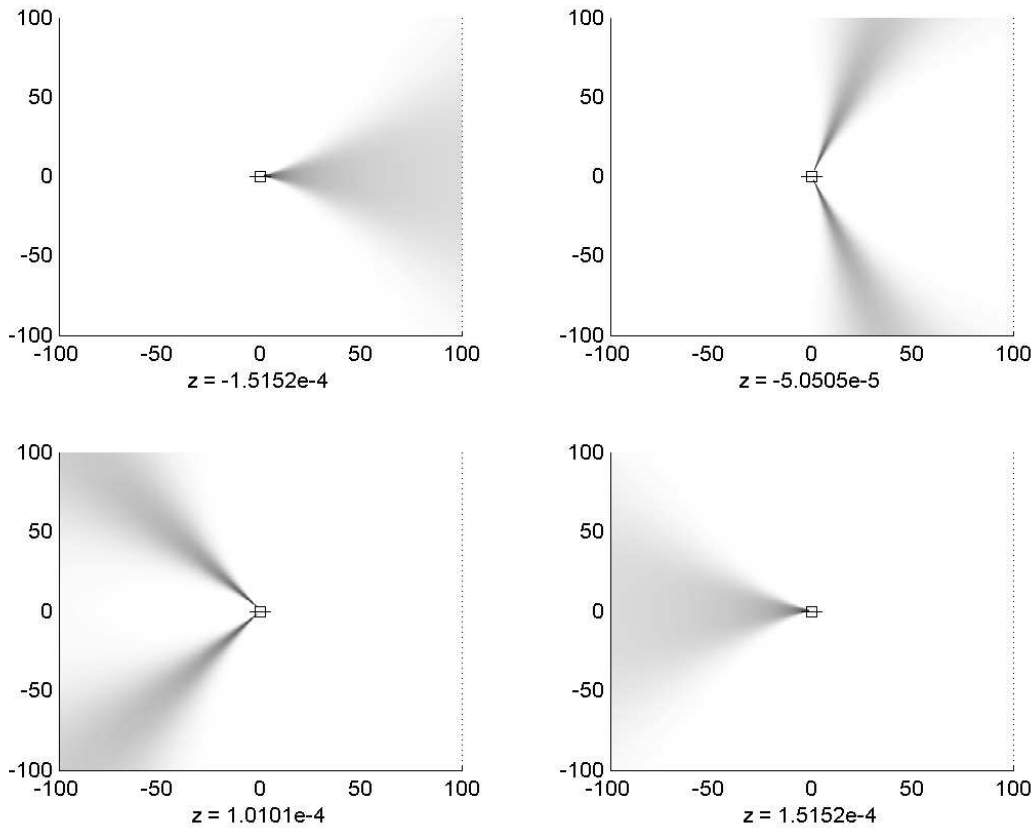

Figure 2-5: The TDOA likelihood function as a function of target position x for various values of time difference z. The dipole of omni-directional microphones is situated horizontally as shown by the horizontal line drawn over the square which represents the TDOA sensor. The first microphone of the dipole is on the right, and the second microphone of the dipole is on the left.

2.2.2 Magnetometer Measurements

A magnetometer is a device which can measure the magnetic field in a local region of the field. In particular, changes in magnetic field can be detected which means that the presence of a metallic object can be detected in a given local region. For our purposes, we will use a very simple model for magnetometers which can detect the presence of a metallic target in its vicinity. Let us define a Boolean variable W which is equal to 1 if the target is metallic and equal to 0 if the target is not metallic. Given a magnetometer at position $x_i \in \mathbb{R}^2$ with range γ and target located at $x(t) \in \mathbb{R}^2$ at time t, the magnetometer measurement is modelled by

$$Z_i(t) = \begin{cases} 1 - N_i(t) & , \ d(x_i, x(t)) \leq \gamma, W = 1 \\ N_i(t) & , \ \text{otherwise} \end{cases} \tag{2.7}$$

where $N_i(t)$ is a Bernoulli random variable with probability p of equalling 1.

We can define a likelihood function for the magnetometer measurement z given a target at location x,

$$p_{magn}(z|x) = \begin{cases} (1-p)z + p(1-z) & , \ d(x_i, x) \leq \gamma \ , W = 1 \\ pz + (1-p)(1-z) & , \ \text{otherwise} \end{cases} .$$

The model of the magnetometer measurement above is for a single target in the field. With the possibility of multiple targets, we will model the magnetometer measurement by

$$Z_i(t) = \max_{j=1,\dots,m} Z_i^j(t) \tag{2.8}$$

where $Z_i^j(t)$ is the magnetometer reading if it happened to be that target $j \in \{1, \dots, m\}$ was the only target in range of the magnetometer. We associate a boolean variable W_j indicating whether the target j is metallic or not. Thus, if any target would have triggered the magnetometer to be 1, then the magnetometer will return 1. This is an appropriate model because when there are several metallic targets in range of the magnetometer, then we would expect the magnetometer to have a higher probability of measuring a 1.

2.2.3 Discussion

For the measurement models presented above, we can always represent the model according to a likelihood function of the measurement $Z_i(t)$ at time t given the target position $x(t)$. In all of the measurement models shown above, we have assumed stationarity of the measurement models so that the likelihood functions do not depend on time. Incorporating other types of sensors into the sensor network requires specifying an appropriate likelihood function

$$p(Z_i(t)|x(t))$$

of the observed measurement $Z_i(t)$ given the current target position $x(t)$. We hope to have convinced the reader that any kind of sensor measurement model can be represented by a likelihood function.

Another assumption we make for all measurement models is that the noise terms are independent. Thus, for a sensor network with n sensors labelled by $i \in \{1, \ldots, n\}$ and with specified likelihood function

$$p_i(z_i|x)$$

for sensor i, the combined likelihood of all measurements is given by the product of the individual likelihood functions,

$$\prod_{i=1}^{n} p_i(z_i|x)$$

where $\{z_i\}_{i=1}^n$ are the measurements observed by each sensor $i \in \{1, \ldots, n\}$ of the sensor network. The independence assumption of the random terms of all measurements means that the fusion of these measurements is especially easy and can be computed by calculating the product of the individual likelihood functions.

The geometrical interpretation of fusing information about sensor measurements when we can take the product of the likelihood functions (viewed as a function of x when given measurements z_i) is that the resulting high probability region of the possible target locations x that coincide with the observed measurements is the intersection of the high probability regions of all individual likelihoods. Thus, fusing the information of another sensor measurement into the current belief under these independence assumptions corresponds to refining the region of possible target locations that may have generated the observed measurements.

As for the notions of information being local, distributed, and redundant, the information gained about target location from observing a sensor measurement is about some, possibly non-elliptical, region of the Euclidean plane. If the target happens to be very far away from the sensor, then little information is known about the location of the target except that it lies in some large region. Thus, each sensor essentially has an effective range where its observed measurements can provide significant information. We refer to this bounded effective range as the information being local. Certainly, with many sensor measurements, we can determine the possible target locations with higher accuracy so that the information about target location is distributed throughout sensor nodes. Finally, most of the information about target location is provided by a small subset of sensors which are located near the target. Thus, the information provided by sensors that are far away are redundant.

2.3 Localization Formulation

Since we wish to track targets in the field with the sensor network, we need to perform target localization which can be fed as inputs to the tracking algorithms. The purpose

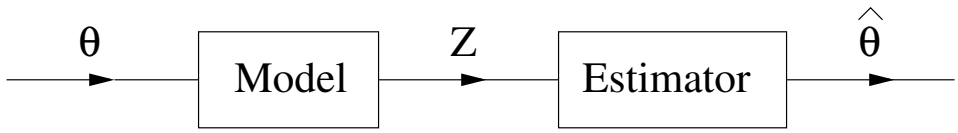

Figure 2-6: Usual parameter estimation process.

of target localization is to convert sensor measurements collected at some time t into an estimate about target location. Specifically, we will formulate the problem of *single* target localization in sensor networks, and generalize this formulation to the problem of parameter estimation in sensor networks. Due to the distributed nature of information in data and the fact that collecting data expends a significant amount of energy in sensor networks, formulating the problem of single target localization must include formulating an efficient data collection mechanism.

We divide this section into two subsections. Section 2.3.1 describes the considerations for a data collection scheme for estimation in sensor networks. Then, Section 2.3.2 formulates the problem of single target localization in sensor networks.

2.3.1 Distributed Estimation in Sensor Networks

We formulate single target localization as a standard parameter estimation problem with an efficient data collection mechanism. This section will discuss the necessity of adding a data collection mechanism to the standard parameter estimation problem due to the distributed nature of data in a sensor network. Thus, the considerations presented in this section can be applied to any parameter estimation problem where data is distributed throughout nodes of the sensor network.

In the standard parameter estimation problem, we have two spaces, the space of parameters Θ and the space of measurements $\mathcal{Z}$. With a given model between parameters and measurements, the problem is to construct an estimator which computes the "best" estimate $\hat{\theta} \in \Theta$ of the parameter generating the observed measurement $z \in \mathcal{Z}$ according to some criterion of the "best" estimate. Figure 2-6 shows the relationship of these different quantities in the usual parameter estimation problem. Some unknown parameter θ generates the observed measurement z from which we must construct an estimator which outputs an estimate $\hat{\theta}$ of θ.

In sensor networks, we now have several sensor measurements observed by several sensor nodes. Figure 2-7 shows the relationship between the various parts of the estimation process, which is the same as before except that we now explicitly show the measurements coming from the different sensor nodes. The measurement space is now

$$\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n \ ,$$

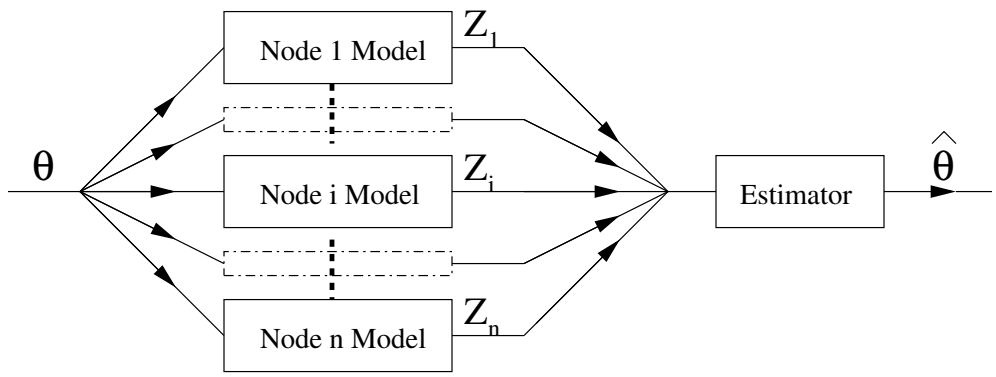

Figure 2-7: Estimation process in a sensor network.

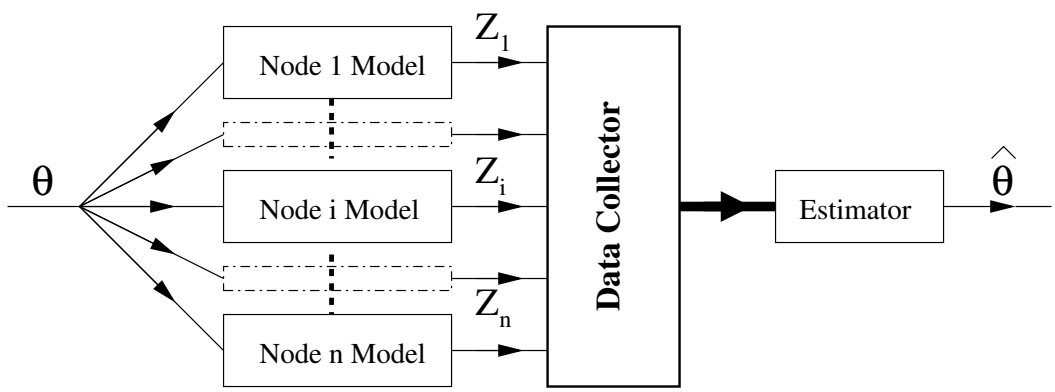

Figure 2-8: Estimation process in a sensor network with data collection.

the Cartesian product of the measurement spaces of each sensor $i \in \{1, \ldots, n\}$. The estimator uses the observed sensor measurements $\{z_i\}_{i=1}^n$ from every sensor node to compute the estimate $\hat{\theta}$. This figure reveals an implied assumption of the usual parameter estimation framework, which is that the estimator has *access to all measurements*. However, in a real sensor network, a sensor node only has *access to its own measurements* initially, and in order to compute the estimate, remote sensor measurements must be collected. Since the process of data collection requires communication and communication is the largest source of power consumption, the data collection algorithm must be efficient. Figure 2-8 shows the estimation process in sensor networks with the added component of data collection.

Thus, in formulating the problem of single target localization in sensor networks, we must formulate an efficient data collection mechanism which collects the data needed to compute the estimate. What makes this problem especially interesting

is due to the fact that information is redundant. Redundancy implies that not all sensor measurements need to be collected all the time. Furthermore, which sensor measurements need to be collected depends on the actual target location. Since we cannot know the actual target location but can infer the target location based on the observed sensor measurements, this means that determining what data to collect should be dependent on what the observed sensor measurements are.

There is a chicken and egg problem inherent in this data collection problem. What measurements to collect can be determined if we know what the measurements are. On the other hand, we cannot know what the measurements are until they have been collected. This chicken and egg problem cannot be made efficient if we restrict ourselves to think of the process of determining what to collect and the process of collecting measurements as two separate black boxes which must be performed one after the other. The key idea to make the data collection mechanism efficient is to open up the black boxes and perform smaller chunks of each computation in an iterative manner so that the resulting algorithm makes more efficient use of the resources available. This idea of opening up the black box and doing little pieces of computation while keeping track of what has been computed so far to determine what computation to perform next is the principle of information-based computation which is the general theme of this thesis in order to design computationally efficient algorithms.

2.3.2 Problem Formulation

The main constraint on sensor nodes of the sensor network is that nodes run on batteries, and to ensure that the sensor network can run as long as possible, we must conserve energy. As discussed before, communication is the most significant consumer of energy so that efficiency will be based solely on the cost attributed to communication.

Let us define some standard notation to be used throughout this section. The sensor network is comprised of n sensor nodes labelled by an integer i from the set $\{1, \ldots, n\}$. Each sensor node i observes measurements $z_i(t) \in \mathcal{Z}_i$ from its sensors at each time $t = 0, 1, \ldots$. We allow the measurement space $\mathcal{Z}_i$ for each sensor node to be different due to the possibility that different sensor nodes can be outfitted with different kinds of sensors. We will denote the collection of all sensor measurements at time t by

$$z(t) = (z_1(t), \ldots, z_n(t)) \ ,$$

and the corresponding combined space is the Cartesian product of all individual sensor measurement spaces

$$\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n \ .$$

Each sensor node i knows its own sensor measurement model which is represented as a likelihood function

$$p_i(z_i|x)$$

where $z_i \in \mathcal{Z}_i$ and $x \in \mathbb{R}^2$ denotes the target location.

As discussed in Section 2.1.2, a configuration phase is assumed to have been completed so that each node i of the network knows the measurement models

$$\{p_j(z_j|x)\}_{j \in \mathcal{N}(i)}$$

generating the data for each of its neighbors $\mathcal{N}(i) \subset \{1, \ldots, n\}$. Typically, the set of neighbors of each sensor node is the subset of sensor nodes which are within a certain radius around the node. We will now begin the formulation of the single target localization problem by considering the two aspects of the problem:

1. What sensor data must be collected so that the computation of an estimate of reasonable quality is possible? Section 2.3.2.1 describes how to formalize this question and determine what data to collect based on what the data is.

2. What are the possible data collection mechanisms on a sensor network? Section 2.3.2.2 describes the general sensor node operations from which a data collection mechanism is constructed.

Finally, Section 2.3.2.3 will state the problem for single target localization as well as extend the problem statement to more general domains.

2.3.2.1 Determining Informative Data

We consider the inference task of single target localization to be the computation of the posterior distribution of the target position $x \in \mathbb{R}^2$ given sensor measurements $z = (z_1, \ldots, z_n) \in \mathcal{Z}$. Given a density of the prior distribution of the target position $p_0(x)$ and under the assumption that the random elements of each sensor measurement are independent, the posterior density of the target position given all sensor measurements is

$$\tilde{p}_z(x) = p_0(x) \prod_{i=1}^{n} p_i(z_i|x)$$

where $p_i(z_i|x)$ is the likelihood for the measurement z_i of sensor node i given the target location x. We have explicitly notated the observed sensor measurements z that the posterior density depends on.

If only a subset $S \subset \{1, \ldots, n\}$ of the sensor measurements is available, the posterior density of the target location based on this subset of sensor data can be computed by

$$\tilde{p}_{z_S}(x) = p_0(x) \prod_{i \in S} p_i(z_i|x)$$

where $z_S = (z_{j_1}, \ldots, z_{j_m})$ when $S = \{j_1, \ldots, j_m\}$. The notion that information is redundant means that there exists a subset $S \subset \{1, \ldots, n\}$ of the sensor nodes such that

$$\tilde{p}_{z_S}(x) \approx \tilde{p}_z(x)$$

where we must specify a notion of approximation $\approx$. For example, one measure of what it means for $\tilde{p}_{z_S}$ to approximate $\tilde{p}_z$ could be based on the Kullback-Leibler distance

$$D(p(x)||q(x)) = \int p(x) \log \frac{p(x)}{q(x)} dx$$

for two densities p and q. That is, we consider $\tilde{p}_{z_S}$ to approximate $\tilde{p}_z$ if

$$D(\tilde{p}_{z_S}(x)||\tilde{p}_z(x)) \leq \gamma$$

for some chosen threshold γ. Another criterion could be based on the sup norm

$$\sup_{x \in \mathbb{R}^2} |\tilde{p}_{z_S}(x) - \tilde{p}_z(x)| \leq \gamma$$

for some chosen threshold γ. The approximation criterion is the formalization of the notion of "an estimate of reasonable quality". Once an appropriate approximation criterion has been chosen, we can then determine the minimal set of data must be collected.

Since what data should be collected depends on what the data is, we will compute the subset of sensor measurements to collect for every likely combination of sensor measurements $z \in \mathcal{Z}_{likely} \subset \mathcal{Z}$. By restricting ourselves to likely combinations of measurements, we are putting no constraints on what data must be collected when it is highly unlikely that any target may have generated the combination of measurements. For example, this set can be

$$\mathcal{Z}_{likely} = \{z \in \mathcal{Z} : \prod_{i=1}^{n} p_i(z_i|x) > \gamma \text{ for some } x \in F\}$$

where γ is a threshold for the likelihood and $F \subset \mathbb{R}^2$ is the field which we define in Definition 2.7.

The minimal subset $S_z \subset \{1, \ldots, n\}$ of sensor measurements to collect when the collective sensor data is $z \in \mathcal{Z}_{likely}$ is given by

$$S_z = \arg_{S \subset \{1,\ldots,n\}: \tilde{p}_{z_S} \approx \tilde{p}_z} \min |S|$$

where $|\cdot|$ denotes the cardinality of the set. For those $z \in \mathcal{Z} - \mathcal{Z}_{likely}$, we can set

$$S_z = \emptyset \ .$$

Thus, in order to compute an estimate of reasonable quality, the data collection mechanism must collect the sensor measurements of the sensor nodes in S_z if for each sensor $i \in S_z$, the measurement observed by sensor i is equal to z_i.

To conclude, based on a measure of approximation to the complete data posterior distribution $\tilde{p}_z$, we have computed a subset of sensor measurements S_z to collect based on the what the data z is which will serve as a constraint on what data must be collected in order for target localization to be considered successful. The next subsection will discuss the mechanism of how the sensor nodes can communicate data to each other.

2.3.2.2 Data Collection Mechanism

In order to describe the data collection mechanism in sensor networks, we need to define a local knowledge state, a communication rule, and a communication algorithm so that each sensor node can decide what data to communicate based on what it currently knows. We then describe how the communication algorithm is used in the operation of a sensor node. The concurrent operation of sensor nodes results in a communication history which will be defined so that the cost of communication can be measured.

Definition 2.1 *The* set of local knowledge states $\mathcal{B}$ *is the collection of subsets of time, sensor node, and measurement value triples defined by*

$$\mathcal{B} \subset 2^{\left\{[0,\infty)\times\{1,\ldots,n\}\times\left(\bigcup_{i=1}^{n}\mathcal{Z}_i\right)\right\}}$$

such that for every local knowledge state $B \in \mathcal{B}$*,*

1. *if* (t, i, z) *is an element of* B*, then* z *is an element of* $\mathcal{Z}_i$ *and*
2. *for every pair of same time elements* (t, i, z_i) *and* (t, j, z_j) *in* B*,* i *is not equal to* j*.*

Condition (1) for a local knowledge state $B \in \mathcal{B}$ is that measurements believed to come from sensor node i must be an element of its sensor measurement space $\mathcal{Z}_i$, and condition (2) is that each sensor node observes a single, unique value at any time t. The local knowledge state of sensor node i at time t is denoted B_i^t, which represents the collection of measurements that node i knows at time t. Initially at time 0, node i knows only its own measurement $z_i \in \mathcal{Z}_i$, so that $B_i^0 = \{(0, i, z_i)\}$. When sensor nodes communicate with each other, local knowledge states of sensor nodes will be augmented appropriately, and thus, nodes can gain knowledge of other sensor node measurements.

We remark here that the way a local knowledge state has been defined in Definition 2.1 is a set theoretic one of conceptual value. In a real implementation, a relevant summary of the information contained in the collection of data stored by the local knowledge state can suffice to represent the local knowledge state for a particular global task. Such summaries are minimal sufficient statistics, and the effect of these partial computations during the data collection process is to reduce the number of bits that must be transmitted since these partial computations essentially extract the relevant information contained within the collection of data and strip away any unnecessary information contained in the data. We will formulate the single target localization problem without the possibility of partial computation first, and later, extend the problem formulation to include the possibility of partial computation. In Chapter 4 where we will solve the target localization problem formulated here, we will take a similar approach and consider data collection mechanisms which pass data

around first. Then, we will refine these data collection mechanisms to information collection mechanisms which include partial computations that only pass around the minimal sufficient statistics needed for computing the global task.

Definition 2.2 *A* communication rule *is a triple* (B, R, Q) *where*

$B \in \mathcal{B}$ *is local knowledge state,*

$R \subset \{1, \ldots, n\}$ *is a set of sensor nodes which should receive the data in* B*,*

$Q \subset \{1, \ldots, n\}$ *is a set of sensor nodes which are queried for their local knowledge state.*

The motivation behind defining a communication rule is that what data to collect should be dependent on what data has been currently collected. Local knowledge states B represent the data that has been currently collected. Collection of data based on this local knowledge state comes in two forms: (1) we can send this local knowledge state to remote sensor nodes R or (2) we can query for the local knowledge state of remote sensor nodes Q. We can now conglomerate a set of communication rules to define a communication algorithm.

Definition 2.3 *A* communication algorithm *is a subset of the set of all communication rules*

$$C \subset \mathcal{B} \times 2^{\{1,\ldots,n\}} \times 2^{\{1,\ldots,n\}} ,$$

such that for any (B_1, R_1, Q_1) *and* (B_2, R_2, Q_2) *in* C*,* $B_1 \neq B_2$*. The* set of all communication algorithms *is denoted* $\mathcal{C}$*.*

The condition $B_1 \neq B_2$ essentially means that a communication algorithm C is the set theoretic equivalent of a partially defined function

$$C : \mathcal{B} \to 2^{\{1,\ldots,n\}} \times 2^{\{1,\ldots,n\}} .$$

We say "partially defined" because C does not necessarily contain a triple $(B, \cdot, \cdot)$ for all elements of $\mathcal{B}$. This can be easily padded by assuming that if there exists no triple $(B, \cdot, \cdot)$ in C for some $B \in \mathcal{B}$, then we assume that the triple $(B, \emptyset, \emptyset)$ is in C. A triple $(B, \emptyset, \emptyset)$ is interpreted as no communication link being established when the local knowledge state is B.

Sensor Node Operation

Let us take a moment now to describe the basic operation of sensor nodes for data collection and estimation. We assume a communication algorithm $C_i \in \mathcal{C}$ is given to each sensor node. We consider the case when sensor nodes operate synchronously and all sensor nodes observe their sensor measurements at time t only. The idea is that the communication algorithms encode a data collection mechanism for the sensor

measurements observed in a single instance of time. Thus, we can extend to the case when we perform localization for several time instances by replicating the following algorithm for each sensor observation time.

Initially, at time t, each sensor node i collects its local measurement from its sensors $z_i(t)$ and stores this as an initial local knowledge state

$$B_i^t = \{(t, i, z_i(t))\}$$

for each $i \in \{1, \ldots, n\}$. Furthermore, each sensor node i must keep track of the set of measurements that node i believes is known by other remote sensor nodes $j \neq i$,

$$D_{i,j}^t \subset \{1, \ldots, n\} \ ,$$

because we do not wish to repeatedly send the same data to the same sensor nodes. Initially, for all sensor nodes $i \in \{1, \ldots, n\}$,

$$D_{i,j}^t = \{j\}$$

for all $j \in \{1, \ldots, n\}$ and $j \neq i$ since every sensor node i knows that any remote sensor node j has knowledge of its own measurements. Then, the following operations occur for each time $\tau = t+1, t+2, \ldots$ in sensor node i.

1. *(Receiving Communications)* At time τ, the communication device may return a local knowledge state $B_{com,i}^\tau$ sent from other sensor nodes. If there is nothing returned, we denote this by setting $B_{com,i}^\tau = \emptyset$. The communicated local knowledge state will augment node i's local knowledge state B_i^τ according to

   $$B_i^\tau = B_i^{\tau-1} \cup B_{com,i}^\tau$$

   where $B_i^{\tau-1}$ is sensor node i's previous local knowledge state. If query messages were received, then we store the set of nodes which were the sources of those query messages in

   $$\tilde{R}_i^\tau \subset \{1, \ldots, n\} \ .$$

   If no query message was received, we set $\tilde{R}_i^\tau = \emptyset$.

2. *(Processing)* Then, the local knowledge state B_i^τ is fed to the communication algorithm C_i in node i. For each $b \subset B_i^\tau$, we compute

   $$(r_b^\tau, q_b^\tau) = C_i(b) \ .$$

   Because the space of local knowledge states $\mathcal{B}$ is a set of subsets, we must perform the above auxiliary evaluations of C_i because any subset $b \subset B_i^\tau$ is consistent with the local knowledge state B_i^τ. Then, we set

   $$R_i^\tau = \bigcup_{b \subset B_i^\tau} r_b^\tau$$

and

$$Q_i^\tau = \bigcup_{b \subset B_i^\tau} q_b^\tau$$

where the subset R_i^τ is the set of nodes to send the local knowledge state B_i^τ and the subset Q_i^τ is the set of nodes to query for their local knowledge state.

3. *(Transmitting Communications)* From the subsets R_i^τ and Q_i^τ returned by the communication algorithm and the nodes which sent query messages $\tilde{R}_i^\tau$, we can determine what data and messages to send to remote sensor nodes. The set of sensor nodes whose measurements observed at time t are elements of the local knowledge state B_i^τ is defined by

$$\mathbf{sensors}_t(B_i^\tau) = \{j \in \{1, \ldots, n\} : (t, j, z_j) \in B_i^\tau \text{ for some } z_j \in \mathcal{Z}_j\} \ . \tag{2.9}$$

Consider the subset of sensor nodes defined by

$$\mathbf{R}_{i,j}^\tau = \mathbf{sensors}_t(B_i^\tau) - D_{i,j}^{\tau-1} \ .$$

$\mathbf{R}_{i,j}^\tau$ is the set of sensor measurements that node i knows at time τ and believes is not known by sensor j. The set of sensor nodes which should be made aware of all the data contained in B_i^τ are the sensor nodes in $R_i^\tau \cup \tilde{R}_i^\tau$. Since sensor i is aware of which sensor measurements are known by remote sensors according to $D_{i,j}^{\tau-1}$, we wish to send only those data which sensor i believes is not known by the remote sensor. Thus, putting this together, the local knowledge state to be sent to sensor node j by sensor node i is

$$\mathbf{B}_{i,j}^\tau = \begin{cases} \{(t, k, z_k) \in B_i^\tau : k \in \mathbf{R}_{i,j}^\tau\} & , \ j \in R_i^\tau \cup \tilde{R}_i^\tau \\ \emptyset & , \ \text{otherwise} \end{cases} \ .$$

Note that the communication protocol being used for nodes in $R_i^\tau \cup \tilde{R}_i^\tau$ is a "push" type of communication because information is pushed out of this sensor node to other remote nodes. Remote nodes $j \in Q_i^\tau$ are sent a query message for the local knowledge state of sensor node j. Note that the communication protocol being used for nodes in Q_i^τ is a "pull" type of communication because information is pulled out of the nodes in Q_i^τ by this sensor node. Finally, we must update the beliefs about the local knowledge state of remote sensors by

$$D_{i,j}^\tau = \begin{cases} D_{i,j}^{\tau-1} \cup \mathbf{R}_{i,j}^\tau & , \ j \in R_i^\tau \cup \tilde{R}_i^\tau \\ D_{i,j}^{\tau-1} & , \ \text{otherwise} \end{cases} \ .$$

4. *(Propagation of Communications)* The result of the transmissions sent out are the following. The local knowledge state received by the wireless receiver of each sensor i at the next time $\tau + 1$ is

$$B_{com,i}^{\tau+1} = \bigcup_{j \in \{1,\ldots,n\} - \{i\}} \mathbf{B}_{j,i}^\tau \ .$$

The query messages received by the wireless receiver of each sensor i at the next time $\tau + 1$ is

$$\tilde{R}_i^{\tau+1} = \{j \in \{1, \ldots, n\} - \{i\} : i \in Q_j^\tau\} \ .$$

With a communication algorithm C_i specified for each sensor node $i \in \{1, \ldots, n\}$ in the sensor network, the concurrent operation of all sensor nodes will result in a passing around of data throughout the sensor network. We can now formally define the history of communications established so that we can measure the cost of the communications occurring during the the concurrent operation of the sensor nodes with communication algorithms $\{C_i\}_{i=1}^n$.

Definition 2.4 *The* communication history of data observed at time t *passed around a sensor network of n nodes where each sensor node $i \in \{1, \ldots, n\}$*

- *has communication algorithm $C_i \in \mathcal{C}$ and*
- *observes data $z_i(t) \in \mathcal{Z}_i$ at time t*

is defined to be a time series of local knowledge states and query messages passed around the network

$$\mathcal{H}_\tau(\{C_i\}_{i=1}^n, z(t)) = (\{\mathbf{B}_{i,j}^\tau\}_{i,j \in \{1,\ldots,n\}}, \{\tilde{R}_i^\tau\}_{i \in \{1,\ldots,n\}})$$

for $\tau = t, t+1, \ldots$ where $\mathbf{B}_{i,j}^\tau$ and $\tilde{R}_i^\tau$ are computed according to the description given about the sensor node operation above.

The dependence of the communication history on the communication algorithms of each sensor node and the observed data has been explicitly shown in the notation above, but we will refer to the history as

$$(\mathcal{H}_t)_{t \in [0,\infty)}$$

when it is obvious what the communication algorithms and data are.

2.3.2.3 Problem Statement

To state the single target localization problem formally, let us define some terms.

- Let $\mathcal{X} \subset \mathbb{R}^2$ be the set of target locations we wish to estimate.
- The measurement space of sensor node i is $\mathcal{Z}_i$. We use the shorthand notation

  $$\mathcal{Z}_i^j = \mathcal{Z}_i \times \mathcal{Z}_{i+1} \times \cdots \times \mathcal{Z}_j \ .$$

  In particular, the combined measurement space is

  $$\mathcal{Z} = \mathcal{Z}_1^n \ .$$

- For each $z \in \mathcal{Z}$, using some appropriate approximation criterion formalizing the notion of an estimate of reasonable quality, we have the set of sensor measurements
$$S_z \subset \{1, \ldots, n\}$$
that must be collected when the observed sensor measurements are z.

- $\{C_j\}$ is shorthand for the collection of communication algorithms $\{C_j\}_{j=1}^n$.

- $z(t)$ is shorthand for the vector of all sensor measurements observed at time t, $(z_1(t), \ldots, z_n(t))$.

- If we allow the sensor network to collect data at time t only, then all communications eventually cease, and each sensor node i will be carrying a local knowledge state
$$B_i^\infty(\{C_j\}, z(t)) \in \mathcal{B}$$
referred to as the *final local knowledge state*, where we show explicitly the dependence of the final local knowledge state on the collection of communication algorithms and observed sensor measurements.

Single Target Localization Problem *The problem is to choose a collection of communication algorithms $C_j \in \mathcal{C}$ for each sensor node $j \in \{1, \ldots, n\}$ so that if data is collected at time t only, once all communications have ceased, there exists some node $i \in \{1, \ldots, n\}$ such that*

$$\mathbf{sensors}_t(B_i^\infty(\{C_j\}, z(t))) \supset S_{z'} \tag{2.10}$$

for some $z' \in \mathcal{Z}$ with
$$z'_k = z_k(t)$$
for all $k \in S_{z'}$. Furthermore, given some distribution P over the space of measurements $\mathcal{Z}$, the average communication cost

$$\int_{\mathcal{Z}} \sum_{\tau=t}^{\infty} \mathbf{cost}\left(\mathcal{H}_\tau(\{C_j\}, z)\right) dP(z) \tag{2.11}$$

of the chosen collection of communication algorithms $\{C_j\}$ is minimal. The cost function
$$\mathbf{cost} : \mathcal{B}^{n^2} \times (2^{\{1,\ldots,n\}})^n \to [0, \infty)$$
is the communication cost incurred by the communication of local knowledge states and query messages during a single iteration of the concurrent operation of the sensor nodes.

The optimization in the problem above is over possible collections of communication algorithms $\{C_j\}$ over the n nodes of the sensor network. The idea is that after

communications have settled down, the node i which contains enough data in its local knowledge state B_i^∞ to compute the posterior density of the target position. The estimated posterior density is computed by

$$\tilde{p}_{est}(x) = p_0(x) \prod_{(t,i,z_i)\in B_i^\infty} p_i(z_i|x) \ .$$

Assuming we can solve the target localization problem, we are guaranteed that this posterior density approximates the posterior density computed with all sensor measurements

$$\tilde{p}_{est} \approx \tilde{p}_z \ .$$

The relation (2.10) is the information constraint of the problem which only allows as solutions those collections of communication algorithms which collect the necessary data so that an estimate of reasonable quality can be computed. However, note that we only ask that some sensor node i has enough data to compute an estimate and do not specify which sensor node must have the data. This is a degree of freedom which is allowed because we assume that there exist information retrieval processes that can retrieve information from the sensor network. Of the collection of communication algorithms that satisfy the information constraint (2.10), we wish to choose the particular collection which minimizes the average amount of communication cost incurred in the history $(\mathcal{H}_\tau)_{\tau=t}^\infty$ averaged over the possible combinations of measurement values observed by the sensors at time t. This minimization of the communication cost is expressed in the cost optimization criterion (2.11). Thus, the optimal collection of communication algorithms is the one which on average expends the least amount of communication cost. Any cost criterion can be considered by changing the expression for (2.11).

Of particular interest will be the cost of communication according to the energy consumption model of Section 2.1.3. Let k_{float} be the number of bits needed to represent a floating point number, $k_{integer}$ be the number of bits needed to represent an integer, and $k_{message}$ be the number of bits needed to differentiate between a local knowledge state message and a query message. Typically, the number of bits to represent the various data types are

$$\begin{aligned} k_{float} &= 32 \\ k_{integer} &= 16 \\ k_{message} &= 1 \ . \end{aligned}$$

In this case, $k_{message}$ is only 1 bit because we have only two message types. The number of bits needed to represent a local knowledge state message $B \in \mathcal{B}$ is

$$k_B = k_{message} + 2 \cdot k_{integer} + |B| \cdot (k_{float} + k_{integer} + k_{float})$$

where we transmit the type of message, the destination sensor node label, the cardinality of the local knowledge state B, and for each element of B, we send the time,

sensor node, and sensor measurement triple. We have assumed that the sensor measurement is an element of the reals $\mathbb{R}$. The number of bits needed to send a query message is

$$k_{query} = k_{message} + 2 \cdot k_{integer} + k_{float}$$

where we transmit the type of message, the source sensor node label, the destination node label, and the time of the sensor measurements to query. We send the source node label because in a query message the destination node must know to which node to send data.

Thus, if the history at time τ is

$$\mathcal{H}_\tau = (\{\mathbf{B}^\tau_{i,j}\}_{i,j\in\{1,\ldots,n\}}, \{\tilde{R}^\tau_i\}_{i\in\{1,\ldots,n\}}) \ ,$$

the total communication cost based on the energy consumption model, where by "total" we mean the amount of energy consumed by the network as a whole, is

$$\begin{aligned}\mathbf{cost}(\mathcal{H}_\tau) \quad = \quad & \sum_{\substack{i,j\,\in\,\{1,\ldots,n\}:\\ \mathbf{B}^\tau_{i,j}\neq\emptyset}} \Big(E_{Tx}(k_{\mathbf{B}^\tau_{i,j}}, d_{i,j}) + E_{Rx}(k_{\mathbf{B}^\tau_{i,j}})\Big) \\ & + \sum_{\substack{i\,\in\,\{1,\ldots,n\}:\\ \tilde{R}^\tau_i\neq\emptyset}} \sum_{j\in\tilde{R}^\tau_i} (E_{Tx}(k_{query}, d_{i,j}) + E_{Rx}(k_{query}))\end{aligned}$$

where $d_{i,j}$ is the distance between sensor nodes i and j,

$$E_{Tx}(k,d) = E_{elec}k + \epsilon_{amp}kd^2 \ ,$$

and

$$E_{Rx}(k) = E_{elec}k$$

which is the energy consumption model reprinted from Section 2.1.3.

Discussion and Extensions

The solution space for this problem is overwhelming. The first problem is that the space of *collections* of communication algorithms is large. Then, determining whether a particular collection of communication algorithms satisfies the information constraint (2.10) requires simulating the communication algorithms and recording communication histories for all time series of sensor measurements. Finally, the minimal cost collection of communication algorithms must be chosen according to the cost criterion (2.11). We will use the hierarchy representation to construct the set of communication algorithms $\{C_i\}_{i=1}^n$ which satisfy the information constraint (2.10) and with which we can heuristically optimize the cost criterion (2.11).

The statement of the single target localization problem is quite general in that the only place where the parameter estimation enters is through the information

constraint (2.10) of the set of sensor measurements that must be collected S_z for each $z \in \mathcal{Z}$. Thus, the problem statement applies for any parameter estimation problem as long as the set of data S_z to be collected are computed.

To extend the problem statement to include the possibility of partial processing so that we do not just pass the observed sensor measurements around, we must modify the sensor node operation to include the possibility of performing partial processing on collections of sensor measurements which return some minimal representation of the information contained in the set of sensor measurements. By minimal representation, we mean that the number of bits needed to represent the information is minimal. It is possible that the minimal representation is exactly the collection of sensor measurements although, in general, this is not true. The effect of allowing such partial processing is that the number of bits needed to transmit a local knowledge state B is no longer k_B as defined above. Instead, if there is some computation

$$\Psi : \mathcal{B} \to \mathcal{I}$$

where $\mathcal{I}$ is some information representation space which converts a local knowledge state into some minimal representation of the information contained in B, then the number of bits needed to transmit the information is $k_{\Psi(B)}$, which we have assumed to be less than or equal to k_B since the representation is minimal. Thus, incorporating the possibility of partial processing of data into the problem formulation means to convert all local knowledge states B into its appropriate minimal representation $\Psi(B)$ with appropriate routines to update the information representation when new information becomes available. It should be clear that allowing partial processing of data can reduce the amount of communication cost incurred. However, there are extra computational resources needed to compute Ψ so that the savings of partial processing are weighed with the cost of computing the minimal information representation.

2.4 Tracking Formulation

In this section, we formulate the problem of multiple target tracking. As in the formulation of target localization, we also put a focus on the computational resources needed in order to perform multiple target tracking up to a reasonable degree of accuracy. Let us define some terms.

Definition 2.5 *A* target trajectory *is a time series of target locations,*

$$\{x(t)\}_{t=0}^{\infty}$$

where at each time, $x(t) \in \mathcal{X}$. The space $\mathcal{X}$ is the space of possible target positions and is equal to some d dimensional Euclidean space $\mathbb{R}^d$.

For example, for target trajectories on the Euclidean plane, $\mathcal{X} = \mathbb{R}^2$.

Definition 2.6 *A* target state trajectory *is a time series of the target state,*

$$\{\theta(t)\}_{t=1}^{\infty}$$

where at each time, $\theta(t) \in \Theta$. The space Θ is the state space of the target which includes all information about the target.

We use the usual notion of state which is that the state of a target captures all information about the target. For example, the state space of a target that includes the target position as well as other information like velocity and acceleration, the nominal intensity emitted by the target, and the nominal frequency bandlimited intensities is

$$\Theta = \mathbb{R}^6 \times \mathbb{R} \times \mathbb{R}^b$$

where we are assuming that targets move on the Euclidean plane so that the first 6 components are the position, velocity, and acceleration components of the target, the 7th component is the nominal intensity, and the last b components are the intensities in b frequency bands. In these examples of the target trajectory and target state trajectory, the relationship between the two is

$$x(t) = \pi_{1,2}(\Theta(t))$$

where $\pi_{1,2}$ pulls out the first and second components of $\Theta(t)$ which correspond to the target position.

Definition 2.7 *The* field *is a bounded subset $F \subset \mathcal{X}$ of the set of possible target positions which are in range of the sensor network.*

For example, if the sensor network is composed of n sensor nodes where each sensor node $i \in \{1, \ldots, n\}$ is located at $x_i \in \mathcal{X}$, then the field can be defined as

$$F = \{x \in \mathcal{X} : d(x, x_i) \leq \gamma \text{ for some } i \in \{1, \ldots, n\}\}$$

where $d(\cdot, \cdot)$ is a metric on $\mathcal{X}$ and γ is the range of the sensors in the sensor network.

Definition 2.8 *A* track *T is a time series of estimated target positions with associated error matrix with a start time, denoted T_{start}, and stop time, denoted T_{stop},*

$$T = \{(x^T(t), \Sigma^T(t))\}_{t=T_{start}}^{t=T_{stop}}$$

where $x^T(t) \in \mathcal{X}$ and $\Sigma^T(t)$ is a $d \times d$ matrix if $\mathcal{X} = \mathbb{R}^d$. Furthermore, the estimated target positions $x^T(t)$ and error matrices $\Sigma^T(t)$ are causal functions of sensor data meaning that these quantites are functions of $\{z(\tau)\}_{\tau=0}^{t}$ where $z(\tau) \in \mathcal{Z}$ is the collection of observed sensor measurements at time τ.

The idea is that a track is something we can compute from the data. Furthermore, we would like these tracks to correspond to an actual target trajectory, so it will be necessary to define some notion of how well a track corresponds to a target trajectory. In particular, we only care that a track matches a target when the target is within the field. The times when a target trajectory $x(t)$ is within the field $F \subset \mathcal{X}$ is

$$\{t \in \{0, 1, \dots\} : x(t) \in F\}$$

which can be represented as the union of several discrete intervals of time

$$\bigcup_{i=1}^{l} \{t_{i,1}, t_{i,1} + 1, \dots, t_{i,2}\}$$

where l is the number of intervals in our representation and for each $i \in \{1, \dots, l-1\}$, $t_{i,2} \leq t_{i+1,1}$ and $t_{i,1} \leq t_{i,2}$. In particular, we will be interested in the representation of the union of the longest intervals of consecutive time. This means that for each $i \in \{1, \dots, l-1\}$, $t_{i,2} + 1 < t_{i,1}$. We will denote each of these intervals by

$$I_i^{x(t)} = \{t_{i,1}, t_{i,1} + 1, \dots, t_{i,2} - 1, t_{i,2}\}$$

for $i \in \{1, \dots, l\}$ where l is the number of intervals. We define a matching function $M(T, x(t), I)$ which returns a nonnegative real number corresponding to how well the track T matches the target trajectory $x(t)$ over the time interval I. Lower values of $M(T, x(t), I)$ correspond to better matches between the track and target trajectory, and a value of 0 means a perfect match. One example of a matching function is

$$\begin{aligned} M(T, x(t), I) &= c_1 |(I^T \Delta I)| \\ &\quad + c_2 \sum_{\tau \in I \cap I^T} (x(\tau) - x^T(\tau))' (\Sigma^T(\tau))^{-1} (x(\tau) - x^T(\tau)) \\ &\quad + c_3 \sum_{\tau \in I \cap I^T} \det(\Sigma^T(\tau)) \end{aligned}$$

where

$$I^T = \{T_{start}, T_{start} + 1, \dots, T_{stop}\}$$

is the interval of time corresponding to the track T, $(I^T \Delta I)$ is the symmetric difference, and c_1, c_2, and c_3 are constants which weigh the three terms. The matching function M above contains three terms which penalize the match between track and target trajectory.

1. The first term incurs a penalty if the time interval I^T of the track does not match the time interval I for which we wish to compare the track and target trajectory.

2. The second term measures how well the target trajectory matches the estimated track position weighed by how much error we expect. Note that a track with very large eigenvalues for each $\Sigma^T(\tau)$ will necessarily incur a small error. In order to avoid making this term go artificially to zero, we include the third term.

3. The third term measures the quality of the estimated track. Thus, we penalize those tracks which have large $\Sigma^T(\tau)$.

It will be useful to consider the adjoining of two tracks.

Definition 2.9 *Two tracks T and R are said to be* adjoinable *if*

$$T_{stop} = R_{start} - 1 \ .$$

The resulting adjoined track *is denoted $T \circ R$ and is defined by*

$$T \circ R = \{(x^{T\circ R}(t), \Sigma^{T\circ R}(t))\}_{t=(T\circ R)_{start}}^{(T\circ R)_{stop}}$$

where

$$x^{T\circ R}(t) = \begin{cases} x^T(t) & , \ T_{start} \leq t \leq T_{stop} \\ x^R(t) & , \ R_{start} \leq t \leq R_{stop} \end{cases}$$

$$\Sigma^{T\circ R}(t) = \begin{cases} \Sigma^T(t) & , \ T_{start} \leq t \leq T_{stop} \\ \Sigma^R(t) & , \ R_{start} \leq t \leq R_{stop} \end{cases}$$

$$(T \circ R)_{start} = T_{start}$$

$$(T \circ R)_{stop} = R_{stop} \ .$$

Finally, we define a hypothesis.

Definition 2.10 *A* hypothesis *is a collection of tracks. The space of all hypotheses is denoted $\mathcal{H}$.*

Now, we can state the problem of multiple target tracking in sensor networks.

Multiple Target Tracking Problem *Given*

- *any set of m target trajectories $\{x_j(t)\}_{j=1}^m$ for $t = 0, 1, \ldots$ in the space target positions $\mathcal{X}$,*

- *a field $F \subset \mathcal{X}$,*

- *a set of n sensor nodes which generate a time series of measurements $z(t) \in \mathcal{Z}$ for $t = 0, 1, \ldots$ where $z(t) = (z_1(t), \ldots, z_n(t))$,*

we wish to construct a tracking algorithm

$$\mathbf{T} : \mathcal{Z}^{\{0,1,\dots\}} \to \mathcal{H}$$

which takes as input the observed time series of sensor measurements $z(t)$ and outputs a hypothesis H,

$$H = \mathbf{T}(\{z(t)\}_{t=0}^{\infty}) \ ,$$

such that H is a hypothesis with a minimal number of tracks and for each maximally connected interval of time $I_i^{x_j(t)}$ that target j is within the field F, there exists a track $T \in H$ or an adjoined track $T = T_1 \circ \cdots \circ T_k$ where $T_1, \dots, T_k \in H$ such that

$$M(T, x_j(t), I_i^{x_j(t)}) \leq \gamma \tag{2.12}$$

for some chosen γ which controls how well tracks must match the targets. Furthermore, of all tracking algorithms $\mathbf{T}$ which satisfy the constraint (2.12), we want to choose the tracking algorithm which minimizes the amount of computational resources needed for computing the hypothesis.

The statement of the multiple target tracking problem defines the requirement of what we consider a valid solution by the tracking algorithm as stated in the requirements of the hypothesis H. This solution is motivated by the fact that in some cases it is impossible to perform data association because discriminatory information is unavailable. Thus, the best we can do is to return tracks which correspond to parts of the target trajectory so that by adjoining the adjoinable tracks in H, we can recover the true target trajectory if extra information is available later.

The multiple target tracking problem is the central problem we intend to solve using the methodology presented in this thesis. However, in order to solve the multiple target tracking problem, we also need to solve the target localization problem and classification problems. The next section will formulate a classification problem.

2.5 Classification Formulation

In this section, we will formulate two classification problems.

- The first problem is the problem of classifying the type of target based on the frequency content of the acoustic signal measured by the sensors. This problem is non-trivial due to the possibility of multiple targets in the field.

- The second problem is the problem of classifying whether a target is metallic or not based on measurements from a magnetometer. Again, this problem is non-trivial due to the possibility of multiple targets in range of a magnetometer and the $0-1$ output of the magnetometer.

The classification problems above are parameter estimation problems based on the frequency bandlimited intensity measurement model and the magnetometer measurement model. Without any other information about targets, classifying the types of targets in the field is difficult. However, we will only perform classification when target positions are known. This localization information greatly simplifies the classification problems so that the solutions are straightforward.

2.5.1 Classifying Frequency Signatures

To formulate the problem of classifying frequency signatures of multiple targets in the field, let us define some notation. We assume that the number of frequency bands b is finite so that the frequency signature $\{A_\phi\}_{\phi=1}^b$ and the frequency bandlimited intensity measurement from a sensor $\{z_{i,\phi}\}_{\phi=1}^b$ will be represented as vectors

$$\underline{A} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_b \end{bmatrix} \quad \text{and} \quad \underline{z}_i = \begin{bmatrix} z_{i,1} \\ z_{i,2} \\ \vdots \\ z_{i,b} \end{bmatrix} .$$

Classification of Frequency Signatures Problem
Given

- *m targets with known positions $\{x^j\}_{j=1}^m$,*
- *observations from n frequency bandlimited intensity sensors $\{\underline{z}_i\}_{i=1}^n$,*
- *the frequency bandlimited intensity measurement model for multiple targets for each of the i sensors*
$$\underline{p}_i(\underline{z}_i | \{x^j\}_{j=1}^m; \{\underline{A}^j\}_{j=1}^m)$$
where we have shown the dependence of the likelihood on the frequency signature of each target explicitly, and
- *a set of possible nominal frequency signatures $\{\underline{C}^k\}_{k=1}^K$ where $\underline{C}^k$ is the frequency signature of targets from class k,*

the problem is to compute the maximum likelihood estimate of the frequency signature $\hat{\underline{A}}^j$ of each target $j \in \{1, \ldots, m\}$,

$$(\hat{\underline{A}}^1, \ldots, \hat{\underline{A}}^m) = \arg_{\underline{B}^1, \ldots, \underline{B}^j \in \{\underline{C}^k\}_{k=1}^K} \max \prod_{i=1}^n \underline{p}_i(\underline{z}_i | \{x^j\}_{j=1}^m; \{\underline{B}^j\}_{j=1}^m) .$$

We will solve variants of this basic problem when implementing this classification routine on sensor networks. For example, we will be unable to access frequency

bandlimited intensity measurements from all nodes of the network in general so that we must classify targets based on a subset of sensor measurements.

There is a tradeoff emerging in the classification routine. The more sensor measurements we use for classification, the more accurate is our computation of the target class. However, this accuracy comes at the expense of having to collect more sensor measurements. This tradeoff is not surprising since classification is an instance of parameter estimation, and it is exactly this tradeoff which was exploited in the formulation of the target localization problem to determine the minimal set of sensor measurements to collect.

2.5.2 Classifying Metallic Targets

Classifying metallic targets reveals a different problem than classifying target frequency signatures. The difference is due to the fact that when there are several targets in range of a magnetometer and the magnetometer measures a 1, the only conclusion we can draw from this measurement is that at least one of the targets within range is metallic. The problem is that such a hypothesis is a global one regarding the set of targets as a whole. In a sensor network, since we must store global information in some local fashion, we encounter representation issues of these global hypotheses.

To be more precise, for each of the m targets which are within range of the magnetometer, labelled by $j \in \{1, \ldots, m\}$, we associate a Boolean variable W_j where the interpretation is that W_j is equal to 1 if target j is metallic and W_j is equal to 0 if target j is non-metallic. The likelihood function of the magnetometer measurement given m targets within range of the magnetometer is

$$p_{magn}(z|W_1, \ldots, W_m)$$

where the magnetometer measurement z can take value in $\{0, 1\}$ and we show the explicit dependence of the likelihood on the Boolean variables $\{W_j\}_{j=1}^m$. If we take a Bayesian approach, then given some prior joint distribution of the combination of values of the $\{W_j\}_{j=1}^m$ at time t,

$$p_t(W_1, \ldots, W_m) \ ,$$

the posterior distribution of the $\{W_j\}_{j=1}^m$ after incorporating the magnetometer measurement $z(t)$ is given by

$$\tilde{p}_t(W_1, \ldots, W_m) = \alpha p_{magn}(z(t)|W_1, \ldots, W_m) p_t(W_1, \ldots, W_m)$$

where α is a normalization factor so that

$$\sum_{(W_1, \ldots, W_m) \in \{0,1\}^m} \tilde{p}_t(W_1, \ldots, W_m) = 1 \ .$$

The problem is that we must represent this joint posterior distribution $\tilde{p}_t$ in some distributed fashion for each target. One idea is to compute marginals and approximate the joint posterior by the product of the marginals. That is, for each target j, we compute

$$\tilde{p}_t^j(W_j) = \sum_{W_k \in \{0,1\}: k \in \{1,\ldots,m\} - \{j\}} \tilde{p}_t(W_1, \ldots, W_m) \ .$$

We store the marginal distribution $\tilde{p}_t^j$ with the target j which is a distributed representation that is an approximation to the true joint distribution. Then, if we get another observation $z(t+1)$ from the magnetometer when these m targets are in range, then the prior joint distribution we use to calculate the new posterior is the product of the marginals

$$p_{t+1}(W_1, \ldots, W_m) = \prod_{j=1}^{m} \tilde{p}_t^j(W_j) \ .$$

We can then iterate this computation which gives us a sequence over time t of marginal distributions $\tilde{p}_t^j$ for each target j which we hope is close to the true marginals at each time t.

The success of this distributed representation depends on how well the joint distribution is approximated by the product of the marginals

$$\tilde{p}_t(W_1, \ldots, W_m) \approx \prod_{j=1}^{m} \tilde{p}_t^j(W_j) \ .$$

For magnetometer measurements, this approximation is particularly bad, so alternative distributed representations must be found.

2.6 Solution Methodology: Layered Information Processing Architecture

In this section, we propose a layered information processing architecture from which the various inference tasks of localization, tracking, and classification are pieced together to result in a solution for the general inference task for battlefield awareness. We will defer discussion of the hierarchy representation for Chapter 4 where the hierarchy will be used to construct communication algorithms for target localization.

2.6.1 Overall Architecture

To manage the various inference tasks to be performed in the sensor network, we present a layered information processing architecture as proposed in Section 1.5.2 for

the inference tasks of localization, tracking, and classification with the possibility of adding other inference tasks later.

We will see that this layered architecture allows us to systematically exploit the phenomenon that information from one inference task allows us to compute another inference task much more easily. To illustrate, the results of the localization task certainly aid the tracking task since the input to a tracker is a time series of estimated target locations. The reverse is also true. Given a track, we can predict the next possible target location so that localization is confined to a much smaller region, and thus, the process of localization is confined to a smaller search space. More interestingly, if targets have been classified and a model of the sensor measurements for the target type is known, then the task of localization can be more accurate since some filtering of the raw data is possible. By exploiting these relationships between inference tasks which is the motivation behind designing the proposed inter-layer interactions of the layered information processing architecture, we will demonstrate that difficult problems like data association in multiple target tracking can be solved by exploiting discriminatory information if available.

The first step in designing the layered information processing architecture is to arrange the inference tasks into layers. The considerations for choosing which layer an inference task should be placed are the following.

1. Put those tasks which are "less costly to compute" from the data near the bottom and those that require more processing and hence are "more costly to compute" near the top. Certain inference tasks are easy in the sense that developing an algorithm for the task is straightforward whereas some inference tasks are hard because developing an algorithm for them are difficult. By placing the easy inference tasks below and the hard ones above, we hope to take advantage of the information extracted from the easy inference tasks which may make the harder inference tasks easier.

2. Those inference tasks which extract information which are used as inputs to other inference tasks should be placed lower. This is a rather straightforward consideration since we must compute the inputs in order to compute the outputs.

From our intuition, we arrange the information processing tasks in the following order from bottom to top: localization, tracking, classification, and high-level behavior as shown in Figure 2-9. Choosing the layers to put inference tasks in depends on the specific nature of data and the inference tasks. For example, we may wish to place classification below localization if it happened to be that classifying targets was easy without knowing the position of the targets. This information could then be passed to the localization task which could then use the classification information to perform some kind of matched filtering based on the target class in order to localize the target. In our case, since we consider intensity measurements to be the least costly

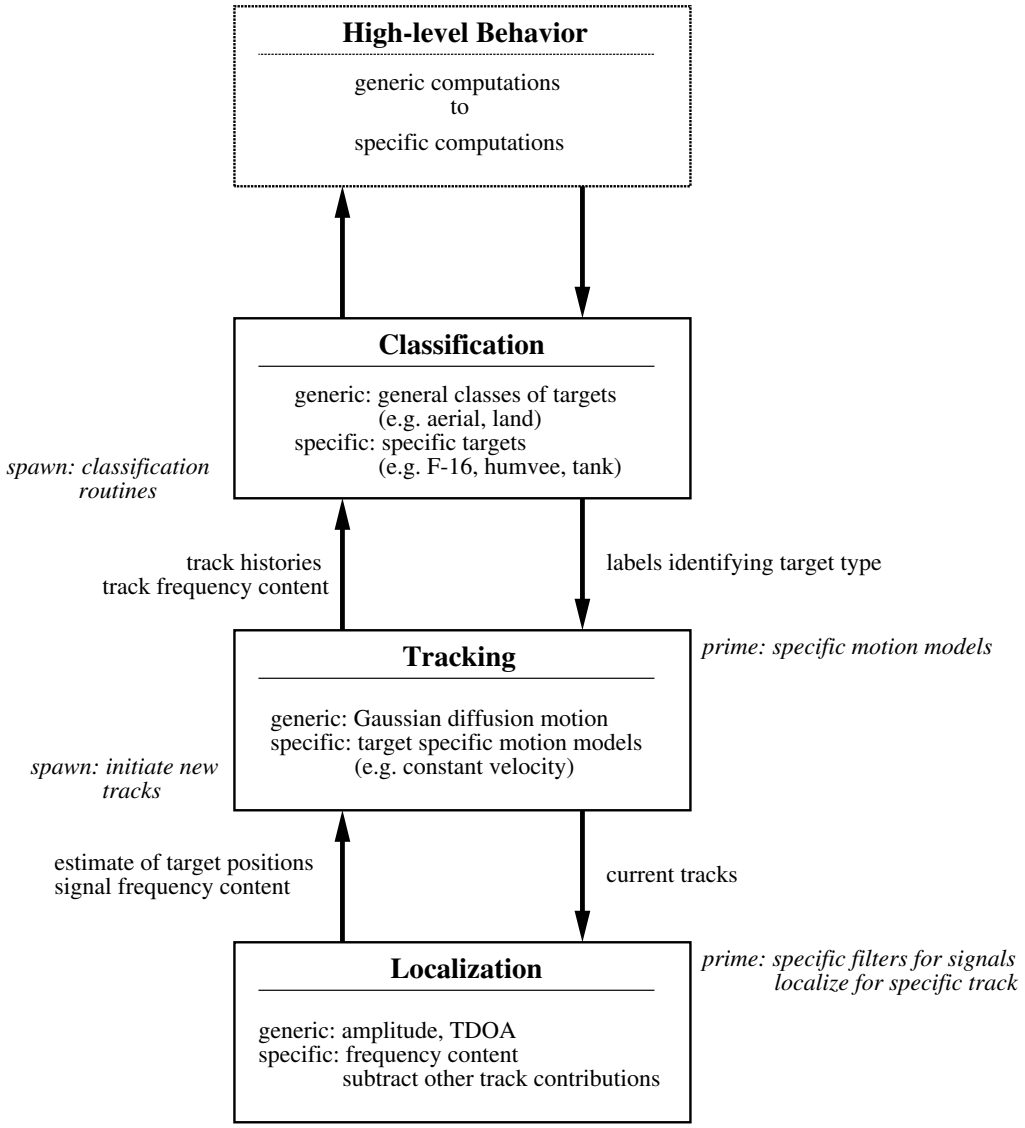

Figure 2-9: The proposed layered information processing architecture for the inference tasks of localization, tracking, and classification on sensor networks. Each layer consists of computations which range from generic to specific. The arrows indicate what information is passed between layers, and the italic text indicates either the spawning or priming effect of computations in the respective layer based on the information passed.

measurement we can extract from the acoustic sensor and because these intensity measurements contain no information about the frequency content about the target (which is the information we use for classification), localization should be placed in a lower layer than classification. The conclusion is that choosing layers is task specific and depends on what information in the data can be extracted cheaply.

Recall that the proposed layered information processing architecture consisted of three specifications.

1. **Intra-layer** The computations in a layer range from generic to specific.

2. **Top-down Priming** Information in this layer has the effect of priming more specific computations in the layer below.

3. **Bottom-up Spawning** Information in this layer has the effect of spawning generic computations in the layer above.

The next sections will describe the above three aspects for each of the layers. Each of the following sections are organized into four paragraphs describing the different parts of Figure 2-9.

1. The first paragraph describes the inference task to to be computed at the respective layer. The main description is regarding the range of computations from generic information to specific information.

2. The second paragraph describes the information that is to pass down to the layer below and the effect the information has on priming the computations of the lower layer.

3. The third paragraph describes the information that is to pass up to the layer above and the effect the information has on spawning generic computations of the higher layer.

4. Finally, the last paragraph describes the intended effect on computational resources that results from the interactions of this layer with its adjacent layers.

Chapter 3 will present the details of the discussion in the following subsections.

2.6.2 Localization

The task of localization is to compute the positions of any targets that may be in the area based on the data collected by the sensor network. Generic computations include localization with intensity or time difference of arrival estimates of the acoustic signal received by microphones. More specific computations include localization based on filtered acoustic signals for specific frequency content or other low-level signal processing based on knowing the frequency signature of specific targets. Of particular

importance will be the process of localization for a specific track based on intensity or time difference of arrival estimates by estimating this track's contribution to the sensor measurements. This is an example of localization for specific information and demonstrates the priming effect of information passed down from the tracking layer to the localization layer.

Since localization is the lowest layer, there is no information to send down to lower layers.

The next layer up is the tracking layer. The possible target localizations computed at each time step is exactly the information needed to perform tracking, and this is the information passed up to the tracking layer. Also, the frequency content of the signals measured by the sensors can be passed up for eventual use in the classification layer. The spawning effect of passing estimated target positions up to the tracking layer is to initiate new tracks.

Since data is distributed throughout nodes of the sensor network, performing localization requires communication between nodes for collecting data as formulated in Section 2.3. The subject of Chapter 4 is to convert the generic inference task of localization to an algorithm on the sensor network which passes data around the sensor network and performs localization in a distributed manner. This section demonstrates the use of the hierarchy representation as a tool for translating an information process on the layered architecture to an algorithm on the system architecture with distributed autonomous agents.

2.6.3 Tracking

The task of tracking is to compute a hypothesis of tracks which contain information about the set of target trajectories moving through the sensor network as formulated in Section 2.4. Generic computations include tracking with a Gaussian diffusion motion model. More specific computations involve choosing more specific motion models based on the classification of the target.

The immediate layer below is the localization layer. The information passed down is the set of current tracks which may or may not have been classified by the classification layer. If the target has been classified, then the priming effect of this information is to perform localization based on more sophisticated signal processing like filtering for certain frequency content. Regardless of whether a target has been classified, the information about the current state of a track primes the localization layer to compute a specific localization for each track.

The next layer up is the classification layer. The information that is passed up are track histories and possibly other information like the frequency content originating from the track which would be useful for classification. The effect of information on track histories or frequency content passed up to the classification layer is to initiate classification routines for identifying general classes of targets. For example, this can include a motion detector based on the recent trajectory of the target as well as a

sound signature detector based on frequency content.

The interaction between the tracking and localization layers can reduce the cases of when multiple target localization is needed. We shall see that if targets enter the field spaced far enough apart, we can track multiple targets using only a single target localization routine.

2.6.4 Classification and Beyond

The task of classification is to compute the identity of targets in the sensor network as formulated in Section 2.5. Generic to specific computations range from classification of generic classes of targets to classification of specific types of targets. For example, generic classes of targets can be troops, ground vehicles, or aerial vehicles whereas more specific types of targets can be a a sniper, hum-vee, or an F-16 fighter jet.

The immediate layer below is the tracking layer. The information passed down to the tracking layer is information about the identity of the track. This information has the effect of priming the motion model of tracks and providing discriminatory information so that certain cases of the data association problem can be solved. The effect of priming the motion model will be to reduce the computational requirements for tracking. In particular, we refer to the result that less particles in a particle filter are needed for tracking when the motion models are more detailed, which is consistent with the results in [18]. Also, by providing discriminatory information about tracks when they are available, the number of tracks in the hypothesis computed by the tracking algorithm can be kept as small as possible.

The layers above can use the information computed in this classification layer to determine various kinds of high-level behavior, like characteristics of a convoy of vehicles. We will not delve into the details of any other higher layers of inference tasks above classification, but it should be clear that we can add other inference tasks to this system by adding layers on top of the existing layered information processing architecture.

2.6.5 Conclusions

Although we only discuss specific problems in designing algorithms for the three layers of localization, tracking, and classification, the layered information architecture is a general one and one we claim to be a good way of designing information processing systems. The abstraction between layers allows us to design algorithms for each information process independently. The glue which brings all of these processes together is the inter-layer interaction mechanism. The top-down element of layer interactions primes more specific computations to occur, and the bottom-up element of layer interactions spawns generic computations in higher layers. These interactions are formalized when we present the abstract hierarchy representation at the end of this thesis in Chapter 5. For this reason, we consider designing layered architectures

to be a general methodology for solving these distributed inference problems.

Chapter 3

Tracking and Classification

In this chapter, we provide a distributed solution to the problem of target tracking and classification as formulated in Section 2.4 and Section 2.5. By the end of this chapter, we will have provided a solution which integrates the inference tasks of localization, tracking, and classification into a general inference system for gathering intelligence about targets in the battlefield awareness application. This general system will be able to handle the canonical scenarios presented in Section 1.3.

The solution methodology is based on the layered information processing architecture. The main idea is to take a holistic view of all information processing tasks and exploit the information extracted from the different processes. By exploiting this information, we mean that we can implement each of the information processing tasks under the assumption of some extra information being available. By systematically exploiting what information is available for computing each of the individual information processing tasks, we will observe that implementing the individual tasks can be significantly simplified.

We will provide simple solutions to the individual information processes, and, as mentioned in Section 1.8.4, we will primarily focus on the interactions we wish to design between them and not on their most optimal solutions. These high-level interactions have been mapped out in the proposed layered information processing architecture (Figure 2-9) of Section 2.6 which will be made specific in this chapter. Then, we integrate the individual information processing tasks by designing "triggers". By triggers, we refer to conditions which determine whether the computation of a particular individual information processing task should be initiated. These triggers embody the principle of information-based computation in that computations are initiated only when there is enough evidence that the computation will be useful. Furthermore, we consider the layered information processing architecture as a macro-hierarchy. The main difference is that we will handcraft the layered architecture rather than automating the construction as in the hierarchy.

Target tracking occupies the second layer in the layered information processing architecture. We focus on how the layer interactions between the tracking layer and

the localization layer can result in a multiple target tracker without the need for a multiple stationary target localization routine oblivious to currently known targets under a condition of sparse entry of targets into the field. The idea is to estimate an intermediate quantity which we refer to as the individual contribution by a target to a sensor.

Classification occupies the third layer in the layered information processing architecture. We focus on how the layer interactions between the classification layer and the tracking layer can aid in the problem of data association. In particular, we show how to incorporate frequency information from the frequency bandlimited intensity measurements in order to discriminate between targets with different frequency signatures. Furthermore, classification information can prime particular motion models for tracks so that the track estimates are more accurate and computational burden is reduced.

We begin this chapter with some background on the problem of single target tracking and multiple target tracking in Section 3.1. The single target tracking problem is posed as a filtering problem for a dynamically evolving process. For practical implementation, we will use the particle filtering algorithm which can be easily derived from the recursive update equations for filtering. Details of the particle filtering algorithm will be left for Appendix C. The background for the multiple target tracking problem will focus primarily on the emerging data association problem. Existing algorithms like the multiple hypothesis tracker (MHT) and joint probabilistic data association filter (JPDAF) will be discussed.

To contrast the proposed interactions between information processing tasks in the layered information processing architecture, we discuss the limitations of a completely bottom-up processing scheme and a completely top-down processing scheme in Section 3.2. If we allow only bottom-up processing from layer to layer, then the problem of multiple target localization is unavoidable and data association requires the enumeration of many hypotheses. On the other hand, if we allow only top-down processing, we will be unable to initiate new tracks efficiently. These deficiencies motivate the proposed interaction between the localization layer and the tracking layer as introduced in Section 2.6 and will be presented in detail in the latter sections of this chapter. We should note that the interaction between bottom-up and top-down processing are actually incorporated in solutions to many problems; however, we are explicitly identifying these bottom-up and top-down elements interacting so that we can systematically design interactions between information processing tasks into these systems.

The latter sections of this chapter are the following. Section 3.4 presents solutions to the individual information processing tasks which compose the layered information processing architecture and describes what information is assumed to be available for implementing them. Then, Section 3.5 presents the triggers which initiate the information processing tasks. Finally, Section 3.6 describes how to implement the individual information processing tasks and associated triggers in a distributed fashion

in the sensor network.

Finally, Section 3.7 presents simulations of the proposed solution for tracking and classification designed with the layered information processing architecture.

3.1 Background

We will present some background to single target tracking and multiple target tracking in this section. Single target tracking is discussed in Section 3.1.1, and multiple target tracking is discussed in Section 3.1.2. We leave details for particle filters in Appendix C which will be the algorithm we use for the filtering problem of tracking.

Let us introduce some notation to be used in the following background discussion. Random variables are denoted by capital letters, say X, and are functions

$$X : \Omega \to \mathcal{X}$$

from some sample space Ω to its space of values, denoted in calligraphic letters as in $\mathcal{X}$. We assume that there is some appropriate underlying probability space $(\Omega, \mathcal{F}, P)$ where the probability measure P is implicit when we speak of random variables. Each element ω in Ω corresponds to a possible outcome of an experiment, and a particular outcome ω of an experiment gives a particular realization $X(\omega)$ of the random variable X. Since we will usually not explicitly refer to the underlying sample space of random variables, we will denote a particular realization of a random variable by lower case letters so that x is a particular realization of the random variable X. Note that x is an element of $\mathcal{X}$. We denote the distribution of a random variable X by

$$P(X) \ .$$

In particular to the discussion to follow, we will only consider continuous random variables for which there are probability densities. By

$$P(X = x) \sim p(x) \ ,$$

we mean that $p(x)$ is the probability density of the random variable X at $X = x$. We do not necessarily use lower case letters to denote a probability density. For discrete random variables, $p(x)$ would correspond to the probability mass function.

Furthermore, we use the shorthand X_i^j and x_i^j to refer to the sequence of random variables

$$X_i, X_{i+1}, \ldots, X_j$$

and sequence of realizations

$$x_i, x_{i+1}, \ldots, x_j$$

respectively.

3.1.1 Single Target Tracking

The problem of single target tracking is to compute a trajectory of the time history of a target's motion through some state space. Since the quantities of interest are usually regarding the current state of the target, single target tracking is formulated as a filtering problem from noisy measurements of a dynamically evolving state process. We will only discuss the discrete time problem here.

- The target state evolves according to a Markov process, referred to as the *state process*,

  $$\{X_t\}_{t=0}^{\infty} ,$$

  where each X_t is a random variable taking value in some state space $\mathcal{X}$. If we specify the transition probabilities

  $$P(X_{t+1} = x_{t+1}|X_t = x_t) \sim q_t(x_{t+1}|x_t)$$

  for all $t \geq 0$ and the initial distribution of X_0,

  $$P(X_0 = x_0) \sim q_{-1}(x_0) ,$$

  then we have specified the statistics of the entire process. We refer to q_t as the *motion model* or the *state dynamics* of the process. In particular, we will assume stationarity so that each q_t is the same, which we refer to as q.

  For an example of a typical state space, the state space corresponding to the problem of tracking targets on the Euclidean plane, $\mathbb{R}^2$, usually has components corresponding to the position, velocity, and possibly acceleration of the target.

- The measurement space is $\mathcal{Y}$, and at each time t, we make a noisy measurement Y_t of the state X_t which results in a *measurement process*,

  $$\{Y_t\}_{t=1}^{\infty} ,$$

  which is related to the state process according to some measurement model described below. The statistics of the measurement process can be characterized by a probability transition function called the *measurement model* or *likelihood*

  $$P(Y_t = y_t|X_t = x_t) \sim L_t(y_t|x_t) .$$

  We will again assume stationarity so that each L_t is the same, which we refer to as L.

  For the discussion to follow, we consider a specific measurement model which picks out only the position components from the state X_t of a target. That is, Y_t is a partial observation of the track state X_t. We mention this because we will differentiate the measurement Y_t from the raw sensor node measurements $\{Z_t^i\}_{i=1}^n$.

The problem of single target tracking can be decomposed into three related problems.

1. **Filtering** Given a realization $\{y_\tau\}_{\tau=0}^t$ of the measurement process up to time t, find an estimate of the current state X_t.

2. **Smoothing** Given a realization $\{y_\tau\}_{\tau=0}^t$ of the measurement process up to time t, find an estimate of some past state $X_{t'}$ where $t' < t$.

3. **Prediction** Given a realization $\{y_\tau\}_{\tau=0}^t$ of the measurement process up to time t, find an estimate of some future state $X_{t'}$ where $t' > t$.

We will focus on the problem of filtering since the other problems are related. The estimate of X_t given a realization of the measurement process up to time t under Bayesian estimation generally boils down to being a function of the posterior distribution,

$$P(X_t = x_t | Y_0^t = y_0^t) \sim \tilde{p}_t(x_t | y_0^t) \ .$$

For example, the maximum a posteriori (MAP) estimate is the maximum of the posterior density

$$\hat{X}_t^{MAP} = \max_{x \in \mathcal{X}} \tilde{p}_t(x | y_0^t) \ ,$$

and the minimum mean squared error (MMSE) estimate is the mean of the posterior density

$$\hat{X}_t^{MMSE} = \int_{x \in \mathcal{X}} x \cdot \tilde{p}_t(x | y_0^t) dx \ .$$

for the case when the state space $\mathcal{X}$ is some d-dimensional Euclidean space. The point is that the posterior distribution is the underlying quantity from which estimates are based in the Bayesian setting.

Many standard algorithms for filtering exist like the Kalman filter for linear state dynamics and linear measurement models with Gaussian disturbances, the extended Kalman filter which linearize nonlinear models around nominal values, and the condensation algorithm, also referred to as particle filtering, which can handle nonlinear state dynamics and measurement models by representing probability distributions by a particle set (See Appendix C). All of these algorithms are recursive algorithms in that the filtered state estimate of the current time is a function of the filtered state estimate of the previous time and the current measurement. This allows us a huge savings in terms of computation because it is not necessary to recompute the effect of past measurements into the current filtered estimate. To put it another way, the effect of past measurements for all filtered estimates in the future is completely summarized by the previous time filtered estimate. The reason these recursive algorithms are possible is because of the assumptions of a Markov state process and the assumption that the measurement Y_t at each time t is conditionally independent of the state at other times given the current state X_t. By using Bayes' rule and the assumptions listed above, we can write the posterior density $\tilde{p}_t(x_t | y_0^t)$ in terms of the contribution

of the current measurement, namely the likelihood $L_t(y_t|x_t)$, and the previous filtered distribution $\tilde{p}_{t-1}(x_{t-1}|y_0^{t-1})$ which is given by

$$\tilde{p}_t(x_t|y_0^t) = \alpha L_t(y_t|x_t)p_t(x_t|y_0^{t-1}) \tag{3.1}$$

where

$$p_t(x_t|y_0^{t-1}) = \int_{\mathcal{X}} q_{t-1}(x_t|x_{t-1})\tilde{p}_{t-1}(x_{t-1}|y_0^{t-1})dx_{t-1} \tag{3.2}$$

and α is an appropriate normalization constant to make $\tilde{p}_t$ integrate to one,

$$\int_{\mathcal{X}} \tilde{p}_t(x|y_0^t) = 1 \ ,$$

for any given realization y_0^t of the measurement process up to time t. The density p_t is called the *predicted density* of X_t given measurements up to time $t-1$.

The track as defined in Definition 2.8 is a function of the of these filtered distributions,

$$\{\tilde{p}_t\}_{t=0}^{\infty} \ .$$

That is, a track T is the time series of mean and covariance pairs

$$T = \{(x^T(t), \Sigma^T(t))\}_{t=T_{start}}^{T_{stop}}$$

where

$$\begin{aligned} x^T(t) &= \int_{\mathcal{X}} x\tilde{p}_t(x|y_0^t)dx \\ \Sigma^T(t) &= \int_{\mathcal{X}} (x - x^T(t))(x - x^T(t))'\tilde{p}_t(x|y_0^t)dx \end{aligned}$$

If we require smoothed estimates, then we must include the effect of future measurements for each filtered estimate. We refer the reader to the literature regarding computing smoothed estimates where techniques like the backward filter can smooth filtered estimates by running an appropriately constructed filter backwards in time.

Practical Implementation by Particle Filters

We will adopt particle filters as our tracking algorithms because

1. they are suited for filtering problems with nonlinear state dynamics and measurement models with non-Gaussian noise and

2. the computational resources required for processing is directly related to the number of particles used.

The measurement models for intensity, frequency bandlimited intensity, TDOA, and magnetometer sensors in the sensor network are highly non-Gaussian so that we require an algorithm which can handle non-Gaussian measurement models. Furthermore, one of the main issues we explore in this thesis is how to design algorithms which are efficient, and particle filters offer an especially easy measure of its computational efficiency, namely the number of particles used in the particle filter.

The particle filtering algorithm essentially represents the posterior densities $\tilde{p}_t$ by a set of sample points $\tilde{\mathbf{P}}_t$ referred to as the particle set. A particle set is a finite set of points in the state space $\mathcal{X}$ with an associated weight. The number of particles in the particle set is a parameter which we can adjust. The recursive update equations for the filter, Eqn. 3.1 and Eqn. 3.2, essentially define the steps for updating the particle set $\tilde{\mathbf{P}}_{t-1}$ corresponding to $\tilde{p}_{t-1}$ from the previous time to the particle set $\tilde{\mathbf{P}}_t$ corresponding to $\tilde{p}_t$ for the current time. In particular, Eqn. 3.2 corresponds to a resampling procedure which propagates the particles in $\tilde{\mathbf{P}}_{t-1}$ forward by the motion model q_{t-1}. Then, Eqn. 3.1 corresponds to a reweighting of these resampled particles according to the likelihood L_t. The reweighting of the samples according to the likelihood L_t is how the effect of the observation y_t is incorporated into the resulting particle set $\tilde{\mathbf{P}}_t$.

We will leave the details of the particle filtering algorithm for Appendix C. The reason we mention particle filters here is because the particle filtering algorithm can be immediately derived given the recursive equations for filtering. In particular, the specific task of track continuation in Section 3.4.2 presents equations to compute the current estimate of the track position given the previous time estimate of the track position. These equations will be implemented by a particle filter in the simulations. The computational resources needed to compute the particle filter and the quality of approximation of the particle set to the true posterior density depends on the size of the particle set $\tilde{\mathbf{P}}_t$. The larger the size of the particle set is, the better the quality of the approximation at the cost of more computation. We will exploit this tradeoff to minimize the computation of the track updates so that targets can be tracked with reasonable quality.

3.1.2 Multiple Target Tracking

The extension of single target tracking to multiple target tracking introduces an added problem known as the *data association* problem. Let us illustrate how the data association problem emerges. The multiple target tracking problem is usually posed in the following manner.

- With m targets, there are m target state processes evolving according to m Markov processes,

$$\{X_t^j\}_{t=0}^{\infty} \text{ for } j \in \{1, \ldots, m\}$$

  Assuming stationarity, each Markov process $\{X_t^j\}_{t=0}^{\infty}$ can be characterized by a

probability transition function

$$P(X_{t+1}^j = x_{t+1}^j | X_t^j = x_t^j) \sim q^j(x_{t+1}^j | x_t^j) \ .$$

- Assuming that there are no missed measurements from targets and no possibility of false alarms, each of the m targets generates a measurement Y_t^j for $j \in \{1, \ldots, m\}$ according to

  $$P(Y_t^j = y_t^j | X_t^j = x_t^j) \sim L^j(y_t^j | x_t^j)$$

  where we have again assumed stationarity of the measurement model.

  We make a distinction between the measurements Y_t^j and the raw sensor measurements $\{Z_t^i\}_{i=1}^n$ in the discussion to follow.

The filtering problem for multiple target tracking is

Given measurements $\{y_\tau^j\}_{\tau=0, j\in\{1,\ldots,m\}}^t$ up to time t, find an estimate for each of the m target states X_t^j at time t for $j \in \{1, \ldots, m\}$.

From the statement above, multiple target tracking may seem like m independent single target tracking problems; however, the measurements $\{y_\tau^j\}_{\tau,j}$ are generally unlabelled so that it is unknown which of the measurements $\{y_t^1, \ldots, y_t^m\}$ at time t correspond to each of the m targets. Thus, the data association problem is finding the right associations between measurements y_t^j and targets so that the filters which estimate the m target states can be updated appropriately.

To discuss the data association problem, let us define the sequence function

$$s : \{0, 1, \ldots, t\} \to \{1, \ldots, m\}$$

and its associated sequence of measurements

$$y_0^{s(0)}, y_1^{s(1)}, \ldots, y_{t-1}^{s(t-1)}, y_t^{s(t)} \ .$$

Let $\mathcal{S}$ be the set of all sequence functions with domain $\{0, \ldots, t\}$ and range $\{1, \ldots, n\}$.

If we knew the particular sequence of measurements $s_j \in \mathcal{S}$ generated by each target $j \in \{1, \ldots, m\}$, then the multiple target tracking problem decomposes into m independent single target tracking problems. However, for most multiple target tracking problems, we do not know the sequence of measurements that correspond to each target, and thus, the data association problem is to find a set of sequences

$$\{s_j\}_{j=1}^m$$

where each $s_j \in \mathcal{S}$ indicates the sequence of measurements corresponding to each of the m targets. Some simplifying assumptions like no measurements being assigned

to more than one track are generally made when choosing a set of sequence functions $\{s_j\}_{j=1}^m$. That is, for each pair of tracks $i, j \in \{1, \ldots, m\}$, the sequence of measurements associated with each track consist of different measurements,

$$s_i(\tau) \neq s_j(\tau) \text{ for all } \tau \in \{0, \ldots, t\} .$$

However, such assumptions do not prevent the combinatorial explosion of the number of sets of sequence functions as t increases. This exponential increase of the possible number of sequences of measurements which could correspond to a track is the primary reason that a practical implementation for the multiple target tracking problem has been unsuccessful.

There are predominantly two methods for solving the data association problem. The multiple hypothesis tracker (MHT) by Reid [22] enumerates a set of likely sequence functions in $\mathcal{S}$ which could possibly correspond to the track of a target. The idea for handling the combinatorial explosion of sequences in $\mathcal{S}$ as t grows is to rank sequences according to the likelihood of corresponding to a target and prune away low ranking sequences. The main issue involved with this enumeration method is to be sure to keep enough of the sequences so that the sequence corresponding to the track of a true target is not inadvertently pruned. Furthermore, to handle the possibility of new targets entering the field, track initiations are hypothesized, and to handle the possibility that no measurement is received for updating a track, missing measurements for tracks are also hypothesized. Incorporating these possibilities further compounds the exponential explosion of the number of sequences so that pruning becomes a necessary and difficult problem. The primary benefit of enumerating these sequences is that we can keep several of them at each iteration so that the effect is to delay making decisions about certain data associations until more evidence is collected later.

The other predominant method, known as the joint probabilistic data association filter (JPDAF) of Bar-Shalom [1], updates track states with the set of measurements received at each time by an appropriately chosen convex combination of all measurements. Thus, every measurement at each time makes some contribution to the state of every track currently in the JPDAF. By using a convex combination of the effects of all measurement at each time, this data association filter is avoiding the exponential explosion that is characteristic of an enumeration method like the MHT.

Once a set of measurements are used to update the state of each track in the JPDAF, the measurements are erased from memory. The result is that the effect of a measurement on the track states is immediate with no possibility of being used again. This can be a problem if the update to a track at some time is particularly bad because there is no way to recover from the bad update. However, incorporating measurements only when they are received is desirable in terms of controlling the computational complexity of this algorithm. In contrast, the MHT can delay decisions on data association until a later time because we keep a subset of the best ranking sequences which virtually allows us to backtrack if necessary. This allows for more

accurate data association at the expense of the extra bookkeeping, and hence, added computational complexity for keeping multiple sequences of measurements.

Both of these approaches begin with measurements $\{y_t^j\}_{j=1}^m$ at time t as they are given by some localization process. The measurements $\{y_t^j\}_{j=1}^m$ are estimates of multiple target positions at time t based on the raw sensor measurements $\{z_t^i\}_{i=1}^n$ at time t from sensor nodes $i \in \{1, \ldots, n\}$. Thus, we consider both of these approaches to be instances of a purely bottom-up approach because the only interaction between the localization and tracking processes is the passing of multiple target localization information from the localization process up to the tracking process. There is no complementary information being passed from the tracking process to the localization process. Certainly, the information about the current tracks known to be in the field can aid the implicit multiple target localization process. We shall see in latter sections of this chapter starting from Section 3.4 that by adding this complementary pathway between the processes of localization and tracking as well as tracking and classification, we will be able to make the algorithms implementing these tasks more efficient.

Before presenting the solution for tracking and classification on the sensor network, the next section will describe how a purely bottom-up or a purely top-down approach is undesirable.

3.2 One Way Processes on the Layered Architecture

3.2.1 Purely Bottom-up Approach

In a purely bottom-up approach, the processes in the localization layer pass information up to the tracking layer. The constraint of a purely bottom-up approach is that the tracking layer has no effect on what computations are being performed in the localization layer. Let us elaborate on the consequences of this constraint for the problem of multiple target tracking.

Essentially, the tracking problem is divided into two separate tasks, one for each layer.

1. **Localization Layer** The localization layer is faced with the problem of multiple target localization which must return a set of estimated target locations. That is, we must estimate at each time t from the realization of sensor measurements $\{z_t^i\}_{i=1}^n$, the number of targets l and an estimate of those l target positions $\{Y_t^j\}_{j=1}^l$. That is, the localization layer computes

$$P(\{Y_t^j\}_{j=1}^l | \{Z_t^i = z_t^i\}_{i=1}^n) \ . \tag{3.3}$$

We use l to denote the number of targets estimated by the localization process rather than m due to the possibility that the localization process may return a

different number of target position estimates than the number of targets in the field.

2. **Tracking Layer** The estimates of $\{Y_t^j\}_{l=1}^l$ are then fed into the tracking layer from which new tracks are initiated, the set of current track states $\{X_t^j\}_{j=1}^m$ are updated, and tracks with no evidence are deleted. The results of the localization layer are considered the measurements for the multiple target tracking problem of the tracking layer.

Note that the MHT and JPDAF approaches solve the multiple target tracking problem using the results of localization layer process which returns an estimate of possible target positions $\{Y_t^j\}_{j=1}^l$ at each time t given raw sensor measurements. The multiple target localization process which generates these target positions is considered a black box which the tracking layer may not affect in any way. Thus, we consider the standard MHT and JPDAF approaches as purely bottom-up approaches in this sense.

It is clear that with this one way information flow from localization to tracking, there is no way to avoid the need for a multiple target localization routine for multiple target tracking. For this reason, we claim that the complexity of the processes in the localization and tracking layers is due to the constraint imposed by a purely bottom-up approach.

3.2.2 Purely Top-down Approach

In a purely top-down approach, the processes in the tracking layer can query for information from the localization layer. The localization layer can then perform the appropriate processes and send the results of the query back up to the tracking layer. The constraint of this approach is that the only way information is passed up to the tracking layer from the localization layer is if the tracking layer queried for it.

Since no information is passed from the localization layer to the tracking layer unless it is queried, the main problem with this approach is how to initiate a track. Of course, a hack could be for the tracking layer to always query for the possibility of new tracks, but this is better viewed as a bottom-up process from the localization layer which is exactly what our proposed interaction between the localization layer and the tracking layer will be as described in the latter sections of this chapter.

What we wish to take away from considering a purely top-down approach is the idea that for each track in the tracking layer we prime the localization layer to perform a specific localization task for the track of interest.

3.3 Tracking Overview

Let us describe the overall idea behind our multiple target tracking scheme using the layered information processing architecture. We begin with an abstract discussion of

the solution methodology for target tracking and the role that the layered information processing architecture plays in Section 3.3.1. Then, a description of the model from multiple targets to multiple heterogeneous sensors in the sensor network is given in Section 3.3.2. The model can be represented graphically as a Markov random field. Section 3.3.3 formulates what inference on this model corresponds to tracking. Due to the complexity of the calculations involved for computing the exact inference, we present an approximate inference scheme which performs tracking in Section 3.3.4. This approximate inference scheme is an example of a belief propagation type of approximation.

3.3.1 The Abstract Idea

The goal of tracking as formulated in Section 2.4 is to construct a hypothesis $H \in \mathcal{H}$ so that there exists a track or an adjoined track in H for every part of a target trajectory that traverses the field. The main idea for constructing such an H will be to iteratively update the hypothesis. That is, given the hypothesis H_{t-1} from the previous time $t-1$ and the observed joint sensor measurement $z(t) \in \mathcal{Z}$ at time t, we wish to construct a tracking algorithm

$$\mathbf{T} : \mathcal{H} \times \mathcal{Z} \to \mathcal{H}$$

which returns the updated hypothesis at time t,

$$H_t = \mathbf{T}(H_{t-1}, z(t)) \ .$$

The one-step tracking algorithm $\mathbf{T}$ involves the following two primary tasks for iteratively updating the hypothesis from H_{t-1} to H_t.

1. **Initiation of New Tracks** At time t, new targets may enter the field so that new tracks which correspond to these targets must be initiated and added to H_t.

2. **Update of Existing Tracks** The tracks in the hypothesis H_{t-1} from the previous time need to be updated to be included in the hypothesis H_t. Updating existing tracks involves the following two tasks.

   (a) Since multiple targets can be present in the field, we must be able to extract localization information for each specific target from the observations of the sensor nodes.

   (b) Due to the possibility that targets may cross, we wish to update targets using as much discriminatory information as needed to ensure proper track updates. Thus, incorporating classification routines during tracking is needed.

To design the tracking algorithm, we propose to break up the problem into specific tasks and arrange them into a layered information processing architecture as was proposed in Section 2.6. Then, by designing intra-layer computations which range from generic to specific and designing bottom-up spawning and top-down priming interactions, we will be able to design the complex tracking algorithm in a systematic fashion. Furthermore, expanding the functionality of the resulting general inference system is easy by adding tasks within layers or designing a new layer on top of the existing layered architecture.

The primary observation that the layered information processing architecture takes advantage of is that the implementation of a task can be significantly simplified when extra information is available. The layered information processing architecture allows us to systematically design that certain information be available to compute certain tasks. These are exactly the role of bottom-up spawning and top-down priming interactions.

3.3.2 Multiple Target to Multiple Sensor Model

The state space Θ is the state of a target. The state of a target can include spatial information like position, velocity, and acceleration components and other information like the nominal intensity, the frequency signature, and whether the target is metallic or not. When there are m targets, the state θ_t^j of each target $j \in \{1, \ldots, m\}$ evolves, in discrete time steps, according to its own state transition model, which we represent by a transition density

$$q_t^j : \Theta \times \Theta \to [0, \infty)$$

usually written $q_t^j(\theta_{t+1}|\theta_t)$ which represents the density that the state of the target at time $t+1$ is θ_{t+1} given that the state of the target at time t was θ_t.

The relevant information from each of the m tracks at time t with state θ_t^j which can affect the measurement observed by sensor node i includes information about the target position x_t^j, the nominal intensity A^j emanating from the target, the frequency signature $\{A_\phi^j\}_{\phi=1}^b$ of the target where b is the number of frequency bands, and whether the target is metallic or not $W^j \in \{0, 1\}$. Because we assume that all relevant information about the target other than the target's position is stationary, we will represent the relationship between the state of target j and the observation of sensor node i by a likelihood function

$$p_i^j(z_i|x)$$

where only the target's position x is shown explicitly in the notation.

The model for the sensor measurements of all n sensor nodes at some time t in the sensor network given m targets is shown in a Markov random field representation in Figure 3-1. For each target j, we represent the position by a random variable X^j. If target j was the only target in the sensor network, then it will generate a contribution to the measurement of each sensor node i which is represented by a

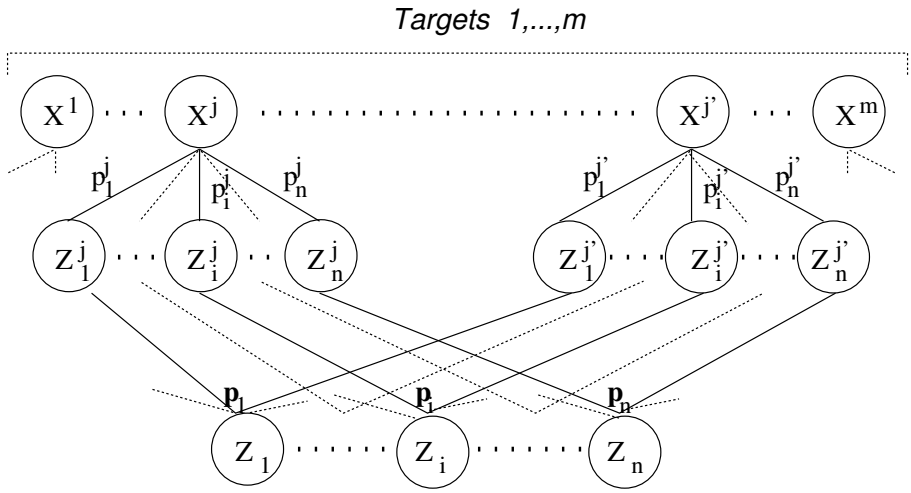

Figure 3-1: The Markov random field representation of the random variables defined in the model from target positions X^j to sensor measurements Z_i. We label edges of the Markov random field with the densities that define the relationship between the random variables.

random variable Z_i^j. The relationship between X^j and Z_i^j is given by a likelihood function $p_i^j(z_i^j|x^j)$ for all $i \in \{1, \ldots, n\}$ which labels the edges between X^j and Z_i^j. The measurement observed by sensor node i, denoted Z_i is some aggregation of the individual contributions $\{Z_i^j\}_{j=1}^m$ from each target $j \in \{1, \ldots, m\}$. We represent this aggregation by a transition density

$$\mathbf{p}(z_i|z_i^1, \ldots, z_i^m) \ .$$

3.3.3 Inference on the Model

For the purpose of tracking, the inference task we would like to perform with this model is to compute the posterior distribution of each X^j given sensor measurements $\{Z_i\}_{i=1}^n$, denoted

$$P(X^j|Z_1, \ldots, Z_n).$$

We assume that the random variables X^j are independent which is a decent assumption since we believe that the target states are independent of each other. Let us derive the posterior distribution for the case when there are two targets and two sensor nodes and all random variables are discrete. By Bayes' rule, the posterior

distribution of X^1 is given by

$$\begin{aligned} P(X^1|Z_1,Z_2) &= \sum_{X^2} P(X^1,X^2|Z_1,Z_2) \\ &= \sum_{X^2} \alpha P(Z_1,Z_2|X^1,X^2)P(X^1,X^2) \\ &= \alpha P(X^1)\sum_{X^2} P(Z_1,Z_2|X^1,X^2)P(X^2) \end{aligned}$$

where α is a normalization constant and the third equality follows from the assumption of independence between X^1 and X^2. Writing Z_1 and Z_2 in terms of the individual target contributions Z_1^1, Z_1^2, Z_2^1, and Z_2^2 gives

$$\begin{aligned} P(Z_1,Z_2|X^1,X^2) &= \sum_{Z_i^j:i,j\in\{1,2\}} P(\{Z_i\}_{i=1}^2|\{Z_i^j\}_{i,j\in\{1,2\}},X^1,X^2)P(\{Z_i^j\}_{i,j\in\{1,2\}}|X^1,X^2) \\ &= \sum_{Z_i^j:i,j\in\{1,2\}} P(Z_1|\{Z_1^j\}_{j=1}^2)P(Z_2|\{Z_2^j\}_{j=1}^2)P(\{Z_i^1\}_{i=1}^2|X^1)P(\{Z_i^2\}_{i=1}^2|X^2)\ . \end{aligned}$$

Expanding to the case when there are m targets and n sensor nodes, we have

$$P(X^j|\{Z_i\}_{i=1}^n) = \alpha P(X^j)P(\{Z_i\}_{i=1}^n|X^j) \tag{3.4}$$

$$P(\{Z_i\}_{i=1}^n|X^j) = \sum_{X^k:k\in\{1,\dots,m\}-\{j\}} P(\{Z_i\}_{i=1}^n|\{X^k\}_{k=1}^m)P(\{X^k\}_{k\in\{1,\dots,m\}-\{j\}}) \tag{3.5}$$

where

$$P(\{Z_i\}_{i=1}^n|\{X^j\}_{j=1}^m) = \sum_{\substack{Z_i^j:\ i\in\{1,\dots,n\},\\ j\in\{1,\dots,m\}}} \left[\prod_{i=1}^n P(Z_i|\{Z_i^j\}_{j=1}^m)\right]\left[\prod_{j=1}^m P(\{Z_i^j\}_{i=1}^n|X^j)\right]$$

We refer to $P(\{Z_i\}_{i=1}^n|X^j)$ as the likelihood of target j. Because the Markov random field of Figure 3-1 is loopy, we are unable to simplify the expressions above further. As the number of targets m or the number of sensors n grows, the computation above scales exponentially. Thus, we will make an approximation to the above true expression for the posterior distribution of each target location.

3.3.4 Approximate Inference on the Model

We use a belief propagation type of approximation. The idea is to propagate messages starting from the target location nodes X^j down the graph in Figure 3-1. Then, propagate messages back up so that an approximation to the posterior distribution of each X^j is computed. Let us describe this approximation.

At each time t, we have a prior distribution for each target's position given by a density

$$p_X^j(x) \ .$$

We propagate these prior densities of the position of target j downwards to compute the density of the individual contributions $\{Z_i^j\}_{i=1}^n$, which are given by

$$P_i^j(z) = \int_{\mathbb{R}^2} p_i^j(z|x) p_X^j(x) dx \ .$$

Then, given the prior density of each individual contribution $\{Z_i^j\}_{j=1}^m$ to sensor node i, we can compute the posterior density of each $\{Z_i^j\}_{j=1}^m$ given that $Z_i = z_i$ by

$$\tilde{P}_i^j(z_i^j) = \int_{\mathbb{R}^{m-1}} \mathbf{p}(z_i|z_i^1, \ldots, z_i^m) \left[\prod_{k=1}^m P_i^k(z_i^k) \right] dz_i^1 \cdots dz_i^{j-1} dz_i^{j+1} \cdots dz_i^m \ .$$

Note that in the expression above for $\tilde{P}_i^j$ that the prior target contributions P_i^j to sensor node i are independent. We have not yet made any approximations.

On the way up the graph of Figure 3-1, however, we will make the approximation that the posterior density of the individual contributions $\{Z_i^j\}_{i=1}^n$ by target j is the product of the marginals,

$$\prod_{i=1}^n \tilde{P}_i^j(z_i^j) \ .$$

This is an approximation which comes from an independence assumption that belief propagation makes. Then, the posterior density of the position of target j under this approximation is given by

$$\tilde{p}_X^j(x) = \alpha p_X^j(x) L^j(x) \tag{3.6}$$

where

$$L^j(x) = \int \left[\prod_{i=1}^n p_i(z_i^j|x) \tilde{P}_i^j(z_i^j) \right] dz_1^j \cdots dz_n^j \ . \tag{3.7}$$

Eqn. 3.6 and Eqn. 3.7 are comparable to Eqn. 3.4 and Eqn. 3.5 respectively. L^j is an approximation to $P(\{Z_i\}_{i=1}^n|X^j)$ which is referred to as the likelihood of target j. The key step in this approximation to the posterior density of the position of target j is the computation of the posterior density $\tilde{P}_i^j$ of the individual contribution to the observation of sensor node i by target j. Section 3.4.1, which presents how to initiate tracks for new targets entering the field, and Section 3.4.3, which presents how to localize for a specific target, will describe how to compute $\tilde{P}_i^j$ for various kinds of sensing modalities.

To summarize the above discussion, we have described the model from a set of target positions $\{X^j\}_{j=1}^m$ to the sensor measurements $\{Z_i\}_{i=1}^n$ through an intermediate set of random variables $\{Z_i^j\}_{i=1,j=1}^{n,m}$ which correspond to the individual contributions to the observation of sensor i by target j. We then presented the calculation of an approximate density $\tilde{p}_X^j$ to the true posterior density of each target position X^j given observations of all sensor measurements $\{Z_i = z_i\}_{i=1}^n$. To summarize, the computations involved were

$$p_X^j \Rightarrow P_i^j \Rightarrow \tilde{P}_i^j \Rightarrow L^j \Rightarrow \tilde{p}_X^j \ .$$

The approximation makes independence assumptions which are not true and is the kind of approximation that is made by belief propagation. However, as long as targets are spaced far enough apart, we will see experimentally that the approximation is good enough to successfully perform tracking.

3.4 Individual Information Processing Tasks

In this section, we will discuss the different individual information processing tasks to be performed in the layered information processing architecture. The individual tasks presented here should be considered to be *examples* of the types of information processing tasks that may be encountered and how the information between them are to be exploited when designing a general inference system for a particular application.

The solutions to each of these tasks are given separately to demonstrate that the layered information processing architecture is a modular one. Recall that our focus is not on the optimal implementation of these individual processing tasks as mentioned in Section 1.8.4. Because the layered information processing architecture is modular, we will leave the optimal implementation of each of these tasks for future work. Instead, our focus will be on how these tasks will interact with each other which in many instances means we only need to solve an easier version of the problem. By an "easier version", we mean that we solve the problem assuming the availability of extra information. The effect of this extra information is to reduce the complexity of the problem to one which is significantly easier to solve. We will point out the extra information assumed to be available in the various information processing tasks we consider in this section.

Although tracks have been defined in Definition 2.8 to consist only of a time series of mean and error covariance pairs of the track position, we also associate a particle set representation of the posterior density $\tilde{p}_X^j$ of the current track position to compute better estimates of the track contributions to sensors. Furthermore, we also implicitly associate a nominal intensity, frequency signature, and an indication of whether the track corresponds to a metallic target. These other information about the track will be important for serving as discriminatory information for aiding the data association problem.

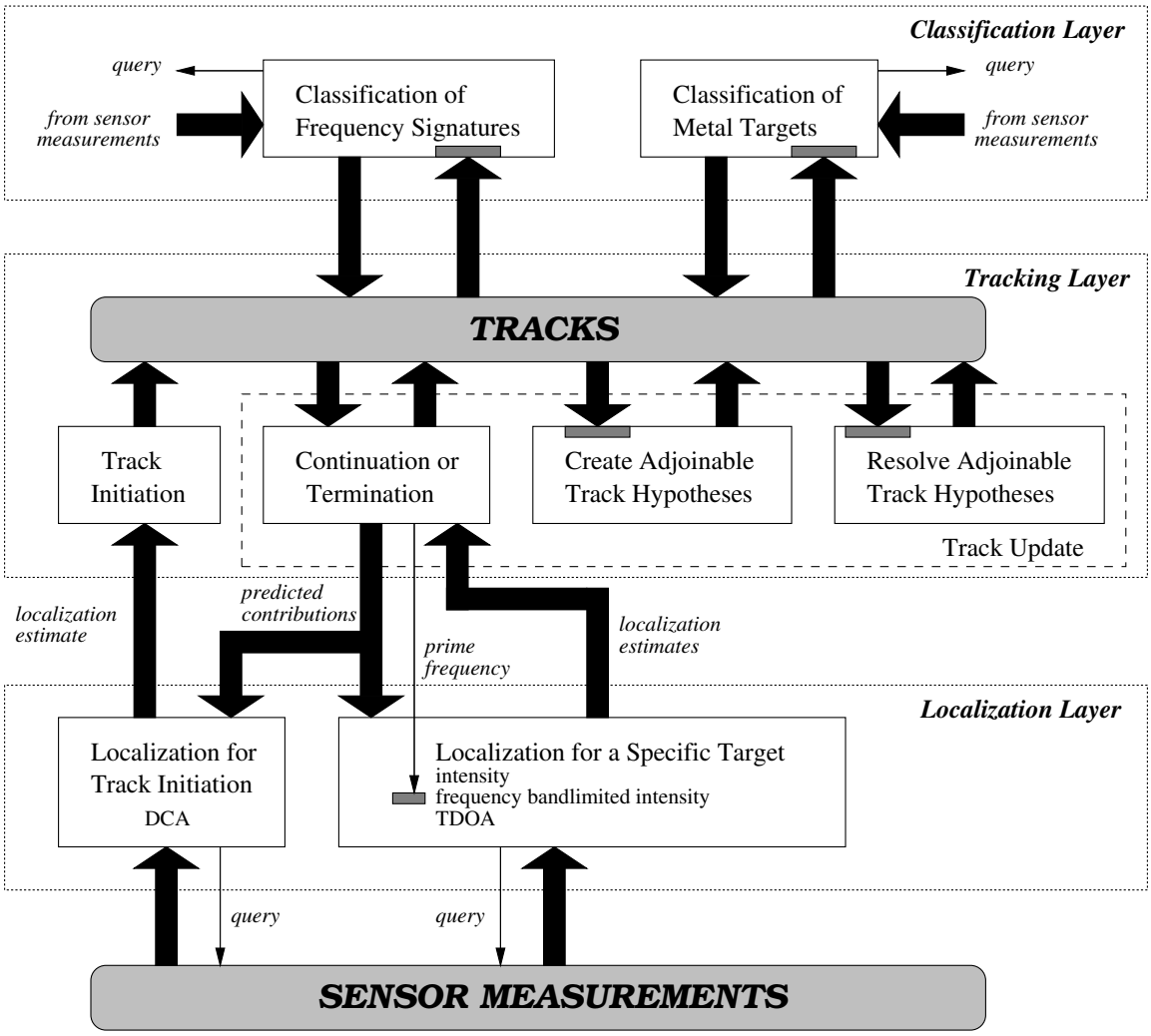

Figure 3-2: The individual information processing tasks in each layer of the layered information processing architecture and the information dependencies between them. Black block arrows indicate information being exchanged between the different processes. Thin black arrows indicate possible top-down priming and bottom-up spawning interactions. The gray boxes indicate that there is a trigger for initiating the corresponding processing task. The large rounded gray boxes labelled by "Sensor Measurements" and "Tracks" refer to the data measured by the sensor network and the representation of tracks constructed from the data respectively by this inference system.

Figure 3-2 shows the layered information processing architecture which corresponds to the abstract tracking algorithm **T** we referred to in Section 3.3.1. At each time t, the sensor measurements, indicated by the large rounded gray box labelled by "Sensor Measurements", are incorporated by the inference system according to this layered information processing architecture to update the current hypothesis about tracks, which is indicated by the large rounded gray box labelled by "Tracks".

The specific tasks are the following. In the descriptions, we implicitly refer to Figure 3-2 which show these individual information processing tasks and the interactions to occur among them.

1. **Track Initiation and Localization for Track Initiation** These are the tasks in the localization layer and tracking layer which uses the most generic information, intensity and TDOA measurements. The effect of this localization on the tracking layer is a bottom-up spawning interaction which initiates new tracks. The DCA's for localization will query the appropriate sensor measurements to compute the localization for possible new targets.

2. **Track Updating** This is the primary task of the tracking layer. This processing task is responsible for continuing or terminating existing tracks, creating adjoinable track hypotheses when tracks are about to cross and cannot be discriminated, and resolving these hypotheses when discriminatory information is available.[1] Generic and specific computations in this layer correspond to a choice of motion model for track continuation. Also, if a track is near enough to other tracks that there is the possibility of encountering a data association problem, using more specific information like frequency bandlimited intensity measurements can be primed.

3. **Localization for a Specific Target** This is a task in the localization layer which estimates the localization for a specific target. In particular, when primed by the track continuation processing task, more specific information like frequency bandlimited intensity measurements can be exploited for localizing targets. This processing task is initiated whenever a track must be continued or terminated so that this task can be considered to be initiated by a top-down priming effect from the tracking layer. To compute a localization for a specific target, the sensor measurements within range of the target are queried.

4. **Classification of Frequency Signatures** This task resides in the classification layer and is spawned when an unclassified track is within range of frequency bandlimited intensity sensors. This classification information is extra information associated with tracks which can be used to localize for specific targets using frequency bandlimited intensity measurements. The appropriate sensor measurements are queried to compute the classification.

[1] An *adjoinable track hypothesis* is defined in Section 3.4.2 in Definition 3.1 so we will defer discussion of this until then.

5. **Classification of Metallic Targets** This is another task which resides in the classification layer. This task is spawned for tracks which are within range of a magnetometer sensor, and the information gained is about whether the target corresponding to the track is metallic. This extra information can be used to disambiguate crossing tracks when one target is metallic and the other is not. The appropriate magnetometer sensors are queried for computing this classification.

Approximations of densities to a Gaussian density will be used often in the following sections. Let us define the following notation for a Gaussian density where $x \in \mathbb{R}^d$, $\mu \in \mathbb{R}^d$, and Σ is a $d \times d$ covariance matrix:

$$G(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)\right] .$$

The symbol $'$ denotes transpose.

3.4.1 Track Initiation

The initiation of tracks is where the entire process of gaining intelligence about targets in the field begins. There are two possible scenarios where we must initiate tracks.

1. The first scenario is when there are no targets currently in the field, and a new target appears in the field.

2. The second scenario is when there are already targets present in the field, and an additional new target enters the field.

We will present a solution for the above scenarios where a single target localization algorithm is sufficient enough to initiate tracks successfully if the targets enter in a *sparse fashion*. Let us describe what we mean by sparse. The intuitive idea is that any target only affects the measurement of a local neighborhood of sensor nodes. That is, if the target is currently located at $x \in \mathbb{R}^2$, then only those sensor nodes whose positions lie within some radius $r > 0$ from x will actually sense the target. We say that targets are *sparse* in the field if for any two targets located at $x_1, x_2 \in \mathbb{R}^2$, the target positions differ in distance by more than $2r$,

$$d(x_1, x_2) > 2r$$

where $d(\cdot, \cdot)$ is the Euclidean distance. Notice that when targets are sparse, the sensor measurement of every sensor node is essentially affected by only a single target. This is why only a single target localization algorithm is sufficient if targets enter the field in a sparse fashion.

We will not explicitly consider the case when targets enter the field in a non-sparse, or dense, fashion. This case requires a solution to the multiple target localization problem which is a problem we leave open. However, with only a single target localization solution, we will be able to demonstrate the interactions occurring in the layered information processing architecture which is the primary focus of this chapter. Furthermore, since a single target localization solution can handle the case when targets enter the field in a sparse fashion, the only case for which we need a solution to the multiple target localization problem is when multiple targets enter the field *simultaneously* within a distance of $2r$ from each other. Notice that this cuts down the possible space of multiple target locations for which we must localize multiple targets which demonstrates a reduction in complexity for designing a multiple target localization algorithm.

No Targets in the Field

The bulk of the solution for initiating a track in this scenario will be given in Chapter 4. We can run the distributed communication algorithms (DCA) constructed which perform target localization based on intensity and TDOA sensor measurements for each time t. If a sensor node, say $i \in \{1, \ldots, n\}$, declares that the global localization task has been completed for measurements collected at time t, then sensor node i initiates a track based on the posterior density of the target location estimated from the localization task. That is, if the local knowledge state in node i is $B \in \mathcal{B}$, then sensor node i computes the posterior density

$$\tilde{p}_B(x) = p_0(x) \prod_{(t,j,z_j) \in B} p_j(z_j|x) \ .$$

Recall the definition of a track in Definition 2.8. We create a new track T with start time

$$T_{start} = t \ .$$

For now, T_{stop} is undefined because the track has not terminated. Then, for time t, we can compute the mean and covariance of this new track's position

$$\begin{aligned} x^T(t) &= \int_{\mathcal{X}} x \tilde{p}_B(x) dx \\ \Sigma^T(t) &= \int_{\mathcal{X}} (x - x^T(t))(x - x^T(t))' \tilde{p}_B(x) dx \end{aligned}$$

where the symbol $'$ denotes transpose. We have thus successfully initiated a track for a new target entering the field in a sparse fashion.

Other Targets Existing in the Field

We will assume that all targets currently in the field are being tracked accurately so that we have the current density of the target position p^j of the m targets in the

field. With this information, the idea is to subtract out the contribution these targets have on the observed measurement of each sensor node so that the remaining part of the sensor measurement should correspond to any new targets entering the field. Then, we have essentially reduced the problem to the case when no targets are in the field. That is, we treat the estimated sensor measurements with the effect of all known tracks subtracted out as if they were the real sensor measurements and run the DCA's for single target localization. If any sensor node declares that it has computed the global localization task, then that sensor node will initiate a new track as described in the first scenario for track initiation.

It is because of using the information of known targets currently in the field that the problem of track initiation when targets exist in the field has been reduced to the problem of track initiation when there are no targets in the field. Without using this information of known targets, we have no choice but to solve the multiple target localization problem which is a difficult problem due to the parameter space of target locations being the space of finite Cartesian products of target positions. That is, if the space of target locations is $\mathcal{X}$, the space of finite multiple target locations is

$$\bigcup_{k=1}^{m} \mathcal{X}^k \ .$$

This is an example of how exploiting available information can allow us to solve an "easier" version of the problem.

Let us describe how to subtract out the contribution of known tracks in the field from each sensor measurement. We have two types of sensor measurements with which we perform localization for track initiation: intensity measurements and TDOA measurements.

Intensity Measurements

For a sensor node $i \in \{1, \ldots, n\}$ which observes intensity measurements, the measurement observed when there are m targets within range is the superposition of the individual contributions from each. The individual contribution to sensor node i by a target $j \in \{1, \ldots, m\}$, which we represent by a random variable Z_i^j, is the measurement that sensor node i would observe if target j was the only target within range of i. Then, the intensity measurement observed by sensor node i in the presence of m targets is given by

$$Z_i = \sum_{j=1}^{m} Z_i^j \ .$$

If there is a new target within range of sensor node i, then it contributes to the observation of sensor node i as well. Denoting this contribution of the new target by

Z_i^0, the intensity measurement in this case is given by

$$Z_i = Z_i^0 + \sum_{j=1}^{m} Z_i^j \ .$$

For the purpose of track initiation, we will assume that the above model for the intensity measurement of sensor node i. Then, the problem is:

Given an intensity measurement observation from sensor node i, say $Z_i = z_i$ where $z_i \in \mathbb{R}$, what is the posterior distribution of Z_i^0?

To solve this problem we must specify the prior distributions of Z_i^0 and the random variables in $\{Z_i^j\}_{j=1}^m$. The likelihood function of sensor node i observing z_i when there is a single target located at x is

$$p_i(z_i|x)$$

given by the intensity measurement model (Eqn. 2.3). Since the position of target j is distributed according to the density p^j, the prior density of the individual contribution Z_i^j is given by

$$P_i^j(z) = \int_{\mathbb{R}^2} p_i(z|x)p^j(x)dx \ .$$

We will approximate this density by a Gaussian so that computing the posterior density of Z_i^0 will be easy. The Gaussian approximation for the density of Z_i^j has the same mean and variance as P_i^j which are

$$\mu_{i,j} = \int_{\mathbb{R}} z P_i^j(z)dz \tag{3.8}$$

$$\sigma_{i,j}^2 = \int_{\mathbb{R}} (z - \mu_{i,j})^2 P_i^j(z)dz \ . \tag{3.9}$$

For the posterior density of Z_i^0, we also assume a Gaussian density. We choose a small mean $\mu_{i,0}$ because we assume that the target is far away when it first comes in range of the sensor. We choose the standard deviation $\sigma_{i,0}$ to be large enough so that there is a significant probability that the intensity measurement is near or less than or equal to 0.[2] For example, choosing $\sigma_{i,0}$ equal to $\mu_{i,0}$ is a good option.

Since all random variables Z_i^0 and $\{Z_i^j\}_{j=1}^m$ are Gaussian and assuming independence of these random variables, the posterior density of Z_i^0 given $Z_i = z_i$ is also a

[2]In our model of the intensity measurement (See Eqn. 2.3), there is the possibility of measuring a negative intensity of the signal. This certainly does not model reality because the intensity of the signal will always be non-negative. However, we ignore this deviation from reality in our intensity measurement model to simplify the computation of the posterior density.

Gaussian density with mean and variance given by

$$\begin{aligned}\tilde{\mu}_{i,0} &= \frac{\sigma_{i,0}^2(z_i - \sum_{k=1}^m \mu_{i,k}) + (\sum_{k=1}^m \sigma_{i,k}^2)\mu_{i,0}}{\sigma_{i,0}^2 + \sum_{k=1}^m \sigma_{i,k}^2} \\ \tilde{\sigma}_{i,0}^2 &= \frac{\sigma_{i,0}^2(\sum_{k=1}^m \sigma_{i,k}^2)}{\sigma_{i,0}^2 + \sum_{k=1}^m \sigma_{i,k}^2} \ .\end{aligned}$$

The above posterior density calculations for Z_i^0 is performed for each sensor node $i \in \{1, \ldots, n\}$ of the sensor network. Then, the sensor network can run the DCA's treating the sensor measurement observed by sensor node i to be $\tilde{\mu}_{i,0}$ and adding $\tilde{\sigma}_{i,0}^2$ to the variance of the noise term in the intensity measurement model for sensor node i. If some sensor node declares that it has computed the global localization task, then a new track can be initiated.

TDOA measurements

For a sensor node $i \in \{1, \ldots, n\}$ which observes TDOA measurements, the measurement observed when there are m targets within range is the union of the individual contributions from each. Denoting the individual contribution by target $j \in \{1, \ldots, m\}$ to sensor node i by a random variable Z_i^j, the measurement observed by sensor node i in the presence of m targets is given by

$$Z_i = \{Z_i^1, \ldots, Z_i^m\} \ .$$

If there is a new target within range of sensor node i, then denoting its contribution by Z_i^0, the TDOA measurement in this case is given by

$$Z_i = \{Z_i^0, Z_i^1, \ldots, Z_i^m\} \ .$$

The problem is similar to the case of intensity measurements.

Given a TDOA measurement observation from sensor node i, say $Z_i = z_i$ where z_i is a finite subset of $\mathbb{R}$, what is the posterior distribution of Z_i^0?

We specify the prior distributions of $\{Z_i^j\}_{j=1}^m$ just as before except that the likelihood function

$$p_i(z_i|x)$$

is given by the TDOA measurement model (Eqn. 2.6). We will again approximate the density $P_i^j$of Z_i^j by a Gaussian with mean and variance $\mu_{i,j}$ and $\sigma_{i,j}^2$ as given in Eqn. 3.8 and Eqn. 3.9 respectively. When the observation of sensor node i is $z_i = \{v_1, \ldots, v_l\}$, the idea for "subtracting out the effect of target j's contribution" to sensor node i's measurement is to remove the $v_k \in z_i$ which is within a window of two standard deviations of the mean of Z_i^j. If there are multiple elements in z_i which

are within this window, we remove one of them. Thus, the problem of subtracting out the effect of m targets is to associate m elements in z_i with the m means $\{\mu_{i,j}\}_{j=1}^{m}$ so that each element/mean pair is within two standard deviations of one another. If there are many such associations, then we can choose some regularization criterion to choose one of them. If there are no such associations, choose as many element/mean pair associations as possible.

We present an algorithm below which approximates the solution of subtracting out the effect of m targets from a TDOA measurement. The algorithm removes elements in z_i iteratively over the known targets $j \in \{1, \ldots, m\}$ by the following algorithm.

for $j := 1$ **to** m
  $\hat{v} = \arg_{v \in z_i} \min \; d(\mu_{i,j}, v)$
  if $d(\hat{v}, \mu_{i,j}) \leq 2\sigma_{i,j}$
    then $z_i := z_i - \{v\}$

The resulting z_i is the TDOA measurement observed by sensor node i with the contributions from the m known targets subtracted out. The above algorithm only approximates the solution due to the following pathological case. In the above algorithm, the order of the targets over which we loop can make a difference in which elements of z_i are removed. For example, if there are two targets whose contributions have mean and variance given by

$$\begin{aligned} \mu_{i,1} &= 0 \\ \mu_{i,2} &= 1 \\ \sigma_{i,1}^2 &= .5 \\ \sigma_{i,2}^2 &= .5 \ , \end{aligned}$$

then, if the observed TDOA measurement is

$$z_i = \{.1, -.9\} \ ,$$

and we iterate the above algorithm first with target 1 and then with target 2, we first take out .1 from z_i. Then, for target 2 since $\mu_{i,2}$ is greater than 2 standard deviations away from $-.9$, the result of the algorithm is

$$z_i = \{-.9\} \ .$$

On the other hand, if we iterate the above algorithm first with target 2 and then with target 1, we first take out .1 from z_i. Then, for target 1 since $\mu_{i,1}$ is within 2 standard deviations away from $-.9$, the result of the algorithm is

$$z_i = \emptyset \ .$$

This pathology occurs if there are many targets with nearly the same bearing from the TDOA sensor; however, we will not worry about this so much because the left over sensor measurement only contributes a part of the information for target localization.

Finally, as before, the DCA's can be run treating these leftover elements in z_i as the real observation. A track can be initiated if some sensor node declares it has computed the global localization task.

3.4.2 Track Updating

Although track initiations can only occur if targets enter the field in a sparse fashion, we will be able to track targets within the field regardless of whether they are sparse or dense. The purpose of track updating is to update the existing tracks in the previous time hypothesis H_{t-1} to the current time hypothesis H_t. We denote the current time by t.

We call a track $T \in H_{t-1}$ an *active track* if its stopping time T_{stop} has not yet been defined. An active track corresponds to a real target which is currently in the field. A track is referred to as a *terminated track* if the stopping time is defined and $T_{stop} \leq t$. A terminated track corresponds to a target which is no longer in the field or the previous history of positions of a target which is currently in the field.

The terminated tracks in H_{t-1} are kept unchanged in H_t. The only tracks which need to be updated from hypothesis H_{t-1} to H_t are the active tracks. Thus, for every active track $T \in H_{t-1}$, we have the following possibilities of updating T to T'.

1. **Continuation** The current position of track T based on sensor measurements is estimated and appended to the track.

2. **Termination** The track T is deemed to no longer correspond to a target within the field so that the track is terminated.

3. **Adjoinable Track Hypothesis** When tracks are about to cross with the potential for data association ambiguities and it is determined that there is currently no discriminatory information to disambiguate the data association, then the proposed solution is to terminate the tracks that are about to cross and initiate new ones. The idea is that the resulting hypothesis H_t contains an adjoined track which corresponds to a true target trajectory. This particular track update task handles the scenarios when no discriminating information is available when tracks cross.

4. **Resolving Adjoinable Track Hypotheses** For tracks which are attached with an adjoinable track hypothesis and discriminating information becomes available, the hypothesis can be resolved which has the effect of connecting certain adjoinable tracks together. This particular subtask takes care of the scenario when discriminating information becomes available after ambiguity in data association has occurred.

Continuation

The problem of continuing a track $T \in H_{t-1}$ is

Given the posterior density of the track position $\tilde{p}_{t-1}^T$ at time $t-1$, of track T from the previous time $t-1$, compute an estimate of its current track position $\tilde{p}_t^T$ at time t, given the observed joint measurement $z = (z_1, \ldots, z_n)$ at time t of all sensor nodes in the sensor network which are within range of track T.

We are given the posterior density $\tilde{p}_{t-1}^T$ of the track position from the previous time $t-1$ which corresponds to a target $j \in \{1, \ldots, m\}$ currently in the field. So, the first thing we must do is predict the density p_t^T of the track position for the *current* time. For each target j, we have a motion model represented as a transition density

$$q^j(x_t|x_{t-1})$$

which is the density that the target is located at x_t at the current time given that the target was located at x_{t-1} in the previous time. Depending on the information known about the target, the motion model q^j can be chosen accordingly. For example, if nothing is known about the motion of the target except that the next position should be within some distance σ from where the target was before, the suitable motion model would be a Gaussian diffusion such as

$$q^j(x_t|x_{t-1}) = G(x_t; x_{t-1}, diag(\sigma^2, \sigma^2)) \ .$$

If we know the target moves at constant velocity $v \in \mathbb{R}^2$, then a constant velocity motion model with Gaussian disturbance is given by

$$q^j(x_t|x_{t-1}; v) = G(x_t; x_{t-1} + v, diag(\sigma^2, \sigma^2))$$

for some appropriately chosen σ^2 and where we explicitly show the dependence of the motion model on the velocity as well as the previous position. Since we will use particle filters to implement track continuation, we also have a choice of number of particles to use when using different motion models. If the parameter σ^2 in the motion models above is small, then less particles are needed since the motion model becomes more deterministic. On the other hand, if the parameter σ^2 is large, more particles are needed in order to ensure that the region of high probability is well-populated by particles.

Then, the predicted density of the position of target j for the current time is given by

$$p^j(x_t) = \int_{\mathbb{R}^2} q^j(x_t|x_{t-1}) \tilde{p}_{t-1}^j(x_{t-1}) dx_{t-1}$$

where we have assumed that the location of track T corresponding to target j at the previous time was distributed according to the density

$$\tilde{p}_{t-1}^j = \tilde{p}_{t-1}^T$$

where we have just changed the label of the track T to the corresponding target j. The predicted density p^j is passed to a localization subtask which computes the likelihood L^j of the position of target j based on sensor measurements. This localization for a specific target will be discussed in detail in Section 3.4.3. Once L^j is received from the subtask, the posterior density of the position of target j is given by Eqn. 3.6 which is reprinted here

$$\tilde{p}^j(x) = \alpha p^j(x) L^j(x)$$

where α is a normalization constant.

Then, the track T corresponding to target j is updated by adding a new position and error covariance for the current time.

$$\begin{aligned} x^T(t) &= \int_{\mathbb{R}^2} x\tilde{p}^j(x)dx \\ \Sigma^T(t) &= \int_{\mathbb{R}^2} (x - x^T(t))(x - x^T(t))'\tilde{p}^j(x)dx \end{aligned}$$

The position and error covariance correspond to the the mean and covariance of the posterior position density $\tilde{p}^j$.

Termination

A track T should be terminated for two reasons:

1. if the target is no longer within the field or

2. there is evidence that the track does not correspond to a real target.

Testing for termination of a track is performed after the steps of track continuation have been complete. Assuming that there exists a target j in the field that corresponds to track T, from the computations involved in track continuation, we have the resulting likelihood for target j, L^j, and the new position and error covariance for track T, $x^T(t)$ and $\Sigma^T(t)$.

The test for whether the target is in the field is performed by checking whether $x^T(t)$ is in the field $F \subset \mathbb{R}^2$. If $x^T(t) \notin F$, then this track is terminated by setting $T_{stop} = t$.

The test for whether there is evidence that the track corresponds to a real target is to compare the likelihood L^j with the predicted density p^j. If the likelihood and predicted density are similar in the sense that the regions where we expect the target to lie with high probability overlap, then we are inclined to believe that there is lots of evidence that the track corresponds to a real target. On the other hand, if the likelihood and predicted density are not similar, then we are inclined to believe that the track may not correspond to a real target. To formalize the notion of similarity,

we need a measure of similarity between two densities. The Kullback-Leibler distance is a good choice,

$$D(\alpha L^j \| p^j) = \int_{\mathbb{R}^2} \alpha L^j(x) \log \frac{\alpha L^j(x)}{p^j(x)} dx$$

where α is a normalization factor to satisfy

$$\int_{\mathbb{R}^2} \alpha L^j(x) dx = 1 \ .$$

Since p^j is a probability density, it already integrates to 1. Then, given a threshold γ, we test if

$$D(\alpha L^j \| p^j) \geq \gamma \ .$$

If true, we terminate track T by setting $T_{stop} = t$.

Adjoinable Track Hypothesis

When two tracks are about to cross in position and there is no discriminatory information to disambiguate between the two tracks, our solution is to form an adjoinable track hypothesis.

Definition 3.1 *An* adjoinable track hypothesis *for a track T is a set of tracks*

$$\{T_1, \ldots, T_m\}$$

which can be adjoined to a track T. That is, $T_i \circ T$ is an adjoined track for all $i \in \{1, \ldots, m\}$.

The idea is that an adjoinable track hypothesis S for a track T represents the set of possible beginnings of a track T. We will attach an adjoinable track hypothesis to tracks which cross to represent the possible data associations of tracks before and after crossing.

After continuing all tracks in H_{t-1} and terminating the appropriate ones, we will check to see if any adjoinable track hypotheses need to be formed. For each active track $T \in H_t$, collect the set of active tracks S_T which are within a radius of r around T and are associated with targets for which there is no discriminating information. S_T are those tracks where there is the possibility of a data association problem. We add a track R corresponding to the last position and error covariance of track T. That is, R is defined to be

$$\begin{aligned} R &= \{(x^R(t), \Sigma^R(t))\} \\ R_{start} &= t \\ R_{stop} & \quad \text{undefined} \end{aligned}$$

where

$$\begin{aligned} x^R(t) &= x^T(t) \\ \Sigma^R(t) &= \Sigma^T(t) \ . \end{aligned}$$

Then, we terminate the track T and the tracks $T' \in S_T$ by setting

$$\begin{aligned} T_{stop} &= t-1 \\ T'_{stop} &= t-1 \ . \end{aligned}$$

We then create an adjoinable track hypothesis for track R,

$$\{T\} \cup S_T$$

which is additional information that is attached to track R.

Resolving Adjoinable Track Hypotheses

For any active tracks $T \in H_t$ to which there is attached an adjoinable track hypothesis S, we wish to check if there is any discriminating information we can collect which can rule out tracks from S which are to be adjoined to T.

Let us consider an example of a particular scenario where an adjoinable track hypothesis can be resolved. In the presentation of the scenarios in Section 1.3, we presented an example of two targets crossing where one, which we denote by T^1, was known to be metallic and the other, which we denote by T^2, was known to be non-metallic. When the targets actually cross, say at time t, there is no discriminating information available since there is no magnetometer in range. What will happen in this case is that an adjoinable track hypothesis is formed

$$S = \{T^1, T^2\} \ ,$$

the previous tracks are terminated

$$\begin{aligned} T^1_{stop} &= t-1 \\ T^2_{stop} &= t-1 \ , \end{aligned}$$

and new tracks are formed, T^3 and T^4 where

$$\begin{aligned} T^3_{start} &= t \\ T^4_{start} &= t \ . \end{aligned}$$

Both of these two tracks are attached with the adjoinable track hypothesis S. As the tracking progresses and the two targets begin to separate, one of the targets, say T^3, enters the range of a magnetometer. Then, there are two possibilities.

1. If the magnetometer reading is that the target within range is metallic, then we know that T^3 should be adjoined with T^1 since T^1 corresponded with a metallic target. Furthermore, we can infer that T^4 should be adjoined with T^2. Resolving the adjoinable track hypothesis S in this case is to remove T^1, T^2, T^3, and T^4 from the hypothesis and add the adjoined tracks $T^1 \circ T^3$ and $T^2 \circ T^4$.

2. If the magnetometer reading is that the target within range is non-metallic, then we remove T^1, T^2, T^3, and T^4 from the hypothesis and add the adjoined tracks $T^1 \circ T^4$ and $T^2 \circ T^3$.

The general case for resolving an adjoinable track hypothesis using magnetometer measurements is the following. For each active track T in the current hypothesis H_t with an associated adjoinable track hypothesis

$$S = S_{metal} \cup S_{non}$$

where S_{metal} represents the set of tracks in S which are associated with metallic targets and S_{non} represents the set of tracks in S which are associated with non-metallic targets, we test if track T is the only track within range of a magnetometer sensor i and the adjoinable track hypothesis S contains tracks associated wtih both metallic and non-metallic targets. If true, the following steps are performed.

1. If the observation z_i of magnetometer sensor i is equal to 1, then we update the adjoinable track hypothesis S of track T to

$$S' = S_{metal} \ .$$

2. If the observation z_i of magnetometer sensor i is equal to 0, then we update the adjoinable track hypothesis S of track T to

$$S' = S_{non} \ .$$

Then, for the case when the updated adjoinable track hypothesis S' of track T contains only one track, say R, we perform the following.

1. Remove T and R from the current hypothesis H_t, and add $R \circ T$.

2. For all tracks T' in H_t for which there is an adjoinable track hypothesis $S_{T'}$, we remove R from it.

$$S_{T'} := S_{T'} - \{R\}$$

   The reason R is removed is because we have resolved the association of R to T. Note that this step may cause $S_{T'}$ to contain only a single track so that the previous step will have to be iterated.

3.4.3 Localization for a Specific Target

Localizing for a specific target is a subtask of the track updating task. The track updating task passes the prior densities $\{p^j\}_{j=1}^m$ of the estimated positions of targets to this task as well as other information like the nominal intensity and frequency signature of the targets. The task of localizing for a specific target is to compute the following:

$$p^j \Rightarrow P_i^j \Rightarrow \tilde{P}_i^j \Rightarrow L^j \ .$$

Given the prior densities $\{p^j\}_{j=1}^m$ of target positions which are in range of sensor node i, the idea for localizing for a specific target is to compute the posterior density $\tilde{P}_i^j$ of the the individual contribution Z_i^j given the observation $Z_i = z_i$ of sensor node i from the prior density P_i^j. Then, we can compute the approximate likelihood L^j of the position of target j from $\{\tilde{P}_i^j\}_{i=1}^n$ and the measurement likelihoods $\{p_i\}_{i=1}^n$ of each sensor node according to Eqn. 3.7. L^j is then passed passed back to the track updating task which will update the track corresponding to target j with this likelihood appropriately.

Note that the localization L^j for a specific target j would not be possible without the availability of the prior densities $\{p^j\}_{j=1}^m$ of the target locations. Without using this information, we are left with no choice but to perform multiple target tracking using as inputs the output of some multiple target localization algorithm. By exploiting this information, we are able to avoid having to implement a multiple target localization algorithm and still be able to track multiple targets in the field as long as they are tracked with reasonable accuracy and enter the field in a sparse fashion.

Since there is the possibility that targets have been classified for their frequency signatures and because we will want to use this extra information about targets if it is available and needed (e.g. for data association), we have four possibilities for computing the posterior density $\tilde{P}_i^j$ of the individual contribution from a target.

1. **Intensity Only** Sensor node i can measure intensity measurements. Only the intensities and positions of all targets within range of sensor node i are known.

2. **Frequency Bandlimited Intensity Only** Sensor node i can measure frequency bandlimited measurements. The frequency signatures and positions of all targets within range of sensor node i are known.

3. **Mixed Intensity and Frequency** Sensor node i can measure both intensity and frequency bandlimited measurements. For some of the targets within range of sensor node i, only the intensity is known, and for the rest of the targets within range, the frequency signatures are also known.

4. **TDOA** Sensor node i can measure TDOA measurements. Bearings of all targets within range are measured.

The rest of this section discusses how to compute $\{\tilde{P}_i^j\}_{j=1}^m$ for targets within range of a given sensor node i and specifies the associated likelihood p_i for the various cases above.

Intensity Only

When there are m targets labelled $j \in \{1, \ldots, m\}$ within range of sensor node i which observes intensity measurements, the observation of sensor node i is modelled by

$$Z_i = \sum_{j=1}^{m} Z_i^j \tag{3.10}$$

where Z_i^j is the contribution to sensor node i by target j.

Given the prior density p^j of the location of target j, we can compute the density of the individual contribution Z_i^j by

$$P_i^j(z) = \int_{\mathbb{R}^2} p_i(z|x) p^j(x) dx$$

where the likelihood $p_i(z|x)$ is given by the intensity measurement model (Eqn. 2.3). We approximate P_i^j by a Gaussian with same mean $\mu_{i,j}$ and variance $\sigma_{i,j}^2$ as shown in Eqn. 3.8 and Eqn. 3.9 respectively.

Then, given that sensor node i observed the measurement $Z_i = z_i$ where the model of the sensor measurement is given by Eqn. 3.10 and assuming that the individual contributions $\{Z_i^j\}_{j=1}^m$ are independent Gaussians with mean $\mu_{i,j}$ and variance $\sigma_{i,j}^2$, the posterior density $\tilde{P}_i^j$ of Z_i^j is also a Gaussian with mean and variance given by

$$\begin{aligned}
\tilde{\mu}_{i,j} &= \frac{\sigma_{i,j}^2 (z_i - \sum_{k=1,k\neq j}^m \mu_{i,k}) + (\sum_{k=1,k\neq j}^m \sigma_{i,k}^2)\mu_{i,j}}{\sum_{k=1}^m \sigma_{i,k}^2} \\
\tilde{\sigma}_{i,j}^2 &= \frac{\sigma_{i,j}^2 (\sum_{k=1,k\neq j}^m \sigma_{i,k}^2)}{\sum_{k=1}^m \sigma_{i,k}^2} .
\end{aligned}$$

Thus, we have computed the posterior density for $\tilde{P}_i^j$ given by

$$\tilde{P}_i^j(z) = G(z; \tilde{\mu}_{i,j}, \tilde{\sigma}_{i,j}^2)$$

for $j \in \{1, \ldots, m\}$. The associated likelihood p_i is given by the intensity measurement model of Eqn. 2.3.

Frequency Bandlimited Intensity Only

In this case, there are m targets within range of sensor node i which can measure frequency bandlimited intensity, and the frequency signature $\underline{A}^j$ of each target $j \in$

$\{1, \ldots, m\}$ is also known. $\underline{A}^j$ is a vector of the nominal intensities emitted by target j in each of b frequency bands given by

$$\underline{A}^j = \begin{bmatrix} A_1^j \\ A_2^j \\ \vdots \\ A_b^j \end{bmatrix} . \tag{3.11}$$

The frequency bandlimited measurement observed by sensor node i is given by

$$\underline{Z}_i = \sum_{j=1}^{m} \underline{Z}_i^j \tag{3.12}$$

where $\underline{Z}_i$ and $\underline{Z}_i^j$ are vectors of length b where each component corresponds to the intensity measured in one of the b frequency bands. $\underline{Z}_i^j$ is the individual contribution to sensor measurement i by target j which was given in Eqn. 2.5. We will make a simplification to this frequency bandlimited measurement model and assume that for all frequency bands $\phi \in \{1, \ldots, b\}$, the intensity scaling factors $\{f_\phi\}_{\phi=1}^b$ and the atmospheric attenuation coefficients $\{\alpha_\phi\}_{\phi=1}^b$ are the same. That is,

$$\begin{aligned} f_\phi &= f \\ \alpha_\phi &= \alpha \end{aligned}$$

for all $\phi \in \{1, \ldots, b\}$. Then, the model for the frequency bandlimited intensity sensor i when m targets are in range is given by

$$\underline{Z}_i = \sum_{j=1}^{m} \underline{A}^j g_i(x^j) + \underline{N}_i \tag{3.13}$$

where x^j is the position of target j, $\underline{N}_i$ is a multi-variate zero mean Gaussian random vector of length b with covariance

$$\Sigma = diag(\sigma^2, \ldots, \sigma^2)$$

and

$$g_i(x) = f(d(x_i, x)^\alpha) .$$

Notice that the contribution of track j to sensor node i in the model above is through the function g_i. Thus, letting X^j be the random variable associated with the position of target j, then the individual contribution Z_i^j by target j can be considered to be

$$Z_i^j = g_i(X^j)$$

where $Z_i^j \in \mathbb{R}$ is a scalar rather than the original vector individual contribution $\underline{Z}_i^j$ in Eqn. 3.12. Thus, rewriting the model given by Eqn. 3.13 above, we have

$$\underline{Z}_i = \sum_{j=1}^{m} \underline{A}^j Z_i^j + \underline{N}_i \ . \tag{3.14}$$

The likelihood function associated with Z_i^j is given by

$$p_i(z|x) = \delta_0(z - f(d(x_i, x)^\alpha)) \tag{3.15}$$

where $\delta_0(\cdot)$ denotes the Dirac delta function centered at 0. Let us now calculate the posterior density $\tilde{P}_i^j$ for each individual contribution Z_i^j.

Given the prior density p^j of the position of target j, we compute the prior density of the individual contribution Z_i^j by

$$P_i^j(z) = \int_{\mathbb{R}^2} p_i(z|x) p^j(x) dx$$

where p_i is given by Eqn. 3.15 above. As usual, we will approximate P_i^j by a Gaussian density with the same mean and variance $\mu_{i,j}$ and $\sigma_{i,j}^2$ as given by Eqn. 3.8 and Eqn. 3.9 respectively. Then, given the observation $\underline{Z}_i = \underline{z}_i$ of sensor node i, we wish to compute the posterior density $\tilde{P}_i^j$ of Z_i^j. Because $\{Z_i^j\}_{j=1}^m$ are Gaussian, the noise term $\underline{N}_i$ is a Gaussian, and they are independent, the posterior density $\tilde{P}_i^j$ is also Gaussian which we will now derive.

Denote the vector of random variables $\{Z_i^j\}_{j=1}^m$ by

$$\underline{Y}_i = \begin{bmatrix} Z_i^1 \\ \vdots \\ Z_i^m \end{bmatrix}$$

and the vector of realizations of $\underline{Y}_i$ by the lower case

$$\underline{y}_i = \begin{bmatrix} z_i^1 \\ \vdots \\ z_i^m \end{bmatrix}$$

where $\{z_i^j\}_{j=1}^m$ are the realizations of the random variables $\{Z_i^j\}_{j=1}^m$. Also, let us define the matrix

$$\mathbf{A} = \begin{bmatrix} | & \cdots & | \\ \underline{A}^1 & \cdots & \underline{A}^m \\ | & \cdots & | \end{bmatrix}$$

given by the known frequency signatures $\{\underline{A}^j\}_{j=1}^m$ of the m targets. We also collect the means $\mu_{i,j}$ and variances $\sigma_{i,j}^2$ of P_i^j by

$$\bar{\mu}_i = \begin{bmatrix} \mu_{i,1} \\ \vdots \\ \mu_{i,m} \end{bmatrix}$$

and

$$D = diag(\sigma_{i,1}^2, \ldots, \sigma_{i,m}^2) \ .$$

Then, from the model for $\underline{Z}_i$ given in Eqn. 3.14,

$$P(\underline{Z}_i = \underline{z}_i | \underline{Y}_i = \underline{y}_i) \sim G(\underline{z}_i; \mathbf{A}\underline{y}_i, \Sigma)$$

where by $\sim$ we mean that the right hand side is the corresponding density. Then, the posterior distribution has density

$$\begin{aligned} P(\underline{Y}_i = \underline{y}_i | \underline{Z}_i = \underline{z}_i) &= \alpha P(\underline{Z}_i = \underline{z}_i | \underline{Y}_i = \underline{y}_i) P(\underline{Y}_i = \underline{y}_i) \\ &\sim \alpha G(\underline{z}_i; \mathbf{A}\underline{y}_i, \Sigma) \prod_{j=1}^m G(z_i^j; \mu_{i,j}, \sigma_{i,j}^2) \end{aligned} \tag{3.16}$$

where α is a normalization constant. Thus, the joint posterior distribution of $\underline{Y}_i$ is Gaussian so that all we need is the mean and variance of this posterior distribution. The exponent of the density in Eqn. 3.16 is given by

$$-\frac{1}{2}(\underline{z}_i - \mathbf{A}\underline{y}_i)'\Sigma^{-1}(\underline{z}_i - \mathbf{A}\underline{y}_i) - \frac{1}{2}(\underline{y}_i - \underline{\mu}_i)'D^{-1}(\underline{y}_i - \underline{\mu}_i)$$

where the symbol $'$ denotes transpose. The idea to find the mean and variance is to rewrite the above exponent in the form

$$(\underline{y}_i - \underline{c})'F^{-1}(\underline{y}_i - \underline{c}) + d_1$$

where d_1 is some term not depending on $\underline{y}_i$. Then, $\underline{c}$ will correspond to the mean and F will correspond to the variance. Any term without $\underline{y}_i$ can be disregarded because this term is in the constant d_1 which only affects the Gaussian density as part of the normalization constant.

Dropping the $-\frac{1}{2}$ term, we begin with

$$(\underline{z}_i - \mathbf{A}\underline{y}_i)'\Sigma^{-1}(\underline{z}_i - \mathbf{A}\underline{y}_i) + (\underline{y}_i - \underline{\mu}_i)'D^{-1}(\underline{y}_i - \underline{\mu}_i) \ .$$

Expanding, we have

$$\underline{z}_i'\Sigma^{-1}\underline{z}_i - 2\underline{z}_i'\Sigma^{-1}\mathbf{A}\underline{y}_i + \underline{y}_i'\mathbf{A}'\Sigma^{-1}\mathbf{A}\underline{y}_i + \underline{y}_i'D^{-1}\underline{y}_i - 2\underline{\mu}_i'D^{-1}\underline{y}_i + \underline{\mu}_i'D^{-1}\underline{\mu}_i \ .$$

Taking out terms which do not depend on $\underline{y}_i$ and collecting the quadratic and linear terms, we have

$$\underline{y}_i'(D^{-1} + \mathbf{A}'\Sigma^{-1}\mathbf{A})\underline{y}_i - 2(\underline{\mu}_i'D^{-1} + \underline{z}_i'\Sigma^{-1}\mathbf{A})\underline{y}_i + d_2 \ .$$

Then, since

$$(\underline{y}_i - \underline{c})'F^{-1}(\underline{y}_i - \underline{c}) = \underline{y}_i'F^{-1}\underline{y} - 2\underline{c}'F^{-1}\underline{y}_i + d_3$$

where d_3 again indicates terms that do not depend on $\underline{y}_i$, we can read off F and $\underline{c}$ which are

$$\begin{aligned} F &= (D^{-1} + \mathbf{A}'\Sigma^{-1}\mathbf{A})^{-1} \\ \underline{c} &= (D^{-1} + \mathbf{A}'\Sigma^{-1}\mathbf{A})^{-1}(D^{-1}\underline{\mu}_i + \mathbf{A}'\Sigma^{-1}\underline{z}_i) \end{aligned}$$

Note that F is not diagonal in general so that the individual contributions $\{Z_i^j\}_{j=1}^m$ are not conditionally independent given $\underline{Z}_i$. For each target $j \in \{1, \ldots, m\}$, the posterior density of the individual contribution to sensor node i is given by

$$\tilde{P}_i^j(z) = G(z; \underline{c}_j, F_{j,j})$$

where $\underline{c}_j$ is the jth component of

$$\underline{c} = \begin{bmatrix} \underline{c}_1 \\ \vdots \\ \underline{c}_m \end{bmatrix}$$

and $F_{j,j}$ is the jth diagonal element of

$$F = \begin{bmatrix} F_{1,1} & \cdots & F_{1,m} \\ \vdots & \ddots & \vdots \\ F_{m,1} & \cdots & F_{m,m} \end{bmatrix} \ .$$

The likelihood p_i for these contributions is given by Eqn. 3.15.

Mixed Intensity and Frequency

We now consider the case when we know the frequency signature of only some of the targets that are within range of a sensor node i which can measure intensity and frequency bandlimited intensity measurements. Say that there are m targets within range and that we know the frequency signatures for l of them, where $l < m$. We will label the targets for which we know the frequency signature by an integer $j \in \{1, \ldots, l\}$ with associated frequency signature $\underline{A}^j$. For the targets for which we do not know the frequency signature, we will label them by an integer $j \in \{l+1, \ldots, m\}$ and refer to their unknown frequency signature by $\underline{B}^j$.

Then, the simple frequency bandlimited measurement model (Eqn. 3.14) for the measurement received by this sensor is

$$\underline{Z}_i = \sum_{i=1}^{l} \underline{A}^j Z_i^j + \sum_{i=l+1}^{m} \underline{B}^j Z_i^j + \underline{N}_i$$

where $\underline{N}_i$ is a multi-variate zero mean Gaussian random vector of length b with diagonal covariance Σ as described before. Since we would like to use all information available to us, we would like to make use of the frequency signature information about targets $j \in \{1, \ldots, l\}$. Since the frequency signature of targets in $\{l+1, \ldots, m\}$ are not known, we must first estimate the frequency signatures $\{\underline{B}^j\}_{j=l+1}^m$ in the model above. Once we estimate the unknown frequency signatures, we are in the case of frequency bandlimited measurements only for estimating the posterior densities of the individual contributions Z_i^j. This was described above under the **Frequency Bandlimited Intensity Only** part of this section. Thus, the only new problem presented by mixed intensity and frequency information is how to choose an estimate for $\{\underline{B}^j\}_{j=l+1}^m$ which we will now present.

We compute the posterior densities P_i^j of the individual contributions Z_i^j as before where the likelihood function is given by Eqn. 3.15. Then, the mean and variance of P_i^j, denoted by $\mu_{i,j}$ and $\sigma_{i,j}^2$, can be computed as shown in Eqn. 3.8 and Eqn. 3.9. For estimating the unknown frequency signatures, we will assume that the Z_i^j are exactly the mean value $\mu_{i,j}$. Thus, we have the following model for the measurement of sensor node i.

$$\underline{Z}_i = \sum_{j=1}^{l} \underline{A}^j \mu_{i,j} + \sum_{j=l+1}^{m} \underline{B}^j \mu_{i,j} + \underline{N}_i$$

We will assume that the intensity of all targets are known even though the frequency signature is not all known. Thus, we assume that

$$\begin{aligned} \sum_{\phi=1}^{b} A_\phi^j &= A_{tot} \\ \sum_{\phi=1}^{b} B_\phi^j &= A_{tot} \end{aligned}$$

where $A_{tot} \in \mathbb{R}$ is the known nominal intensity of all targets in the field, $\{A_\phi^j\}_{\phi=1}^b$ are the components of $\underline{A}^j$, and $\{B_\phi^j\}_{\phi=1}^b$ are the components of $\underline{B}^j$ as given in Eqn. 3.11. In general, the nominal intensities of different targets can be different; however, we will present the simple case of when all targets have the same nominal intensity.

Furthermore, we assume that the frequency signatures of $\{\underline{B}^j\}_{j=l+1}^m$ are all the same. That is,

$$\underline{B}^j = \underline{B}$$

so that the measurement model becomes

$$\underline{Z}_i = \sum_{j=1}^{l} \underline{A}^j \mu_{i,j} + \underline{B} \left(\sum_{j=l+1}^{m} \mu_{i,j} \right) + \underline{N}_i \ .$$

Then, we can define a likelihood function for $\underline{Z}_i$ given $\underline{B}$ which is

$$f(\underline{z}_i|\underline{B}) = G\left(\underline{z}_i; \sum_{j=1}^{l} \underline{A}^j \mu_{i,j} + \underline{B} \left(\sum_{j=l+1}^{m} \mu_{i,j} \right), \Sigma \right) \ .$$

The idea behind this assumption is to be consistent with the fact that there is no information about the frequency signatures of the targets $\{l+1, \ldots, m\}$ that can distinguish between them. If the $\underline{B}^j$ were estimated to be different, then there is implicitly frequency information being used to determine the posterior density of Z_i^j. We choose all $\underline{B}^j$ to be the same for the purpose of localizing the targets under this mixed intensity and frequency case. We do not suggest that the estimated $\underline{B}^j$ is the frequency signature of the targets $j \in \{l+1, \ldots, m\}$.

We will pose the problem of estimating $\underline{B}$ as a parameter estimation problem.

Given the frequency bandlimited sensor measurement $\underline{Z}_i = \underline{z}_i$, compute the maximum likelihood estimate of $\underline{B}$ given by

$$\underline{B}^* = \arg_{\underline{B} \in \mathbf{B}} \max f(\underline{z}_i|\underline{B})$$

where $\mathbf{B}$ is a subset of vectors in $\mathbb{R}^b$ such that for all $\underline{B} \in \mathbf{B}$,

$$\begin{aligned} \sum_{\phi=1}^{b} B_\phi &= A_{tot} \\ B_\phi &\geq 0 \ for \ all \ \phi \in \{1, \ldots, b\} \ . \end{aligned}$$

We assume that covariance Σ is diagonal with equal valued elements on the diagonal. Then, because f is a Gaussian density, the maximization problem becomes the following minimization problem

$$\underline{B}^* = \arg_{\underline{B} \in \mathbf{B}} \min \ (\alpha \underline{B} - \underline{D})'(\alpha \underline{B} - \underline{D})$$

where

$$\begin{aligned} \alpha &= \sum_{j=l+1}^{m} \mu_{i,j} \\ \underline{D} &= \underline{z} - \sum_{j=1}^{l} \underline{A}^j \mu_{i,j} \ . \end{aligned}$$

This is a quadratic minimization problem with equality and inequality constraints which can be solved by setting the derivatives of the following Lagrangian to zero.

$$\mathcal{L}(\underline{B}, \mu, \{\lambda_\phi\}_{\phi=1}^b) = (\alpha\underline{B} - \underline{D})'(\alpha\underline{B} - \underline{D}) - \mu\left(A_{tot} - \sum_{\phi=1}^b B_\phi\right) - \sum_{\phi=1}^b \lambda_\phi B_\phi$$

where the Kuhn-Tucker conditions of the optimum $\underline{B}^*$ and λ_ϕ^* are

$$\begin{array}{l} (1) \text{ if } B_\phi^* > 0, \text{ then } \lambda_\phi = 0 \\ (2) \text{ if } B_\phi^* = 0, \text{ then } \lambda_\phi \geq 0 \ . \end{array}$$

An Algorithm for Solving the Quadratic Minimization

The actual algorithm to solve this quadratic minimization problem is easy. We first consider the minimization with only the equality constraints on $\underline{B}$. The Lagrangian for this simpler problem is given by

$$\mathcal{L}(\underline{B}, \mu) = (\alpha\underline{B} - \underline{D})'(\alpha\underline{B} - \underline{D}) - \mu\left(A_{tot} - \sum_{\phi=1}^b B_\phi\right) \ .$$

The solution $\hat{\underline{B}}$ is given by setting the derivatives of the Lagrangian to zero.

$$\begin{aligned} 2\alpha^2\hat{B}_\phi - 2\alpha D_\phi - \mu &= 0 \text{ for all } \phi \in \{1, \ldots, b\} \\ \sum_{\phi=1}^b \hat{B}_\phi &= A_{tot} \ . \end{aligned}$$

Then, if any components of $\hat{\underline{B}}$ are negative, we set those components to be 0 and solve a reduced minimization problem with only the positive components of $\hat{\underline{B}}$. That is, if $S \subset \{1, \ldots, b\}$ are the components of $\hat{\underline{B}}$ which are positive, we assume that the components other components $\{1, \ldots, b\} - S$ are zero and solve

$$\begin{aligned} 2\alpha^2 B_\phi - 2\alpha D_\phi - \mu &= 0 \text{ for all } \phi \in S \\ \sum_{\phi \in S} B_\phi &= A_{tot} \\ B_\phi &= 0 \text{ for all } \phi \in \{1, \ldots, b\} - S \ . \end{aligned}$$

Calling the result $\hat{\hat{\underline{B}}}$, if there are any negative components, we repeat the same steps above. This is iterated until all components are non-negative. Call this result $\underline{B}^*$. It is easy to verify that the resulting $\underline{B}^*$ satisfies the original quadratic programming problem.

Once the solution $\underline{B}^*$ is obtained, we can use

$$\underline{B}_j = \underline{B}^*$$

for all $j \in \{l+1, \ldots, m\}$. Then, the posterior densities of the individual contributions Z_i^j can be computed.

TDOA

When there are m targets within range of a sensor node i which observes TDOA measurements, the model for the measurement is given by

$$Z_i = \{Z_i^1, \ldots, Z_i^m\}$$

where Z_i^j represents the time difference of arrival observed by sensor i from target j. When given the observation of the TDOA sensor $Z_i = z_i$, the idea is to associate for each target j the element $v \in z_i$ which is within two standard deviations of the mean of prior distribution of Z_i^j and closest to the mean. If there is no such $v \in z_i$ which satisfies this, we will just return the prior density of Z_i^j as the posterior density because there was no additional information to update the density of Z_i^j.

To formalize the above idea, we compute the prior density P_i^j of each Z_i^j. Then, approximate P_i^j by a Gaussian with the same mean and variance $\mu_{i,j}$ and $\sigma_{i,j}^2$ as given by Eqn. 3.8 and Eqn. 3.9. Given that the observation of TDOA sensor node i is $Z_i = z_i$, for each target $j \in \{1, \ldots, m\}$, compute the possible subset of time difference of arrival estimates which can be associated with target j,

$$V_j = \{v \in z_i : d(v, \mu_{i,j}) \leq 2\sigma_{i,j}^2\} \ .$$

If V_j is empty, then the posterior density of Z_i^j is

$$\tilde{P}_i^j(z) = G(z; \mu_{i,j}, \sigma_{i,j}^2) \ .$$

If V_j is not empty, compute the closest $\hat{v}_j$ in V_j by

$$\hat{v}_j = \arg_{v \in V_j} \min d(v, \mu_{i,j}) \ .$$

Then, the posterior density of Z_i^j is

$$\tilde{P}_i^j(z) = \delta_0(z - \hat{v}_j)$$

where $\delta_0(\cdot)$ is the Dirac delta function centered at 0. The associated likelihood function p_i for this TDOA sensor node i is given by the TDOA measurement model in Eqn. 2.6.

3.4.4 Classification of Frequency Signatures

In this section, we give a solution to the problem of classifying frequency signatures for m targets in the field based on frequency bandlimited intensity measurements from n sensor nodes labelled by $i \in \{1, \ldots, n\}$ as formulated in Section 2.5.1. We will use the simple model for frequency bandlimited measurements of Eqn. 3.14, reprinted here

$$\underline{Z}_i = \sum_{j=1}^{m} \underline{A}^j Z_i^j + \underline{N}_i \ .$$

We assume the availability of accurate position estimates of the m targets which will enable us to classify for the frequency signature of the targets due to their differences in position from a set of frequency bandlimited intensity sensors. These estimates give a distribution for the individual contributions Z_i^j above so that the only parameters are the frequency signatures $\{\underline{A}^j\}_{j=1}^m$. Without a distribution for the individual contributions, estimation of the frequency signatures is ill-posed unless estimation is performed jointly for the frequency signatures and the track contributions. By exploiting the information about target positions, we are able to classify for the frequency signatures of specific targets, and hence, we have reduced the problem of classifying for frequency signatures to an "easier" version.

Given that the individual contributions Z_i^j are distributed as a Gaussian with mean $\mu_{i,j}$ and $\sigma_{i,j}^2$ and $\underline{N}_i$ is a zero-mean Gaussian vector of length b with diagonal covariance

$$\Sigma = diag(\sigma^2, \ldots, \sigma^2)$$

and all variables are independent, the likelihood of observing $\underline{Z}_i = \underline{z}_i$ given the frequency signatures $\{\underline{A}^j\}_{j=1}^m$ is

$$f_i(\underline{z}_i|\{\underline{A}^j\}_{j=1}^m) = G(\underline{z}_i; \mathbf{A}\underline{\mu}_i, \mathbf{A}\Sigma_i\mathbf{A}' + \Sigma)$$

where

$$\mathbf{A} = \begin{bmatrix} | & \cdots & | \\ \underline{A}^1 & \cdots & \underline{A}^m \\ | & \cdots & | \end{bmatrix} ,$$

$$\underline{\mu}_i = \begin{bmatrix} \mu_{i,1} \\ \vdots \\ \mu_{i,m} \end{bmatrix} ,$$

and

$$\Sigma_i = diag(\sigma_{i,1}^2, \ldots, \sigma_{i,m}^2) \ .$$

Then, the likelihood of observing $\{\underline{Z}_i = \underline{z}_i\}_{i=1}^n$ from n sensor nodes, under the assumption that all random variables Z_i^j and $\underline{N}_i$ are independent, is given by

$$\begin{aligned} f(\underline{z}_1, \ldots, \underline{z}_n|\{\underline{A}^j\}_{j=1}^m) &= \prod_{i=1}^n f_i(\underline{z}_i|\{\underline{A}^j\}_{j=1}^m) \\ &= \prod_{i=1}^n G(\underline{z}_i; \mathbf{A}\underline{\mu}_i, \mathbf{A}\Sigma_i\mathbf{A}' + \Sigma) \end{aligned}$$

Given a finite set of nominal frequency signatures $\{\underline{C}^k\}_{k=1}^K \subset \mathbb{R}^b$ for K known classes of targets where b is the number of frequency bands. The problem is to solve

$$(\hat{\underline{A}}^1, \ldots, \hat{\underline{A}}^m) = \arg_{\underline{B}^1, \ldots, \underline{B}^m \in \{\underline{C}^k\}_{k=1}^K} \max f(\underline{z}_1, \ldots, \underline{z}_n|\{\underline{B}^j\}_{j=1}^m)$$

Notice that the time complexity of the brute force solution to the above problem requires checking for Km combinations of values for elements in $\{\underline{B}^j\}_{j=1}^m$. With large K and m, the time required to compute this classification may be unnecessarily long. Thus, we propose solving the relaxed problem where $\hat{\underline{A}}^j$ can take any value in $\mathbb{R}^b$.

Maximizing the likelihood f is the same as minimizing the following because the likelihood of f is the product of several Gaussians.

$$\bar{\mathbf{A}} = \arg_{\mathbf{A}} \min \sum_{i=1}^{n} (\underline{z}_i - \mathbf{A}\underline{\mu}_i)'(\mathbf{A}\Sigma_i\mathbf{A}' + \Sigma)^{-1}(\underline{z}_i - \mathbf{A}\underline{\mu}_i)$$

where the minimization is over all matrices $b \times m$ matrices $\mathbf{A}$. This is a difficult optimization so we will solve a simpler problem where the covariance Σ_i is equal to 0. This is an assumption that the contributions by each Z_i^j is exactly the mean $\mu_{i,j}$, and since Σ is diagonal with all diagonal components equal, we can reduce the optimization to

$$\bar{\mathbf{A}} = \arg_{\mathbf{A}} \min \sum_{i=1}^{n} (\underline{z}_i - \mathbf{A}\underline{\mu}_i)'(\underline{z}_i - \mathbf{A}\underline{\mu}_i) \ .$$

Since this expression is quadratic in $\mathbf{A}$, we set the derivative of the expression above with respect to the elements of $\mathbf{A}$ to zero to find the minimum. Computing the derivative, setting them to zero, and arranging the terms into vectors and matrices, we have

$$H\mathbf{Z}' = HH'\bar{\mathbf{A}}'$$

where

$$H = \begin{bmatrix} | & \cdots & | \\ \underline{\mu}_1 & \cdots & \underline{\mu}_n \\ | & \cdots & | \end{bmatrix}$$

and

$$\mathbf{Z} = \begin{bmatrix} | & \cdots & | \\ \underline{z}_1 & \cdots & \underline{z}_n \\ | & \cdots & | \end{bmatrix} \ .$$

Then, the solution is given by

$$\bar{\mathbf{A}}' = (HH')^{-1}H\mathbf{Z}' \ .$$

Since we know the nominal frequency signature of K classes of targets, we choose the estimated frequency signature for target j by

$$\hat{A}^j = \arg_{\underline{B} \in \{\underline{C}^k\}_{k=1}^K} \min \ d(\underline{B}, \bar{A}^j)$$

where $\bar{A}^j$ is the jth column of $\bar{\mathbf{A}}$.

3.4.5 Classification of Metallic Targets

For the classification of metallic targets, we use information about target positions to determine when tracks should be updated for their likelihood of corresponding to a metallic target. Without exploiting information about the position of the target, updating this likelihood is difficult, if not impossible.

In the magnetometer measurement model (Eqn. 2.7) when there is exactly one target within range of the magnetometer which is metallic, then with probability $1-p$ the sensor observes a 1, and with probability p the sensor observes a 0. On the other hand, if the target is not metallic, then with probability p the sensor observes a 0, and with probability $1-p$ the sensor observes a 1. Thus, p represents the probability that noise perturbs the measurement observed by the magnetometer sensor node.

In the presence of multiple targets, the sensor measurement is the maximum of the individual contributions from each target as in Eqn. 2.8. Thus, when there are exactly m targets within range of a magnetometer sensor node, the only information we can gain about the targets is regarding the number l of the m targets that are metallic. No information is gained that can determine which *particular subset* of those m targets is metallic or not. Let us denote the observation of the magnetometer sensor at time t by the random variable $Z(t)$. Given that $0 \leq l \leq m$ of these targets are metallic, then the likelihood that the magnetometer observes $Z(t) = z$ is given by

$$g_m(z|l) = \begin{cases} 1 - p^l(1-p)^{m-l} & , \ z = 1 \\ p^l(1-p)^{m-l} & , \ z = 0 \end{cases}$$

where

$$p^l(1-p)^{m-l}$$

represents the probability that each of the l metallic targets return an individual contribution of 0 and that each of the $m-l$ non-metallic targets return an individual contribution of 0. To expand to the case when the magnetometer observes a sequence of measurements $\{z(\tau)\}_{\tau=t}^{t+k-1}$ when the same m targets are within range, the likelihood of observing this sequence of measurements given that l of the m targets are metallic is given by the product

$$\prod_{\tau=t}^{t+k-1} g_m(z(\tau)|l)$$

since we assume that the noise is independent.

Let us represent whether target j is metallic or not by a binary random variable $W_j \in \{0, 1\}$, where $W_j = 0$ means target j is non-metallic and $W_j = 1$ means target j is metallic. Collecting these random variables into a vector, we have

$$\underline{W} = \begin{bmatrix} W_1 \\ \vdots \\ W_m \end{bmatrix}$$

where $\underline{W}$ is a random vector representing the possible configuration of the m targets being metallic or not. Then, the likelihood that the m targets are configured according to $\underline{w}$ is given by

$$f_m(z|\underline{w}) = g_m\left(z \mid \sum_{i=1}^{m} w_i\right) .$$

When we have a sequence of measurements $\{z(\tau)\}_{\tau=t}^{t+k-1}$, the likelihood of the configuration $\underline{w}$ is given by

$$\prod_{\tau=t}^{t+k-1} f_m(z(\tau)|\underline{w})$$

since we assume the noise terms in the model are independent.

Let us now generalize the likelihood function to the case when targets move in and out of range of the magnetometer sensor. Say, that there are a total of m targets each given a distinct label $j \in \{1, \ldots, m\}$ that move in and out of range of the magnetometer. At each time t, we will denote by $S(t) \subset \{1, \ldots, m\}$, the subset of targets that are within range of the magnetometer. Then, the posterior distribution of $\underline{W}$ given the observations $\{z(\tau)\}_{\tau=0}^{t}$ from time 0 to t is given by

$$\tilde{P}_t(\underline{w}|\{z(\tau)\}_{\tau=0}^{t}) = \alpha P_0(\underline{w}) \prod_{\tau=0}^{t} f_{|S(\tau)|}(z(\tau)|\underline{w}_{S(\tau)})$$

where α represents a normalization constant, P_0 represents the prior distribution of the possible configurations $\underline{W}$, and if $S = \{j_1, \ldots, j_n\}$, then

$$\underline{w}_S = \begin{bmatrix} w_{j_1} \\ \vdots \\ w_{j_n} \end{bmatrix} .$$

Since $S(\tau) = \emptyset$ indicates that no targets in range, the measurement $z(\tau)$ should not affect the posterior probability of the configuration $\underline{W}$. Thus, we use the convention that if $S(\tau) = \emptyset$, then

$$f_{|S(\tau)|} = 1 .$$

The expression $\tilde{P}_t$ holds because all noise terms are considered to be independent.

Because of the independence assumption of the noise terms in the magnetometer measurement model, we have a recursive equation for updating $\tilde{P}_{t-1}$ to $\tilde{P}_t$ given the new sensor measurement $z(t)$ and the subset of targets $S(t)$ within range,

$$\tilde{P}_t(\underline{w}|\{z(\tau)\}_{\tau=0}^{t}) = \alpha f_{|S(t)|}(z(t)|\underline{w}_{S(t)})\tilde{P}_{t-1}(\underline{w}|\{z(\tau)\}_{\tau=0}^{t-1} .$$

The information we wish to gain from $\tilde{P}_t$ are the marginals

$$\tilde{P}_t^j(w_j|\{z(\tau)\}_{\tau=0}^{t}) = \sum_{\substack{w_i \in \{0,1\}: \\ i \in \{1,\ldots,m\} - \{j\}}} \tilde{P}_t(\underline{w}|\{z(\tau)\}_{\tau=0}^{t}) .$$

because $\tilde{P}_t^j$ is the posterior distribution of whether target j is metallic or not.

As mentioned in the formulation of this problem in Section 2.5.2, the difficulty in computing $\tilde{P}_t^j$ over time is that there is no good way of representing the joint distribution $\tilde{P}_t$ in terms of the marginals $\{\tilde{P}_t^j\}_{j=1}^m$. For example, approximating by the product of marginals

$$\tilde{P}_t \approx \prod_{j=1}^m \tilde{P}_t^j$$

is a very poor approximation. Thus, finding an efficient way to incorporate the information provided by the magnetometer measurements so that the marginals $\{\tilde{P}_t^j\}_{j=1}^m$ are computed approximately correctly is needed. We leave this problem open.

However, there exists a special scenario when we can represent $\tilde{P}_t$ in terms of the marginals $\tilde{P}_t^j$. This scenario is exactly when we only choose to incorporate those observations of the magnetometer for which there exists exactly a single target within range of the magnetometer. To formalize, let

$$T = \{t \in 0, 1, \ldots : |S(t)| = 1\}$$

be the set of times for which there is exactly a single target within range of the magnetometer. For all $j \in \{1, \ldots, m\}$ define

$$T_j = \{t \in T : S(t) = \{j\}\}$$

be the subset of times when target j is the only target within range of the magnetometer. Then, it is true that

$$\tilde{P}_t(\underline{w}|\{z(\tau)\}_{\mathbb{Z}^+\cap T}) = \prod_{j=1}^m \tilde{P}_t^j(w_j|\{z(\tau)\}_{\mathbb{Z}^+\cap T_j})$$

where $\mathbb{Z}^+$ represents the set of non-negative integers. Thus, we only need to keep the posterior marginal distributions $\tilde{P}_t^j$. Furthermore, when target j is the only target within range of the magnetometer, only the posterior marginal $\tilde{P}_t^j$ needs to be updated.

The above scheme throws away a lot of information from the magnetometer which is undesirable especially because gaining information about whether a target is metallic or not relies on the chance that the targets move into range of a magnetometer. Although more sophisticated approximation schemes of $\tilde{P}_t$ are needed, another alternative is to deploy many magnetometer sensors densely throughout the sensor network so that the special case when exactly a single target is within range of a magnetometer is more likely to occur.

3.5 Integration of Individual Tasks with Triggers

The five individual tasks described in Section 3.4 can now be combined together so that the result is an algorithm which tracks and classifies targets within the field

which was pictured in Figure 3-2. We will specify the triggers which will initiate the different tasks as indicated by the gray boxes within the tasks of Figure 3-2.

In designing triggers for the different tasks, it is important to take into consideration the following aspects of the task.

1. What information is assumed to be available by the task?
2. What information does the task provide?
3. When is the information extracted from the task needed?

Thus, we should design triggers so that the tasks are initiated only when the appropriate information is available and the information provided by the computation of the task is needed.

In each of the subsections below, we will answer the above three questions for the respective tasks and propose a reasonable trigger for initiating this task.

3.5.1 Track Initiation Triggers

Aspects

1. Track initiation assumes the availability of intensity and TDOA measurements and estimates of the current locations of targets in the field.
2. The track initiation task returns a new track which corresponds to a new target entering the field. Specifically, an estimate of the new locations of the new target is given.
3. Since targets may enter the field at any time, track initiation must be running at all times.

Proposed Trigger

Track initiation routines must always be run. Each sensor node must subtract out the effect of individual contributions of all targets within range of it and run its communication algorithm at all times. Because the track initiation routine must always be running, it is very important that the DCA for target localization is efficient. The subject of Chapter 4 is devoted to constructing efficient distributed communication algorithms which perform target localization.

3.5.2 Track Updating

Aspects

1. Track updating assumes the availability of an estimate of the track position from the previous time.

2. The result of track updating is to update the active tracks in the hypothesis of the previous time step to the current time step.

3. The track updating task must be run at all times for all currently active tracks.

Proposed Trigger

Track updating routines must be run for all active tracks so that the trigger for updating a track T is

1. Track T is an active track.

Once tracks are terminated, they will no longer be updated since these tracks are no longer active.

The track updating task has the "Localization for a Specific Track" task as a subtask. Since any sensor node that can observe intensity measurements can also observe frequency bandlimited intensity measurements, there is a choice in the track updating task whether to use this frequency information. Assuming that observing frequency bandlimited intensity measurements is more expensive than observing intensity measurements, we would like to minimize how often frequency information is used. Thus, the only time when frequency information is needed is when tracks are about to cross and their frequency signatures are different. Then, this extra frequency information is discriminatory to solve the data association problem. Thus, the trigger for using frequency bandlimited intensity measurements in the localization for an active track T is if

1. the frequency signature for track T is known and

2. there exists another active track T' which is close to T. That is,

$$d(x^T, x^{T'}) \leq r$$

   where $d(\cdot, \cdot)$ is Euclidean distance, x^T and $x^{T'}$ are the current positions of tracks T and T' respectively, and $r > 0$ is a chosen radius which indicates when targets are considered close enough that discriminatory frequency information is needed.

This decision of whether to use the frequency signature information of track T is passed to the subtask which localizes for a specific track.

3.5.3 Localization for a Specific Track

Aspects

1. Localization for a specific track assumes the availability of the prior density of the individual contribution from the track to all sensor measurements.

2. The result of this task is an estimate of the likelihood of the position of a specific track based on the observations of sensor nodes which are within range of the track.

3. Localization for a specific track is needed in order to update active tracks.

Proposed Trigger

Localizing for a specific track occurs any time a track is to be updated. Thus, this task is triggered by a track update task. Whether to use frequency information for localization of a specific track is passed down from the track update task.

3.5.4 Classification of Frequency Signatures

Aspects

1. To classify frequency signatures of tracks, we assume the existence of several sensors which can observe frequency bandlimited intensity measurements within range of the tracks and the current estimate of the positions of these tracks.

2. This task assigns a frequency signature to the tracks.

3. The frequency signature of a track is needed for handling data association if the tracks that are crossing have distinct frequency signatures. Furthermore, this information can be used for other classification tasks.

Proposed Trigger

The trigger for classifying the frequency signature of a track must first ensure that the position of the track is known with reasonable accuracy and that there exist sensors which can observe frequency bandlimited intensity measurements. Thus, a simple trigger for classifying the frequency signature of a track T is the following.

1. There exist k sensor nodes which can observe frequency bandlimited intensity measurements within range of the track T.

2. Track T is an active track, and the length of track T is greater than or equal to some integer L. The *length* of an active track T is defined as

$$t - T_{start}$$

where t is the current time and T_{start} is the start time of track T.

3. The frequency signature of track T is unknown.

The first condition of the trigger ensures that there exists enough data so that the estimate of the frequency signature for track T is reasonably accurate. When k is large, we have more independent observations of the frequency content of the track which means the estimate of the frequency signature is better. The second condition of the trigger ensures that the estimate of the position of the track is of reasonable quality. The idea is that if a track has been active for a long time, then it is more likely that the estimated track position corresponds to the true position of the target. Finally, we only wish to classify the frequency signature of a track if it has not already been determined.

Thus, the trigger above can considered to be that classification of the frequency signature of a track is initiated as soon as there is enough information available to make a reasonable quality estimate of the frequency signature. It may seem that we should add another condition which only allows classification when tracks are about to cross. However, we choose the above trigger because of the possibility that, when tracks are about to cross, there may not be enough information to make a reasonable quality estimate of the frequency signature. Thus, we take advantage of information when it is available.

3.5.5 Classification of Metallic Targets

Aspects

1. To classify whether a track is metallic, we assume that the current track position is within range of a magnetometer sensor.

2. This task provides information about whether a track corresponds to a metallic target.

3. This information about whether a target is metallic is useful for the problem of data association when it is the only discriminating information. Furthermore, this information can be used for other classification tasks.

Proposed Trigger

Since we will not deal with the complicated case when there are several targets within range of a magnetometer sensor, the proposed trigger is the following.

1. There exists exactly one track T whose position is within range of the magnetometer sensor.

Then, the reading from the magnetometer sensor can be used to compute an estimate of the probability that the track corresponds to a metallic target.

3.6 Distributed Implementation of the Tracking Algorithm

In this section, we describe how to implement each of the individual information processing tasks in a distributed fashion in the sensor network. We must be sure that information that is needed for a particular task is available to the sensor node responsible for performing the task. To efficiently use resources, we also desire that the amount of communication occurring is minimized.

Each track T in the current hypothesis H_t is known by a single sensor node of the sensor network which we refer to as the *track holder* and which we denote by $\mathbf{id}^T(t) \in \{1, \ldots, n\}$. The explicit dependence on time of the track holder of T is shown because it will be possible and necessary for the track holder to change. It will be useful to take the track's point of view of the sensor network. If the track holder is currently $\mathbf{id}^T(t)$, then the only sensor nodes whose measurements can be accessed to update the track T by a single one-hop communication is the set of the neighbors of $\mathbf{id}^T(t)$. For each sensor node $i \in \{1, \ldots, n\}$, we denote by $\mathcal{N}(i) \subset \{1, \ldots, n\}$ the set of neighbor nodes of i.

All targets contribute to the measurement of sensor nodes within a certain radius $r > 0$ from their position.[3] If the position of the target is currently x, the set of sensor nodes which are affected by the target are

$$\mathcal{M}(x) = \{i \in \{1, \ldots, n\} : d(x, x_i) \leq r\}$$

where x_i is the position of sensor node i and $d(\cdot, \cdot)$ is the Euclidean distance. Thus, if the current track holder $\mathbf{id}^T(t)$ of track T is always within a distance r from the target's position as given by the current track position $x^T(t)$, as long as each sensor node knows all neighboring nodes within a radius of $2r$, then all sensor nodes $\mathcal{M}(x^T(t))$ whose measurements are affected by the target will be neighbors of the track holder. Formally, this means that

$$\mathcal{M}(x^T(t)) \subset \mathcal{N}(\mathbf{id}^T(t)) \cup \{\mathbf{id}^T(t)\} \ .$$

Therefore, we will assume that the set of neighbors of each sensor node i is given by

$$\mathcal{N}(i) = \{j \in \{1, \ldots, n\} - \{i\} : d(x_i, x_j) \leq 2r\}$$

where x_i is the position of sensor node i for all $i \in \{1, \ldots, n\}$. We assume that from the configuration phase that every sensor node i knows the sensor measurement

[3] In reality, the radius r can depend on the type of sensor so that we have a set of radii $\{r_i\}_{i=1}^l$ for l different types of sensing modalities. We can extend the discussion of all of the following subsections to such a case. However, to keep the discussion simple we will present the distributed implementation of the various individual tasks assuming a single r. In fact, choosing $r = \max_{i \in \{1, \ldots, l\}} r_i$ can work for the following subsections although there are more communications occurring than necessary for certain tasks.

model of itself as well as its neighbors. Knowledge of the measurement models of the neighbors will be important in order to distribute computations. With the above assumptions about the neighboring nodes of any sensor node, we can assume that the track holder $\mathbf{id}^T(t)$ knows the measurement models of all sensors $\mathcal{M}(x^T(t))$ which are affected by the track.

Note that this representation of the current hypothesis H_t is a distributed representation of the set of tracks known globally by the sensor network. The idea is that the track holder is the sensor node whose neighbors contain the most relevant information needed to update the track. This is efficient from the point of view of incorporating information from sensor nodes to update each track in the hypothesis; however, a negative side effect of this distributed representation is the necessity for an information retrieval task to pull out this distributed representation of the hypothesis to some central receiver. We leave this problem of retrieving information distributed throughout a sensor network open in this thesis. However, there is much work on this problem in the literature such as LEACH [9], GEAR [28], CADR [7], and directed data diffusion [12].

The following subsections will describe how to implement the individual tasks discussed in Section 3.4 in a distributed manner. We refer to the description and notation of the respective subsections in Section 3.4 in the following.

3.6.1 Track Initiation

For the case when there are no targets currently in the field, there are no extra communications necessary because the DCA's for target localization will initiate the proper communications in order to collect the needed information for computing an estimate of reasonable quality.

For the case when there are targets currently in the field for which we have an associated active track T in the current hypothesis H_t, in order to perform track initiation, the individual contribution of each active track must be subtracted out of the appropriate sensor measurements. What is needed by each sensor node i is the mean and variance, $\mu_{i,j}$ and $\sigma^2_{i,j}$, of the density P_i^j of the individual contribution Z_i^j from a target j which is within range of sensor node i. Thus, if the track T is associated with target j, the track holder $\mathbf{id}^T(t)$ performs the following.

1. Compute $\mu_{i,j}$ and $\sigma^2_{i,j}$ for those sensor nodes $i \in \mathcal{M}(x^T(t))$.
2. Then, the track holder sends $\mu_{i,j}$ and $\sigma^2_{i,j}$ to sensor node i for each $i \in \mathcal{M}(x^T(t))$.

Note that the track holder has also computed the individual contribution of the track T to its own sensor measurement.

If all track holders of tracks in the current hypothesis perform the above, the result is that every sensor node of the sensor network will have received the mean and variance of the individual contribution from all tracks which are within range. Then, each sensor node performs the following.

1. Subtract out the effect of all known tracks within range.

2. Run the DCA's and initiate a new track if a sensor node declares it has computed the global localization task.

3.6.2 Track Updating

In track updating, we are updating a track T from the previous time $t-1$ whose track holder is $\mathbf{id}^T(t-1)$ to the current time t. The update of track T is performed by the track holder $\mathbf{id}^T(t-1)$. Let T be the track associated with target j.

The first step the track holder must perform is to determine whether frequency signature information should be used for updating the track by testing whether a track is nearby. This can be implemented by a broadcast message of the track's current mean position by the track holder to all sensor nodes within range of the track. If the track holder receives a message from another track holder and it is determined that the remote track's mean position is near to its track's mean position, frequency signature information is primed to be used by the localization for a specific target subtask.

Then, for track continuation, the following steps are performed by the track holder.

1. The track holder propagates the previous time density $\tilde{p}^j_{t-1}$ of the position of the track by the motion model q^j which results in the predicted density p^j of the position of target j.

2. This predicted density is passed to the **Localization for a Specific Target** subtask in the track holder. We defer discussion of the specific localization subtask for the next section (Section 3.6.3).

After the localization subtask has been completed, the track holder will have knowledge of the likelihood L^j of target j. At this point, all the information needed to perform the rest of track continuation, termination, and creation of adjoinable track hypotheses can be performed by the track holder $\mathbf{id}^T(t-1)$.

As for resolving adjoinable track hypotheses, the track holder $\mathbf{id}^T(t-1)$ performs the following steps if the track T is associated with an adjoinable track hypothesis containing tracks which correspond to both metallic and non-metallic targets.

1. The track holder determines whether track T is within range of a magnetometer sensor i by testing
$$d(x^T(t), x_i) \leq r'$$
where $r' > 0$ is the range of the magnetometer sensor.

2. Then, the track holder queries for the magnetometer measurement of sensor node i.

Then, the magnetometer sensor node i performs the following steps.

1. Determine whether the track T is the only one within range.

2. If it is, then responds back to the querying track holder with the observation from the magnetometer.

3. If not, the response is a message stating that the track is not the only within range.

Then, the track holder finishes with the following steps.

1. If the response by the magnetometer sensor i was a measurement, then the track holder updates the adjoinable track hypothesis of T.

2. If the resulting adjoinable track hypothesis of T contains only one track, say R, then T is replaced with the adjoined track $R \circ T$. Then, a message is broadcast to the sensor network that R should be removed from all adjoinable track hypothesis.

Other track holders can respond to this broadcast message appropriately.

The one other thing we must do is to update the track holder $\mathbf{id}^T(t)$ of the current time since we must ensure that our assumption that the track holder of T is always within a radius r from the current position $x^T(t)$ of the track. Thus, it may be necessary to switch track holders from time $t-1$ to t. The scheme we adopt is to make the sensor node which is closest to the current position $x^T(t)$ of the track to be the track holder. The previous track holder computes

$$\mathbf{id}^T(t) = \arg_{j \in \mathcal{N}(\mathbf{id}^T(t-1))} d(x_j, x^T(t))$$

where x_j is the position of sensor node j. If $\mathbf{id}^T(t)$ is different from $\mathbf{id}^T(t-1)$, then the previous track holder must send all information about track T to sensor node $\mathbf{id}^T(t)$. The information about track T not only includes the history of position and error covariances of the track but also includes other information such as the nominal intensity, the frequency signature, whether the target is metallic, and possibly any attached adjoinable track hypotheses. After sending this information about T to $\mathbf{id}^T(t)$, the old track holder $\mathbf{id}^T(t-1)$ erases from memory all knowledge about the track T.

The above exchange of track holders for a track T only needs to be performed if track T is an active track. A terminated track no longer needs to be updated so that it will forever remain in the last track holder before it was terminated.

3.6.3 Localization for a Specific Target

This task is a subtask of track continuation in the track update task. From track continuation, the task of localizing for a specific target is passed the predicted density p^j of target j. p^j is known in the track holder $\mathbf{id}^T(t-1)$ of the track T which is associated with target j. We will describe how to compute the likelihood L^j for target j in a distributed manner.

The track holder $\mathbf{id}^T(t-1)$ performs the following steps.

1. The track holder computes the mean and variance of P_i^j, denoted by

$$\{\mu_{i,j}, \sigma_{i,j}^2\}_{i \in \mathcal{M}(x^T(t-1))} \ ,$$

   for the individual contribution of target j to all sensor nodes within range of the target.

2. Then, the track holder sends these prior means and variances to the respective sensor nodes in $\mathcal{M}(x^T(t-1))$.

Then, the sensor nodes perform the following steps.

1. Based on the measurement observed by these sensor nodes and the set of prior means and variances received by the sensor about targets within range, the posterior means and variances, $\tilde{\mu}_{i,j}$ and $\tilde{\sigma}_{i,j}^2$, are computed.

2. Then, these posterior means and variances are sent back to the appropriate track holders.

The track holder $\mathbf{id}^T(t-1)$ now has knowledge of the posterior means and variances of target j's individual contribution,

$$\{\tilde{\mu}_{i,j}, \tilde{\sigma}_{i,j}^2\}_{i \in \mathcal{M}(x^T(t-1))} \ .$$

From this information, the track holder can now perform the following steps.

1. The track holder computes the likelihood L^j for target j.

2. This likelihood is returned to the track continuation task which called this specific localization subtask.

3.6.4 Classification of Frequency Signatures

For classifying the frequency signatures of targets, we will only allow the track holder for a track T which is associated with target j to estimate the frequency signature to be associated with T. Let us define the subset of targets within range of a sensor node i by

$$\mathcal{T}(i) = \{j \in \{1, \ldots, m\} : d(x^{T_j}(t), x_i) \leq r\}$$

where T_j is the track associated with target j.

As before, all track holders of active tracks in the hypothesis will compute the prior means and variances, $\mu_{i,j}$ and $\sigma^2_{i,j}$, and send these to the appropriate sensor node i. Thus, each sensor node i is aware of the prior means $\{\mu_{i,j}\}_{j\in\mathcal{T}(i)}$ and error covariances $\{\sigma^2_{i,j}\}_{j\in\mathcal{T}(i)}$.

The information needed to estimate the frequency signature for track T which is associated to a target j are the means $\{\mu_{i,j'}\}_{j'\in\mathcal{T}(i)}$ and frequency bandlimited intensity measurements $\underline{z}_i$ for all sensor nodes $i \in \mathcal{M}(x^T(t))$ which are within range of the track and which are frequency bandlimited intensity sensors. Let us assume that all sensor nodes in $\mathcal{M}(x^T(t))$ are frequency bandlimited intensity sensors. For the case when they are not, we can just restrict the set $\mathcal{M}(x^T(t))$ to those that are. Thus, the track holder performs the following steps.

1. The track holder $\mathbf{id}^T(t)$ queries all sensor nodes $i \in \mathcal{M}(x^T(t))$ for their prior means $\{\mu_{i,j'}\}_{j'\in\mathcal{T}(i)}$ and frequency bandlimited intensity measurements $\underline{z}_i$ at the current time t.

2. With this information, the track holder computes an estimate of the frequency signature for target j. We will elaborate on this below.

To elaborate on how to compute an estimate of the frequency signature, we will construct the matrices H and $\mathbf{Z}$ from Section 3.4.4. Let us represent the set of sensor nodes within range of target j by

$$\mathcal{M}(x^T(t)) = \{i_1, \ldots, i_l\} \ .$$

The set of all targets within range of any sensor node in $\mathcal{M}(x^T(t))$ is

$$\bigcup_{i\in\mathcal{M}(x^T(t))} \mathcal{T}(i) = \{j_1, \ldots, j_k\} \ .$$

Then, H is a $k \times l$ matrix

$$H = [h_{c,d}]$$

where the component of the cth row and d^{th} column is given by

$$h_{c,d} = \begin{cases} \mu_{i_d,j_c} & , \ j_c \in \mathcal{T}(i_d) \\ 0 & , \ \text{otherwise} \end{cases} \ .$$

Essentially, if target j_c is not within range of sensor node i_d, then we assume that the mean contribution μ_{i_d,j_c} is zero.

The matrix $\mathbf{Z}$ is a $b \times l$ matrix where b is the number of frequency bands given by

$$\mathbf{Z} = \begin{bmatrix} | & \cdots & | \\ \underline{z}_{i_1} & \cdots & \underline{z}_{i_l} \\ | & \cdots & | \end{bmatrix} \ .$$

Then, $\bar{\mathbf{A}}$ can be computed. If $j_c = j$, we estimate the frequency signature for target j by

$$\hat{A}^j = \arg_{\underline{B} \in \{\underline{C}^k\}_{k=1}^K} \min \; d(\underline{B}, \bar{A}^{j_c})$$

where $\bar{A}^{j_c}$ is the j_c^{th} column of $\bar{\mathbf{A}}$.

3.6.5 Classification of Metallic Targets

For the classification of whether a target j is metallic, we consider classification only for the special case when there is exactly a single target within range of the magnetometer. We propose the following distributed implementation of this classification. Let the set of all magnetometer sensor nodes be denoted by

$$M = \{i \in \{1, \ldots, n\} : i \text{ is a magnetometer}\} \; .$$

For the track T associated with target j, the current track holder $\mathbf{id}^T(t)$ can determine whether there exists any magnetometer sensors M which are within range of the track

$$M^T(t) = \{i \in M : d(x^T(t), x_i) \leq r'\}$$

where x_i is the position of magnetometer sensor i and $r' > 0$ is the range of the magnetometer sensor. The track holder for track T performs the following steps.

1. Compute $M^T(t) \subset M$.
2. If $M^T(t)$ is non-empty, the track holder $\mathbf{id}^T(t)$ sends a message to the magnetometer sensors in $M^T(t)$ stating that track T is within range.

If all track holders send messages to magnetometer sensors within range of the track, each magnetometer sensor will have received a message from all tracks which are within range of the magnetometer. That is, for each magnetometer sensor $i \in M$, the set of tracks which are within range of magnetometer sensor i is given by

$$S_i(t) = \{T \in H_t : i \in M^T(t)\}$$

where H_t is the current hypothesis. Then, each magnetometer sensor $i \in M$ performs the following steps.

1. If $|S_i(t)|$ is equal to one, magnetometer sensor node i can send its observation back to the track holder of the track in $S_i(t)$.
2. If $|S_i(t)|$ is not equal to one, magnetometer sensor node i sends a message to all track holders of the tracks in $S_i(t)$ stating that there is no update for whether the track is metallic.

Then, each track holder performs the following step.

1. If a magnetometer sensor node responded with an observation, the track holder $\mathbf{id}^T(t)$ can update the distribution of whether the track T corresponds to a metallic target as presented in Section 3.4.5.

3.7 Simulations

In this section, we demonstrate several simulations of the resulting distributed inference system implementing localization, tracking and classification inference tasks described in this chapter and pictured in Figure 3-2. The only information processing task we have not described in this chapter is the "Localization for Track Initiation" task which is implemented by DCA's to be presented in Chapter 4. Let us consider this task a black box for the simulations shown in this section. We split the set of simulations into two parts:

1. a set of simulations demonstrating the intended nominal behavior of the inference system and

2. a set of simulations demonstrating the limits of the proposed inference system.

All simulations to be presented use simulated data based on the sensor measurement models described in Section 2.2. Let us describe the specific parameters of targets and sensor measurement models used to simulate observations measured by sensor nodes. We assume that we have three kinds of sensors: (1) intensity sensors, which can measure both intensity measurements and frequency bandlimited intensity measurements; (2) TDOA sensors, which can only measure TDOA measurements; and (3) magnetometer sensors, which can only measure whether a target is metallic.

3.7.1 Nominal Measurement Model Parameters

Target Parameters We assume that all targets have the same nominal intensity,

$$A_{tot} = 10 \ .$$

Furthermore, we will use three frequency bands so that the frequency signature of any target is a three component vector

$$\underline{A} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

where, to be consistent with the nominal intensity A_{tot}, the frequency components satisfy

$$a_1 + a_2 + a_3 = A_{tot}$$

and each a_i is non-negative. Furthermore, each target is known to be metallic or not which we indicate by $W \in \{0, 1\}$, where $W = 1$ means the target is metallic and $W = 0$ means the target is not metallic. The number of particles used to represent the density of the current position of the track is 50. The motion model we use for tracks is a simple Gaussian diffusion,

$$x(t+1) = x(t) + N(t)$$

where $N(t)$ is a bivariate zero mean Gaussian with diagonal covariance

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix}$$

where we choose $\sigma = 4$.

Intensity Measurements Given the nominal intensity A_{tot} of a target, its current position $x(t) \in \mathbb{R}^2$, and the position of the intensity sensor $x_i \in \mathbb{R}^2$, the simulated measurement of this intensity sensor if this is the only target is given by Eqn. 2.3, reprinted here,

$$z_i(t) = A_{tot} f(d(x_i, x(t))^\alpha) + n_i(t)$$

where f was given in Eqn. 2.4, $d(\cdot,\cdot)$ is Euclidean distance, and $n_i(t)$ is a sample from a Gaussian distribution with zero mean and variance σ_i^2. We choose the following parameter values for all intensity measurements.

$$\begin{aligned} \alpha &= 2 \\ \sigma_i &= .5 \end{aligned}$$

For multiple targets located at $\{x^j(t)\}_{j=1}^m$, the simulated intensity measurement is the sum of the contributions from each given by

$$z_i(t) = \sum_{j=1}^{m} A_{tot} f(d(x_i, x^j(t))^\alpha) + n_i(t) \ .$$

Frequency Bandlimited Intensity Measurements Given the frequency signature $\underline{A}$ of a target, its current position $x(t) \in \mathbb{R}^2$, and the position of the intensity sensor $x_i \in \mathbb{R}^2$, the simulated frequency bandlimited intensity measurement of this sensor is given by Eqn. 2.5, reprinted here in vector form,

$$\underline{z}_i(t) = \underline{A} f(d(x_i, x(t))^\alpha) + \underline{n}_i(t)$$

where all vectors consist of three components, f is given in Eqn. 2.4, $d(\cdot,\cdot)$ is Euclidean distance, and the three components of $\underline{n}_i(t)$ are independent samples from a Gaussian distribution with zero mean and variance $\frac{\sigma_i^2}{3}$. We choose the following parameter values for frequency bandlimited intensity measurements.

$$\begin{aligned} \alpha &= 2 \\ \sigma_i &= .5 \end{aligned}$$

Note that the sum of the three components of the simulated measurement $\underline{z}_i(t)$ has the same statistics as the intensity measurement $z_i(t)$ which makes these frequency bandlimited intensity measurements a refinement of the intensity measurements. For multiple targets located at $\{x^j(t)\}_{j=1}^m$ with frequency signatures

$\{\underline{A}^j\}_{j=1}^m$, the simulated frequency bandlimited intensity measurement is the sum of the contributions from each given by

$$z_i(t) = \sum_{j=1}^{m} \underline{A}^j f(d(x_i, x^j(t))^\alpha) + \underline{n}_i(t) \ .$$

TDOA measurements Given the current position of the target $x(t) \in \mathbb{R}^2$ and the positions of the dipole of microphones of the TDOA sensor $x_{i,1}, x_{i,2} \in \mathbb{R}^2$, the simulated TDOA measurement is given by Eqn. 2.6, reprinted here in slightly different notation,

$$z_i(t) = \begin{cases} T_{i,1}(t) - T_{i,2}(t) & , \ \max_{j=1,2} d(x(t), x_{i,j}) \leq \gamma_{tdoa} \\ \emptyset & , \ \text{otherwise} \end{cases}$$

where

$$T_{i,j}(t) = \frac{d(x(t), x_{i,j})}{c} + \left(\frac{d(x(t), x_{i,j})}{c}\right)^{\frac{\beta}{2}} n_{i,j}(t)$$

and $d(\cdot,\cdot)$ is Euclidean distance and $n_{i,1}(t)$ and $n_{i,2}(t)$ are independent samples from a Gaussian distribution with zero mean and variance σ_i^2. We choose the following parameter values for TDOA measurements.

$$\begin{aligned} \gamma_{tdoa} &= 70 \\ c &= 1100 \\ \beta &= 1 \\ \sigma_i &= 5 \times 10^{-5} \\ d(x_{i,1}, x_{i,2}) &= \frac{1}{6} \end{aligned}$$

We choose the dipole of microphones to be a fixed distance apart and only allow their orientation to vary. For multiple targets, the simulated TDOA measurement is the union of the simulated TDOA measurements from each.

Magnetometer Measurements Given the current position of the target $x(t) \in \mathbb{R}^2$, an indication of whether the target is metallic $W \in \{0, 1\}$, and the position of the magnetometer sensor $x_i \in \mathbb{R}^2$, the simulated magnetometer measurement is given by Eqn. 2.7, reprinted here,

$$z_i(t) = \begin{cases} 1 - n_i(t) & , \ d(x_i, x(t)) \leq \gamma_{mag}, \ W = 1 \\ n_i(t) & , \ \text{otherwise} \end{cases}$$

where $n_i(t)$ is a Bernoulli random variable with probability p of equalling 1. We choose the following parameter values for magnetometer measurements.

$$\begin{aligned} p &= .1 \\ \gamma_{mag} &= 20 \end{aligned}$$

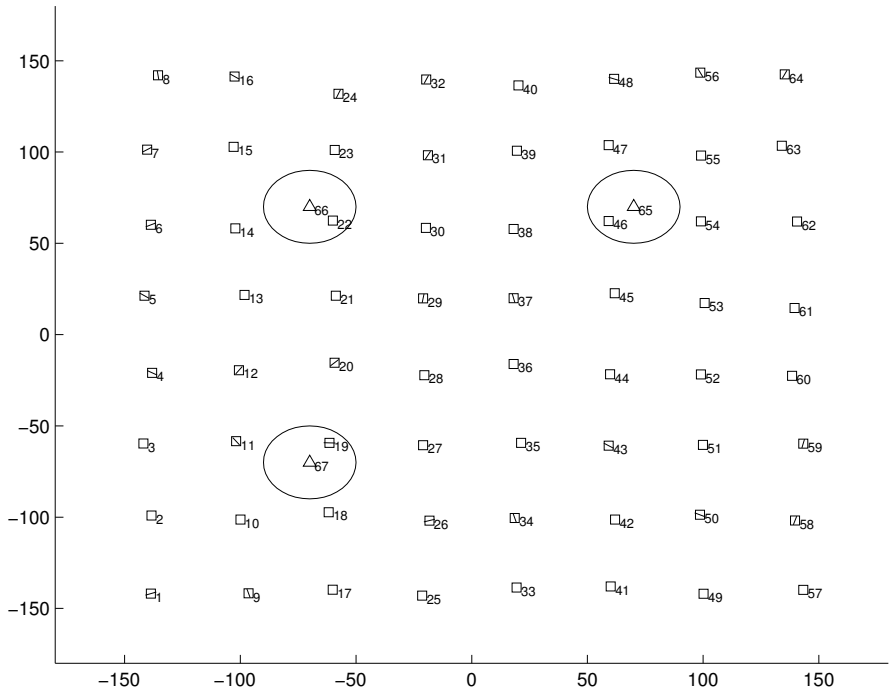

Figure 3-3: Example sensor network of 67 sensor nodes. Numbers next to sensor nodes indicate the ID number of the sensor node.

For multiple targets, the simulated magnetometer measurement is the maximum of the contributions from each target.

All simulations to be shown are runs of the inference system described in this chapter for different sensor network topologies, by which we refer to placement of sensor nodes and types of sensor nodes in the field, and different target trajectories with different parameters like frequency signature and being metallic.

3.7.2 Demonstrations of Nominal Behavior of the Inference System

The simulations that demonstrate desired nominal behavior of the inference system in this section will use a sensor network consisting of 64 intensity and TDOA sensors arranged in a square grid slightly perturbed in position with magnetometer sensors like the one shown in Figure 3-3. Intensity sensors are denoted by a square, TDOA sensors are denoted by a square with a line denoting the orientation of the dipole of microphone sensors, and magnetometer sensors are denoted by a triangle with a circle around it representing the range of the magnetometer sensor. In the following simulations, we will use a version of this sensor network by varying the type of sensor

at each sensor node as well as adding or taking away magnetometer sensors. Unlike the simulations to be presented in Section 3.7.3, we will only spot check the following simulations to determine whether they are running properly.

3.7.2.1 Distributed Multiple Target Tracking

In this section, we demonstrate the multiple target tracking abilities of the distributed inference system. We have 6 targets whose target trajectories $\{x^j(t)\}_{j=1}^6$ over time $t = 0, 1, \ldots$ are given by

$$\begin{aligned}
x^1(t) &= \begin{bmatrix} -150 \\ 150 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} t \\
x^2(t) &= \begin{bmatrix} -100 \\ 120 \end{bmatrix} + \begin{bmatrix} -.25 \\ 2 \end{bmatrix} t \\
x^3(t) &= \begin{bmatrix} 120 \\ 120 \end{bmatrix} + \begin{bmatrix} -2 \\ -.5 \end{bmatrix} t \\
x^4(t) &= \begin{bmatrix} 0 \\ -60 \end{bmatrix} + \begin{bmatrix} 2 \\ -1 \end{bmatrix} t \\
x^5(t) &= \begin{bmatrix} 110 \\ -50 \end{bmatrix} + \begin{bmatrix} 0 \\ 2 \end{bmatrix} t \\
x^6(t) &= \begin{bmatrix} -50 \\ -100 \end{bmatrix} + \begin{bmatrix} 1.5 \\ 1 \end{bmatrix} t
\end{aligned}$$

These target trajectories over time are shown in Figure 3-4. These targets are dense in the sense that sensors will measure contributions from several targets. The range of each intensity and TDOA sensor is 70 for the choice of parameters of the measurement models. Thus, when targets are within $2 \cdot 70$ distance from each other, the observations of some sensor nodes will be the combination of several target contributions. However, we have kept the targets spaced far enough apart so that no ambiguity in data association arises.

Figure 3-5 shows a run of the inference system with the 6 targets on a square grid of sensor nodes randomly chosen to be intensity and TDOA sensors. All targets have the same frequency signature, and all are metallic. The asterisks indicate the ground truth of the target location at the time of the snapshot. The lines with points correspond to the track history up to the current time. A particle set is shown which represents the density of the current track position. A circle of radius 70 is drawn around the mean of each particle set to indicate the set of sensor nodes which are considered to be within range of the track. Only the sensor nodes within these circles receive predicted contributions from the track holder, indicated by a line from the mean of the particle set to the track holder. Then, these sensor nodes will respond back to all track holders within range with an estimate of the posterior contribution from each.

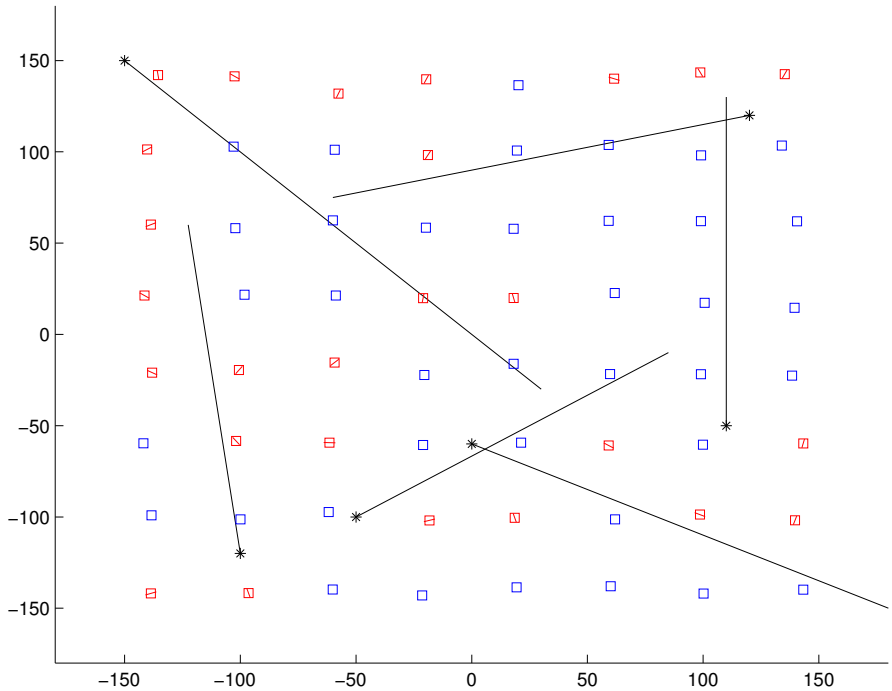

Figure 3-4: The trajectories of the 6 targets to demonstrate the distributed multiple target tracking. Initial target positions are shown with an asterisk, and the lines indicate the target position over time at constant velocity.

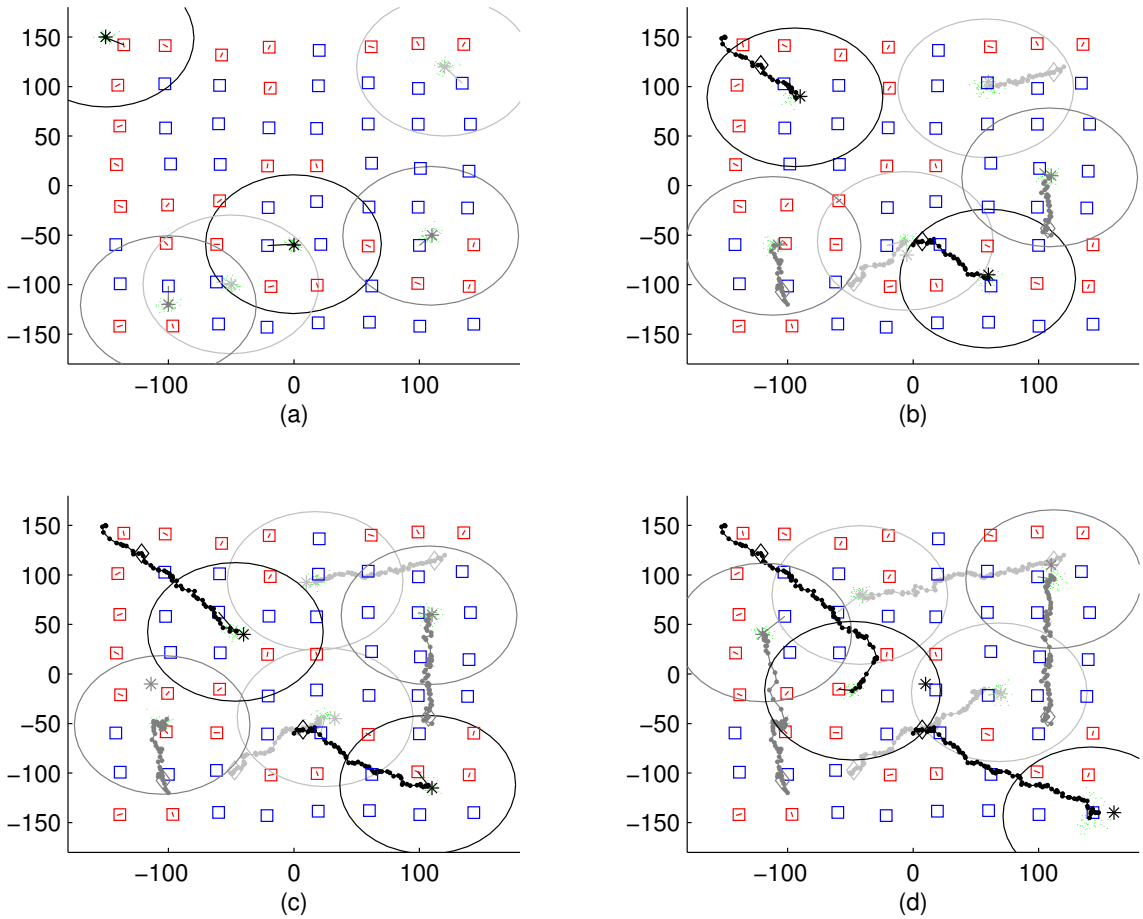

Figure 3-5: Snapshots of running the distributed multiple target tracking algorithm.

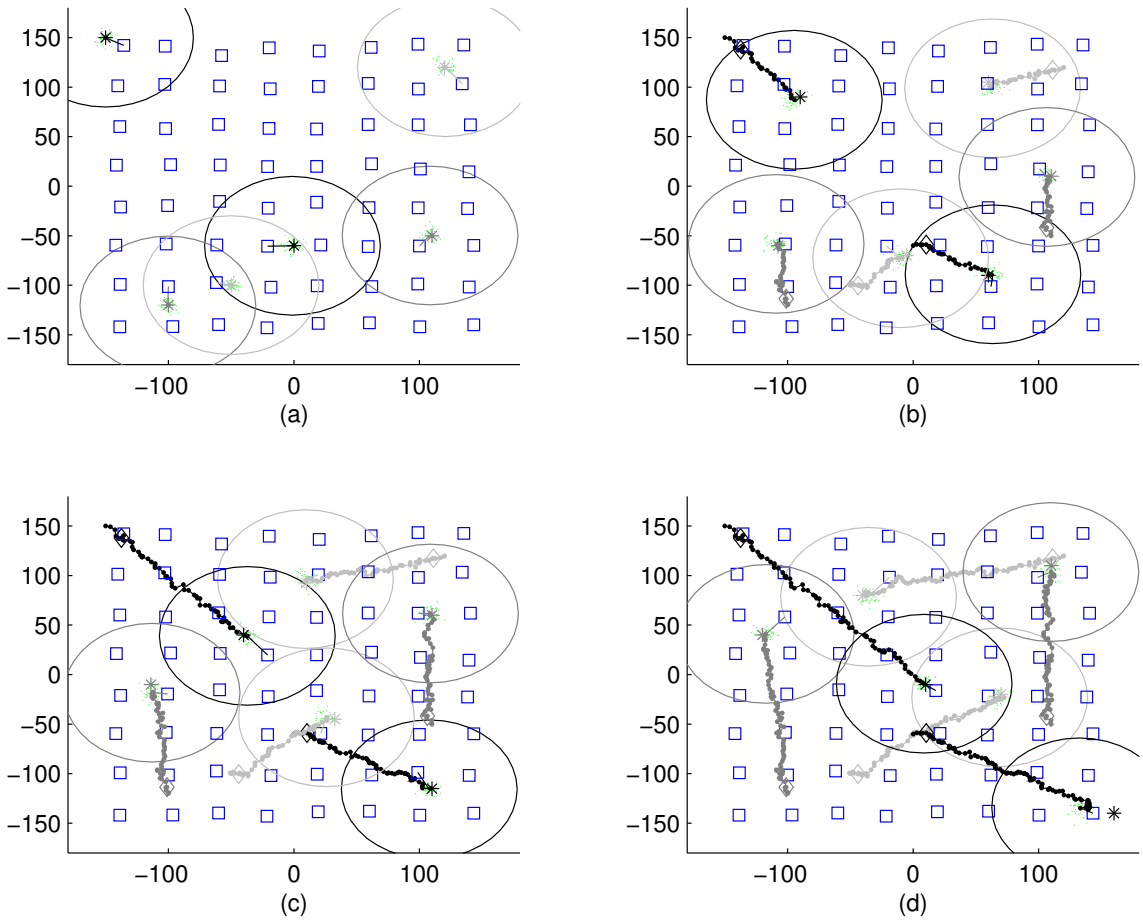

Figure 3-6: Snapshots of running the distributed multiple target tracking algorithm on a sensor network consisting of intensity sensors only.

Note that the track holders for each track change over time which is a distributed representation of the set of all tracks in the sensor network. Initially, we have artificially initiated tracks at the true position of the target at time $t = 0$ as shown in Figure 3-5a. After a short time, we see that the multiple targets are tracked as they travel through the sensor network as shown in Figure 3-5b. As time progresses, the gray track on the bottom left of Figure 3-5c begins to lose the target although it recovers as seen in Figure 3-5d. On the other hand in Figure 3-5d, the black track which began on the top left has lost the true location of the target, which is the black asterisk located at $(12, -12)$.

These losses of the true target position by the track is due to some peculiarities of the measurement model chosen for TDOA sensors. First of all, there is an inherent ambiguity about the axis of the dipole of sensors. Secondly, the likelihood near the

TDOA sensor is large only in a very thin slice (See the top right and bottom left plots in Figure 2-5). This is a problem because we are using particle sets to represent the density of a track position. When the true density of the track position is on top of the TDOA sensor, there is a very small probability that a particle of the particle set lies in the thin slice of high likelihood near the TDOA sensor. This means that those particles which are farther from the location of the TDOA sensor have a higher chance of receiving a larger weight so that the particle set is unnecessarily biased away from the TDOA sensor.

We show another simulation when all sensor nodes of the sensor network are intensity sensors as shown in Figure 3-6. In this simulation, the tracks follow the true positions of the targets better. Comparing Figure 3-6 and Figure 3-4, we see that the six tracks correspond well to the six targets.

3.7.2.2 Intelligence Updating

In this simulation, we will demonstrate the nominal behavior of tracking a single target through a sensor network where the classification of frequency signatures and classification for metal targets is performed when such information is available as described in the canonical scenario of Section 1.3.1. The specific sensor network we use is shown in Figure 3-7. The magnetometer sensors are shown as triangles and the circle around it represents the range of the magnetometer sensor. The 32 nodes arranged in a square grid on the left side of the figure are chosen to be TDOA sensors and the 32 nodes arranged in a square grid on the right side of the figure are chosen to be intensity sensors to demonstrate that the classification for frequency signature is performed when there are enough intensity sensors nearby. Furthermore, we have a magnetometer sensor on the top right so that, when the track is within range of the magnetometer, the distribution of whether the track corresponds to a metallic target is updated.

The classification of the frequency signature of a track is performed when there exist at least 3 intensity sensor nodes within range of the track and the track is at least of length 5. The classification for whether a track corresponds to a metal target is performed when 90% of the weight of the particles representing the current density of the position of the track is within range of the magnetometer. In this simulation, the target moves from position $(-125, -125)$ towards the magnetometer sensor located at $(70, 70)$ along a curved path. Its frequency signature is

$$\underline{A} = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}$$

and it is metallic. We have artificially initiated a track from the true location of the target in the beginning at $(-125, -125)$. Figure 3-8a shows the initial part of the track. Since there are no intensity sensors within range of the track, as indicated by

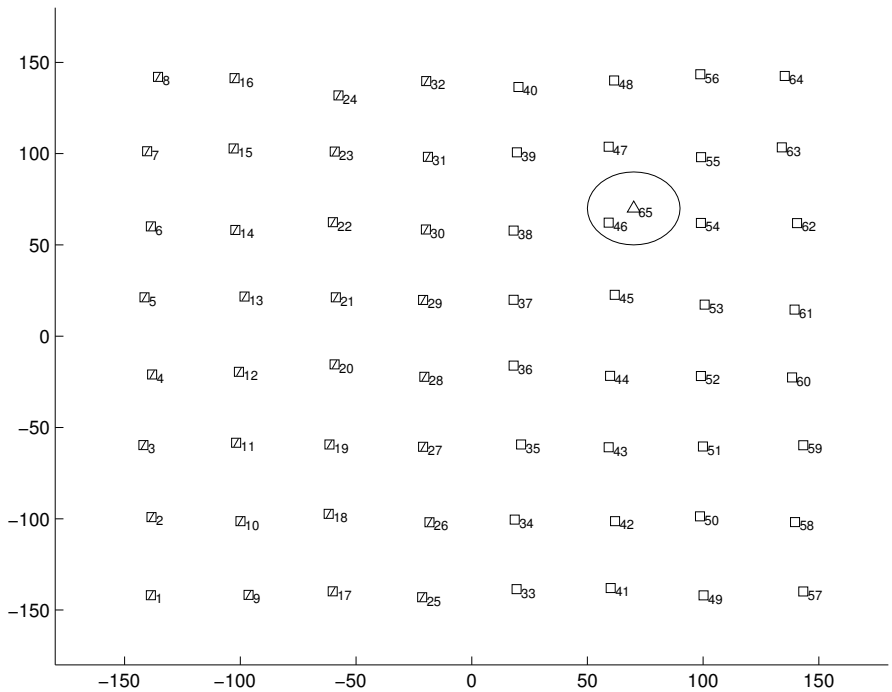

Figure 3-7: Sensor network to demonstrate the canonical scenario of intelligence updating. Numbers next to sensor nodes indicate the ID number of the sensor node.

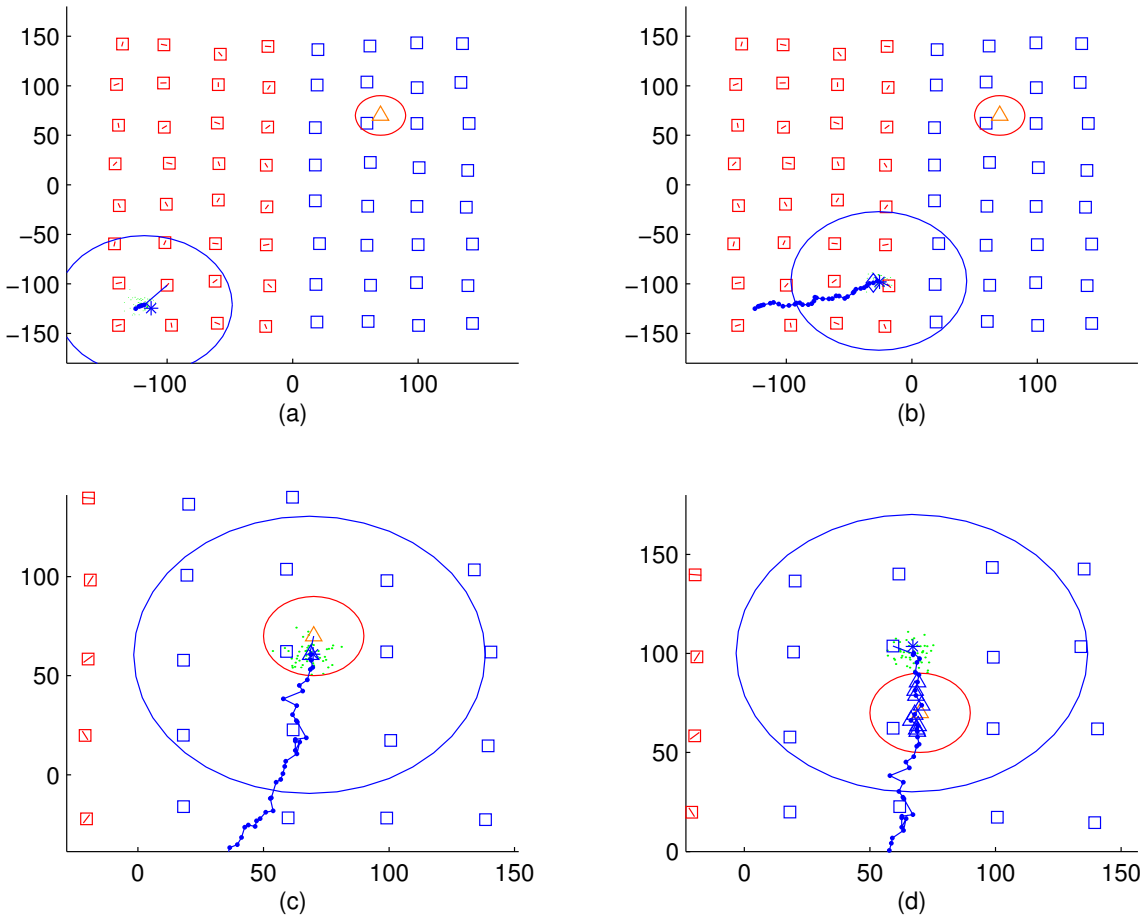

Figure 3-8: Snapshots of demonstrating classification occurring when information is available.

the circle around the mean of the particles, classification of the frequency signature is not initiated. Figure 3-8b shows when the track has come in range of 3 intensity sensors. Since the track has not yet been classified for its frequency signature, the inference system initiates the classification routine and determines that the frequency signature of the track is

$$\begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix}$$

which is indicated by a diamond along the track history. Now that the frequency signature of this track has been classified, the task of classifying for the frequency signature of this track will not be initiated again which is indicated by the absence of a diamond along the track history of subsequent position estimates. Figure 3-8c shows when the track has entered the range of the magnetometer sensor. At this point, classifying for whether this track corresponds to a metallic target is run. We denote this along the track history with a triangle. As long as the track is within range of the magnetometer, this classification routine is run. Figure 3-8d shows when the track has gone out of range of the magnetometer sensor. The classification for metal targets is no longer run when this occurs. The result of the classification for metal targets is that it is metallic with probability $1 - \epsilon$ where $\epsilon = 2.09 \times 10^{-7}$.

3.7.2.3 Data Association: No Discriminatory Information

In this simulation, we demonstrate that if targets come near each other and there is no discriminatory information to disambiguate the two targets so that we encounter a data association problem, our solution is to create an adjoinable track hypothesis by terminating the tracks and beginning new ones. We use the sensor network consisting only of intensity sensors as shown in Figure 3-9. We have two targets $j \in \{1, 2\}$ with the same frequency signature

$$\underline{A}^1 = \underline{A}^2$$

with positions over time $t = 0, 1, \ldots$ given by

$$x^1(t) = \begin{bmatrix} -90 \\ 90 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} t \tag{3.17}$$

$$x^2(t) = \begin{bmatrix} -90 \\ -90 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} t \tag{3.18}$$

Target 1 travels at constant velocity from the top left of Figure 3-9a to the bottom right of Figure 3-9d. Target 2 travels at constant velocity from the bottom left of Figure 3-9a to the top right of Figure 3-9d.

Figure 3-9a shows the initial trajectories of each track which have been artificially initialized at the true position of each target at time $t = 0$. After the track has progressed for a little while, both tracks are classified for their frequency signature.

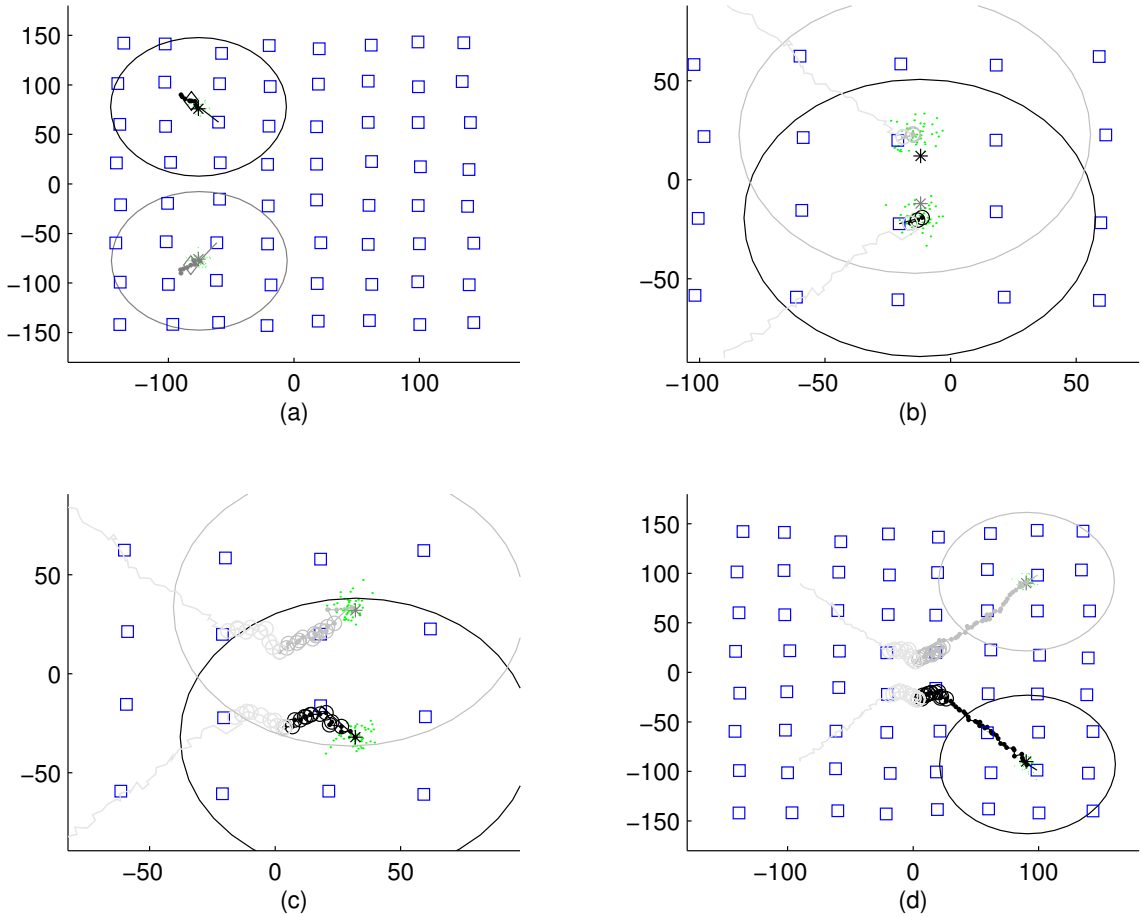

Figure 3-9: Snapshots of demonstrating handling of data association problem when no discriminatory information is available.

In this case, we have chosen the frequency signature of both targets to be the same so that the tracks are classified with the same frequency signature.

As the tracks near each other, when the means of the particle sets representing the density of the each track's position are closer than a distance of 50, then since the two tracks have the same frequency signature, an adjoinable track hypothesis is formed which is shown implicitly in Figure 3-9b. The tracks which were near each other are terminated. An adjoinable track hypothesis containing these two tracks is formed. Then, new tracks are initiated with the density of the current position being assigned to the particle sets of the tracks which were terminated. Finally, the adjoinable track hypothesis is associated with both tracks. The tracks which were terminated and which are now part of the adjoinable track hypothesis is indicated in Figure 3-9b by the light gray track histories. The new tracks formed are indicated by the black and gray track histories. Figure 3-9b indicates that frequency signature information is to be used to localize these two tracks by circles around the appropriate part of the track histories. Although frequency bandlimited intensity measurements are being used in this case, they are of no help for data association because the frequency signature of the two targets are the same.

As the tracks begin to diverge and their current mean positions become farther than a distance of 50, classification information is no longer used to localize for the tracks which is indicated by the absence of a circle around the more recent track histories in Figure 3-9c. Even though it is not possible to determine which previous track history corresponds with one of the latter track histories, it would seem that there should not be a gap between the two tracks as shown in Figure 3-9c. The reason this has occurred is due to the limits of the approximate inference performed when localizing for specific targets (See Section 3.3.4). Because we only represent the marginal densities of track locations rather than the joint densities of multiple track locations, we incur errors in representing the true marginal densities of each track location. We refer to the work of Wainwright [26] and Yedidia [27] for characterizing how well this approximate inference scheme corresponds to the true inference.

Finally, as the tracks diverge from each other, targets are tracked well again as shown in Figure 3-9d. The adjoinable track hypothesis associated with each of these tracks encode the ambiguity of data association that occurred, which is the best we can do since no discriminatory information is available to resolve the ambiguity.

3.7.2.4 Data Association: Discriminatory Information Immediately Available

In this simulation, we demonstrate how discriminatory information that is immediately available, which in this case is the differing frequency signatures of the targets, can handle a potential data association problem. As in Section 3.7.2.3, we have two target $j \in \{1, 2\}$ with positions, $x^1(t)$ and $x^2(t)$, over time $t = 0, 1, \ldots$ given by Eqn. 3.17 and Eqn. 3.18. For this simulation, the frequency signatures of the targets

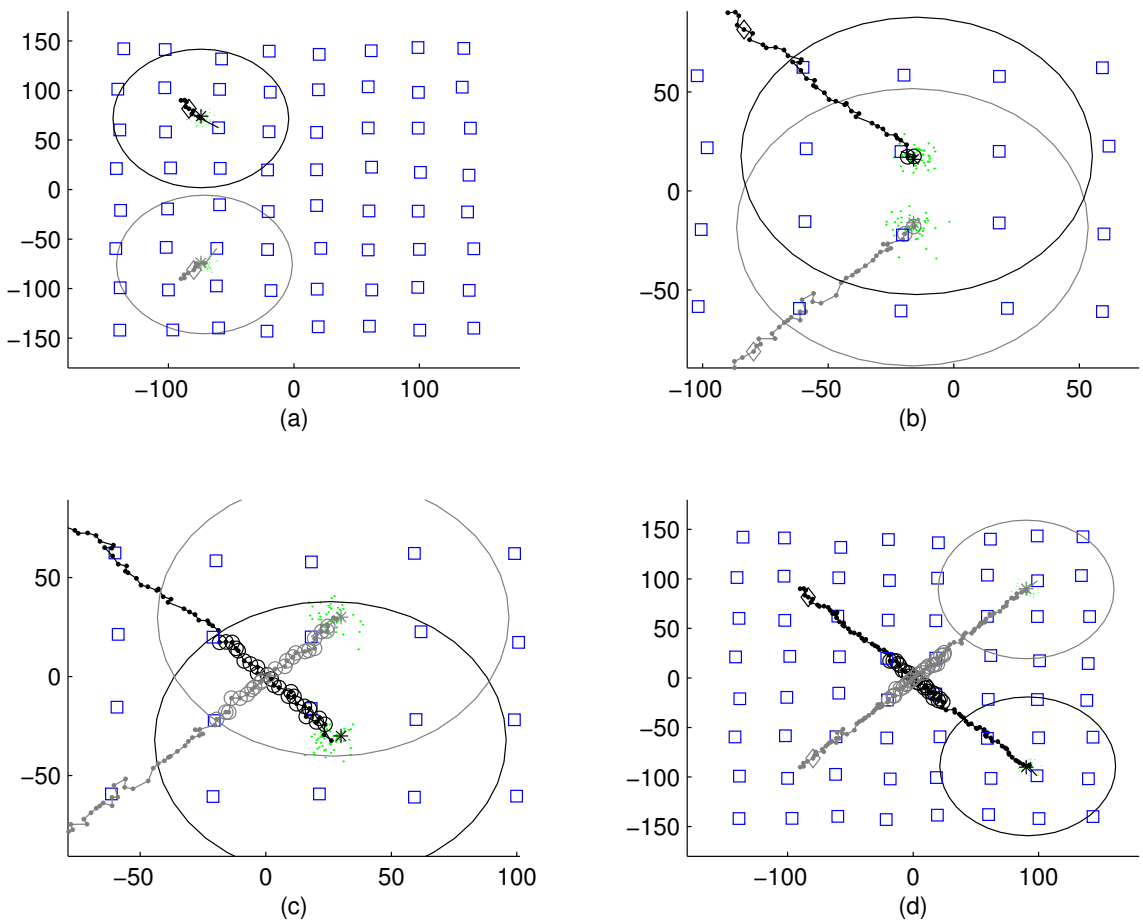

Figure 3-10: Snapshots of demonstrating handling of data association problem when discriminatory information is available. Targets travel at constant velocity and cross.

are different and chosen to be

$$\underline{A}^1 = \begin{bmatrix} 10 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \underline{A}^2 = \begin{bmatrix} 0 \\ 10 \\ 0 \end{bmatrix} .$$

Figure 3-10 shows snapshots of the resulting output of the inference system.

The tracks have been artificially initialized at the true positions of the two targets at time $t = 0$. After a few iterations, classification for the frequency signatures of each track is initiated as shown in Figure 3-10a. The result of the classification is that target 1, which corresponds to the black track history, has frequency signature equal to $(10, 0, 0)'$ and that target 2, which corresponds to the gray track history, has frequency signature equal to $(0, 10, 0)'$.

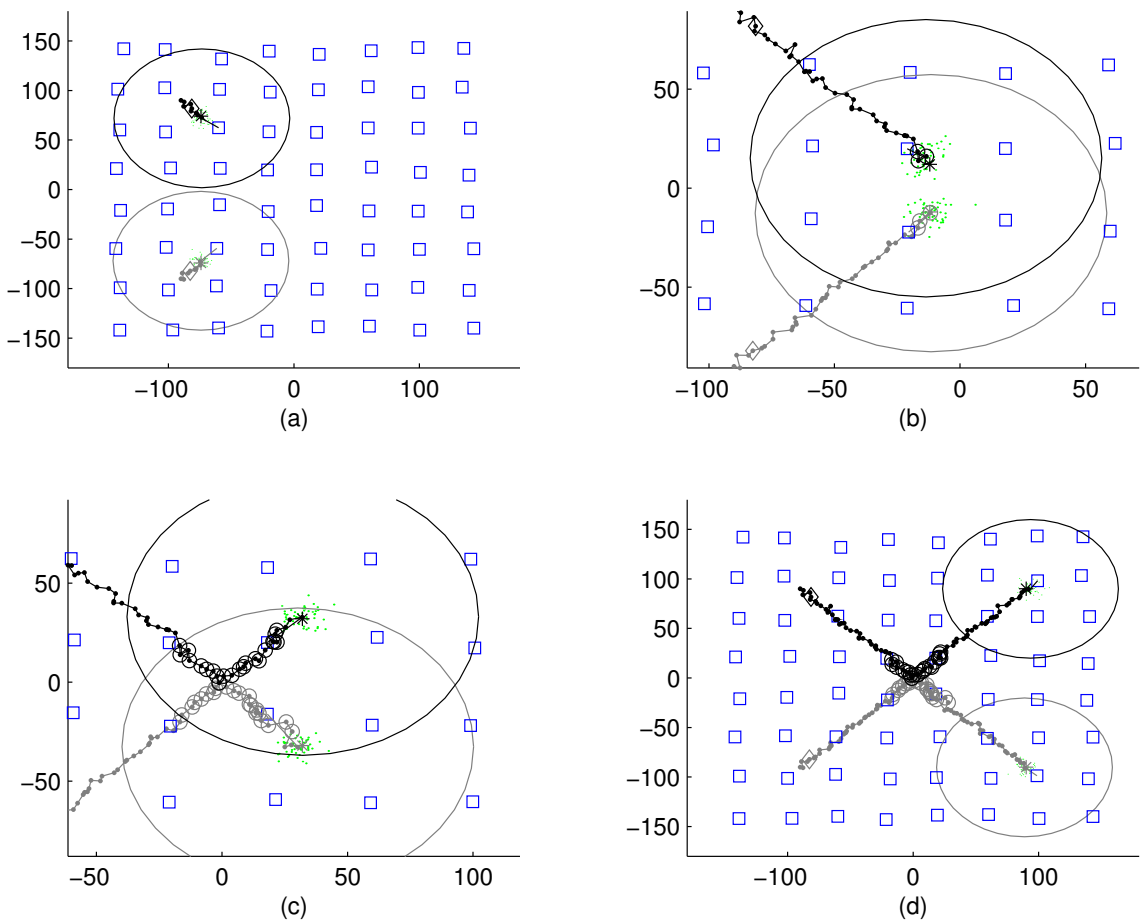

Figure 3-11: Snapshots of demonstrating handling of data association problem when discriminatory information is available. Targets travel at constant velocity, converge, and split apart.

When the tracks become near each other as shown in Figure 3-10b, the frequency signatures are used to localize the tracks as indicated by the circles around the most recent portion of the track histories. Since the tracks correspond to targets with different frequency signatures, neither track is deleted, and no adjoinable track hypothesis is formed.

As the tracks begin to diverge from each other as shown in Figure 3-10c and are no longer considered close (distance between mean positions greater than 50), frequency signature information is no longer used which is indicated by no circles around the most recent part of the track histories.

Finally, Figure 3-10d shows the resulting two tracks which correspond to the true trajectory of the targets. Note that the limitation of the approximate inference for

localization of a specific track is not observed in this simulation as it was in the simulation of Section 3.7.2.3 where no discriminatory information is available. The reason is that since frequency signature information is being used and the targets have different frequency signatures, the estimates of the posterior contribution by each track to every sensor node is more accurate.

To show that the targets are being tracked correctly, we present the following motion of the two targets.

$$x^1(t) = \begin{cases} \begin{bmatrix} -90 \\ 90 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} t & , \ t \leq 45 \\ \begin{bmatrix} -90 \\ -90 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} t & , \ t > 45 \end{cases} \tag{3.19}$$

$$x^2(t) = \begin{cases} \begin{bmatrix} -90 \\ -90 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} t & , \ t \leq 45 \\ \begin{bmatrix} -90 \\ 90 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} t & , \ t > 45 \end{cases} \tag{3.20}$$

The target trajectories above are the same as the target trajectories of Eqn. 3.17 and Eqn. 3.18 for time $t < 45$. For time $t > 45$, the latter half of the target trajectories have been switched. Since the frequency signatures of the two targets are different, this information should result in two tracks which converge towards the middle of the field and diverge away from each other rather than crossing as in Figure 3-10.

Figure 3-11 shows the resulting tracks from this scenario with the latter half of the target trajectories switched. Indeed, the resulting tracks shown in Figure 3-11d show that the targets have converged and then diverged from each other without crossing.

3.7.2.5 Data Association: Discriminatory Information Delayed

In this simulation, we demonstrate how discriminatory information which is available after an ambiguity in data association arises can be used to resolve the data association. We use a sensor network consisting of a square grid of intensity sensors and 3 magnetometer sensors as shown in Figure 3-12a. We have two targets $j \in \{1, 2\}$ travelling at constant velocity with target trajectories, $x^1(t)$ and $x^2(t)$, over time $t = 0, 1, \ldots$ given by Eqn. 3.17 and Eqn. 3.18. For this simulation, the targets have the same frequency signature

$$\underline{A}^1 = \underline{A}^2 \ ,$$

but target 1 is metallic, $W^1 = 1$, and target 2 is non-metallic, $W^2 = 0$.

Figure 3-12 shows the progression of the resulting tracks as these two targets move through the field. The two tracks have been artificially initialized at the true location of the targets at time $t = 0$. Initially, the two tracks move within range of two separate magnetometers so that the classification for whether the tracks correspond to a metal target is initiated as shown by the triangles around those parts of the track history

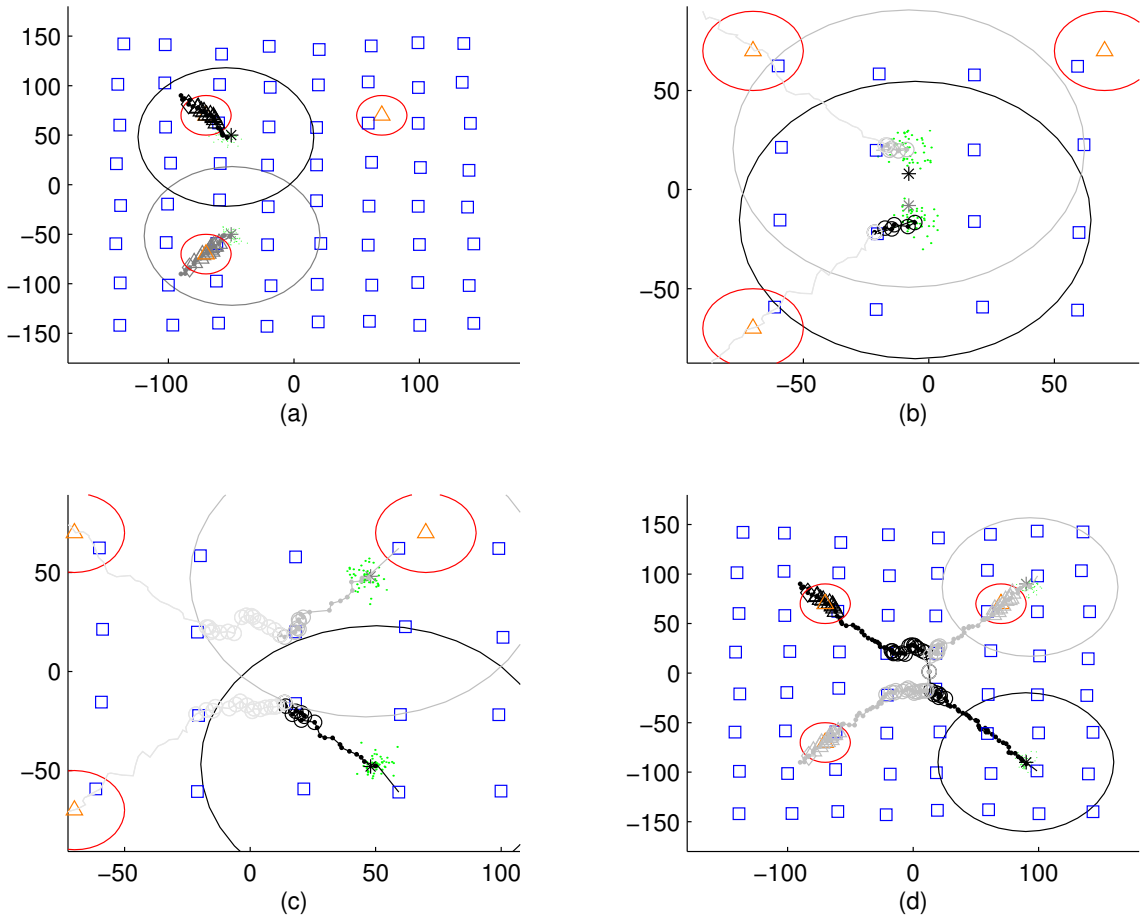

Figure 3-12: Snapshots of demonstrating handling of data association problem when discriminatory information is not available until after tracks have diverged. Targets travel at constant velocity and cross.

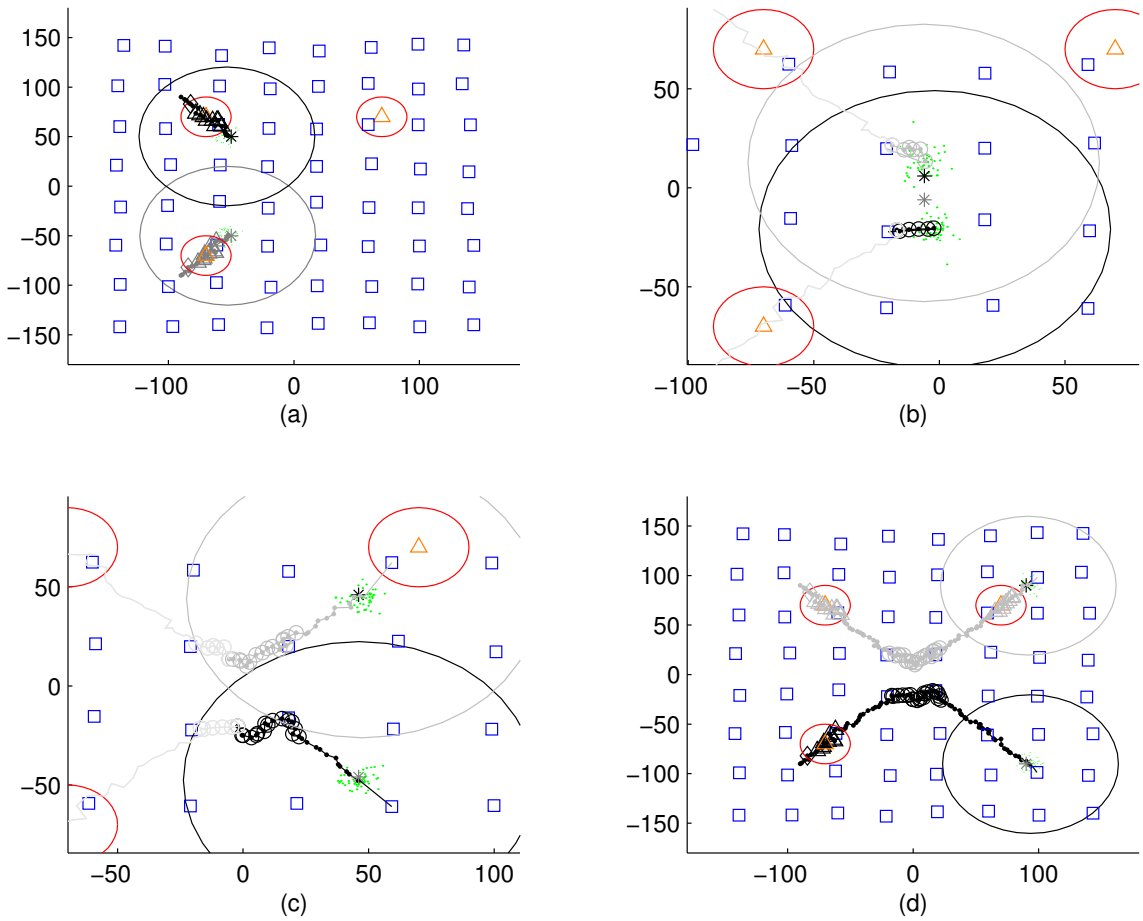

Figure 3-13: Snapshots of demonstrating handling of data association problem when discriminatory information is not available until after tracks have diverged. Targets travel at constant velocity, converge, and split apart.

for which this classification was performed in Figure 3-12a. The result is that, with high probability, the track with track history shown in black corresponds to a metal target, and the track with track history shown in gray corresponds to a non-metallic target.

Just as in the case when no discriminatory information is available, when the tracks near each other, the original two tracks are terminated, an adjoinable track hypothesis is created, and two new tracks are formed and associated with the adjoinable track hypothesis. Figure 3-12b shows this occurring. Classification information is being used to localize for both tracks although it is of no aid for resolving the ambiguity in data association.

When the tracks diverge again as shown in Figure 3-12c, the tracks are updated

without using frequency signature information. Then, when one of the tracks enters within range of the magnetometer at the top right of the field as shown in Figure 3-12d, classification for whether this track corresponds to a metal target is initiated. After several updates, it is found that the target which entered within range of this magnetometer corresponds to a non-metallic target. Then, the adjoinable track hypothesis associated with this track can be resolved since one of the tracks in the hypothesis corresponds to a metal target and the other track in the hypothesis corresponds to a non-metallic target. Furthermore, resolution of this hypothesis leads to the resolution of the hypothesis for the other track. The result is shown in Figure 3-12d, where the inference system believes that the tracks have crossed each other. Note however, that there is a sudden jump in the track history for both tracks so that such resolutions of an adjoinable track hypothesis could be followed by adjusting the track history in some way.

To show that the targets are being tracked correctly, we present the same scenario except that the last half of the target trajectories are flipped. That is, the target positions over time are given by Eqn. 3.19 and Eqn. 3.20. The progression of tracking these targets is shown in Figure 3-13. The same sequence of steps occur as before, except that when one of the tracks is within range of the magnetometer sensor at the top right of the field in Figure 3-13d, the track is found to be metallic. Thus, resolution of the adjoinable track hypothesis results in the opposite resolution to that of Figure 3-12d.

3.7.3 Demonstrations of Limits of Proposed Inference System

3.7.3.1 Track Quality

To measure the performance of the tracking part of the inference system, we will consider how the track quality changes under different perturbations of the simulated measurements. We will consider the following measures of the track quality. Recall that a track T has associated with it a history of the position $x^T(t)$ and error covariance $\Sigma^T(t)$.

1. **Time series of the deviation of mean of track estimates from the true location of the target.** If the true location of the target is $x(t)$, then the time series deviation over time $\varepsilon(t)$ is given by

$$\varepsilon(t) = d(x(t), x^T(t)) \ .$$

2. **Mean error in distance integrated over time.** For a time window

$$t_{start} \leq t \leq t_{stop} \ ,$$

the *mean track error* is given by

$$MTE = \frac{1}{t_{stop} - t_{start} + 1} \sum_{t=t_{start}}^{t_{stop}} \varepsilon(t) \ . \tag{3.21}$$

This quantity measures the average deviation in distance between the track mean and the true target position over a window of time.

3. **Length of track before loss.** We will consider a loss to occur when the true position of the target is farther than γ away, and refer to the first time to a loss as the *track length* ,

$$TL = \min_{t=0,1,\ldots:d(x^T(t),x(t))>\gamma} t \ , \tag{3.22}$$

where we will use $\gamma = 20$.

3.7.3.2 Degradation of Multiple Targets with Simulated Noise

In this simulation, we consider the effect of more simulated noise than modelled in the simulated sensor measurements for tracking multiple targets which are close enough that there always exists a sensor node which must compute the posterior contribution from multiple tracks but far enough that an ambiguity in data association does not arise. Recall that for our choice of parameters, the range of each sensor node is 70. We simulated targets in the sensor network of Figure 3-3.

For each tracking run, we chose an angle ϕ uniformly in the range $[0, 2\pi)$. The initial target positions were chosen to lie on the line

$$\left\{ (x, y) \in \mathbb{R}^2 : y = \frac{\sin(\phi - \frac{\pi}{2})}{\cos(\phi - \frac{\pi}{2})} x \right\}$$

spaced 70 units apart all with the same velocity

$$v = 3 \begin{bmatrix} \cos(\phi) \\ \sin(\phi) \end{bmatrix} \ .$$

Furthermore, all targets were given the same frequency signature. There are no magnetometer sensors in the sensor network so there is no need to specify whether the targets are metallic.

We artificially initiated tracks at the true location of the targets at time $t = 0$ and ran the inference system in integer time steps to time $t = 30$. We then computed the mean track error of all tracks for 20 tracking runs. The standard deviation of the noise terms were scaled by a factor of $\alpha \in \{1, 2, 4, 8\}$ when generating the simulated measurements to test how robust the multiple target tracking part of the inference system is when the measurement models do not correspond to how the simulated

measurements were generated. The measurement models used to update the tracks were the assumed noise parameters given in Section 3.7.1. Figure 3-14 shows the mean track error plotted as a function of $\log_2(\alpha)$ when there are 1, 2, 3, 4, and 5 tracks moving in parallel through the sensor network. The gray dots indicate the mean track errors for the twenty different tracking runs, and the circles connected by a line indicate the mean of those twenty tracking runs for different values of α.

As expected, mean track error grows with more noise in the simulated measurements. With every doubling of the simulated noise, the mean track error approximately doubles. Figure 3-15 shows the track length, which was defined in Eqn. 3.22. As the scaling on the noise terms of the simulated measurements grows, the track length decreases as expected.

To show the performance of tracking multiple targets, we compare the mean track errors and track lengths for different numbers of targets in Figure 3-16. The degradation of the mean track error due to tracking multiple targets in the sensor network is small when the targets are spaced far enough apart that no data association problem arises. The mean track error grows additively with the number of targets independent of the growing noise term in the simulated measurements. Furthermore, the track length becomes smaller as the number of targets increases but is independent of the growing noise term in the simulated measurements. These plots show that the performance for tracking multiple targets is comparable to tracking a single target when the true targets are close but not so close that a data association problem arises.

3.7.3.3 Degradation with Similarity in Frequency Signature

In this simulation, we explore how well data association is handled when tracks cross and the frequency signature of the two targets are varied in their similarity. The target trajectories are given by

$$\begin{aligned} x^1(t) &= \begin{bmatrix} -35 \\ 35 \end{bmatrix} + \begin{bmatrix} 2 \\ -2 \end{bmatrix} t \\ x^2(t) &= \begin{bmatrix} -35 \\ -35 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} t \end{aligned}$$

and move through a sensor network like the one shown in Figure 3-3 except that all sensor nodes were chosen to be intensity sensors and the magnetometer sensors were taken out.

We vary the frequency signature of the targets with a parameter $p \in [0, .5]$ where

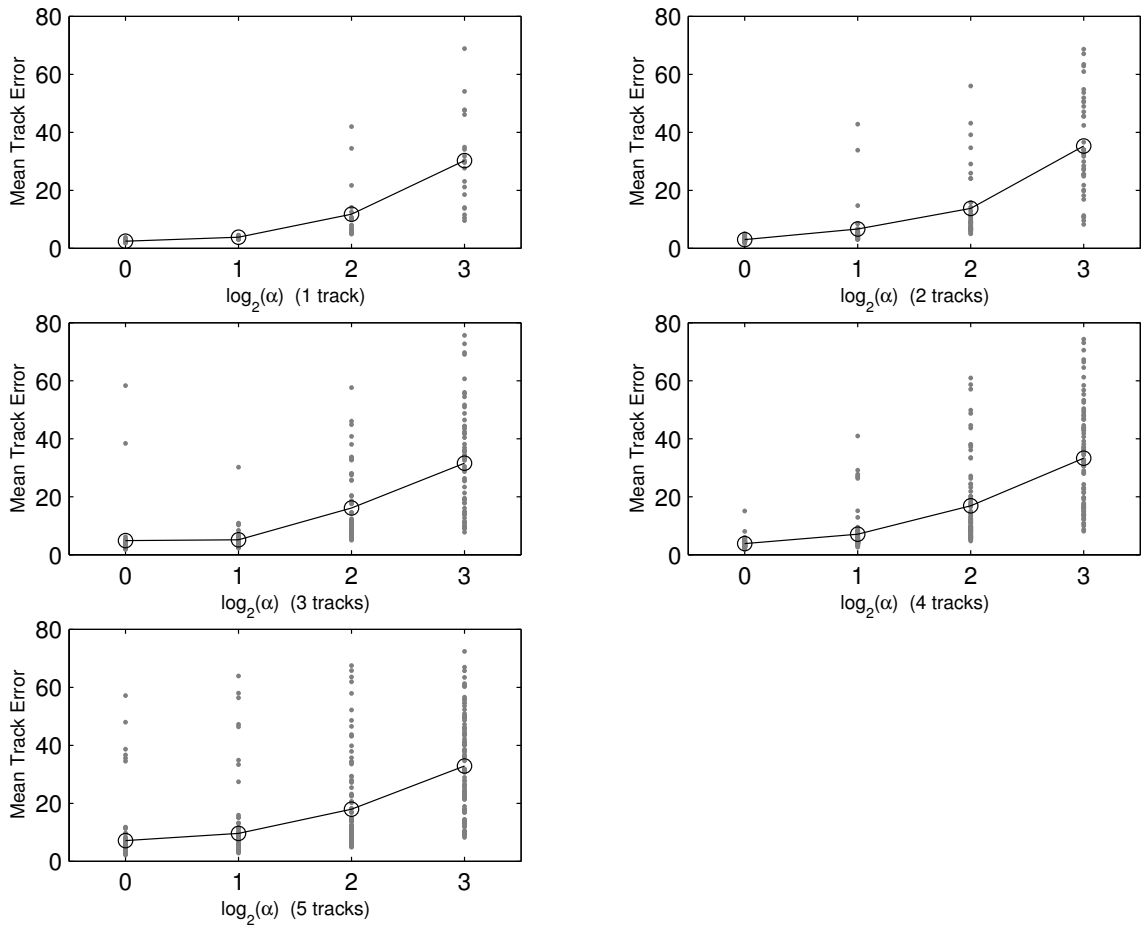

Figure 3-14: Mean track error plotted as a function of α, the scaling on the noise standard deviation of the simulated measurements. Plots show the individual mean track errors over twenty runs as well as the average of those twenty runs for 1, 2, 3, 4, and 5 tracks moving in parallel.

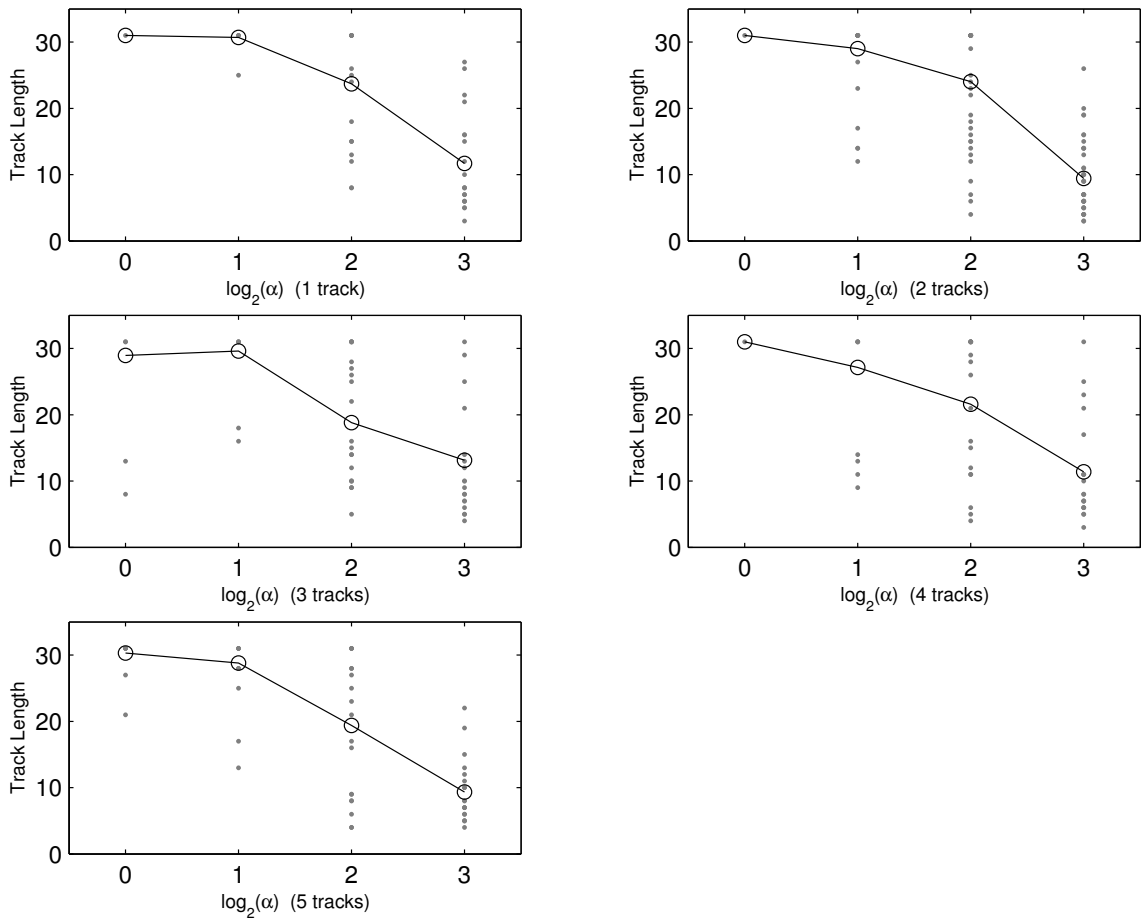

Figure 3-15: Track length plotted as a function of α, the scaling on the noise standard deviation of the simulated measurements. Plots show the individual track lengths over twenty runs as well as the average track length over those twenty runs for 1, 2, 3, 4, and 5 tracks moving in parallel.

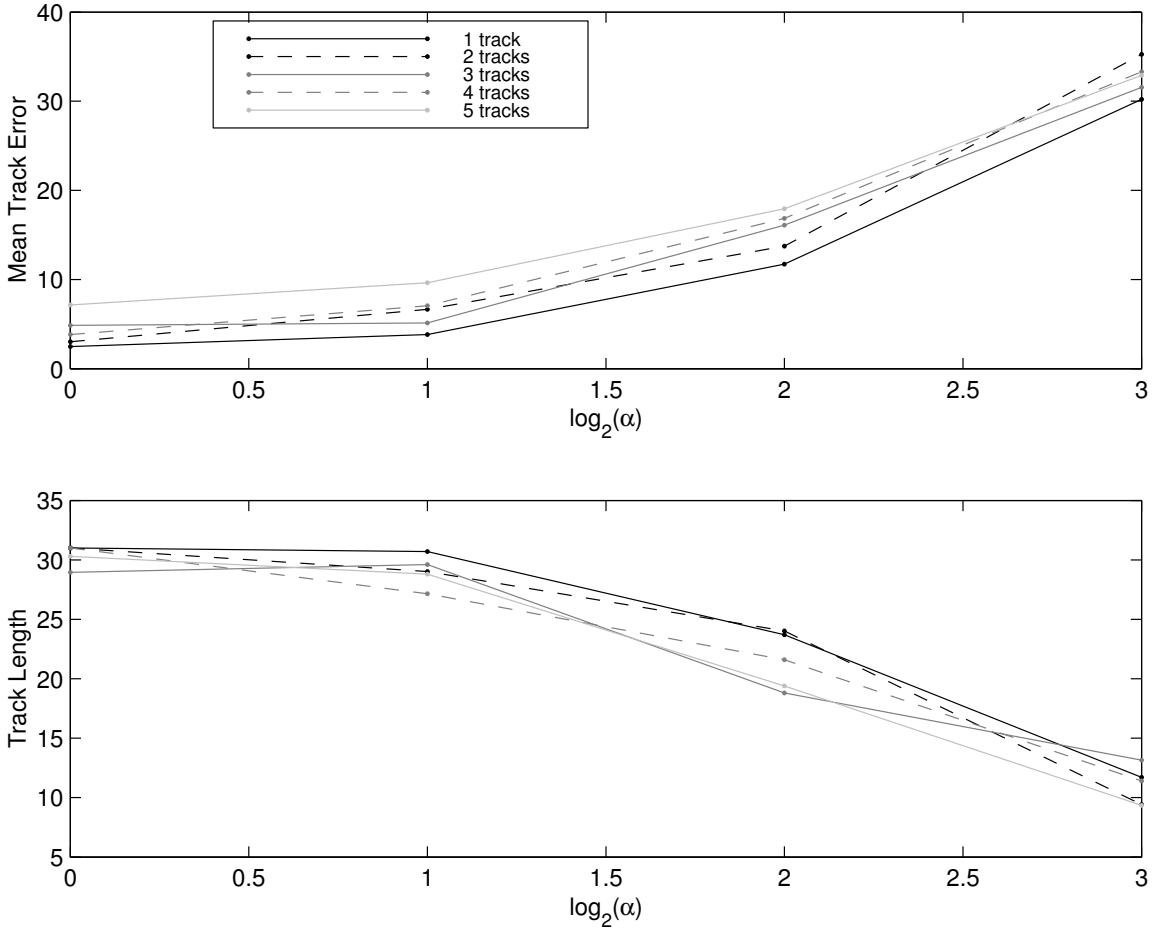

Figure 3-16: Comparison of the average mean track error and average track length for simulations with 1, 2, 3, 4, and 5 tracks moving in parallel.

the frequency signature of each target is

$$\begin{aligned}\underline{A}^1 &= p\begin{bmatrix}10\\0\\0\end{bmatrix} + (1-p)\begin{bmatrix}0\\10\\0\end{bmatrix}\\ \underline{A}^2 &= (1-p)\begin{bmatrix}10\\0\\0\end{bmatrix} + p\begin{bmatrix}0\\10\\0\end{bmatrix}\end{aligned}$$

Measuring the similarity of the two frequency signatures by the Euclidean norm,

$$\|\underline{A}^1 - \underline{A}^2\| \ ,$$

the frequency signatures are most different when $p = 0$ and are exactly the same when $p = .5$. We will measure the mean track error, the track length, and the misclassification error over 20 runs for different values of $p \in [0, .5]$. Sensor measurements were simulated according to the nominal parameters for the measurement models as described in Section 3.7.1. Tracks were artificially initiated at the true locations of the targets, and the inference system was allowed to run. After 5 iterations, the classification for the frequency signature task was initiated and tracks classified for their frequency signature. Due to the noise in the simulated measurements, it is possible for the classification routine to assign the incorrect frequency signature to the targets.

Figure 3-17 plots these statistics over values of p. The transition between the ability to use frequency signature information to disambiguate tracks when they cross occurs between $p = .4$ and $p = .45$. Thus, using frequency signature information to disambiguate tracks which are near each other should only be used if the frequency signatures of the tracks are dissimilar enough. Otherwise, the inference system should consider these tracks as having no discriminatory information so that the tracks are terminated, an adjoinable track hypothesis is created, and new tracks are initiated and attached with the adjoinable track hypothesis.

The misclassification of the frequency signature is computed by the percentage of runs for which the frequency signature computed for a track is mismatched with the true frequency signature of the associated target. When $p > .45$, misclassification errors begin to occur. Note that for $p = .5$, there is no misclassification error. This happens because in this case there is only one frequency signature that both tracks can be classified, namely

$$\begin{bmatrix}5\\5\\0\end{bmatrix} .$$

To handle the possibility of misclassification error, a more robust method for classifying frequency signatures should be to periodically update the frequency signature.

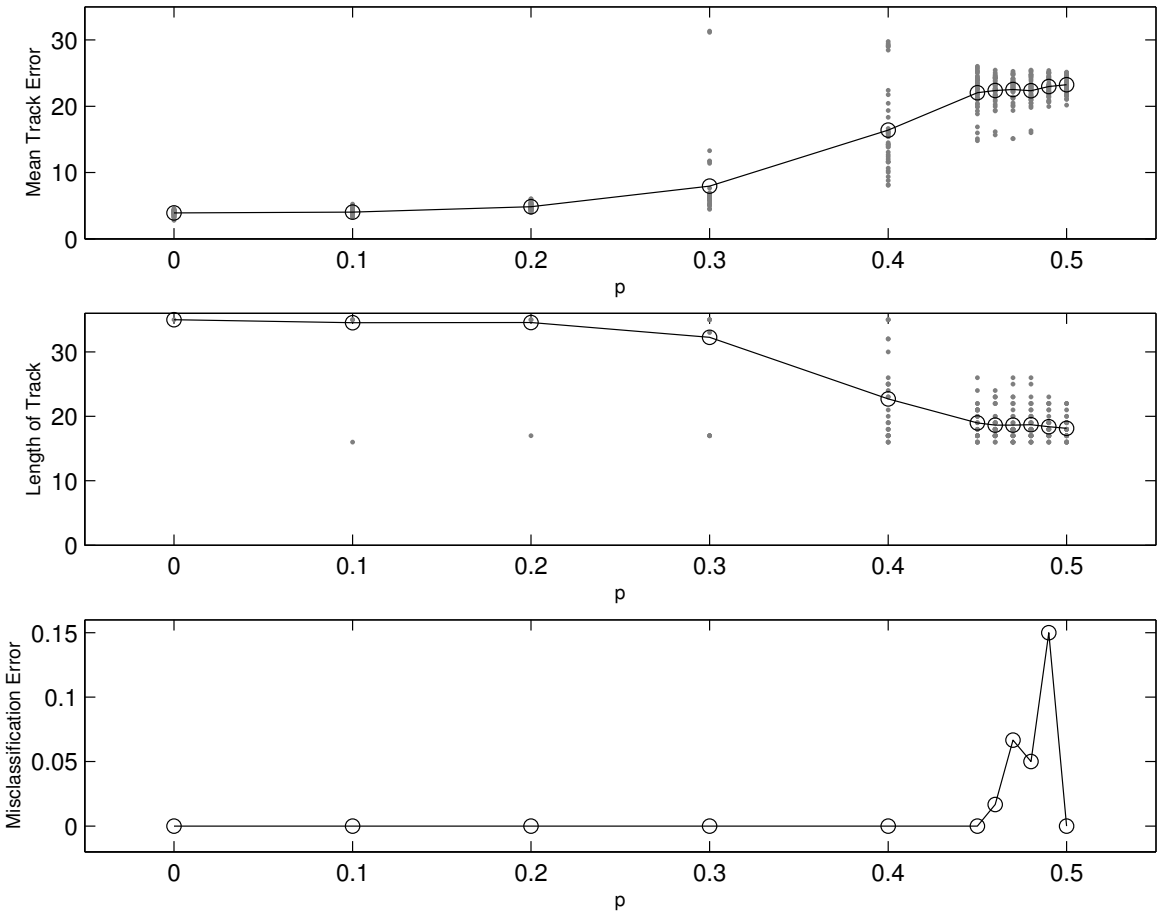

Figure 3-17: Mean track error, track length, and misclassification error over values of p.

3.7.3.4 Degradation of Track Quality with Changes in Intensity Attenuation Coefficient

In this simulation, we explore how variabilities in the intensity attenuation coefficient α in the intensity (Eqn. 2.3) and frequency bandlimited intensity (Eqn. 2.5) measurement models affects the mean track error and the track length. We simulated a single target moving through a sensor network like the one shown in Figure 3-3 except that all sensor nodes were chosen to be intensity sensors and the magnetometer sensors were taken out. For the different runs, the target trajectory was chosen to be

$$x(t) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 3 \begin{bmatrix} \cos(\phi) \\ \sin(\phi) \end{bmatrix} t$$

over time $t = 0, 1, \ldots, 30$ and ϕ was chosen at random from a uniform distribution over the interval $[0, 2\pi)$.

The simulated measurements were generated by varying the intensity attenuation coefficient α. Targets were tracked under the nominal parameters of the measurement models described in Section 3.7.1. In particular, the assumed parameter value for α in the intensity and frequency bandlimited intensity measurement models was 2. Figure 3-18 plots the mean track error and track length over various values of the α generating the simulated measurements.

We see that in the range $\alpha \in [1.8, 2.0]$, the mean track error is fairly low and the track length is near the maximum possible. Outside of this range, the track quality degrades quickly for small values of α and degrades more gradually for larger values of α. Since the track quality is sensitive to variations in α, it will be critical that a good estimate of the intensity attenuation coefficient is available in order to track targets using intensity sensors.

3.7.3.5 Degradation of Track Quality with Spacing Between Targets

In this simulation, we determine the limits of tracking multiple targets moving in parallel depending on the spacing between the targets. It was seen in the Section 3.7.3.2 that multiple targets are tracked well when they are spaced far enough apart.

For each tracking run, we simulated the movement of 4 targets travelling in parallel in the sensor network of Figure 3-3 without the magnetometer sensors. All targets were given the same frequency signature, and the targets being metallic are left unspecified because we have no magnetometer measurements in the sensor network.

As in Section 3.7.3.2 the target trajectories were given a random orientation. The distance d between them is varied in this simulation. The simulated measurements are generated according to the measurement models as described in Section 3.7.1 so that the inference system tracks targets with correct knowledge of how the measurements are generated. We have disabled the termination of tracks and the creation of adjoinable track hypotheses in this simulation.

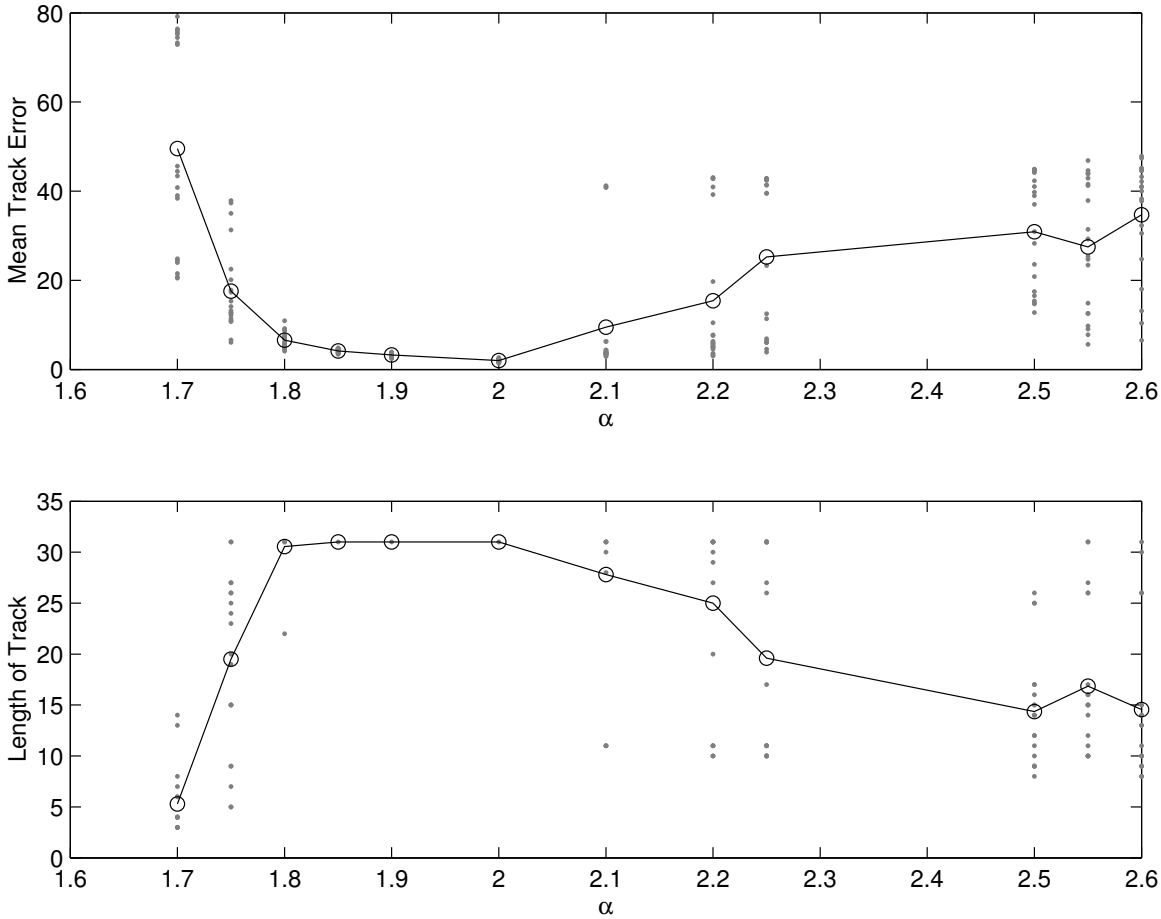

Figure 3-18: Mean track error and track length plotted as a function of intensity attenuation coefficient α generating the simulated measurements.

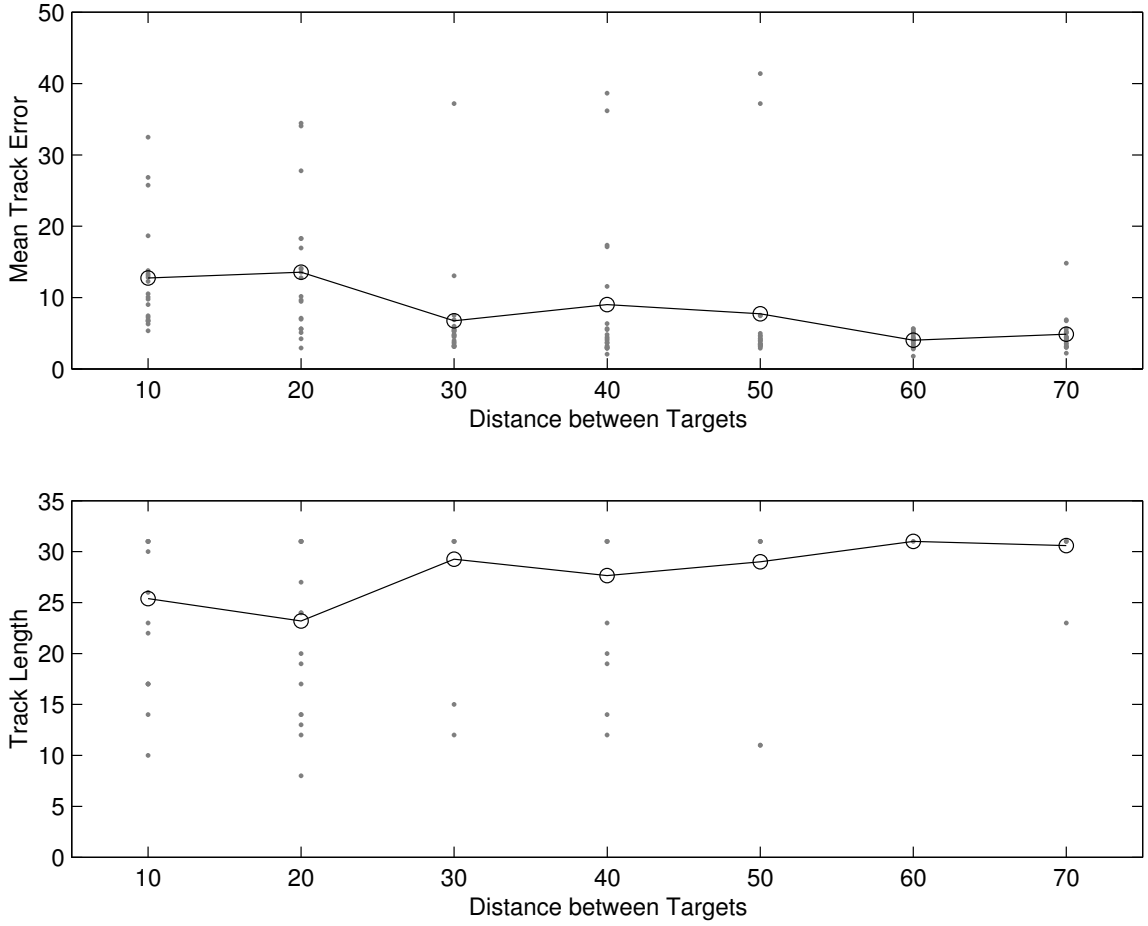

Figure 3-19: Mean track error and track length plotted as a function of the distance d between the 4 targets moving in parallel.

Figure 3-19 shows the plots of mean track error and track length over different values of d. Mean track error does not degrade too much with smaller distance d, although the length of the track degrades. Since small track lengths indicate loss of the target by the track, we should use discriminatory information to localize for tracks when the distance between targets is between 50 and 60.

3.8 Limitations and Extensions of the Overall Inference System

In this section, we will discuss some limitations of the overall inference system presented in Section 3.4, Section 3.5, and Section 3.6 and simulated in Section 3.7.

Furthermore, we will suggest possible ways to extend the inference system to address these limitations.

Sparse Entry
Since the DCA's were constructed for single target localization, the inference system can only track multiple targets if targets enter the field in a sparse fashion. Thus, constructing DCA's for multiple target localization is needed. As mentioned before, since the single target localization can handle the case when targets enter the field in a sparse fashion, multiple target localization need only be concerned with the case when multiple targets enter the field in a dense fashion.

Track Contributions
In order for the multiple target tracking scheme to work, the track holder $\mathbf{id}^T$ of track T must send its predicted contribution to *every* sensor node within range of it, say a radius of r from the current position of the track. If r is large, then a lot of communication cost is incurred in order for the track holder to send the predicted contributions of track T to each of the sensor nodes within a radius of r from it. In order for each sensor node to compute a reasonable estimate of the posterior contributions from each track, the predicted contributions from all tracks within range r of the sensor node must be known. However, we observe that in a dense enough sensor network that it may not be necessary that posterior contributions from all sensor nodes within range of a target are needed in order to estimate the localization for the specific track. Thus, we suggest two possible refinements.

1. The first is that the track holder $\mathbf{id}^T$ still sends the predicted contributions of the track T to all sensor nodes within range r. The refinement is that only those sensor nodes which are closer, say within $r' < r$ from the track T, need to send the posterior contribution of track T back to the track holder. This refinement only reduces the communication cost in sending posterior contributions and does not effect the communication cost of sending the predicted contributions.

2. The second refinement is for the track holder $\mathbf{id}^T$ to select only a subset of sensor nodes within range r from track T to send predicted contributions of the track T. As long as we can guarantee that these subset of sensor nodes have knowledge of all tracks within range of them, then the multiple target tracking scheme should work. Furthermore, since there are sensor nodes which are not receiving predicted contributions from all tracks within range, the track initiation routine must be modified so that the estimated contribution from possibly new targets entering the field is reasonably accurate.

Frequency Dependent Intensity Attenuation
The model for the received frequency bandlimited intensity measurements has an intensity attenuation which is the same for every frequency band. For long ranges, the

attenuation of the intensity is frequency dependent, so that adjustments will have to be made in the localization based on frequency bandlimited intensity measurements task and the classification for frequency signatures task.

Resolving Adjoinable Track Hypothesis Containing a Single Track
When the hypothesis H of a track T is resolved and only a single track T' is in the hypothesis so that

$$H = \{T'\} \ ,$$

other active tracks in the sensor network must be notified to remove T' from their associated hypothesis. Since tracks are represented in a distributed fashion, the track holder $\mathbf{id}^T$ of T must broadcast a message to all other track holders to remove T' from their hypothesis. This broadcast message is an instance of the dissemination information transfer which we refer to the literature for implementing this broadcast message on the sensor network.

Partial Resolution of Adjoinable Track Hypothesis
When a track T with an attached adjoinable track hypothesis H gains information to partially resolve which track in H corresponds to the track T, this information should also be used to partially resolve the adjoinable track hypotheses of other tracks known by the sensor network. This partial resolution can be critical if information, like the information from a magnetometer, is rarely collected. We refer to work in [24] which describes a distributed implementation of partially efficiently resolving hypotheses about target identity in sensor networks.

Communication Delays, Collisions, and Loss
In the inference system, we have not considered the possibility of long delays due to communications involved for the localization for a specific target task, the classification of frequency signature task, and the classification for metal targets task. These delays can have a detrimental effect especially for the localization for a specific target task because track updates must occur periodically in real-time in order to accurately track the target. The delays do not affect the classification tasks as detrimentally since these tasks are performed much less often than the localization task.

We have also assumed that in the communications involved that collisions do not occur. This assumption is not so unreasonable since sensor nodes which are close together can be assigned to different time slots so that concurrent communication is possible. However, to make the system robust, we must take into account the possibility that messages may be lost. For example, any time a track holder sends predicted contributions of a track to sensor nodes within range, it expects response messages from all sensor nodes within range of the track. To make the system robust, the track holder must determine how long to wait for a response and continue with

the track update if the response time is too long.

Computing Distance Between Tracks

The trigger for using frequency bandlimited intensity measurements for localization of a specific target is when there exists another target which is near it. In the distributed implementation, the track holders $\mathbf{id}^{T_1}$ and $\mathbf{id}^{T_2}$ for two tracks T_1 and T_2 which are considered near each other may not be the same, meaning

$$\mathbf{id}^{T_1} \neq \mathbf{id}^{T_2} \ .$$

Thus, in order for a track holder $\mathbf{id}^T$ to determine whether there exists another track near track T, a message must be broadcast to nearby sensor nodes that are track holders which sends the current mean position of its track. Notice that this kind of broadcast message is an instance of the dissemination information transfer.

The communication cost of this broadcast message can be expensive. However, this communication need not occur as often as the communications involved for the localization of a specific target task. Depending on the maximum speed at which targets may converge in position, we can determine the minimal rate for which we must test for whether a track is near another track so that the tracks do not become near each other before the network is able to detect that they will be near.

Limits of the Approximate Inference

The updates to the track state from the sensor measurements were computed according to a belief propagation type of approximation. When tracks are spaced far apart, this approximation is good enough so that targets can be tracked well. However, when tracks are close together and there is no discriminatory information among the tracks, the approximation is bad and the localization estimates tend to be spaced farther away than they truly are. However, once the tracks diverge from each other, the approximation once again becomes good so that the targets are tracked well again. Thus, to make better estimates of the target locations when targets are near each other, we propose two possibilities.

1. When targets are near each other, we can switch to the joint space of multiple target locations so that the true joint distribution is being represented. The main problem with this approach is a computational one in that the number of particles in the particle filter must be made larger in order to fill up the region of high probability in the joint space.

2. We can run the approximate track update with a modification to compensate for the error in the approximation. For example, a simple modification could be to increase the variances of the noise terms in the sensor measurement models so that the track updates are less dependent on the sensor measurements.

Overall System Robustness

Since each of the individual information processing tasks have been solved under the assumption of some extra available information that is assumed to be correct, there is a question of how robust these solutions are when the extra available information is not quite correct. For example, localization for a specific target assumes that the individual predicted contributions from each track are correct in order to compute the posterior contribution of the track. Furthermore, classification of frequency signatures assumes knowledge of the track positions so that their frequency signatures can be estimated. If an error is made, then it is possible that this error has a domino effect to render all subsequent estimates to be incorrect. For example, if the system believes that a certain track T exists but the reality is that T does not correspond to a real target, then this incorrect information can cause errors in track updates to tracks which are near track T. Also, if the track positions used for classifying the frequency signatures are incorrect, then the wrong frequency signature can be assigned to the track. This can have a detrimental effect when the localization for the track is primed to use frequency bandlimited intensity measurements since the localization estimate will be incorrect.

The idea for dealing with these robustness issues is to have routines which test for such inconsistencies of the information assumed and the reality observed. For example, the purpose of track termination is to test whether a track actually corresponds to a true target. For localization with frequency bandlimited intensity measurements, a possible solution is to design a test for how likely the track could have contributed to the frequency bandlimited intensity measurement of a sensor. Based on the results of this test, then the track's frequency signature can be classified again.

Chapter 4

Target Localization

In this chapter, we present a solution to the single target localization problem as formulated in Section 2.3. Since the solution requires the specification of an information collection mechanism, namely the communication algorithms for each sensor node which efficiently pass data or relevant information extracted from the data around the network, we pursue a representation from which possible information collection mechanisms can be enumerated. The representation is referred to as a hierarchy which we had introduced in Section 1.7. In this chapter, we will present the hierarchy for the special case of solving the problem of target localization. A hierarchy corresponds to a particular information collection mechanism so that there will exist many hierarchies for any localization problem. Since every hierarchy encodes a particular information collection mechanism, we can analyze the computational resources required by an information collection mechanism by analyzing the hierarchy representation of it. Thus, constructing efficient communication algorithms for the problem of single target localization is reduced to the problem of constructing an efficient hierarchy. The general principle of information-based computation is at work in the hierarchy which we will see is the underlying principle which allows us to design efficiency into the system.

Abstractly, the hierarchy is a representation of functions as compositions of simpler functions from which computational resources can be measured. Since any inference task, including single target localization, is abstractly a function which maps measurements to some appropriately defined information space, the hierarchy is a representation with more general applications than the construction of efficient communication algorithms for single target localization. Chapter 5 will present these more general aspects of the hierarchy. For this chapter however, we will limit the scope of the presentation of the hierarchy to focus on the construction of efficient communication algorithms.

First, Section 4.1 presents hierarchies which represent simple global functions to illustrate how a hierarchy encodes a specific information collection mechanism and how computational resources required by the information collection mechanism can be

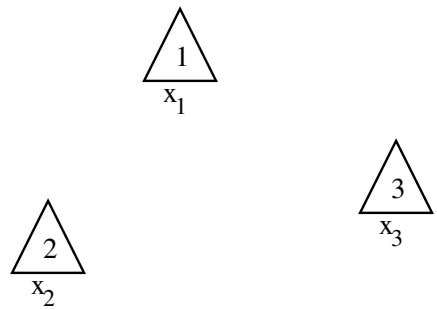

Figure 4-1: A system architecture with three distributed autonomous agents shown as triangles labelled by an integer from $\{1, 2, 3\}$. Initially, each agent can observe the value of its respective variable indicated underneath each triangle.

measured directly from the hierarchy. Second, Section 4.2 discusses how to convert an inference task into a set of computations to encode into a hierarchy. Then, Section 4.3 presents a toy sensor network which describes the solution methodology of using the hierarchy to generate a set of distributed communication algorithms which solves the single target localization problem heuristically.

4.1 Encoding of Information Collection in Hierarchies

In this section, we present how the hierarchy is a representation of an information collection mechanism which includes partial processing of data and information and possible control of sensing for a specific global task on a specified system architecture. Since we wish to address efficiency in computational resource usage with the hierarchy representation, what is especially important is how we can directly measure the computational resource requirements of the resulting algorithms running on the system architecture from the hierarchy representation itself. The presentation uses a simple example to illustrate the hierarchy representation.

Consider a system architecture with three distributed autonomous agents in Figure 4-1. The three distributed autonomous agents are labelled by an integer $i \in \{1, 2, 3\}$. Agent i can observe the variable $x_i \in X_i$ and is initially oblivious of the values of the other two variables. The problem is to evaluate some given function

$$f : X_1 \times X_2 \times X_3 \to Y$$

which depends on the values of the three variables for some range space Y. We assume that all agents have the ability to evaluate f if it knows enough information about the values of all three variables. Thus, the requirement for any information collection

mechanism is that agents must collaborate and communicate the information about the values of the variables x_1, x_2, and x_3 so that at least one agent is aware of enough information to evaluate f.

We make a distinction here between data and information. By data, we refer to the values of the observables of each agent. In this simple example, data would be the observed values of the variables x_1, x_2, and x_3. To speak of information, we must specify what the information is for. Information in this case refers to the information that the observed values of the three variables contain about the output of function f. In particular, the information contained in the observed values of the variables is about which inverse image

$$\{f^{-1}(y)\}_{y \in Y}$$

the triple (x_1, x_2, x_3) is a member. Certainly, the data *contains* all the information needed to evaluate f because, knowing (x_1, x_2, x_3), we can certainly determine which inverse image it lies in; however, the data also contains extraneous information which should be filtered out during information collection so that we do not waste energy communicating this irrelevant information. That is, all we must know is which inverse image $f^{-1}(y)$ that (x_1, x_2, x_3) lies in, and we do not need to know the exact value of the triple (x_1, x_2, x_3). Because of this difference, we term the collection mechanism an *information* collection mechanism rather than a *data* collection mechanism. The essential difference between data collection and information collection is that no partial processing which attempts to extract relevant information from the data is allowed to occur during a data collection process. Because the only difference between information collection and data collection is the possibility of partial processing, we will first consider the problem of representing the set of possible data collection mechanisms. Then, we will see that all information collection mechanisms may be viewed as a refinement of a data collection mechanism to include partial processing on the data so that only the relevant information is passed around the network. Furthermore, the information collection mechanism may also be further refined to include control of sensing when sensing is expensive.

The questions we ask are the following.

1. What is the set of all possible data collection mechanisms of variable values by the three agents so that one of the agents knows all three variable values?

2. How can partial processing and control of sensing be incorporated into a data collection mechanism to result in an information collection mechanism?

3. How do we measure the computational resource requirements of the information collection scheme encoded by a hierarchy?

The next three subsections address the above three questions.

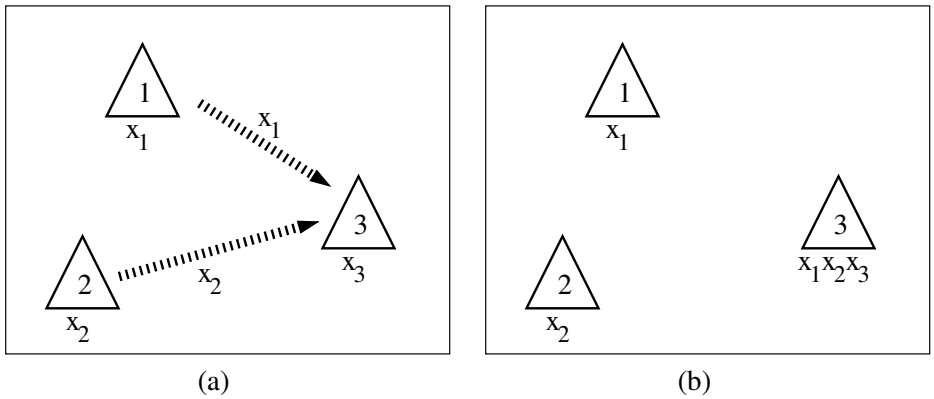

Figure 4-2: One possible mechanism for collecting data. (a) Agents 1 and 2 both send the value of their respective variable to agent 3. (b) Agent 3 is now aware of all variable values which is indicated by the the three variables listed underneath it.

4.1.1 Question 1: Set of Possible Data Collection Mechanisms

Each agent has two communication protocols for communicating data between a remote agent and itself.

1. The first, referred to as a "push" type communication, is for the agent to send data known by it to a remote agent. This is a "push" because it is the agent that is *sending* the data which initiates the communication.

2. The second, referred to as a "pull" type communication, is for the agent to query for data known by a remote agent. This is a "pull" because it is the agent that is *receiving* the data which initiates the communication.

From these two communication protocols, we can imagine the set of possible data collection mechanisms. In the following, when we say that an agent collects, sends, or has knowledge of a variable x_i, we mean that the agent collects, sends, or has knowledge of the observed value of the variable x_i.

One possible data collection mechanism could be that agent 1 sends variable x_1 to agent 2 while agent 3 sends variable x_3 to agent 2 as shown in Figure 4-2. The result is that agent 3 ends up with knowledge of all variables. Then, agent 3 can evaluate the function $f(x_1, x_2, x_3)$ which solves the global computation.

Another possibility is that agent 1 sends variable x_1 to agent 2, and then agent 2, now knowing both x_1 and x_2, sends the variables x_1 and x_2 to agent 3 as shown in Figure 4-3. The result of this scheme is again that agent 3 ends up with knowledge of all variables from which the function f can be evaluated.

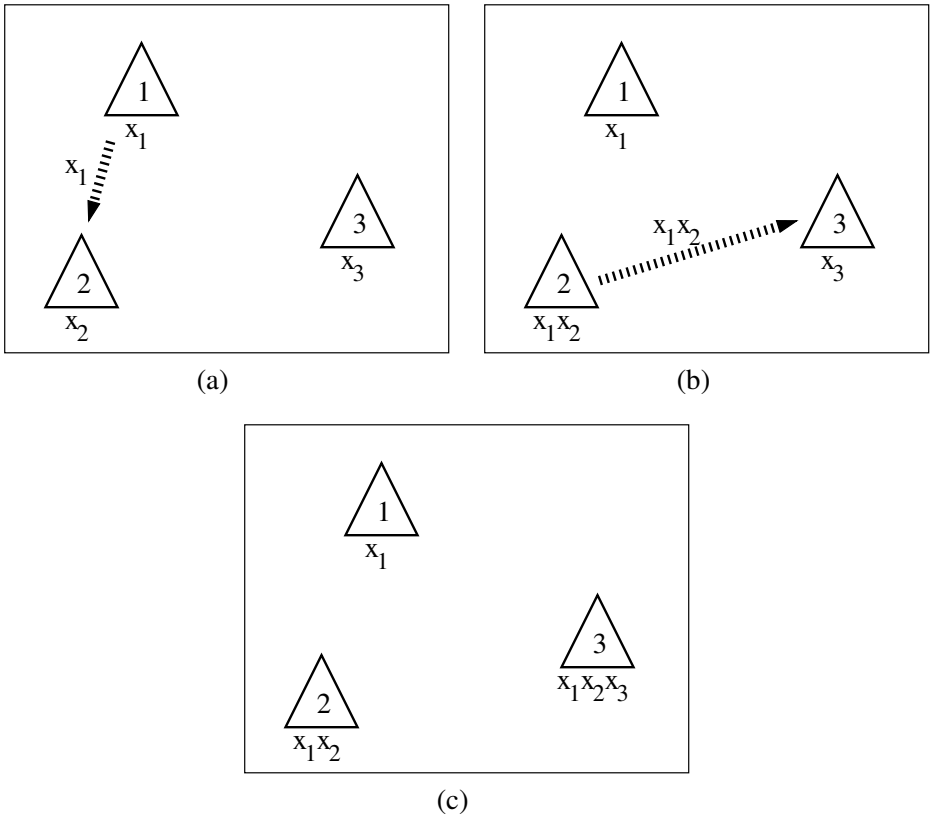

Figure 4-3: Another possible mechanism for collecting data. (a) Agents 1 sends x_1 to agent 2. (b) Then, agent 2 sends x_1 and x_2 to agent 3. (c) Finally, agent 3 is aware of all variable values which is again indicated by the variables listed under the agent. Note that a residual effect of this collection mechanism is that agent 2 is aware of both x_1 and x_2.

What we wish to do is come up with a representation from which we can systematically enumerate reasonable data collection mechanisms. By "reasonable", we mean that unnecessary communications are not taking place. An example of an unreasonable data collection mechanism would be for agent 1 to send x_1 to agent 2 hundreds of times before agent 2 sends x_1 and x_2 to agent 3.

We will use a graph representation to represent the possible data collection mechanisms.

Terminology and Notation

To facilitate the discussion of these graphs, let us define some terminology and notation. The notion of nodes being *above* or *below* one another will be important. Formally, we are imposing a partial ordering of the nodes of the graph which we can informally indicate by the spatial arrangement of the nodes in a picture of the graph. With such an ordering, we can qualify edges incident on a node as either *emanating upwards* or *emanating downwards*. For example, in Figure 4-4, the edge incident on the node associated with the variable x_1 is an edge emanating upwards from x_1. In contrast, the edges incident on the black node are all edges emanating downwards from the black node. Let us define the *down degree* of a node as the number of edges emanating downwards from the node. The black node of Figure 4-4 has down degree 3.

For each node g of the graph, we refer to nodes connected by an edge emanating downwards from g as the *children* of g. Conversely, we refer to the nodes connected by an edge emanating upwards from node g as the *parents* of g. For example, in Figure 4-4, the nodes labelled by the variables x_i for $i \in \{1, 2, 3\}$ are the children of the black node, and the black node is the parent of the three variable nodes. We extend the notion of children and parents to *descendants* and *ancestors*. That is, the descendants of a node are all children of the node, the children of those children, etc. Similarly, the ancestors of a node are parents of the node, the parents of those parents, etc. These definitions can be made because of the fact that we have imposed a partial ordering of nodes in these graphs.

We will always convert a graph containing nodes with down degree greater than 2 into a graph containing nodes with either down degree 2 or down degree 0.[1] The graphs of Figure 4-5 are examples of this conversion from the graph of Figure 4-4. For these graphs, we refer to all nodes with down degree 0 as the *input nodes* which will always be drawn at the bottom of the graph. For all nodes g in the graph with down degree 2, we specify the *left* and *right child* by $l(g)$ and $r(g)$ respectively. The definition of the left and right child is dependent on how we happen to draw the graph in the figures so that $l(g)$ and $r(g)$ are not well-defined unless we draw a picture of the graph.[2] For example, in Figure 4-5a, labelling the black node at the top by g,

[1] None of the graphs we consider contain nodes with down degree 1.

[2] To be completely precise, we should define the two children of any node in some distinct fashion. However, to keep the notation of this discussion simple, we will always refer to a picture if we ever

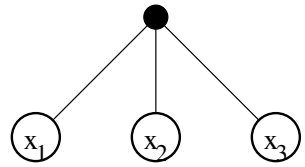

Figure 4-4: Graph representing the three variables to be collected by some agent.

$l(g)$ is the other black node downwards and to the left, and $r(g)$ is the node labelled by x_3.

Regarding terminology, since we will be associating functions and variables to nodes of the graph, we will abuse terminology and refer to the nodes of the graph by the function or variable associated with the node. For example, in Figure 4-4, we will refer to the "node associated with variable x_1" as simply the "node x_1". Let us now begin the discussion of how these graphs can represent possible information collection mechanisms.

Representation by Graphs

Nodes of this graph will represent variables and combinations of variables. (We put off discussion of functions associated with nodes until the next section.) Since there are three variables x_1, x_2, and x_3, we have three nodes corresponding to each variable drawn at the bottom of Figure 4-4. To indicate that the three variables should be collected at one of the agents, we have another node in the graph drawn above the original three nodes with edges connecting this node to the three, which is indicated by the black node and the three edge emanating downwards from it in Figure 4-4. Note that in this graph we have not yet indicated which agent is to have collected all three variables.

The key idea of using this graph to enumerate all reasonable data collection mechanisms is to substitute all black nodes with down degree greater than 2 with a subgraph of black nodes all with down degree 2. Figure 4-5 shows three possibilities of replacing the black node of Figure 4-4 by a subgraph with two black nodes each with down degree 2. Let us interpret the meaning of these resulting graphs which is always interpreted from the lowest nodes to the highest nodes.

- In Figure 4-5a, the interpretation is that some agent, which we refer to as i, collects the variables x_1 and x_2. Then, some agent (possibly i again) collects x_1 and x_2 from agent i (agent i collecting from itself means that agent i needs to do nothing) and collects the variable x_3.

- In Figure 4-5b, the interpretation is similar to that of Figure 4-5a except that

refer to a right or left child.

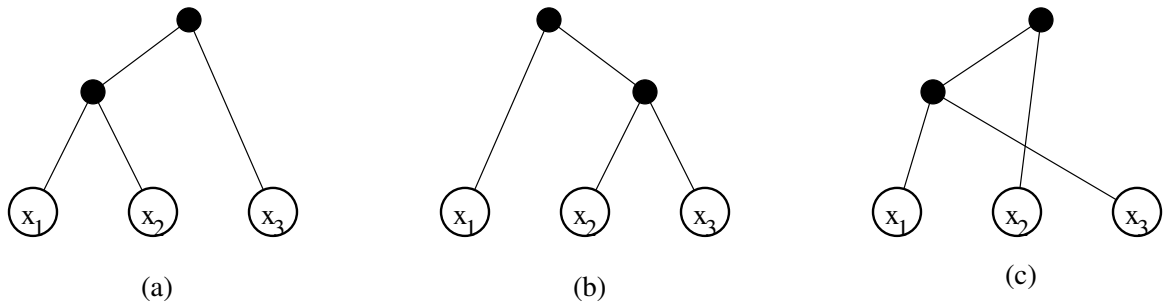

Figure 4-5: Graph of Figure 4-4 with all black nodes of down degree greater than 2 converted into a subgraph of several black nodes with down degree 2. The three graphs (a), (b), and (c) show the resulting graphs.

we have a permutation of variables. That is, some agent collects the variables x_2 and x_3. Then, some agent collects those two variables and collects x_1.

- Similarly, in Figure 4-5c, the interpretation is that some agent collects the variables x_1 and x_3. Then, some agent collects those variables and collects x_2.

At this point, we wish to specify exactly which agent is responsible for each of the intermediate collection steps. This is performed by assigning an agent to each node of the graph. The bottom three nodes corresponding to the variables are certainly constrained to be assigned to their respective agent. That is, the node x_i must be assigned to agent i for all $i \in \{1, 2, 3\}$. We will make a further restriction on the set of all possible *agent assignments* of nodes of the graph to agents. Each black node must be assigned to an agent assigned to one of its two children. Since there are at most two choices of agents for each black node, there are at most 2^b unique agent assignments for each of the graphs in Figure 4-5 where b is the number of black nodes.

Since $b = 2$ in our example, from the three graphs of Figure 4-5, we have a total of $3 \cdot 2^b = 12$ unique graphs with different agent assignments which are shown in Figure 4-6. Real data transfers are encoded on edges which connect nodes assigned to different agents. These edges are shown in bold in Figure 4-6. Let us interpret the data transfer encoded in each of these 12 graphs.

We will define some notation to represent the data transfers taking place. To represent agent $i \in \{1, 2, 3\}$ transferring a subset of variables $S \subset \{x_1, x_2, x_3\}$ to agent $j \in \{1, 2, 3\}$, we use the notation

$$i \xrightarrow{S} j \ .$$

To indicate the possibility of data transfers occurring concurrently, we write the data transfers in the same column. To indicate data transfers that must occur in sequence, we separate the data transfers by the symbol "$\Rightarrow$". Table 4.1 shows the data transfer

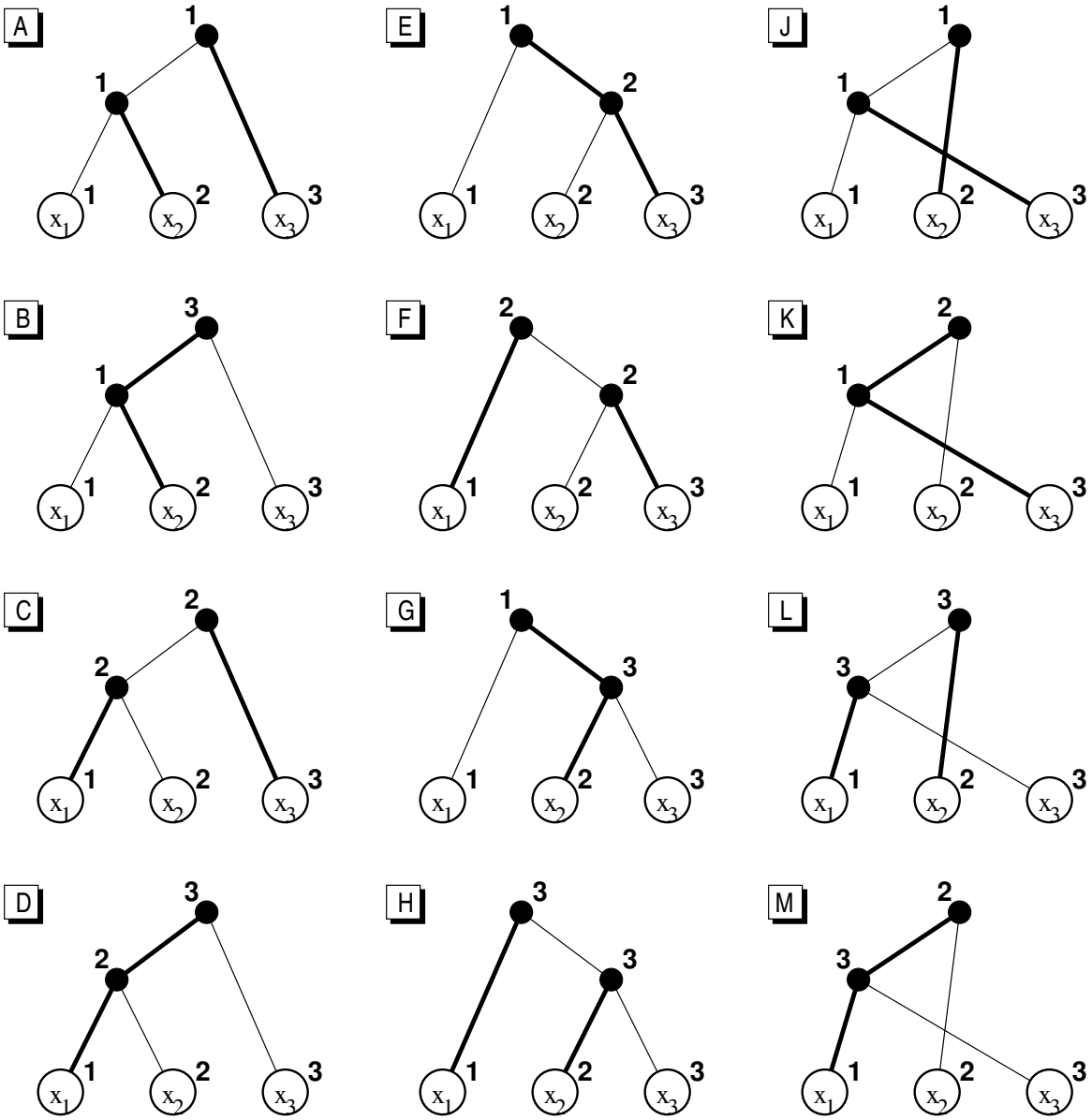

Figure 4-6: The 12 different graphs with unique agent assignments denoted by an agent label from $\{1, 2, 3\}$ next to each node. The four graphs in each column correspond to the respective unassigned graphs of Figure 4-5. The bold edges correspond to real information transfers encoded in the graph.

A	$2 \xrightarrow{\{x_2\}} 1$ $3 \xrightarrow{\{x_3\}} 1$	**G**	$2 \xrightarrow{\{x_2\}} 3 \Rightarrow 3 \xrightarrow{\{x_2,x_3\}} 1$
B	$2 \xrightarrow{\{x_2\}} 1 \Rightarrow 1 \xrightarrow{\{x_1,x_2\}} 3$	**H**	$1 \xrightarrow{\{x_1\}} 3$ $2 \xrightarrow{\{x_2\}} 3$
C	$1 \xrightarrow{\{x_1\}} 2$ $3 \xrightarrow{\{x_3\}} 2$	**J**	$3 \xrightarrow{\{x_3\}} 1$ $2 \xrightarrow{\{x_2\}} 1$
D	$1 \xrightarrow{\{x_1\}} 2 \Rightarrow 2 \xrightarrow{\{x_1,x_2\}} 3$	**K**	$3 \xrightarrow{\{x_3\}} 1 \Rightarrow 1 \xrightarrow{\{x_1,x_3\}} 2$
E	$3 \xrightarrow{\{x_3\}} 2 \Rightarrow 2 \xrightarrow{\{x_2,x_3\}} 1$	**L**	$1 \xrightarrow{\{x_1\}} 3$ $2 \xrightarrow{\{x_2\}} 3$
F	$1 \xrightarrow{\{x_1\}} 2$ $3 \xrightarrow{\{x_3\}} 2$	**M**	$1 \xrightarrow{\{x_1\}} 3 \Rightarrow 3 \xrightarrow{\{x_1,x_3\}} 2$

Table 4.1: The data collection mechanism corresponding to each graph of Figure 4-6.

encoded by each graph of Figure 4-6. Note that the data transfers taking place are the same for A and J, C and F, and H and L which says that distinct graphs with unique agent assignments do not necessarily lead to distinct data transfers. These 6 graphs encode a data collection mechanism where data transfers can occur concurrently. Furthermore, they are equivalent up to a permutation of agent labels. That is, for any choice of distinct labels $i, j, k \in \{1, 2, 3\}$, the data transfers occurring in A, C, F, H, J, and L are of the form,

$$\boxed{\begin{array}{l} i \xrightarrow{\{x_i\}} k \\ j \xrightarrow{\{x_j\}} k \end{array}} \,. \tag{4.1}$$

The other 6 graphs, namely B, D, E, G, K, and M, encode a data collection mechanism where data transfers must occur in sequence. These collection mechanisms can be considered a multi-hop type of data routing. For example, in B, variable x_2 hops from agent 2 to agent 1 followed by a hop from agent 1 to agent 3. These 6 graphs are equivalent up to a permutation of agent labels by which we mean that for any choice of distinct labels $i, j, k \in \{1, 2, 3\}$, the data transfers in B, D, E, G, K, and M are of the form,

$$\boxed{i \xrightarrow{\{x_i\}} j \;\Rightarrow\; j \xrightarrow{\{x_i,x_j\}} k} \,. \tag{4.2}$$

Now, we wish to make specific the communications involved that implement the encoded data transfers. We will refer to any specific implementation of the communications involved for data transfer as a data collection mechanism. Recall that

each agent has the ability to query values from other agents and send values to other agents so that we have two choices of implementing each data transfer

$$i \xrightarrow{S} j \ .$$

1. *("Pull" type)* We can implement a query, where agent j would query agent i for information about the variables in S. Such a query would be followed by a response by agent i which sends the values of the variables in S to agent j.

2. *("Push" type)* The other choice is for agent i to merely send the values of the variables in S to agent j.

Although the end result in terms of what data has been transferred is the same for either the "pull" type communication or the "push" type communication, there is a difference of what initiates the communication. To illustrate the difference, let us consider the variants of the data transfers encoded in mechanism A of Figure 4-6.

In the graph A, the bold edges are the real data transfers taking place, and we have discussed that each of these data transfers can be implemented as a "pull"-type or a "push"-type communication. We will indicate the particular choice of communication type by denoting a bold edge of the graph with either an arrow downward or an arrow upward. Edges denoted by an arrow downwards, which we refer to as *downlinks*, correspond to a "pull"-type of communication, and edges denoted by an arrow upwards, which we refer to as *uplinks*, correspond to a "push"-type of communication. Figure 4-7 shows the four possible choices of uplinks and downlinks for graph A.

To interpret the specific type of communication used to collect the variables, we interpret each of the nodes of the graph as encoding a local knowledge state[3] of the agent assigned to the node. For example, the nodes labelled with a variable x_i assigned to agent i encodes the local knowledge state that variable x_i is known by agent i for each $i \in \{1, 2, 3\}$. Furthermore, the node labelled by b_1 in Figure 4-7 encodes the local knowledge state that agent 1 has knowledge of both x_1 and x_2, and the node labelled by b_2 encodes the local knowledge state that agent 1 has knowledge of all variables x_1, x_2, and x_3. We can now interpret the specific type of communications encoded by each of the graphs in Figure 4-7. Let us introduce a notation for edges. We denote an edge which connects two nodes of the graph, say a and b, by a pair (a, b) or (b, a).[4]

(a) Since the edge (x_2, b_1) is an uplink, agent 2 will send the value of variable x_2 when it has observed x_2. Since the edge (x_3, b_2) is an uplink, agent 3 will send the value of variable x_3 to agent 1 when it has observed x_3. Notice that the two communications are initiated by the sender of data which is why we associate uplinks with a "push" type of communication.

[3]See Definition 2.1 for definition of a local knowledge state.

[4]We will not worry about directed edges in this discussion.

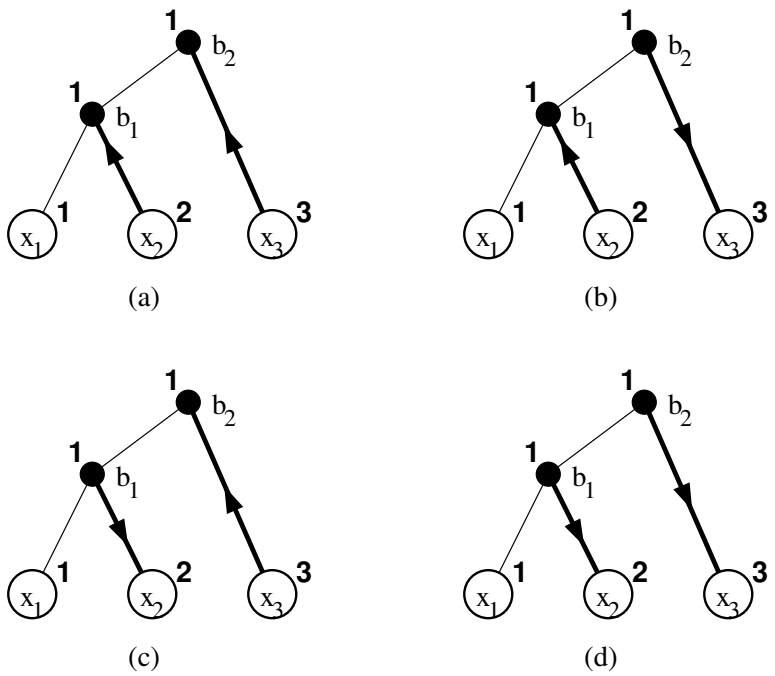

Figure 4-7: Graphs for data collection mechanism A with edges labelled by uplinks and downlinks. (a) Agents 2 and 3 send to agent 1. (b) Agent 2 sends to agent 1. Agent 1 queries agent 3. (c) Agent 1 queries agent 2. Agent 3 sends to agent 1. (d) Agent 1 queries agent 2. Then, agent 1 queries agent 3.

(b) Since the edge (x_2, b_1) is an uplink, agent 2 will send the value of variable x_2 when it has observed x_2. The downlink (b_2, x_3) encodes a query by agent 1 since node b_2 is assigned to agent 1. The local knowledge state of agent 1 which initiates the query encoded by the downlink is the local knowledge state encoded by the node b_1 and not b_2. The reason for this is that the local knowledge state encoded by b_2 would be the local knowledge state of agent 1 *after* the query has taken place. However, we need to know the local knowledge state of agent 1 which initiates the query *before* the query takes place. Thus, the downlink encodes that agent 1 is to query agent 3 for x_3 when agent 1 knows both x_1 and x_2.

To summarize, agent 2 sends x_2 to agent 1 once it has observed x_2. Agent 1 queries agent 3 for x_3 when it knows both x_1 and x_2. We associate downlinks with a "pull" type of communication because the communication is initiated by the receiver of data.

(c) Since (b_1, x_2) is a downlink, agent 1 will query agent 2 for x_2 when agent 1 observes x_1. Since (x_3, b_2) is an uplink, agent 3 sends x_3 to agent 1 when agent 3 observes x_3.

(d) Since (b_1, x_2) is a downlink, agent 1 will query agent 2 for x_2 when agent 1 observes x_1. Since (b_2, x_3) is a downlink, agent 1 will query agent 3 for x_3 when agent 1 knows both x_1 and x_2.

Note that the above four data collection mechanisms are all derived from the same graph A which means that the sequence of data transfers is the same for all four mechanisms although the sequence of "push" and "pull" type communications for implementing the data transfers differ. We mentioned before that graphs A and J of Figure 4-6 encode the same data transfer; however, the set of data collection mechanisms derived from the two graphs are different. For example, if all bold edges of A are chosen to be downlinks, the resulting collection mechanism is that agent 1 queries for x_2 from agent 2 followed by a query for x_3 from agent 3. On the other hand, if all bold edges of J are chosen to be downlinks, the resulting collection mechanism is that agent 1 queries for x_3 from agent 3 followed by a query for x_2 from agent 2. Thus, the order of queries from the remote agents 2 and 3 is different. The conclusion is that the specific topology of the graph will result in a different set of possible data collection mechanisms.

By extracting what local knowledge states initiate communications from the graphs, we have essentially defined a set of communication algorithms, one for each agent. In particular to the four data collection mechanisms described above, the communication algorithms for each agent are summarized below.

(a) Communication Algorithms

Agent 1 *nothing*

Agent 2 Given x_2, send x_2 to agent 1.

Agent 3 Given x_3, send x_3 to agent 1.

(b) Communication Algorithms

Agent 1 Given x_1, x_2, query for x_3 from agent 3.

Agent 2 Given x_2, send x_2 to agent 1.

Agent 3 *nothing*

(c) Communication Algorithms

Agent 1 Given x_1, query for x_2 from agent 2.

Agent 2 *nothing*

Agent 3 Given x_3, send x_3 to agent 1.

(d) Communication Algorithms

Agent 1 Given x_1, query for x_2 from agent 2.
Given x_1, x_2, query for x_3 from agent 3.

Agent 2 *nothing*

Agent 3 *nothing*

These communication algorithms are distributed algorithms where each agent can locally decide what communications to initiate. Furthermore, the global behavior which was designed into this collection mechanism is that one of the agents ends up with knowledge of all variables x_1, x_2, and x_3 so that the function f can be evaluated. We were able to design this global behavior into the communication algorithms because all graph topologies were constructed from the original graph of Figure 4-4.

Note that the above communication algorithms were deterministically extracted from the graphs of Figure 4-7. Thus, we have reduced the problem of constructing a set of communication algorithms to the problem of constructing a particular kind of graph representation. To summarize, the goal for determining the most efficient data collection mechanism will be to construct a graph topology, choose an agent assignment, and finally, choose uplinks and downlinks so that the extracted set of communication algorithms minimizes the computational resources expended.

The graphical representation presented in this section which represents a set of reasonable data collection mechanisms is a part of what we refer to as the hierarchy. What is missing in the representation so far is a representation of the partial processing that can occur during data collection to result in an information collection mechanism. This will be accomplished by associating functions with nodes of the graph. The resulting representation is what we refer to as a hierarchy. We will introduce the idea behind incorporating partial processing into a data collection mechanism in the next section.

4.1.2 Question 2: Incorporating Partial Processing

The graphs of Figure 4-6, pending a choice of uplinks and downlinks, represent a set of reasonable data collection mechanisms for collecting data from 3 distributed autonomous agents in order to calculate any 3 input function $f(x_1, x_2, x_3)$. To reap the possible benefits of partial processing, it is necessary to know more about the function than the inputs it depends on.

Concept of Partial Processing

Let us first discuss the conceptual idea behind partial processing before presenting a specific example. The fundamental problem is the following.

Given a surjective function of two variables

$$f : X_1 \times X_2 \to Y$$

written

$$y = f(x_1, x_2)$$

where all the sets are finite with cardinality given by

$$\begin{aligned} |X_1| &= n_1 \\ |X_2| &= n_2 \\ |Y| &= m \ , \end{aligned}$$

when given $x_1 \in X_1$, what partial processing on x_1 can we perform so that the representation of the information in x_1 about f is minimal?

By minimal representation of the information in x_1, we mean the smallest cardinality space Z such that there exists a pair of functions

$$f' : Z \times X_2 \to Y$$

and

$$g : X_1 \to Z$$

such that

$$f(x_1, x_2) = f'(g(x_1), x_2) \ .$$

The function g is the partial processing on x_1.

Let us give the solution to an easy case first. Say we knew the value of x_2. Let us rewrite the notation $f(x_1, x_2)$ as $f_{x_2}(x_1)$ so that we consider f_{x_2} to be a function of x_1 parameterized by x_2. Then, clearly, the minimal cardinality of Z is the cardinality of Y,

$$|Z| = |Y|$$

because f is surjective. As for the functions f' and g, the following definitions work. Let the space Z be $\{z^1, \ldots, z^m\}$ and Y be $\{y^1, \ldots, y^m\}$.

$$g(x_1) = \begin{cases} z^1 & , \ x_1 \in f_{x_2}^{-1}(y^1) \\ z^2 & , \ x_1 \in f_{x_2}^{-1}(y^2) \\ \vdots & \vdots \\ z^m & , \ x_1 \in f_{x_2}^{-1}(y^m) \end{cases} \quad , \quad f'(z, x_2) = \begin{cases} y^1 & , \ z = z^1 \\ y^2 & , \ z = z^2 \\ \vdots & \vdots \\ y^m & , \ z = z^m \end{cases}$$

where $f_{x_2}^{-1}$ denotes the set inverse of the function f_{x_2}. We can verify that given x_2

$$f(x_1, x_2) = f'(g(x_1), x_2) \ .$$

Note that the definition of g depends on x_2. The essential aspect of this partial processing on x_1 is that a partition is set up over the elements of X_1 so that "information about x_1 for evaluating f" is about the membership of x_1 in one of the partitions. The partition is

$$P_{x_2} = \{f_{x_2}^{-1}(y)\}_{y \in Y} \ . \tag{4.3}$$

What the exact elements of Z are is of no consequence as long as elements of X_1 in the same partition get mapped to the same element in Z and elements of X_1 in different partitions get mapped to different elements of Z. For instance, we can switch z^1 and z^2 in the definition of g above to get

$$g'(x_1) \quad = \quad \begin{cases} z^2 & , \ x_1 \in f_{x_2}^{-1}(y^1) \\ z^1 & , \ x_1 \in f_{x_2}^{-1}(y^2) \\ \vdots & \vdots \\ z^m & , \ x_1 \in f_{x_2}^{-1}(y^m) \end{cases}$$

and, as long as f' is changed correctly as in

$$f''(z, x_2) \quad = \quad \begin{cases} y^1 & , \ z = z^2 \\ y^2 & , \ z = z^1 \\ \vdots & \vdots \\ y^m & , \ z = z^m \end{cases} \quad ,$$

we can verify that

$$f(x_1, x_2) = f''(g'(x_1), x_2) \ .$$

Thus, the invariant aspect is the partition of X_1.

Now, let us consider the case when we do not know x_2. Then, the minimal partition, which is the collection of distinct subsets of X_1 which cover X_1 with the minimum number of elements, is given by

$$P = \bigwedge_{x_2 \in X_2} P_{x_2}$$

where P_{x_2} was defined in Eqn. 4.3. The operator $\wedge$ on two partitions P_1 and P_2 is given by

$$P_1 \wedge P_2 = \{p \subset X_1 : p \neq \emptyset, p = p_1 \cap p_2 \text{ for some } p_1 \in P_1 \text{ and} p_2 \in P_2\} \ .$$

$P_1 \wedge P_2$ is the minimal partition which is a refinement of both P_1 and P_2. Thus, P above is the minimal partition which is a refinement of all partitions P_{x_2} for $x_2 \in X_2$. Thus, because P is a refinement of all minimal partitions for any value of x_2, we can conclude that there exists a g and f' with $|Z| = |P|$ such that

$$f(x_1, x_2) = f'(g(x_1, x_2)) \ .$$

Thus, in the following text, when we refer to partial processing, we mean a function g above. Also, the information contained in the data refers to the output of the partial processing g.

The above discussion extends to functions of more variables since the spaces X_1 and X_2 could both be Cartesian products of several spaces. Furthermore, we can extend to the case when all spaces are continuous, where the notion of minimality of a partition must be replaced with minimal σ-algebras.

Partial Processing in Information Collection

To demonstrate the possible benefits of partial processing, let us choose a specific function for the system to evaluate. The system task is to evaluate the following function dependent on the three variables x_1, x_2, and x_3

$$f(x_1, x_2, x_3) = \begin{cases} (x_1 - x_2)^2 - x_3 & , \ x_1 + x_2 > 0 \\ (x_1 - x_2)^2 & , \ x_1 + x_2 \leq 0 \end{cases} \ . \tag{4.4}$$

There are two pieces in the expression for f above that are important. The first is what we refer to as the *computations*, which are the expressions

$$\begin{array}{l} (x_1 - x_2)^2 - x_3 \\ (x_1 - x_2)^2 \ . \end{array} \tag{4.5}$$

The results of these computations are what we are interested in evaluating. The second part, which is the more important part in terms of determining the best data collection mechanism, is what we refer to as the *triggers*, which are the expressions

$$\begin{array}{l} x_1 + x_2 > 0 \\ x_1 + x_2 \leq 0 \ . \end{array} \tag{4.6}$$

These triggers determine when the associated computation should be evaluated. Note that the triggers are the concrete embodiment of information-based computation. That is, based on the information that the trigger $x_1 + x_2 > 0$ is true, we must evaluate

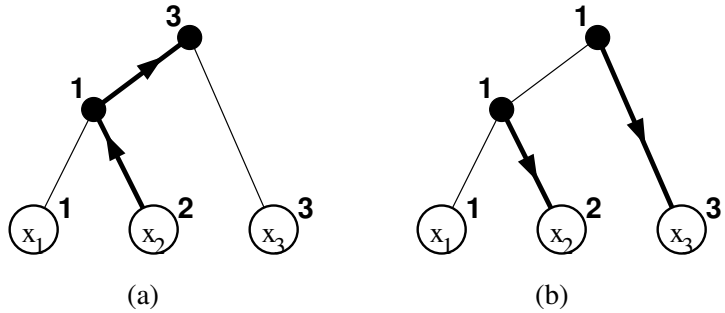

Figure 4-8: (a) Graph B from Figure 4-6 with all bold edges chosen to be uplinks. (b) Graph A from Figure 4-6 with all bold edges chosen to be downlinks.

the computation $(x_1 - x_2)^2 - x_3$. On the other hand, based on the information that the trigger $x_1 + x_2 \leq 0$ is true, we must evaluate the computation $(x_1 - x_2)^2$. We will use the term "trigger" more broadly than specifically referring to the conditions of a function definition as in Eqn. 4.4 above.

Note that to evaluate a trigger or a computation, we must collect the appropriate data or information needed to evaluate them. We say that the triggers are more important for data collection because the triggers determine which computation is to be evaluated, and thus, we must evaluate the triggers *before* evaluating the computation. The effect of this is that there are actually two kinds of partial processing that can occur during data collection: (1) partial processing to evaluate the triggers and (2) partial processing to evaluate the computations.

We intend to demonstrate the following two effects partial processing can have during data collection.

1. The first effect is that partial processing can reduce the amount of bits that must be communicated because the information contained in the data requires less bits to represent than the data itself.

2. The second effect is that partial processing can determine what information is needed so that the sensing process of the agents can be controlled intelligently. This effect can be important if observing measurements from sensors is expensive.

We address each effect in the respective sections below.

4.1.2.1 Reducing Data Transfer through Partial Processing

Let us consider the completely "push"-type data collection mechanism associated with graph B of Figure 4-6 which is shown in Figure 4-8a. This data collection mechanism involves two communications.

- First, agent 2 sends x_2 to agent 1.
- Then, agent 1 sends both x_1 and x_2 to agent 3.

Partial processing can be incorporated into this data collection mechanism before each communication.

- The partial processing that can occur before sending x_2 is to extract the information in x_2 needed to evaluate the computations (Eqn. 4.5) and the information in x_2 needed to evaluate the triggers (Eqn. 4.6). It is clear from the expressions of the computations and triggers of f that we must send x_2 as is, so that there is no partial processing to be performed.
- In the second communication, agent 1 is to send the information contained in x_1 and x_2 to agent 3. Since the computations depend on x_1 and x_2 only on their squared difference, the information about x_1 and x_2 is contained in the value of

  $$(x_1 - x_2)^2 \ .$$

  Thus, the information to be sent in the second communication is the value of $(x_1+x_2)^2$. Assuming that sending a floating point number requires sending k_{float} bits, the partial processing has reduced the number of bits sent from $2k_{float}$ bits, for sending the values of the variables in $\{x_1, x_2\}$, by a factor of $\frac{1}{2}$ to k_{float} bits, for sending the value of $(x_1 - x_2)^2$. Hence, we have demonstrated the possible reduction in the number of bits to communicate when partial processing of the computations is incorporated into the data collection mechanism.

  We can also perform partial processing for evaluating the triggers. When agent 1 has knowledge of both x_1 and x_2, it can evaluate the triggers. If it turned out that $x_1+x_2 \leq 0$, then another possible reduction in communication is that agent 1 need not send $(x_1 - x_2)^2$ to agent 3 because the function f does not depend on x_3 in this case. Thus, certain communications can become unnecessary depending entirely on the results of partial processing on the triggers. Notice that if agent 1 does not send $(x_1 - x_2)^2$ when $x_1 + x_2 \leq 0$ that the evaluation of f ultimately resides in agent 1. On the other hand, when $x_1 + x_2 > 0$, agent 1 will send the information for the computations, namely $(x_1 - x_2)^2$, contained in the two variables x_1 and x_2 to agent 3. Then, the evaluation of f ultimately resides in agent 3. As for the information about the triggers to be sent to agent 3, there is no extra information to be sent because both triggers have already been evaluated by agent 1. Just the mere fact that agent 3 receives the information $(x_1 - x_2)^2$ indicates to agent 3 that $x_1 + x_2 > 0$. Note that determining whether certain communications are needed or not is determined completely by partial processing of the triggers and not by partial processing of the computations.

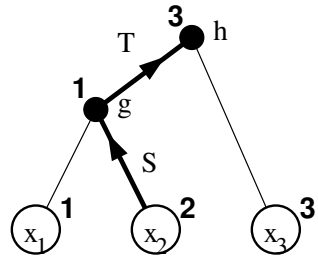

Figure 4-9: The hierarchy for the graph of Figure 4-8a showing the partial computations and trigger sets which initiate communications.

The reductions in communication described above come at the expense of performing partial processing so that the complexity of partial processing must be weighed with the reduction in communication. In our sensor networks, the reduction in communication generally outweighs the extra partial processing needed so that performing as much partial processing as possible is desirable.

We can indicate the partial processing occurring by associating two functions

$$g_a : \mathcal{Z}_l \times \mathcal{Z}_r \to \mathcal{Z}_o$$
$$R_a : \mathcal{Z}_l \times \mathcal{Z}_r \to \mathcal{Z}_o$$

with every node a of the graph which is not an input node. The function g_a represents the partial processing of the computations, and the function R_a is the partial processing of the triggers. The space $\mathcal{Z}_l$ is the space of data or information coming from the left child of node a, and $\mathcal{Z}_r$ is the space of data or information coming from the right child of node a. Finally, the space $\mathcal{Z}_o$ is the space that the result of both partial processing lies in. Secondly, to indicate the possibility of forgoing certain communications entirely, edges of the graph will be labelled with a subset of values, referred to as the *trigger sets*, which indicates the subset of values resulting from partial processing of the triggers for which the associated communication is to be initiated.

Figure 4-9 shows these functions associated with nodes and trigger sets associated with uplinks for the graph of Figure 4-8a. In this example, the spaces for the left input, right input, and output of all functions is the space of reals, $\mathbb{R}$. We consider the value traversing each edge (a, b) where a and b are nodes and a is below b to be the value output from the computation associated with node a. In Figure 4-9, the value traversing edge (x_1, g) is the value of variable x_1. The value traversing edge (x_2, g) is the value of variable x_2. The value traversing edge (x_3, h) is the value of variable x_3. Finally, the value traversing edge (g, h) is the output of the evaluation of g.[5] Since we

[5]By associating functions with nodes, the graph has a combinational circuit interpretation where nodes are gates and edges are interconnections. Edges emanating upwards from a node can be con-

have two kinds of partial processing, one for computations and another for triggers, we have two values associated with each edge which we specify by superscript, x^c for computations and x^t for triggers. For each node with down degree 2, we denote by x_l the value of the edge which connects this node to its left child, and we denote by x_r the value of the edge which connects this node to its right child. Thus, for every node, there are four possible values which the two functions associated with the node may depend on, x_l^c, x_r^c, x_l^t, and x_l^t.

Formalizing the explanation of the partial processing involved for the computations, we specify the function g and h associated with the black nodes of Figure 4-9 by

$$\begin{aligned} g(x_l^c, x_r^c, x_l^t, x_r^t) &= (x_l^c - x_r^c)^2 \\ h(x_l^c, x_r^c, x_l^t, x_r^t) &= x_l^c - x_r^c \quad . \end{aligned}$$

Substituting the particular values of x_l^c and x_r^c and getting rid of the dependence on x_l^t and x_r^t according to the graph in Figure 4-9, we have

$$\begin{aligned} g(x_1, x_2) &= (x_1 - x_2)^2 \\ h(g(x_1, x_2), x_3) &= g(x_1, x_2) - x_3 = (x_1 - x_2)^2 - x_3 \quad . \end{aligned}$$

The partial processing associated with the triggers, which we denote by R_g and R_h for nodes g and h respectively in the figure, are

$$\begin{aligned} R_g(x_l^c, x_r^c, x_l^t, x_r^t) &= x_l^t + x_r^t \\ R_h(x_l^c, x_r^c, x_l^t, x_r^t) & \quad \textit{no partial processing at node } h \end{aligned}$$

where R_g is specifically

$$R_g(x_1, x_2) = x_1 + x_2 \ .$$

The trigger sets encoding what initiates the respective communications are

$$\begin{aligned} S &= \mathbb{R} \\ T &= \{x \in \mathbb{R} : x > 0\} \quad . \end{aligned}$$

Let us explain how these specifications work together in determining the two communications taking place as described above.

1. In the first communication where agent 2 sends x_2 to agent 1, the formalism is that the uplink labelled with S encodes that agent 2 is to send the variable x_2 to agent 1 if

$$x_2 \in S \ .$$

   Since $S = \mathbb{R}$, agent 2 always sends x_2 to agent 1.

sidered to be interconnections connected to the output of the gate, and edges emanating downwards from a node can be considered to be interconnections connected to the input of the gate. However, this interpretation is not to be taken too literally because we have uplinks and downlinks which control the flow of computations in this graph.

2. Upon receiving x_2, agent 1 then performs the partial processing for computation in node g[6] which is
$$g(x_1, x_2) = (x_1 - x_2)^2$$
since $x_l = x_1$ and $x_r = x_2$. Furthermore, the partial processing for triggers in node g is also evaluated,
$$R_g(x_1, x_2) = x_1 + x_2 \ .$$
Now, if $R_g(x_1, x_2) \in T$, we send the results of $g(x_1, x_2)$ to agent 3 as indicated by the uplink labelled with T if
$$R_g(x_1, x_2) \in T \ .$$
Thus, agent 1 sends $g(x_1, x_2)$ to agent 3 if $x_1 + x_2 > 0$, which is exactly the kind of behavior we desired. For general graphs, we also need to send the results of $R_g(x_1, x_2)$ if there are more triggers to evaluate at higher nodes.

What is not shown explicitly in the hierarchy is the evaluation of the function f by agent 1 in the node g when $(x_1 + x_2) \leq 0$. Also not shown is some declaration by the agent if they have determined locally that they have completed the global task. Thus, with each node, we also associate a *stopping condition* which is a subset of local knowledge states from which the agent will declare that the global task has been completed. Furthermore, there may be some residual computation that should be performed once the stopping condition has been satisfied. Let us give a description of the stopping conditions for the two black nodes.

1. *(Node g)* The stopping condition is
$$STOP_g = \{(x_l^t, x_r^t) \in \mathbb{R}^2 : R_g(x_l^t, x_r^t) \leq 0\} \ .$$
Then, since we have already computed g, agent 1 will declare that it has computed the global task.

2. *(Node h)* The stopping condition is when agent 3 is aware of information sent by agent 1
$$STOP_h = \{\text{event that value of } g \text{ was sent by agent 1}\} \ .$$
After evaluating
$$h = g - x_3 \ ,$$
agent 3 will declare that it has computed the global task.

[6]We abuse notation here. By "node g", we refer to the node of the graph labelled by a "g" in Figure 4-9. In the expression to follow, $g(x_1, x_2)$ refers to the function.

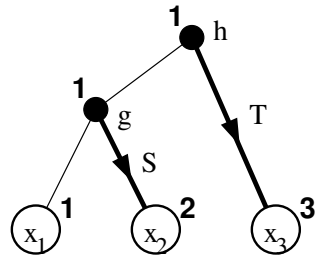

Figure 4-10: The hierarchy for the graph of Figure 4-8b showing the partial computations and trigger sets which initiate communications.

By choosing stopping conditions, the distributed system can declare when the global task has been computed. There is the possibility that more than one agent or no agents may declare that the global task has been computed. To ensure that only a single agent declares that the global task has been computed can be designed into the system by designing the stopping conditions to be mutually exclusive and collectively exhaustive. For example, the stopping conditions above are equivalent to

$$\begin{aligned} STOP_g &= \{(x_1, x_2) \in \mathbb{R}^2 : x_1 + x_2 \leq 0\} \\ STOP_h &= \{(x_1, x_2) \in \mathbb{R}^2 : x_1 + x_2 > 0\} \ , \end{aligned}$$

and since $STOP_g \cap STOP_h = \emptyset$ and $STOP_g \cup STOP_h = \mathbb{R}^2$, which means that the stopping conditions are mutually exclusive and collectively exhaustive, either agent 1 will declare to have computed the global task or agent 3 will declare to have computed the global task. This completes the presentation of the hierarchy for the data collection mechanism of Figure 4-8a.

Let us quickly present the completely "pull"-type communication example of Figure 4-8b which differs from the "push"-type communication in that the trigger sets S and T associated with downlinks are tested with values traversing different edges. Figure 4-10 shows the graph with computations at nodes and trigger sets associated with downlinks. The partial processing of the computations, g and h, are

$$\begin{aligned} g(x_l^c, x_r^c, x_l^t, x_r^t) &= (x_l^c - x_r^c)^2 \\ h(x_l^c, x_r^c, x_l^t, x_r^t) &= x_l^c - x_r^c \ . \end{aligned}$$

The partial processing of the triggers, R_g and R_h, are

$$\begin{aligned} R_g(x_l^c, x_r^c, x_l^t, x_r^t) &= x_l^t + x_r^t \\ R_h(x_l^c, x_r^c, x_l^t, x_r^t) & \quad \textit{no partial processing at node } h \ . \end{aligned}$$

Since the trigger sets S and T are associated with downlinks, S and T are not subsets of the space of values for variables x_1 and x_3 respectively as they would have been if

the edges were uplinks. Instead, S is a subset of the information traversing the edge (x_1, g), which is the space of values for x_1. T is a subset of the information traversing the edge (g, h) which is the space of values for the output of R_g. In general, the trigger set associated with a downlink on edge (a, b) where node a is below node b are subsets of the collective space of values on the other edges which are not downlinks emanating downwards from node b. The trigger sets encoding when the respective communications are to take place are

$$\begin{aligned} S &= \mathbb{R} \\ T &= \{x \in \mathbb{R} : x > 0\} \ . \end{aligned}$$

As before, there are two stopping conditions associated with nodes g and h which are

$$\begin{aligned} STOP_g &= \{(x_l^t, x_r^t) \in \mathbb{R}^2 : R_g(x_l^t, x_r^t) \leq 0\} \\ STOP_h &= \{x_l^t \in \mathbb{R} : x_l^t > 0\} \ . \end{aligned}$$

Let us now describe the data collection and partial processing mechanism encoded by this hierarchy. We can imagine that the nodes of the graph are like states in a finite state machine. So, initially we start in the node x_1 in the hierarchy of Figure 4-10. At first, agent 1 has access to its own variable x_1. Since node x_1 is connected by an edge upwards to node g and node g is also assigned to agent 1, we can imagine transitioning "states" to that node. Since the right edge downward from node g is a downlink, agent 1 tests whether its current local knowledge state satisfies S. That is, the test is to see if x_1, being the only variable that agent 1 knows currently, is an element of S. Since $S = \mathbb{R}$, the test is true, and agent 1 queries agent 2 for x_2. After receiving x_2, agent 1 evaluates the partial processing for computations g and triggers R_g, so that agent 1 has evaluated

$$\begin{aligned} g(x_1, x_2) &= (x_1 - x_2)^2 \\ R_g(x_1, x_2) &= x_1 + x_2 \ . \end{aligned}$$

At this point, agent 1 is currently in the "state" of node g. So, we test the stopping condition of node g,

$$(x_l^t, x_r^t) = (x_1, x_2) \in STOP_g$$

which reduces to testing whether

$$x_1 + x_2 \leq 0 \ .$$

If true, agent 1 declares it has computed the global task which is the evaluation of $g(x_1, x_2) = (x_1 - x_2)^2$. If not true, agent 1 continues.

Since node h is connected by an edge upwards from node g and the agent assignments are both to agent 1, we can imagine transitioning "states" again to node h. At this point, agent 1 has knowledge of x_1, x_2, $R_g(x_1, x_2) = x_1 + x_2$, and

$g(x_1, x_2) = (x_1 - x_2)^2$. Since there is a downlink to node x_3 from node h, agent 1 tests whether the value from partial processing on triggers on the edge (g, h), which is $R_g(x_1, x_2)$, satisfies T which is to test

$$R_g(x_1, x_2) \in T \ .$$

This reduces to testing whether

$$x_1 + x_2 > 0 \ .$$

If true, agent 1 queries for x_3 and evaluates h. If false, no query is made, and agent 1 can not evaluate h. However, since agent 1 would have stopped at node g if $x_1+x_2 \leq 0$, this test will always be true, and agent 1 will query for x_3. Finally, agent 1 tests for the stopping condition which is that the left input from the trigger partial processing satisfies $STOP_h$, which is

$$x_l^t = R_g(x_1, x_2) \in STOP_h \ .$$

Since the stopping condition reduces to

$$x_1 + x_2 > 0$$

which is always true if agent 1 had not stopped at node g, agent 1 declares that it has computed the global task. The result of the global task is the output of the computation h which is

$$(x_1 - x_2)^2 - x_3 \ .$$

To summarize, each node of the graph is associated with two functions, one for partial processing of the computations and another for partial processing of the triggers. The ones which we did not explicitly specify like for the nodes associated with each variable x_i for $i \in \{1, 2, 3\}$ can be considered to be computing the identity function for both computations and triggers,

$$x = f_{identity}(x)$$

for whatever space x lies in. Furthermore, we associated a trigger set for each uplink or downlink which indicated when the respective communication is to take place. Finally, each node is given a stopping condition which indicates when the agent assigned to the node should declare that it has computed the global task. Note that the trigger sets associated with uplinks and downlinks and the stopping conditions are used to test the results from partial processing of the triggers. Thus, the triggers play a primary role in determining the resulting information collection mechanism encoded by this representation. With all of these elements added to the graph, the resulting representation is what refer to as the hierarchy.

4.1.2.2 Controlling Sensing through Partial Processing

Each of the three agents can sense its respective variable, meaning that it can observe its respective variable. As with any operation performed by an agent, sensing also consumes energy so that minimizing the amount of sensing is desirable for constructing efficient algorithms. A particularly interesting case is when different kinds of sensing are associated with different computational costs, so that choosing what to sense based on what has already been sensed in order to extract maximal information with minimal cost is the goal. This is not unlike the goal of collecting needed data while minimizing the cost of communication, and, in fact, the hierarchy can be used for this problem of sensing as well. Let us consider the hierarchies of Figure 4-9 and Figure 4-10 to demonstrate this idea.

In the hierarchy of Figure 4-9, the variables x_1 and x_2 must always be sensed in order for the information collection mechanism to compute the global task correctly. Thus, agents 1 and 2 must always sense their respective variables. On the other hand, the variable x_3 affects the resulting evaluation of the function f only when $x_1+x_2 \geq 0$. Thus, agent 3 should not sense the value of x_3 unless agent 3 knows that $x_1+x_2 \geq 0$. Note that the triggers again are the more important functions to evaluate in order to determine what to sense. For the information collection mechanism encoded by this hierarchy, the test for when agent 3 should sense for x_3 is when agent 3 receives the output of g from agent 1 since this information is sent exactly when $x_1 + x_2 \geq 0$.

Now, considering the hierarchy of Figure 4-10 where the information collection mechanism is essentially for agent 1 to query for x_2 from agent 2, and based on the value of $x_1 + x_2$, to query for x_3 from agent 3, the simple sensing rule could be that

1. agent 1 always senses for x_1,
2. agent 2 senses for x_2 only if queried, and
3. agent 3 also senses for x_3 only if queried.

Essentially, agent 2 will always sense for x_2 because agent 1 always queries for x_2. On the other hand, agent 1 will only query for x_3 if $x_1 + x_2 > 0$ so that agent 3 does not always observe x_3. Again, note that it is the trigger $x_1 + x_2 > 0$ that is determining when agent 3 should observe x_3.

Now that we have discussed how communications can be reduced through partial processing and discussed how sensing can also be refined for any given hierarchy, we will discuss how to measure the computational resource requirements of the encoded information collection mechanism directly from these hierarchies in the next section.

4.1.3 Question 3: Measuring Computational Resources

The computational resources we wish to measure from the hierarchies which represent an information collection mechanism, will be the computational resource cost due to

sensing, processing, and communication by each of the agents. There are several choices for optimizing some global statistic of the computational cost encoded by a hierarchy. These include minimizing the worst case maximal computational cost or minimizing the average computational cost. The constituent cost elements for determining any of these statistics is based on two categories of elementary quantities:

- the trigger which initiates sensing, processing, or communication, and
- the computational cost of the particular sensing, processing, or communication operation.

In particular, we can reduce triggers to be events $\mathcal{F}$ of some sample space Ω of the possible realizations of sensed measurements collected by the system where $(\Omega, \mathcal{F})$ is a measurable space. The global cost is a measurable function

$$\mathbf{cost}_H : \Omega \to \mathbb{R}^+$$

returning non-negative reals derived for a particular hierarchy H. Given a finite index set I which indexes the set of elementary operations of sensing, processing, and communication encoded in the hierarchy H, we have a collection of triggers $\{F_i^H\}_{i \in I}$, where each $F_i^H \in \mathcal{F}$, and a collection of computational costs $\{c_i^H\}_{i \in I}$, where each $c_i^H \in \mathbb{R}^+$, associated with each elementary operation $i \in I$. We assume that the global cost function has the form

$$\mathbf{cost}_H(\omega) = \sum_{i \in I} 1_{F_i^H}(\omega) c_i^H \tag{4.7}$$

where $1_{F_i^H} : \Omega \to \{0, 1\}$ is the indicator function of the event F_i^H defined by

$$1_{F_i^H}(\omega) = \begin{cases} 1 & , \ \omega \in F_i^H \\ 0 & , \ \text{otherwise} \end{cases} .$$

Each individual operation of sensing, processing, and communication contributes additively to the global cost depending on when these operations are triggered. Such an additive model is reasonable for global cost statistics like energy consumption.

The worst case cost of a hierarchy H is given by

$$\sup_{\omega \in \Omega} \mathbf{cost}_H(\omega) \ .$$

By specifiying a probability measure P on the measurable space $(\Omega, \mathcal{F})$, the average cost of a hierarchy H is given by

$$\int_\Omega \mathbf{cost}_H(\omega) dP(\omega) = \sum_{i \in I} c_i^H P(F_i^H)$$

where the right hand side holds true if the global cost function is of the form in Eqn. 4.7. Then, the optimal solution for the most efficient information collection mechanism is given by the hierarchy $\hat{H}$ which satisfies

$$\begin{aligned}
(i) \quad \hat{H} &= \arg_{H\in\mathcal{H}} \min \left[\sup_{\omega\in\Omega} \mathbf{cost}_H(\omega)\right] \\
(ii) \quad \hat{H} &= \arg_{H\in\mathcal{H}} \min \left[\int_{\Omega} \mathbf{cost}_H(\omega) dP(\omega)\right]
\end{aligned}$$

where (i) corresponds to minimizing the worst case and (ii) corresponds to minimizing the average case. $\mathcal{H}$ denotes the set of all hierarchies.

We will discuss how to extract the individual costs $\{c_i^H\}_{i\in I}$ and the triggers $\{F_i^H\}_{i\in I}$ encoded in a hierarchy H in three subsections dedicated to the individual costs due to sensing, processing, and communication. We use as examples the hierarchies of Figure 4-9 and Figure 4-10, which we will refer to as H_{4-9} and H_{4-10} respectively. The measurable space we use is

$$(\mathbb{R}^3, \mathfrak{B}(\mathbb{R}^3))$$

which is the space of all triples (x_1, x_2, x_3) of the three variables observed by the three agents. By $\mathfrak{B}(\mathbb{R}^3)$, we mean the Borel σ-algebra of $\mathbb{R}^3$.

4.1.3.1 Sensing Costs

Costs due to sensing arise whenever the sensor of an agent is turned on and the sensor measurement is observed. The sensor measurements may actually be the result of some low-level signal processing on the actual raw sensor reading, so that resources are expended to process the raw sensor reading into the sensor measurement. For example, the intensity, frequency bandlimited intensity, TDOA, and magnetometer measurements described in Section 2.2 are all computed by some low-level signal processing.

For the example system architecture, the costs due to sensing arise from observing the three variables x_1, x_2, x_3 which we will denote by c_1, c_2, and c_3. From the discussion in Section 4.1.2.2 of when these variables must be observed for the two hierarchies H_{4-9} and H_{4-10}, it was shown that x_1 and x_2 are always observed and x_3 need only be observed when $x_1 + x_2 > 0$. Thus, the three triggers corresponding to c_1, c_2, and c_3 are

$$\begin{aligned}
F_1^{H_{4-9}} = F_1^{H_{4-10}} &= \mathbb{R}^3 \\
F_2^{H_{4-9}} = F_2^{H_{4-10}} &= \mathbb{R}^3 \\
F_3^{H_{4-9}} = F_3^{H_{4-10}} &= \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 + x_2 > 0\}
\end{aligned}$$

4.1.3.2 Processing Costs

Costs due to processing arise any time a function associated with any node of the hierarchy is evaluated and a test is performed to determine whether to initiate a communication or observe a measurement.

For hierarchy H_{4-9}, there is a cost c_g, c_h, and c_{R_g} for evaluating g, h, and R_g respectively. Furthermore, there is a cost c_S and c_T for determining whether $x_2 \in S$ and $R_g(x_1, x_2) \in T$ respectively, which tests whether to initiate the corresponding communications. Furthermore, depending on how detailed we wish to get, there are routines which determine when to sense for the variables x_1, x_2, and x_3, the cost of testing for stopping conditions, and the cost of possible residual computations after a stopping condition is reached; however, we will not go this far into detail. The triggers for the processing costs c_g, c_h, c_{R_g}, c_S, and c_T are

$$\begin{aligned}
F_g^{H_{4-9}} &= \mathbb{R}^3 \\
F_h^{H_{4-9}} &= \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 + x_2 > 0\} \\
F_{R_g}^{H_{4-9}} &= \mathbb{R}^3 \\
F_S^{H_{4-9}} &= \mathbb{R}^3 \\
F_T^{H_{4-9}} &= \mathbb{R}^3 \ .
\end{aligned}$$

The trigger $F_T^{H_{4-9}}$ is the event that agent 1 *tests* whether $R_g(x_1, x_2) \in T$. This test is performed any time R_g is evaluated which is all of $\mathbb{R}^3$.

For hierarchy H_{4-10}, there is again a cost c_g, c_h, and c_{R_g} for evaluating computations g, h, and R_g respectively. Furthermore, there is again a cost c_S and c_T for determining whether $x_1 \in S$ and $R_g(x_1, x_2) \in T$ respectively. Furthermore, other costs for determining when to sense for the variables x_1, x_2, and x_3, testing for stopping conditions, and evaluating residual computations after a stopping condition has been reached can be specified although we will not delve this far into detail. The triggers for the costs c_g, c_h, c_{R_g}, c_S, and c_T are

$$\begin{aligned}
F_g^{H_{4-10}} &= \mathbb{R}^3 \\
F_h^{H_{4-10}} &= \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 + x_2 > 0\} \\
F_{R_g}^{H_{4-10}} &= \mathbb{R}^3 \\
F_S^{H_{4-10}} &= \mathbb{R}^3 \\
F_T^{H_{4-10}} &= \{(x_1, x_2, x_2\} \in \mathbb{R}^3 : x_1 + x_2 > 0\} \ .
\end{aligned}$$

Note that in both H_{4-9} and H_{4-10} that the evaluation of h only occurs some of the time because the associated trigger F_h is a strict subset of $\mathbb{R}^3$. In H_{4-10}, the trigger for evaluating the test $R_g \in T$ is a strict subset of $\mathbb{R}^3$ which means this test is only evaluated some of the time. On the other hand, the evaluation of the test $R_g \in T$ of hierarchy H_{4-9} always occurs. As for which of the two hierarchies is more

efficient (in terms of processing cost alone) depends on the exact costs assigned to these evaluations.

For the sensor networks we are interested in, the costs due to sensing and processing are insignificant compared to the cost of communication which we describe next.

4.1.3.3 Communication Costs

The costs for communication, where we consider cost to be the amount of energy needed for communicating the information reliably, will be based on the energy consumption model for communication as presented in Section 2.1.3. In that model, the energy consumed during transmission depends linearly on the the number of bits k to be transmitted and quadratically on the distance d between the sender and receiver. Furthermore, there is energy consumed during reception which grows linearly with the number of bits k received. As before, we denote these costs by

$$E_{Tx}(k,d) \text{ and } E_{Rx}(k) \quad .$$

For a "pull"-type communication, there is also an added cost for sending a query message since the query message is k_{query} bits.

To compute the cost of communication, we derive the cost and trigger associated with each uplink and downlink in the hierarchy. Let us define some notation.

- k_j, for any positive integer j, represents the number of bits that must be sent to communicate one real number.

- $d_{i,j} \in \mathbb{R}$ represents the distance between agent i and agent j where, in our example, $i, j \in \{1, 2, 3\}$.

In hierarchy H_{4-9}, there are two "push"-type communications encoded corresponding to the trigger sets S and T. The costs associated with these two communications are

$$\begin{aligned} c_S &= E_{Tx}(k_1, d_{1,2}) + E_{Rx}(k_1) \\ c_T &= E_{Tx}(k_1, d_{1,3}) + E_{Rx}(k_1) \ . \end{aligned}$$

In both communications, a single real number is communicated. However, the distance over which the real number is communicated differs. The triggers for these communications are

$$\begin{aligned} F_S^{H_{4-9}} &= \mathbb{R}^3 \\ F_T^{H_{4-9}} &= \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 + x_2 > 0\} \ . \end{aligned}$$

In hierarchy H_{4-10}, there are two "pull"-type communications encoded corresponding to the trigger sets S and T. The costs associated with these two communications are

$$\begin{aligned} c_S &= E_{Tx}(k_{query}, d_{1,2}) + E_{Rx}(k_{query}) + E_{Tx}(k_1, d_{1,2}) + E_{Rx}(k_1) \\ c_T &= E_{Tx}(k_{query}, d_{1,3}) + E_{Rx}(k_{query}) + E_{Tx}(k_1, d_{1,3}) + E_{Rx}(k_1) \ . \end{aligned}$$

In both communications above, there is first the cost of communicating the query message followed by the response which is a single real number. The triggers for these communications are

$$\begin{aligned} F_S^{H_{4-10}} &= \mathbb{R}^3 \\ F_T^{H_{4-10}} &= \{(x_1, x_2, x_3) \in \mathbb{R}^3 : x_1 + x_2 > 0\} \ . \end{aligned}$$

4.1.4 Conclusions

In this section, we have presented how to encode an information collection mechanism into a graphical representation which we call the hierarchy. Abstractly, we can consider all information collection mechanisms to be derived from a data collection mechanism into which partial processing and control of sensing is added to refine the collection mechanism. The point was to demonstrate that there exist many hierarchies for any global task which represent a reasonable set of distributed algorithms for information collection. By demonstrating how to measure the computational resource cost of a hierarchy, and hence, the computational cost of the distributed algorithm represented by the hierarchy, we have reduced the problem of constructing efficient distributed algorithms by constructing a suitable hierarchy.

The general principle followed in reducing the cost of sensing, processing, and communication is to encode a trigger which initiates these basic operations. These triggers are the manifestation of the principle of information-based computation. In fact, when we present the toy example of constructing communication algorithms for the problem of single target localization in Section 4.3, we will again see that the triggers play the important role in the construction of the hierarchy rather than the actual computations to evaluate.

The global task is defined as a function evaluation of many variables which were physically distributed in a system architecture with distributed autonomous agents. Defining global tasks as function evaluations is general in the sense that any computation is a function evaluation. However, there is still work to be done if we are to convert inference tasks into a set of communication algorithms for solving target localization.

- We must convert the notion of "estimates of reasonable quality" for an inference task into a set of computations to be evaluated. This is the subject of Section 4.2.

The running example of this section did not show how to construct a hierarchy representing an efficient information collection mechanism without having to enumerate all possible hierarchies, and it only briefly mentioned how to convert the hierarchy into a set of communication algorithms for each node of the sensor network.

- Section 4.3 will present how to algorithmically construct a hierarchy corresponding to a set of efficient communication algorithms that encodes a set of computations corresponding to the inference task of target localization.

4.2 Converting an Inference Task to a Set of Computations

A word on terminology and notation...

In this section, we will be computing estimates based on a subset of sensor measurements. Let us be precise about what we mean by "measurements" and "subset of sensor measurements". The sensor network consists of n nodes which we label individually by an integer $i \in \{1, \ldots, n\}$. Each sensor node i has sensors attached to it, and the observation of the sensor measurement of sensor i is denoted by z_i. The space that z_i lies in is denoted $\mathcal{Z}_i$. To be completely precise, we may also refer to z_i as the *individual* sensor measurement because it is the measurement from a single sensor node.

The *combined* measurement or *joint* measurement refers to the vector of all individual sensor measurements. We denote this vector by z which is related to the individual sensor measurements by

$$z = (z_1, \ldots, z_n) \ .$$

The space that z lies in is denoted $\mathcal{Z}$ which is defined to be

$$\mathcal{Z} = \mathcal{Z}_1 \times \cdots \mathcal{Z}_n \ .$$

Given the measurement $z \in \mathcal{Z}$, z_i refers to the ith component of z. That is, z is the joint measurement, and z_i is the individual measurement from sensor node i. In the text, we may omit the qualifying measurements being "individual" or "combined" or "joint"; however, the ambiguity should be clear from the mathematical symbols.

Finally, by a "subset of measurements", we refer to two possibilities. The first is a subset of sensor node labels, which we denote by

$$S \subset \{1, \ldots, n\} \ .$$

The second is the subset of individual measurements from a given joint measurement $z \in \mathcal{Z}$ as specified by $S \subset \{1, \ldots, n\}$, which we denote by z_S. If the elements of S are $\{j_1, \ldots, j_l\}$, then

$$z_S = (z_{j_1}, \ldots, z_{j_l})$$

and the space that it lies in is denoted

$$\mathcal{Z}_S = \mathcal{Z}_{j_1} \times \cdots \times \mathcal{Z}_{j_l} \ .$$

Let us now begin the discussion of this section.

Main Idea

We refer to the initial discussion of what measurements to collect as introduced in Section 2.3.2.1. The posterior density of the target location given a subset S of sensor measurements, denoted z_S, with prior density p_0 is given by

$$\tilde{p}_{z_S}(x) = \alpha p_0(x) \prod_{i \in S} p_i(z_i|x)$$

where α is a normalization constant and $p_i(z_i|x)$ is the likelihood function of sensor i observing z_i given the target is located at x. The posterior density of the target location given all measurements is

$$\tilde{p}_z(x) = \alpha p_0(x) \prod_{i=1}^{n} p_i(z_i|x)$$

where α is a normalization constant.

A relation $\approx$ is given which has the intuitive notion that if

$$\tilde{p}_{z_S} \approx \tilde{p}_z \ ,$$

then we say that the posterior density given the subset of sensor measurements S approximates the posterior density given all measurements when the joint measurement observed is z. We discussed some measures for defining $\approx$ in Section 2.3.2.1.

Then, we gave an expression for the minimal subset of measurements (subsets of $\{1, \ldots, n\}$)

$$S_z = \arg_{S \subset \{1,\ldots,n\}: \tilde{p}_{z_S} \approx \tilde{p}_z} \min |S| \ .$$

S_z is the minimal subset of sensor nodes which contains most of the information about the target location $x \in \mathbb{R}^2$. Since the entire measurement space $\mathcal{Z}$ is large, we limit ourselves to a subset of the measurement space, the so-called *likely joint measurements* denoted

$$\mathcal{Z}_{likely} = \bigcup_{x \in F} \hat{Z}_x \tag{4.8}$$

where $F \subset \mathbb{R}^2$ is the field. The intuition of $\hat{Z}_x \subset \mathcal{Z}$ is the subset of joint measurements which are likely to be observed if the true target location was x. Thus, the subset

$\mathcal{Z}_{likely}$ is the set of likely joint measurements observed if the target is anywhere in the field F. The expression for $\hat{Z}_x$ is given by

$$\hat{Z}_x = \{z \in \mathcal{Z} : \prod_{i \in \{1,\ldots,n\}-T_x} p_i(z_i|x) > \gamma_{|T_x|} \text{ and } z_j = \emptyset \text{ for all } j \in T_x \} \tag{4.9}$$

where $T_x \subset \{1, \ldots, n\}$ is defined to be the subset of TDOA sensors which are outside the range of the target located at x and $\gamma_{|T_x|}$ is a threshold which takes into account that $|T_x|$ TDOA sensors are out of range.[7]

Thus, the global inference task is the following.

Global Inference Task

When the joint measurement observed is $z \in \mathcal{Z}$, then

1. *if $z \in \mathcal{Z}_{likely}$, collect at least the minimal subset of sensor measurements S_z, and compute the posterior density given the measurements collected.*

2. *Otherwise, we do not care what happens.*

To be precise, we wish to evaluate a function

$$f : \mathcal{Z} \to \mathcal{P}_d(\mathcal{X}) \ ,$$

where $\mathcal{P}_d(\mathcal{X})$ is the set of probability densities on $\mathcal{X}$ and $\mathcal{X}$ is the set of target locations $\mathbb{R}^2$. This function f is defined by

$$f(z_1, \ldots, z_n) = \begin{cases} \tilde{p}_{z_{R_z}} & , \ z \in \mathcal{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases}$$

where

$$R_z \supset S_z$$

for all $z \in \mathcal{Z}_{likely}$. For any choice of $\{R_z\}_{z \in \mathcal{Z}_{likely}}$ where each R_z satisfies the above condition, this partially defined function f satisfies the requirements of the global task because the posterior densities are computed from a subset of measurements which is at least those in S_z. Thus, there is a whole family of functions $\mathcal{G}$ which satisfy the requirements of the global task so that we can evaluate any one of these functions in

[7] We must use this complicated expression for $\hat{Z}_x$ because the expression

$$\hat{Z}_x = \{z \in \mathcal{Z} : \prod_{i=1}^{n} p_i(z_i|x) > \gamma\}$$

does not taken into account that TDOA measurements lie in $\mathbb{R} \cup \{\emptyset\}$ and not $\mathbb{R}$. The above expression for $\hat{Z}_x$ does not match the notion of the set of likely measurements if the target is outside the range of a TDOA sensor due to the Dirac delta function in the likelihood function of a TDOA measurement. (See Eqn. 4.11.)

the sensor network. Notice that these family of functions $\mathcal{G}$ being parameterized by $\{R_z\}_{z\in\mathcal{Z}_{likely}}$ forms a lattice. That is, for any two functions in $f^1, f^2 \in \mathcal{G}$ parameterized by $\{R_z^1\}_{z\in\mathcal{Z}_{likely}}$ and $\{R_z^2\}_{z\in\mathcal{Z}_{likely}}$ respectively, the partial order $\subset$ is defined by

$$f^1 \subset f^2 \iff R_z^1 \subset R_z^2 \text{ for all } z \in \mathcal{Z}_{likely} \ .$$

The join ($\cup$) of f^1 and f^2,

$$f^3 = f^1 \cup f^2 \ ,$$

returns a function f^3 parameterized by $\{R_z^3\}_{z\in\mathcal{Z}_{likely}}$ where

$$R_z^3 = R_z^1 \cup R_z^2$$

for all $z \in \mathcal{Z}_{likely}$. The meet ($\cap$) of f^1 and f^2,

$$f^3 = f^1 \cap f^2 \ ,$$

returns a function f^3 parameterized by $\{R_z^3\}_{z\in\mathcal{Z}_{likely}}$ where

$$R_z^3 = R_z^1 \cap R_z^2$$

for all $z \in \mathcal{Z}_{likely}$. We can verify the properties of a lattice with the above join and meet operators. There is minimal and maximal element in this lattice of functions. We call the minimal element the *minimal global function* which is defined to be

$$\underline{f}(z_1, \ldots, z_n) = \begin{cases} \tilde{p}_{z_{S_z}} & , \ z \in \mathcal{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ . \tag{4.10}$$

Note that for any $f \in \mathcal{G}$,

$$\underline{f} \subset f \ .$$

The minimal global function $\underline{f}$ is the function which collects the least amount of data and computes an estimate while still satisfying the requirements of the global inference task. At the opposite end is the maximal element which we refer to as the *maximal global function.* It is defined by

$$\bar{f}(z_1, \ldots, z_n) = \begin{cases} \tilde{p}_z & , \ z \in \mathcal{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ .$$

The maximal global function $\bar{f}$ is the function which collects all data to compute the estimate and has the property that for every $f \in \mathcal{G}$,

$$\bar{f} \supset f \ .$$

$\bar{f}$ is also referred to as the *optimal* estimate because $\bar{f}$ computes the optimal posterior density in the Bayesian sense given all data when $z \in \mathcal{Z}_{likely}$.

We have defined the global inference task in the manner above because we are only interested in *estimates of reasonable quality.* We need this extra freedom of choosing any function in $\mathcal{G}$ so that we do not need to collect all the information contained in all the sensor measurements. Instead, we only need to collect a subset of sensor measurements in order to compute the estimate. The partial order has a nice intuitive meaning about the quality of the estimate and the amount of data that needs to be collected. If

$$f^1 \subset f^2 \ ,$$

then computing f^1 requires less collection of data while f^2 computes estimates which are of higher quality since more data is used to compute the estimate.

The particular problem of single target localization we will solve is the case when all sensor measurements are either intensity measurements or TDOA measurements. As usual, we assume the sensor network consists of n sensor nodes labelled by an integer $i \in \{1, \ldots, n\}$. If sensor node i observes intensity measurements, then the measurement space is

$$\mathcal{Z}_i = \mathbb{R}$$

from the measurement model Eqn. 2.3. If sensor node i observes TDOA measurements, then the measurement space is

$$\mathcal{Z}_i = \mathbb{R} \cup \{\emptyset\}$$

from the measurement model Eqn. 2.6.

Hierarchies can be constructed to encode a finite set of computations. Since the measurement space $\mathcal{Z}$ and the parameter space $\mathcal{X}$ (i.e. the space of target locations $\mathbb{R}^2$) are continuous, we must discretize both the measurement and parameter spaces. In Section 4.2.1, we will derive the set of computations and triggers resulting from a given discretization of the measurement and parameter spaces. These computations and triggers will define a discretized version $f_{discrete}$ (See Eqn. 4.15) of the minimal global function $\underline{f}$ that satisfies

$$f_{discrete} \supset \underline{f} \ ,$$

meaning that $f_{discrete}$ satisfies the requirements of the global task. Then, in Section 4.2.2, we will discuss the the issues for choosing an appropriate discretization.

4.2.1 Deriving a Set of Computations from a Given Discretization

For each sensor $i \in \{1, \ldots, n\}$, we will represent a discretization of the measurement space $\mathcal{Z}_i$ by imposing an equivalence relation $\sim_i$. On the other hand, the discretization of the parameter space $\mathcal{X}$ will be represented by a finite sample of points $\mathcal{F} \subset \mathcal{X}$, although this discretization can also be represented by an equivalence relation $\sim_{\mathcal{X}}$.

The equivalence class of $z_i \in \mathcal{Z}_i$, and equivalently for $x \in \mathcal{X}$ is denoted by square brackets

$$\begin{aligned} [z_i] &= \{y \in \mathcal{Z}_i : y \sim_i z_i\} \\ [x] &= \{y \in \mathcal{X} : y \sim_{\mathcal{X}} x\} \end{aligned}$$

where it is understood which equivalence relation is governing the equivalence class. In particular, the parameter space $\mathcal{X}$ is the set of target locations on the Euclidean plane $\mathbb{R}^2$ so we will use $\mathcal{X}$ and $\mathbb{R}^2$ interchangeably in the following discussion. Let us give a particular discretization for intensity and TDOA measurements.

Discretizing Intensity Measurements

The model for an intensity measurement at sensor node i is given by a likelihood function

$$p_i(z_i|x)$$

where $z_i \in \mathbb{R}$ and $x \in \mathbb{R}^2$. Given the prior density $p_0(x)$ of target locations, we can compute the marginal density of the sensor measurement by

$$\bar{p}_i(z_i) = \int_{\mathbb{R}^2} p_i(z_i|x)p_0(x)dx \ .$$

Consider the cumulative probability distribution function of this marginal density given by

$$\bar{P}_i(z_i) = \int_{-\infty}^{z_i} \bar{p}_i(y)dy$$

where the interpretation is that $\bar{P}_i(z_i)$ is the probability that the sensor measurement is less than or equal to z_i. Note that $\bar{P}_i(z_i)$ is a monotonically non-decreasing function of z_i ranging between 0 and 1. We can discretize the measurement space $\mathbb{R}$ into k intervals of equal probability by choosing end points

$$v_j = \inf_{a \in \bar{P}_i^{-1}(\frac{j}{k})} a$$

for $j \in \{1, \ldots, k-1\}$ where $\bar{P}_i^{-1}$ indicates the set inverse of $\bar{P}_i$ since $\bar{P}_i$ is not necessarily one-to-one. Then, the collection of k intervals is

$$W = \{(-\infty, v_1], (v_1, v_2], \ldots, (v_{k-2}, v_{k-1}], (v_{k-1}, \infty)\} \ ,$$

and for each interval $w \in W$,

$$\Pr(w) = \int_w \bar{p}_i(y)dy = \frac{1}{k} \ .$$

The equivalence relation for sensor i corresponding to the discretization of the intervals in W above is defined by

$$z_i^1 \sim_i z_i^2 \iff \text{there exists an interval } w \in W \text{ such that } z_i^1 \in w \text{ and } z_i^2 \in w \ .$$

Discretizing TDOA Measurements

The model for a TDOA measurement at sensor node i is given by a likelihood function of the form

$$p_i(z_i|x) = \begin{cases} p_{tdoa}(z_i|x) & , \ z_i \in \mathbb{R}, \ d(x, x_i) \leq \gamma \\ 0 & , \ z_i = \emptyset, \ d(x, x_i) \leq \gamma \\ \delta_\emptyset(z_i) & , \ d(x, x_i) > \gamma \end{cases} \tag{4.11}$$

where $z_i \in \mathbb{R} \cup \{\emptyset\}$ is the TDOA measurement, $x \in \mathbb{R}^2$ is the target location, x_i is the location of this TDOA sensor, $d(\cdot, \cdot)$ is the Euclidean distance on the plane, γ is the range of the TDOA sensor, and $\delta_\emptyset(\cdot)$ is the Dirac delta function centered at the point $\emptyset$. The model has two cases: (1) the target is in range of the sensor, $d(x, x_i) \leq \gamma$, and (2) the target is out of range of the sensor, $d(x, x_i) > \gamma$. When the target is in range, the TDOA measurement takes value in $\mathbb{R}$ according to the density p_{tdoa}. When the target is out of range, the TDOA measurement is $\emptyset$ with probability 1.

To discretize the measurement space $\mathbb{R} \cup \{\emptyset\}$ into k regions, we set aside one discretization region for when $z_i = \emptyset$ and discretize $\mathbb{R}$ into $k-1$ intervals using the density p_{tdoa}. To elaborate, given the prior density $p_0(x)$ of the target location, we first compute the conditional density of the target location given the target is within range which is given by

$$\tilde{p}_0(x) = \begin{cases} \frac{p_0(x)}{\int_{\bar{B}_\gamma(x_i)} p_0(y)dy} & , \ d(x, x_i) \leq \gamma \\ 0 & , \ \text{otherwise} \end{cases}$$

where

$$\bar{B}_\gamma(x_i) = \{x \in \mathcal{X} : d(x, x_i) \leq \gamma\}$$

is the closed ball of radius γ around x_i. We then compute the conditional density of the TDOA measurement given the target is within range by

$$\bar{p}_i(z_i) = \int_{\mathbb{R}^2} p_{tdoa}(z_i|x)\tilde{p}_0(x)dx \ .$$

Its associated cumulative probability distribution function is given by

$$\bar{P}_i(z_i) = \int_{-\infty}^{z_i} \bar{p}_i(y)dy \ .$$

Then, as before we divide the space $\mathbb{R}$ into $k-1$ intervals of equal probability by choosing end points by

$$v_j = \inf_{a \in \bar{P}_i^{-1}(\frac{j}{k-1})} a$$

for $j \in \{1, \ldots, k-2\}$. Then, the collection of $k-1$ intervals is

$$W = \{(\infty, v_1], (v_1, v_2], \ldots, (v_{k-2}, \infty)\} \ ,$$

and for each interval $w \in W$,

$$\Pr(w) = \int_w \bar{p}_i(y)dy = \frac{1}{k-1} \ .$$

The equivalence relation for the TDOA sensor i representing the discretization of this measurement space is

$$\begin{aligned} z_i^1 \sim_i z_i^2 \quad \Longleftrightarrow \quad & \text{One of the following hold :} \\ & (i) \text{ there exists an interval } w \in W \text{ such that } z_i^1 \in w \text{ and } z_i^2 \in w \\ & (ii) \ z_i^1 = \emptyset \text{ and } z_i^2 = \emptyset \ . \end{aligned}$$

Resulting Discretization of the Joint Measurement Space

Given the equivalence relations $\sim_i$ on the individual measurement space $\mathcal{Z}_i$ for every sensor node $i \in \{1, \ldots, n\}$ of the sensor network, we will define an equivalence relation $\sim$ on the joint measurement space $\mathcal{Z}$ by

$$z \sim z' \iff z_i \sim_i z_i' \text{ for all } i \in \{1, \ldots, n\} \ .$$

Discretizing Target Locations

To discretize the set of target locations, we consider a square grid of points on $\mathbb{R}^2$ given by

$$G_d = \left\{ a \begin{bmatrix} d \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ d \end{bmatrix} : a, b \in \mathbb{Z} \right\} \tag{4.12}$$

where $\mathbb{Z}$ is the set of integers and $d > 0$ is the spacing of the square grid. Recall the field $F \subset \mathbb{R}^2$ defined in Definition 2.7, which is the bounded set of possible target positions which are in range of the sensor network. Because F is bounded, the set of points

$$\mathcal{F} = G_d \cap F$$

is a finite set of points in $\mathbb{R}^2$ which is the sample of points we will use to discretize the set of target positions. Thus, the equivalence relation defined by these set of points is given by

$$x_1 \sim_{\mathcal{X}} x_2 \iff \arg_{x \in \mathcal{F}} \min d(x, x_1) = \arg_{x \in \mathcal{F}} \min d(x, x_2) \ .$$

Note that the partitions set up by this equivalence relation are the Voronoi regions of the points in $\mathcal{F}$ according to the Euclidean metric on the plane.

Derivation of the Set of Computations

We can now derive the set of computations with these discretizations of the combined measurement space, given by $\sim$, and the parameter space, given by a set of sample points $\mathcal{F}$.

The reason for discretizing the joint measurement space $\mathcal{Z}$ is so that we consider "points" under the discretization to be elements of the quotient space

$$\mathcal{Z}/\sim \ .$$

We will abuse notation here. For an element $z \in \mathcal{Z}$, we denote by $[z]$ the equivalence class of z as well as the point in the quotient space $\mathcal{Z}/\sim$. With k equivalence classes per component of $\mathcal{Z}$, the number of elements in $\mathcal{Z}/\sim$ is k^n. More importantly, we assume that the number of equivalence classes is finite. Thus, we will define comparable quantities for $\mathcal{Z}_{likely}$ and S_z to derive a *finite* set of computations with associated triggers.

For each of the sampled target locations $x \in \mathcal{F}$, we compute the set of likely joint measurements that the sensor nodes will observe by $\hat{Z}_x$ as given by Eqn. 4.9. The idea is that the union of these subsets,

$$\bigcup_{x \in \mathcal{F}} \hat{Z}_x \ ,$$

approximates the set $\mathcal{Z}_{likely}$ which was given in Eqn. 4.8 and reprinted below.

$$\mathcal{Z}_{likely} = \bigcup_{x \in F} \hat{Z}_x$$

Now, we must take into account the discretization of $\mathcal{Z}$.

Consider the set of distinct equivalence classes of the points in $\bigcup_{x \in \mathcal{F}} \hat{Z}_x$,

$$\mathbf{Z}_{likely} = \{[z] : z \in \bigcup_{x \in \mathcal{F}} \hat{Z}_x\} \ .$$

$\mathbf{Z}_{likely}$ is a finite set equivalence classes which covers the set $\bigcup_{x \in \mathcal{F}} \hat{Z}_x$. By a covering, we mean that the following relation holds

$$\{z \in \mathcal{Z} : [z] \in \mathbf{Z}_{likely}\} \supset \bigcup_{x \in \mathcal{F}} \hat{Z}_x \ .$$

Hence, we consider $\mathbf{Z}_{likely}$ to be a discretized (in measurement space) and sampled (in parameter space) version of $\mathcal{Z}_{likely}$. In fact, let us assume that we have picked a dense enough sampling of the field F so that $\mathbf{Z}_{likely}$ covers $\mathcal{Z}_{likely}$ as well.

Recall that for each $z \in \mathcal{Z}$, we calculated the minimal subset of measurements to collect by

$$S_z = \arg_{S \subset \{1,\dots,n\} : \tilde{p}_{z_S} \approx \tilde{p}_z} \min |S| \ . \tag{4.13}$$

Notice that the estimate based on the subset of measurement z_S is an "estimate of reasonable quality" to the estimate based on all measurements z as quantified by the relation $\approx$ above. We can now define the minimal subset of measurements to collect for each equivalence class by

$$\mathbf{S}_{[z]} = \bigcup_{z' \in \mathcal{Z}: z' \sim z} S_{z'} \ , \tag{4.14}$$

which is the quantity comparable to S_z. Note that for any $z^1, z^2 \in \mathcal{Z}$ with $z^1 \sim z^2$ that

$$\mathbf{S}_{[z^1]} = \mathbf{S}_{[z^2]} \ .$$

Given a measurement $z \in \mathcal{Z}$ where it is true that $[z] \in \mathbf{Z}_{likely}$, the idea is that we only need to collect the subset of measurements $\mathbf{S}_{[z]}$ and compute the posterior density

$$\tilde{p}_{z_{\mathbf{S}_{[z]}}}(x) = \alpha p_0(x) \prod_{i \in \mathbf{S}_{[z]}} p_i(z_i|x) \ ,$$

where α is a normalization constant. This is an estimate of reasonable quality because the subset of measurements $\mathbf{S}_{[z]}$ used to compute the estimate $\tilde{p}_{z_{\mathbf{S}_{[z]}}}$ for all $z' \in [z]$ is a superset of the minimal subset of measurements $S_{z'}$ required to be collected and used to compute the estimate.

The global function under these discretizations is the function

$$f_{discrete}(z_1, \ldots, z_n) = \begin{cases} \tilde{p}_{z_{\mathbf{S}_{[z]}}} & , \ [z] \in \mathbf{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ . \tag{4.15}$$

Due to the choice of $\mathbf{S}_{[z]}$ and the assumption that $\mathbf{Z}_{likely}$ covers $\mathcal{Z}_{likely}$, it is true that

$$f_{discrete} \supset \underline{f} \ .$$

The set of computations and their associated triggers are expressed in the function $f_{discrete}$ above.

Let us simplify notation a bit in the expression for f above to explicitly show the set of computations and their triggers. With each element of $\mathbf{Z}_{likely}$, let us associate a number $1, \ldots, M$ where

$$M = |\mathbf{Z}_{likely}| \ ,$$

and define a function

$$elem : \{1, \ldots, M\} \to \mathbf{Z}_{likely}$$

which takes a label and returns the associated element of $\mathbf{Z}_{likely}$. For example, if $[z]$ is associated with i, then

$$[z] = elem(i) \ .$$

We also use the simpler notation

$$S_i = \mathbf{S}_{elem(i)}$$

for $i \in \{1, \ldots, M\}$. Let

$$g_i : \mathcal{Z}_{S_i} \to \mathcal{P}_d(\mathcal{X}) \ ,$$

be defined by

$$g_i(z_{S_i}) = \tilde{p}_{z_{S_i}} \ .$$

Also, let

$$T_i : \mathcal{Z} \to \{0, 1\}$$

be defined by

$$T_i(z) = [z \in elem(i)]$$

where we abuse notation and also use the square brackets $[\cdot]$ to indicate Iverson's convention, which returns 1 if the expression inside the square brackets is true and zero otherwise. Then, the function $f_{discrete}$ can be rewritten as

$$f_{discrete}(z_1, \ldots, z_n) = \begin{cases} g_1(z_{S_1}) & , \ T_1(z) = 1 \\ g_2(z_{S_2}) & , \ T_2(z) = 1 \\ \vdots & \vdots \\ g_M(z_{S_M}) & , \ T_M(z) = 1 \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ .$$

Note that the function above is much like the example function Eqn. 4.4 of Section 4.1.2 (reprinted below) in that $f_{discrete}$ is defined piecewise over different regions of the domain.

$$f^*(x_1, x_2, x_3) = \begin{cases} (x_1 - x_2)^2 - x_3 & , \ x_1 + x_2 > 0 \\ (x_1 - x_2)^2 & , \ x_1 + x_2 \leq 0 \end{cases}$$

The added difference is that we now allow the function to be undefined over certain regions of the domain. The reason why we were able to reduce communications in the example function is because the function depended on different variables for different regions of the domain. The view of f^* was an encoding of two computations

$$\begin{aligned} g_1^*(x_1, x_2, x_3) &= (x_1 - x_2)^2 - x_3 \\ g_2^*(x_1, x_2) &= (x_1 - x_2)^2 \end{aligned}$$

where the evaluation of g_1^* is triggered when it is true that $x_1 + x_2 > 0$ and the evaluation of g_2^* is triggered when $x_1 + x_2 \leq 0$.

Going back to our function $f_{discrete}$, we can view this function as encoding the evaluation of the set of computations

$$\{g_i\}_{i=1}^M$$

with associated triggers

$$\{T_i\}_{i=1}^M \ .$$

Now, to evaluate g_i we only need to collect the sensor measurements in S_i. This is desirable because this means we do not have to collect all sensor measurements to evaluate g_i. However, before we evaluate g_i when the observed sensor measurement is $z \in \mathcal{Z}$, we must test whether $T_i(z) = 1$, but testing whether $T_i(z) = 1$ requires collecting *all* sensor measurements! This is certainly not good because we must always collect all measurements all the time. This phenomenon did not occur with the example function f^*, and the reason for this is because the triggers determining whether to evaluate g_1^* and g_2^* depended only on the variables x_1 and x_2 and not on x_3. Since g_2^* is a function of only x_1 and x_2, if it is true that $x_1 + x_2 \leq 0$, then we only need to collect x_1 and x_2 in order to evaluate f^*. Thus, the conclusion is that the set of measurements that must be collected in order to evaluate the computation g_i are the measurements from sensors in S_i *and* the sensor measurements needed for evaluating the associated trigger T_i. Now, we can take advantage of the fact that $f_{discrete}$ is undefined over certain regions of the domain.

The initial idea for fixing this problem is to make the triggers depend only on the components in S_i. Thus, we replace the triggers T_i in $f_{discrete}$ with

$$T_i'(z_{S_i}) = [z_{S_i} \sim z^i_{S_i}]$$

where z^i is any element in the equivalence class $elem(i)$ and

$$z_S \sim z'_S \iff \text{for all } j \in S,\ z_j \sim_j z'_j\ .$$

Thus, the function we consider is

$$f'_{discrete}(z_1, \ldots, z_n) = \begin{cases} g_1(z_{S_1}) & ,\ T_1'(z_{S_1}) = 1 \\ g_2(z_{S_2}) & ,\ T_2'(z_{S_2}) = 1 \\ \vdots & \vdots \\ g_M(z_{S_M}) & ,\ T_M'(z_{S_M}) = 1 \\ \text{undefined} & ,\ \text{otherwise} \end{cases} .$$

However, there is a problem with this idea. It is possible that for a given $z \in \mathcal{Z}$ that

$$T_i'(z_{S_i}) = 1 \text{ and } T_j'(z_{S_j}) = 1$$

for $i, j \in \{1, \ldots, M\}$ and $i \neq j$ which could make the function above not well-defined since the evaluation of $f'_{discrete}(z)$ could either be the output of $g_i(z_{S_i})$ or $g_j(z_{S_j})$ (assuming that these outputs are different).

Example

To illustrate this possibility, consider a system with three boolean variables $z_1, z_2, z_3 \in \{0, 1\}$, define $S_1^* = \{1, 2\}$ and $S_2^* = \{2, 3\}$ and consider the two triggers

$$\begin{array}{ll} (i) & z_{S_1^*} = (1, 0) \\ (ii) & z_{S_2^*} = (0, 0) \end{array}$$

Trigger (i) is equivalent to $z_1 = 1$ and $z_2 = 0$, and trigger (ii) is equivalent to $z_2 = 0$ and $z_3 = 0$. If the observation of the three variables is

$$(z_1, z_2, z_3) = (1, 0, 0) \ ,$$

then both triggers are satisfied. However, if it is known that it is impossible for the observation of the three variables to return $(1, 0, 0)$, then the two triggers never equal 1 at the same time so that the function *is* well-defined.

Thus, we require that exactly one of the triggers T_i' is equal to one only over the domain where the minimal global function $\underline{f}$ was defined, namely $\mathcal{Z}_{likely}$. To be more precise, the condition that the above function $f'_{discrete}$ is well-defined is if for every $z \in \mathcal{Z}_{likely}$, there exists exactly one $i \in \{1, \ldots, M\}$ such that

$$T_i'(z_{S_i}) = 1$$

and for all other $j \in \{1, \ldots, M\} - \{i\}$,

$$T_j'(z_{S_j}) = 0 \ .$$

In the case when the condition above does not hold, we must choose a subset $R_i \subset \{1, \ldots, n\}$ where

$$S_i \subset R_i \subset \{1, \ldots, n\}$$

so that the condition above holds for the triggers

$$\mathbf{T}_i(z_{R_i}) = [z_{R_i} \sim z^i_{R_i}]$$

where z^i is any element of the equivalence class $elem(i)$. Then, the resulting function to evaluate is

$$\mathbf{f}(z_1, \ldots, z_n) = \begin{cases} g_1(z_{S_1}) & , \ \mathbf{T}_1(z_{R_1}) = 1 \\ g_2(z_{S_2}) & , \ \mathbf{T}_2(z_{R_2}) = 1 \\ \vdots & \vdots \\ g_M(z_{S_M}) & , \ \mathbf{T}_M(z_{R_M}) = 1 \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ .$$

There is a straightforward algorithm for generating these R_i.

Algorithm Generating R_i
Initially, start with

$$R_i := S_i$$

for all $i \in \{1, \ldots, M\}$. Then, for each pair $i, j \in \{1, \ldots, M\}$ with $i \neq j$, test if there exists a $z \in \mathcal{Z}_{likely}$ such that

$$\mathbf{T}_i(z_{R_i}) = 1 \text{ and } \mathbf{T}_j(z_{R_j}) = 1 \ .$$

For any pair i, j where the triggers both evaluate to 1, we update the set R_i and R_j by

$$\begin{aligned} R_i &:= R_i \cup \{k\} \\ R_j &:= R_j \cup \{k\} \end{aligned}$$

where $k \in \{1, \ldots, n\}$ is chosen such that $z_k^i \not\sim_k z_k^j$ where $z^i \in elem(i)$ and $z^j \in elem(j)$. We continue to check over all pairs $i, j \in \{1, \ldots, M\}$ with these updated subsets $\{R_i\}_{i=1}^M$ until the condition is satisfied for every pair.

As long as the original triggers $\{T_i\}_{i=1}^M$ were distinct, meaning that for no $z \in \mathcal{Z}_{likely}$ did more than one of the $T_i(z)$ evaluate to 1, this algorithm will compute the subset of measurements R_i which the trigger $\mathbf{T}_i$ should depend on for all $i \in \{1, \ldots, M\}$ such that the resulting function $\mathbf{f}$ is well-defined.

Thus, we have derived a set of computations

$$\{g_i\}_{i=1}^M$$

depending on the subset of measurements $\{S_i\}_{i=1}^M$ and a set of triggers

$$\{\mathbf{T}_i\}_{i=1}^M$$

depending on the subset of measurements $\{R_i\}_{i=1}^M$ where

$$R_i \supset S_i \ .$$

We can now encode these functions into a hierarchy and construct efficient communication algorithms for single target localization.

4.2.2 Issues on Choosing a Discretization

To summarize the previous section, the minimal global function of the inference task (Eqn. 4.10) is reprinted below.

$$f(z_1, \ldots, z_n) = \begin{cases} \tilde{p}_{z_{S_z}} & , \ z \in \mathcal{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ .$$

After choosing a discretization of $\mathcal{Z}$, represented by an equivalence relation $\sim$, and choosing a sampling $\mathcal{F} \subset F$ of points in the field F, we have replaced the function f above by the discretized version (Eqn. 4.15) reprinted below.

$$f_{discrete}(z_1, \ldots, z_n) = \begin{cases} \tilde{p}_{z_{\mathbf{S}_{[z]}}} & , \ [z] \in \mathbf{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ .$$

Then, the rest of the previous section goes on to make the triggers

$$[z] \in \mathbf{Z}_{likely}$$

depend only on a subset of measurements.

In this section, we discuss the resulting $f_{discrete}$ for the following two choices:

1. a discretization of the joint measurement space $\mathcal{Z}$ according to some equivalence relation $\sim$ and

2. a set of sample points $\mathcal{F}$ in the field $F \subset \mathcal{X}$.

The goal in choosing the discretization and sampling is to make $\mathbf{S}_{[z]}$ as small as possible for each $[z] \in \mathbf{Z}_{likely}$ and keep the number of equivalence classes in $\mathbf{Z}_{likely}$ small. $\mathbf{S}_{[z]}$ represents the minimal subset of sensor measurements that must be collected when z is an element of some equivalence class in $\mathbf{Z}_{likely}$. $|\mathbf{Z}_{likely}|$ is the number of computations we must encode into a hierarchy which we would like to keep small because this affects the time complexity for constructing a hierarchy.

We discuss the issues for choosing a discretization on the measurement space in Section 4.2.2.1. Then, in Section 4.2.2.2, we discuss the issue of choosing the likely set of measurements $\mathcal{Z}_{likely}$ and the issue of how densely to sample from the field.

4.2.2.1 Discretization of the Measurement Space

To characterize the amount of discretization in the joint measurement space, we consider the parameter k which is the number of equal probability partitions for the individual measurement space of each sensor node. Thus, with n sensor nodes, the number of equivalence classes in the joint measurement space is k^n. In this section, we wish to discuss the effect of k alone, so we will assume that for any chosen k, the corresponding $\mathbf{Z}_{likely}$, denoted $\mathbf{Z}^k_{likely}$ to show the explicit dependence on k, is the minimal set of equivalence classes which covers the given subset of likely measurements $\mathcal{Z}_{likely}$.

Since $\mathbf{Z}^k_{likely}$ is a subset of all equivalence classes, it certainly true that

$$|\mathbf{Z}^k_{likely}| \leq k^n \ .$$

Although the exact cardinality of $\mathbf{Z}^k_{likely}$ depends on the subset $\mathcal{Z}_{likely}$, $|\mathbf{Z}^k_{likely}|$ grows with larger k. Thus, we wish to keep k as small as possible.

Now, let us discuss the effect that discretization of the measurement space has on the average size of the subset of measurements that must be collected. For a given discretization of the parameter space according to the positive integer parameter k, we obtain an equivalence relation which we denote $\sim^k$. Then, we can compute the minimal subset of measurements in each equivalence class according to

$$\mathbf{S}^k_{[z]} = \bigcup_{z' \in \mathcal{Z}: z' \sim^k z} S_{z'} \ .^{8}$$

[8]Note that the computation of $\mathbf{S}^k_{[z]}$ is intractable because the number of elements in the equivalence class $[z]$ is uncountable. Thus, in a practice, we must estimate $\mathbf{S}^k_{[z]}$ by choosing a finite sample of the elements B in the equivalence class $[z]$ and estimating $\mathbf{S}^k_{[z]}$ by $\bigcup_{z' \in B} S'_z$.

The reason that $\mathbf{S}^k_{[z]}$ is defined to be the union of the minimal subset of measurements $S_{z'}$ for every z' in the equivalence class $[z]$ is so that for all of these z', we collect enough data to make an estimate of reasonable quality, and hence, satisfy the requirements of the inference task. Given a probability distribution P on $\mathcal{Z}$, the average number of sensor measurements that must be collected under this discretization parameter k is given by

$$E[|\mathbf{S}^k_{[Z]}|] = \int_{\mathcal{Z}} |\mathbf{S}^k_{[z]}| dP(z)$$

where Z denotes a random variable in $\mathcal{Z}$ with distribution P.

Thus, if we are given an upper bound M on the number of computations we may encode into a hierarchy, then the best choice of k is given by

$$k = \arg_{l \in \mathbb{Z}^+ : |\mathbf{Z}^l_{likely}| \leq M} \min E[|\mathbf{S}^l_{[Z]}|]$$

where $\mathbb{Z}^+$ denotes the positive integers. Choosing k in this manner necessitates the computation of $E[|\mathbf{S}^l_{[Z]}|]$ for many values of l which means we must compute the subsets $\mathbf{S}^l_{[z]}$. To avoid this computation, let us give some intuition on how k affects the average number of sensor measurements to collect.

We wish to argue that larger k means smaller $E[|\mathbf{S}^k_{[Z]}|]$. However, such a tradeoff is only "loosely" true because $E[|\mathbf{S}^k_{[Z]}|]$ is not a monotonically non-increasing sequence as k grows due to the quantization effects of discretizing the space $\mathcal{Z}$ and the fact that S_z viewed as a function

$$h : \mathcal{Z} \to \{1, \ldots, n\}$$

defined by

$$S_z = h(z)$$

is discontinuous and may not even be measurable. On the other hand, we can make a precise statement for certain subsequences of the parameter k. Consider the sequence $(k_j)_{j=0}^{\infty}$ where

$$k_j = m \cdot a^j$$

for any positive integer m and integer $a \geq 2$. Note that $\sim_{k_j}$ is a refinement of $\sim_{k_{j-1}}$ by which we mean that for any $z^1, z^2 \in \mathcal{Z}$,

$$z^1 \sim_{k_j} z^2 \implies z^1 \sim_{k_{j-1}} z^2 \ .$$

The consequence of this refinement is that

$$\mathbf{S}^{k_{j-1}}_{[z]} = \bigcup_{[z'] \subset [z]} \mathbf{S}^{k_j}_{[z']}$$

which means that

$$|\mathbf{S}^{k_{j-1}}_{[z]}| \geq |\mathbf{S}^{k_j}_{[z']}|$$

for any $[z'] \subset [z]$. Thus, the sequence $(A_j)_{j=0}^{\infty}$ where

$$A_j = E[|\mathbf{S}_{[Z]}^{k_j}|]$$

is a monotonically non-increasing sequence with j.

Loosely speaking, the conclusion is that with larger k comes a reduction in the average number of measurements to collect. This effect, however, is at the cost of having to encode a larger number of computations into a hierarchy. Thus, with an upper bound M of the number of computations we may encode into a hierarchy, our heuristic is to choose the largest discretization parameter k so that $|\mathbf{Z}_{likely}^{k}| \leq M$. That is,

$$k = \arg_{l \in \mathbb{Z}^+ : |\mathbf{Z}_{likely}^{k}| \leq M} \max \ l \ .$$

4.2.2.2 Choosing Likely Measurements $\mathcal{Z}_{likely}$ and Samples of the Field

Choice of Likely Measurements

Let us consider the issue of choosing the set of likely measurements $\mathcal{Z}_{likely}$ first since this set will affect how dense to choose the sample points from the field F. Recall that

$$\mathcal{Z}_{likely} = \bigcup_{x \in F} \hat{Z}_x \ .$$

An expression for $\hat{Z}_x$ was given in Eqn. 4.9. $\hat{Z}_x$ represents the set of likely measurements given that the target is located at x. One measure about how well the set $\hat{Z}_x$ corresponds to the intuitive notion of the "likely set of measurements observed" when the target is located at x is the probability that the measurement generated is an element of $\hat{Z}_x$. We refer to this measure as the *detection probability* defined by

$$P_{detect} = P(\hat{Z}_x|x) = \int_{\hat{Z}_x} \prod_{i=1}^{n} p_i(z_i|x) dz \ .$$

Usually, we have a requirement that

$$P_{detect} > 1 - \epsilon$$

for some small $\epsilon > 0$. The problem with this requirement is that $\hat{Z}_x$ can be very large. Fortunately, during the operation of the sensor network, we are not limited to a single observation of the sensor measurements for localizing a target. Instead, we can loosen the requirement to detecting the target at least once with probability $1-\epsilon$ for every m observations of the sensor measurements. If the probability that we detect the target in a single observation is P_{detect}, then assuming that the m observations are independent, the probability that we detect the target at least once for every m observations is given by

$$1 - (1 - P_{detect})^m \ .$$

The requirement is that

$$1 - (1 - P_{detect})^m > 1 - \epsilon$$

which reduces to

$$P_{detect} > 1 - \epsilon^m \ .$$

Thus, to satisfy this requirement, we must choose $\hat{Z}_x$ (by choosing thresholds in Eqn. 4.9 appropriately) so that

$$P(\hat{Z}_x) > 1 - \epsilon^m \ .$$

From these requirements, we have the set $\mathcal{Z}_{likely}$.

Sampling from the Field

Now, let us consider the choice of samples of target locations. To characterize the density of samples in $\mathcal{F}$, we consider the parameter d which is the spacing of the square grid G_d from Eqn. 4.12. When d is large, we get sparse samples in F, and when d is small, we get dense samples in F. In order to discuss the effect of d alone, let us assume that we have chosen a k which discretizes the measurement space.

For a given d, we obtain a set of sample points $\mathcal{F}^d$ in the field F, where we explicitly show the dependence of the set of sample points on d. From these sample points $x \in \mathcal{F}^d$, we construct the sets $\hat{Z}_x$ as given by Eqn. 4.9. Then, we compute

$$\mathbf{Z}^d_{likely} = \{[z] \subset \mathcal{Z} : z \in \bigcup_{x \in \mathcal{F}^d} \hat{Z}_x\}$$

where we explicitly show the dependence of $\mathbf{Z}_{likely}$ on d. Let us define the notation

$$\bigcup \mathbf{Z}^d_{likely} = \{z \in \mathcal{Z} : [z] \in \mathbf{Z}^d_{likely}\}$$

which is the set of joint measurements which is an element of one of the equivalence classes of $\mathbf{Z}^d_{likely}$. If $\mathbf{Z}^d_{likely}$ covers $\mathcal{Z}_{likely}$, meaning that

$$\bigcup \mathbf{Z}^d_{likely} \supset \mathcal{Z}_{likely} \ ,$$

then we can meet the requirements of the inference task because the function $f_{discrete}$ will compute the posterior density for all joint measurements $z \in \bigcup \mathcal{Z}_{likely}$ which means we compute the posterior density for all joint measurements $z \in \mathcal{Z}_{likely}$.

For the case when $\mathbf{Z}^d_{likely}$ does not cover $\mathcal{Z}_{likely}$, the resulting $f_{discrete}$ does not satisfy the requirements of the inference task. We can quantify by how much the requirements are not met by considering the quantity

$$P(Z^d_{err})$$

where P is a probability distribution on $\mathcal{Z}$ and

$$Z^d_{err} = \mathcal{Z}_{likely} - \bigcup \mathbf{Z}^d_{likely} \ .$$

When the probability of Z^d_{err} is 0, the inference task requirements are satisfied with probability 1. We wish to choose the parameter d so that we can make this probability be equal to 0. However, if we can tolerate not satisfying the requirements of the inference task up to a probability of $\gamma \in [0, 1]$, then we can choose d according to

$$d = \arg_{l \in \mathbb{R}^+ : P(Z^d_{err}) \leq \gamma} \min \ l$$

where $\mathbb{R}^+$ is the positive reals.

In conclusion, the general idea for choosing the spacing d is to choose it small enough so that the resulting $\mathbf{Z}^d_{likely}$ covers the set of likely measurements $\mathcal{Z}_{likely}$. The choice of d depends on the choice of the discretization parameter k. If k is large, then we must make d small in order to ensure that $\mathbf{Z}^d_{likely}$ covers $\mathcal{Z}_{likely}$. On the other hand, if k is small, then d can be larger because there are fewer equivalence classes in $\mathcal{Z}$.

4.3 Toy Example

We will consider a simple sensor network and a simple inference task which will be an example of the method for constructing communication algorithms for data collection. The discussion of how the hierarchy is a representation which encodes an information collection mechanism has been presented in Section 4.1. Furthermore, the conversion of an inference task into a set of computations with associated triggers was discussed in Section 4.2. In this section, we show the complete steps from inference task to set of computations and triggers to hierarchy to, finally, the communication algorithms which solves the single target localization problem formulated in Section 2.3. The novel material in this section include the following.

1. An algorithmic construction of the graph topology of a hierarchy is presented using a heuristic to guide the construction. The need for this heuristical construction is because the set of all graph topologies cannot be enumerated. The hope is that with an appropriately chosen heuristic, the resulting graph topology will admit to an efficient communication algorithm.

2. We show that the all-important triggers for each computation can be represented in the Boolean domain. The effect of moving to the Boolean domain is to simplify the algorithmic construction of the hierarchy.

3. The information collection mechanism encoded in the hierarchy is explicitly converted into a set of communication algorithms, one for each node of the

sensor network, as was defined in Definition 2.3. We refer to the resulting set of communication algorithms which have been constructed using the hierarchy as distributed communication algorithms (DCA).

Furthermore, the toy example of this section serves as a concrete example of the general presentation of the hierarchy which is the subject of Chapter 5.

Let us first describe this simple sensor network and the estimation task.

4.3.1 Description of Simple Network

The sensor network shown in Figure 4-11 consists of 8 sensor nodes labelled $\{1, \ldots, 8\}$ which are depicted as squares. Each sensor node i has a position which we denote by $x_i \in \mathbb{R}^2$ which are shown in Table 4.2. The units for the sensor positions are in terms of tens of meters.

i	x_i	i	x_i
1	$(-2, 1)$	5	$(1, 0)$
2	$(0, 1.5)$	6	$(2, -1)$
3	$(2, 1)$	7	$(0, -1.5)$
4	$(-1, 0)$	8	$(-2, -1)$

Table 4.2: Sensor positions for sensor network of toy example given in units of tens of meters.

We consider each sensor node to be a range detector in a particular region around itself where each node's measurement takes three possible values representing "far", "mid", or "near". In particular, the measurement space for each node $i \in \{1, \ldots, 8\}$ is

$$\mathcal{Z}_i = \{0, 1, 2\}$$

where 0 represents "far", 1 represents "mid", and 2 represents "near". The measurement at sensor i is denoted by $z_i \in \mathcal{Z}_i$.

The parameter space is the continuous space of target locations on the Euclidean plane $\mathbb{R}^2$. As noted in Section 4.2, we must sample the parameter space which is dense enough so that the set of likely measurements $\mathcal{Z}_{likely}$ are covered. However, for the purpose of demonstrating the steps involved in constructing communication algorithms, we will only sample 8 target locations which are the midpoints of the triangular regions enclosed by the three closest sensors as shown in Figure 4-11 by $\times$'s and labelled $\{\theta_i\}_{i=1}^8$ where each $\theta_i \in \mathbb{R}^2$. From here on, we will assume for this toy example that targets can only be located at the 8 sampled positions $\{\theta_i\}_{i=1}^8$, so that the parameter space is

$$\Theta = \{\theta_1, \ldots, \theta_8\} \ .$$

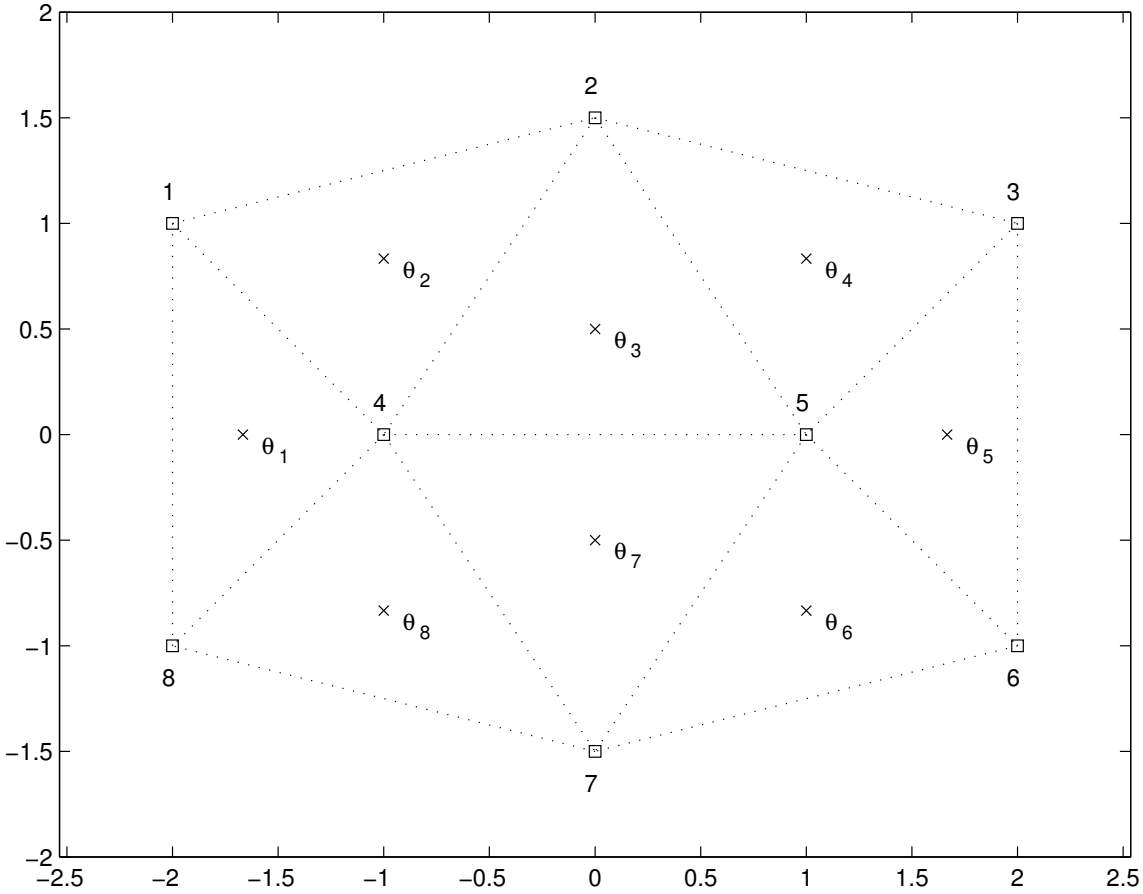

Figure 4-11: Simple sensor network containing 8 sensor nodes shown as squares. The ×'s represent the sampled target locations to serve as the parameter space.

The relationship between parameters and measurements is given by a set of likelihood functions,

$$P_i(z_i|\theta)$$

where in this discrete measurement space the likelihood is the probability that sensor i's measurement is equal to $z_i \in \mathcal{Z}_i$ given the target is located at $\theta \in \Theta$. For each sensor node $i \in \{1, \ldots, 8\}$, these likelihoods are

$$P_i(z_i|\theta) = \begin{cases} 1 - p_2 & , \ z_i = 2, \mathbf{nom}_i(\theta) = 2 \\ \frac{p_2}{2} & , \ z_i \neq 2, \mathbf{nom}_i(\theta) = 2 \\ 1 - p_1 & , \ z_i = 1, \mathbf{nom}_i(\theta) = 1 \\ \frac{p_1}{2} & , \ z_i \neq 1, \mathbf{nom}_i(\theta) = 1 \\ 1 - p_0 & , \ z_i = 0, \mathbf{nom}_i(\theta) = 0 \\ \frac{p_0}{2} & , \ z_i \neq 0, \mathbf{nom}_i(\theta) = 0 \end{cases}$$

where

$$\mathbf{nom}_i(\theta) = \begin{cases} 2 & , \ 0 \leq d(x_i, \theta) \leq 1 \\ 1 & , \ 1 < d(x_i, \theta) \leq 2 \\ 0 & , \ d(x_i, \theta) > 2 \end{cases}$$

is the nominal value of sensor i when the target is located at θ, $d(\cdot, \cdot)$ is the Euclidean distance, and

$$0 \leq p_2 \leq p_1 \leq p_0 \leq 1$$

represents the possibility of noise perturbing the measurement at the sensor. This simple model essentially says that sensor nodes measure the nominal value i with probability $1 - p_i$, and with probability p_i, the sensor node measures one of the other two values. The nominal values for each sensor for the 8 target locations are

θ_i	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8
θ_1	1	0	0	2	0	0	0	1
θ_2	1	1	0	2	0	0	0	0
θ_3	0	2	0	1	1	0	1	0
θ_4	0	1	1	0	2	0	0	0
θ_5	0	0	1	0	2	1	0	0
θ_6	0	0	0	0	2	1	1	0
θ_7	0	1	0	1	1	0	2	0
θ_8	0	0	0	2	0	0	1	1

.

We will assume that the noise at each sensor is independent so that the combined likelihood of all sensor measurements is the product

$$\mathbf{P}(z_1, \ldots z_8|\theta) = \prod_{i=1}^{8} P_i(z_i|\theta) \ .$$

Given a target position $\theta \in \Theta$, the set of likely measurements observed is defined as

$$\hat{Z}_\theta = \{z \in \mathcal{Z} : \mathbf{P}(z) \geq \rho\}$$

for a threshold ρ we will specify later. Thus, the set of likely measurements is

$$\mathcal{Z}_{likely} = \bigcup_{\theta \in \Theta} \hat{Z}_\theta \ .$$

Now, we can state the global inference task for this simple sensor network.

Global Inference Task *Given an observation consisting of sensor measurements $z \in \mathcal{Z}$ and prior distribution of target locations $P_\Theta(\theta)$, we wish to compute the posterior distribution of the target location*

$$\tilde{P}(\theta|z) = \frac{\mathbf{P}(z|\theta)P_\Theta(\theta)}{\sum_{\theta' \in \Theta} \mathbf{P}(z|\theta')P_\Theta(\theta')}$$

up to an accuracy of absolute difference less than γ if it is true that $z \in \mathcal{Z}_{likely}$.

Let us explain "accuracy of absolute difference". For two probability mass functions $P_1, P_2 \in \mathcal{P}(\Theta)$, where $\mathcal{P}(\Theta)$ is the set of probability mass functions on Θ, the absolute difference between them is

$$\max_{\theta \in \Theta} |P_1(\theta) - P_2(\theta)| \ .$$

We say that two functions f_1 and f_2 with domain $\mathcal{Z}$ and range $\mathcal{P}(\Theta)$ are accurate up to an absolute difference less than γ for $z \in \mathcal{Z}_{likely}$ if

$$\max_{z \in \mathcal{Z}_{likely}} \max_{\theta \in \Theta} |f_1(z) - f_2(z)| < \gamma \ .$$

Since data is distributed throughout nodes of the network, it is clear that to satisfy the requirements of the inference task, sensor measurements must be passed around the network. We will construct a data collection mechanism with partial processing of the triggers so that unnecessary communications will not occur; however, in the presentation to follow, we will not incorporate partial processing for computations. The reason is that after understanding the process of constructing a data collection mechanism with partial processing of the triggers only, it will be easy to see how partial processing of the computations can be included into the algorithmic construction of the hierarchy. Thus, we will leave the issue of partial processing of the computations until the summary in Section 4.3.7. We also do not show how to incorporate control of sensing in the presentation for the same reason we do not show how to incorporate partial processing of the computations. For a more practical reason, the energy expended by sensing tends to be nearly insignificant compared to the energy expended in communication.

The rest of this toy example section proceeds in the following steps with an indication of where any discussion of that step was presented in parentheses.

1. Determine the set of computations with associated triggers which satisfy the requirements of the inference task. (Section 4.2)

2. Construct a graph topology for a hierarchy encoding a data collection mechanism with partial processing of the triggers for the purpose of evaluating the computations derived from step 1. The important aspect for the algorithmic construction of the graph topology is to pick heuristics which correspond to optimizing the cost criterion.

3. Assign nodes of the hierarchy to agents of the sensor network. Choose uplinks, downlinks, and trigger sets for each edge of the hierarchy. (Section 4.1)

4. Convert the hierarchy into a set of communication algorithms. (Section 4.1)

All steps 1 through 4 are intended to be automated once the various parameters needed for automation are specified.

Step 1 requires a choice of the discretization parameter k and the target location sample spacing d. Furthermore, the sensor measurement models need to be known, and the thresholds for determining the likely set of measurements must be specified. Then, the set of computations and triggers can be automatically computed.

Step 2 requires a specification of a heuristic to algorithmically construct the graph topology. This step takes the set of computations and triggers as inputs and constructs a graph topology for a hierarchy.

Step 3 requires a specification of the cost criterion of the problem. This step takes the graph topology computed in the previous step and chooses an agent assignment, uplinks, downlinks, and trigger sets by optimizing the cost criterion. The result is a hierarchy.

Finally, step 4 can extract the communication algorithms from the constructed hierarchy.

Steps 2 and 3 encompass the algorithmic construction of the hierarchy which we have broken into two parts: (1) construction of the graph topology and (2) agent assignments, uplinks, downlinks, and trigger sets. The two steps need not be separated so that agent assignments and choice of uplinks, downlinks, and trigger sets can be made while the graph topology is being constructed. Section 4.5 will present one such variation of the construction of the hierarchy which is especially relevant for constructing DCA's for target localization. Furthermore, we address the issue of scalability of the hierarchy construction algorithm in Section 4.6.

4.3.2 Step 1: Set of Computations

In this step, we will derive a set of computations and their triggers for the inference task. We choose the following parameters which were left unspecified before.

$$\begin{aligned} p_2 &= .1 \\ p_1 &= .1 \\ p_0 &= .1 \\ \rho &= .4 \\ \gamma &= .17 \\ P_\Theta(\theta) &= \frac{1}{8} \text{ for all } \theta \in \Theta \end{aligned}$$

With the above parameters,

$$\hat{Z}_\theta = \{(\mathbf{nom}_1(\theta), \ldots, \mathbf{nom}_8(\theta))\}$$

which is the nominal value of the measurement when the target is located at θ. Then, the set of likely measurements

$$\mathcal{Z}_{likely} = \bigcup_{\theta \in \Theta} \hat{Z}_\theta$$

is the set of nominal sensor measurements for each of the eight target locations in Θ. Let us denote the eight nominal sensor measurements corresponding to target location θ_i by z^i. Specifically, these are

$$\begin{aligned} z^1 &= (1,0,0,2,0,0,0,1) \\ z^2 &= (1,1,0,2,0,0,0,0) \\ z^3 &= (0,2,0,1,1,0,1,0) \\ z^4 &= (0,1,1,0,2,0,0,0) \\ z^5 &= (0,0,1,0,2,1,0,0) \\ z^6 &= (0,0,0,0,2,1,1,0) \\ z^7 &= (0,1,0,1,1,0,2,0) \\ z^8 &= (0,0,0,2,0,0,1,1) \end{aligned}$$

Since the measurement space is already discrete, we have no need for discretization, although if the number of elements in the measurement space is large we may wish to choose an equivalence relation to keep the number of "points" to a reasonable number.

For each $z \in \mathcal{Z}_{likely}$, we can compute the minimal subset of measurements to collect

$$S_z = \arg_{S \subset \{1,\ldots,8\}: \tilde{P}_{z_S} \approx \tilde{P}_z} \min |S|$$

where

$$\tilde{P}_{z_S}(\theta) = \alpha P_\Theta(\theta) \prod_{i \in S} P_i(z_i|\theta)$$

is the posterior probability mass function on Θ given sensor measurements in S and α is a normalization constant. By $\approx$, we mean that the two probability mass functions have an absolute difference less than $\gamma = .17$, as specified in the statement of the inference task. The results of the calculation of S_z for $z \in \mathcal{Z}_{likely}$ are

$$\begin{aligned}
S_{z^1} &= \{1, 4, 8\} \\
S_{z^2} &= \{1, 2, 4\} \\
S_{z^3} &= \{2, 4\} \\
S_{z^4} &= \{2, 3, 5\} \\
S_{z^5} &= \{3, 5, 6\} \\
S_{z^6} &= \{5, 6, 7\} \\
S_{z^7} &= \{5, 7\} \\
S_{z^8} &= \{4, 7, 8\} \ .
\end{aligned}$$

We use the simpler notation S_i to denote S_{z^i}. Thus, the function representing the global inference task is

$$f(z_1, \ldots, z_8) = \begin{cases} \tilde{P}_{z_{S_z}} & , \ z \in \mathcal{Z}_{likely} \\ \text{undefined} & , \ \text{otherwise} \end{cases} \ .$$

The set of computations in the above function can be indexed by an integer $i \in \{1, \ldots, 8\}$ corresponding to the eight elements in

$$\mathcal{Z}_{likely} = \{z^1, z^2, \ldots, z^8\} \ .$$

Thus, define the function

$$g_i : \mathcal{Z}_{S_i} \to \mathcal{P}(\Theta)$$

by

$$g_i(z_{S_i}) = \tilde{P}_{z_{S_i}} \ .$$

The notation above means specifically the following. Since the range of g_i is the set of probability mass functions, we will show the dependence on θ in subscript. See

below.

$$
\begin{aligned}
g_1(z_1, z_4, z_8)_{(\theta)} &= \alpha_1 P_\Theta(\theta) P_1(z_1|\theta) P_4(z_4|\theta) P_8(z_8|\theta) \\
g_2(z_1, z_2, z_4)_{(\theta)} &= \alpha_2 P_\Theta(\theta) P_1(z_1|\theta) P_2(z_2|\theta) P_4(z_4|\theta) \\
g_3(z_2, z_4)_{(\theta)} &= \alpha_3 P_\Theta(\theta) P_2(z_2|\theta) P_4(z_4|\theta) \\
g_4(z_2, z_3, z_5)_{(\theta)} &= \alpha_4 P_\Theta(\theta) P_2(z_2|\theta) P_3(z_3|\theta) P_5(z_5|\theta) \\
g_5(z_3, z_5, z_6)_{(\theta)} &= \alpha_5 P_\Theta(\theta) P_3(z_3|\theta) P_5(z_5|\theta) P_6(z_6|\theta) \\
g_6(z_5, z_6, z_7)_{(\theta)} &= \alpha_6 P_\Theta(\theta) P_5(z_5|\theta) P_6(z_6|\theta) P_7(z_7|\theta) \\
g_7(z_5, z_7)_{(\theta)} &= \alpha_7 P_\Theta(\theta) P_5(z_5|\theta) P_7(z_7|\theta) \\
g_8(z_4, z_7, z_8)_{(\theta)} &= \alpha_8 P_\Theta(\theta) P_4(z_4|\theta) P_7(z_7|\theta) P_8(z_8|\theta)
\end{aligned}
$$

where $\{\alpha_i\}_{i=1}^8$ are normalization constants. Thus, we have rewritten the function as

$$
f(z_1, \ldots, z_8) = \begin{cases}
g_1(z_1, z_4, z_8) & , \ z = z^1 \\
g_2(z_1, z_2, z_4) & , \ z = z^2 \\
g_3(z_2, z_4) & , \ z = z^3 \\
g_4(z_2, z_3, z_5) & , \ z = z^4 \\
g_5(z_3, z_5, z_6) & , \ z = z^5 \\
g_6(z_5, z_6, z_7) & , \ z = z^6 \\
g_7(z_5, z_7) & , \ z = z^7 \\
g_8(z_4, z_7, z_8) & , \ z = z^8 \\
\text{undefined} & , \ \text{otherwise}
\end{cases} .
$$

What remains to be done is to make the triggers (i.e. the tests $z = z^i$ for $i \in \{1, \ldots, 8\}$) depend on a subset of measurements. That is, we consider the tests

$$
T_i(z_{S_i}) = [z_{S_i} = z^i_{S_i}]
$$

where $[\cdot]$ is Iverson's convention. Unwrapping the notation above, these tests are specifically,

$$
\begin{aligned}
T_1(z_1, z_4, z_8) &= [z_1 = 1][z_4 = 2][z_8 = 1] \\
T_2(z_1, z_2, z_4) &= [z_1 = 1][z_2 = 1][z_4 = 2] \\
T_3(z_2, z_4) &= [z_2 = 2][z_4 = 1] \\
T_4(z_2, z_3, z_5) &= [z_2 = 1][z_3 = 1][z_5 = 2] \\
T_5(z_3, z_5, z_6) &= [z_3 = 1][z_5 = 2][z_6 = 1] \\
T_6(z_5, z_6, z_7) &= [z_5 = 2][z_6 = 1][z_7 = 1] \\
T_7(z_5, z_7) &= [z_5 = 1][z_7 = 2] \\
T_8(z_4, z_7, z_8) &= [z_4 = 2][z_7 = 1][z_8 = 1]
\end{aligned}
$$

	z^1	z^2	z^3	z^4	z^5	z^6	z^7	z^8
T_1	1	0	0	0	0	0	0	0
T_2	0	1	0	0	0	0	0	0
T_3	0	0	1	0	0	0	0	0
T_4	0	0	0	1	0	0	0	0
T_5	0	0	0	0	1	0	0	0
T_6	0	0	0	0	0	1	0	0
T_7	0	0	0	0	0	0	1	0
T_8	0	0	0	0	0	0	0	1

Table 4.3: Trigger evaluations for each of the eight joint measurements in $\mathcal{Z}_{likely}$.

If for all $z \in \mathcal{Z}_{likely}$, there is exactly one T_i that evaluates to 1 and all others are 0, then we can use the above tests as triggers. Table 4.3 shows the evaluations of each T_i for the eight joint measurements in $\mathcal{Z}_{likely}$. Since these tests satisfy the condition, we have derived the following function which is well-defined over the subset of the domain $\mathcal{Z}_{likely}$ and satisfies the requirements of the inference task.

$$f'(z_1, \ldots, z_8) = \begin{cases} g_1(z_1, z_4, z_8) & , \; T_1(z_1, z_4, z_8) = 1 \\ g_2(z_1, z_2, z_4) & , \; T_2(z_1, z_2, z_4) = 1 \\ g_3(z_2, z_4) & , \; T_3(z_2, z_4) = 1 \\ g_4(z_2, z_3, z_5) & , \; T_4(z_2, z_3, z_5) = 1 \\ g_5(z_3, z_5, z_6) & , \; T_5(z_3, z_5, z_6) = 1 \\ g_6(z_5, z_6, z_7) & , \; T_6(z_5, z_6, z_7) = 1 \\ g_7(z_5, z_7) & , \; T_7(z_5, z_7) = 1 \\ g_8(z_4, z_7, z_8) & , \; T_8(z_4, z_7, z_8) = 1 \\ \text{undefined} & , \; \text{otherwise} \end{cases} .$$

Therefore, the set of computations to encode into a hierarchy is

$$\{g_i\}_{i=1}^8$$

with associated triggers

$$\{T_i\}_{i=1}^8$$

where the computations and triggers depend on the sensor measurements

$$\{S_i\}_{i=1}^8 .$$

4.3.3 Step 2: Graph Topology Construction

The next step is to construct a hierarchy which encodes the set of computations $\{g_i\}_{i=1}^8$ and triggers $\{T_i\}_{i=1}^8$ derived in the previous step. In Section 4.1, we presented the hierarchy in a step by step fashion where we could construct the graph topology

and then add in the effect of partial processing by specifying a computation with each node, specifying uplinks and downlinks to edges which correspond to real communications, and associating a trigger set with each uplink and downlink. We could do the same here, but construction of the hierarchy becomes very complicated because of the multitude of functions we can specify at each node and the set of possible trigger sets to associate with each uplink and downlink.

The idea then is to reduce everything down to 1's and 0's (i.e. to the Boolean domain). Then, the only computations that can be associated with a node are single output Boolean functions and the trigger sets on each uplink or downlink can only be a subset of $\{0, 1\}$.

The first thing we must do is to represent the joint measurement $z \in \mathcal{Z}$ in terms of 1's and 0's. Define the token indicator function

$$\pi_k^v : \mathcal{Z} \to \{0, 1\}$$

by

$$\pi_k^v(z) = [z_k = v]$$

where $k \in \{1, \ldots, n\}$, z_k refers to the kth component of z, $v \in \mathcal{Z}_k$ is an element of the measurement space of sensor node k, and $[\cdot]$ is Iverson's convention. Note that if we are given the evaluations of all $\{\pi_k^v(z)\}_{v \in \mathcal{Z}_k, k \in \{1, \ldots, n\}}$ for some unknown $z \in \mathcal{Z}$, then we can determine what z is.

In the more general case when we have discretized the measurement space according to an equivalence relation $\sim$ as constructed in Section 4.2, the corresponding set of token indicator functions are

$$\pi_k^v : \mathcal{Z} \to \{0, 1\}$$

defined by

$$\pi_k^v(z) = [z_k \in v]$$

where $k \in \{1, \ldots, n\}$, z_k refers to the kth component of z, v refers to an equivalence class in $\mathcal{Z}_k$ derived from the equivalence relation $\sim_k$, and the square brackets $[\cdot]$ is Iverson's convention. Let us define

$$\mathcal{Z}_k = \{[z_k] : z_k \in \mathcal{Z}_k\}$$

to be the set of all equivalence classes of $\mathcal{Z}_k$ according to the equivalence relation $\sim_k$. In this general setting, given the evaluations of all $\{\pi_k^v(z)\}_{v \in \mathcal{Z}_k, k \in \{1, \ldots, n\}}$ for some unknown $z \in \mathcal{Z}$, we can determine in which equivalence class z is a member. This is enough information in order to evaluate the triggers T_i since these triggers are only a function of the equivalence class that each joint measurement lies in. However, this is not enough information to evaluate the associated computation g_i in general.

Let us define a set of Boolean variable s $\{a_k^v\}_{v \in \mathcal{Z}_k, k \in \{1, \ldots, n\}}$ which is related to the observed measurement $z \in \mathcal{Z}$ by

$$a_k^v = \pi_k^v(z) \ .$$

We will rewrite the triggers $\{T_i\}_{i=1}^8$ in terms of the Boolean variables $\{a_k^v\}_{v,k}$. We will abuse notation here and use T_i to stand for the Boolean variable associated with the trigger T_i.

$$\begin{aligned}
T_1 &= a_1^1 \wedge a_4^2 \wedge a_8^1 \\
T_2 &= a_1^1 \wedge a_2^1 \wedge a_4^2 \\
T_3 &= a_2^2 \wedge a_4^1 \\
T_4 &= a_2^1 \wedge a_3^1 \wedge a_5^2 \\
T_5 &= a_3^1 \wedge a_5^2 \wedge a_6^1 \\
T_6 &= a_5^2 \wedge a_6^1 \wedge a_7^1 \\
T_7 &= a_5^1 \wedge a_7^2 \\
T_8 &= a_4^2 \wedge a_7^1 \wedge a_8^1
\end{aligned}$$

The symbol $\wedge$ denotes the Boolean AND operator. Now, we can begin constructing a hierarchy for the Boolean variables $\{T_i\}_{i=1}^8$.

Let us take a moment to explain why we are only concerned with the triggers. One may wonder how we can justify ignoring the computations $\{g_i\}_{i=1}^8$ because these are the computations we must evaluate. Furthermore, partial processing of these computations can reduce the number of bits that must be communicated throughout the network so that the information collection process is certainly affected by the partial processing of the computations. However, the current discussion will present the extreme case when any partial processing of computations during data collection is prohibited. At this extreme, the best we can do is to pass those data that need to be passed around as efficiently as possible. Then, after enough data has been gathered by some agent of the network which is the signal that a stopping condition has been reached, the agent can perform the residual computation of evaluating the appropriate g_i. The notion of what data needs to be passed around is completely captured by the triggers $\{T_i\}_{i=1}^8$ and not by the computations $\{g_i\}_{i=1}^8$. The notion of what information needs to be passed around is still primarily captured by the triggers because the only effect that partial processing of the computations have is to change the cost of passing *data* around to the cost of passing *information* around. In the summary (Section 4.3.7), we will quickly discuss how to add the effect of partial processing of the $\{g_i\}_{i=1}^8$ into the algorithmic hierarchy construction.

However, in the extreme case of no partial processing of the computations, we are not disallowing partial processing of the triggers. Because each trigger T_i is expressed as a conjunction of several Boolean variables, if it is known that one of these Boolean variables evaluated to 0, then we automatically know that T_i evaluates to 0. Thus, the partial processing performed in each node of the hierarchy for the purpose of evaluating triggers is a logical AND operation. Furthermore, the trigger sets associated with each uplink or downlink in the resulting hierarchy is the subset $\{1\}$.

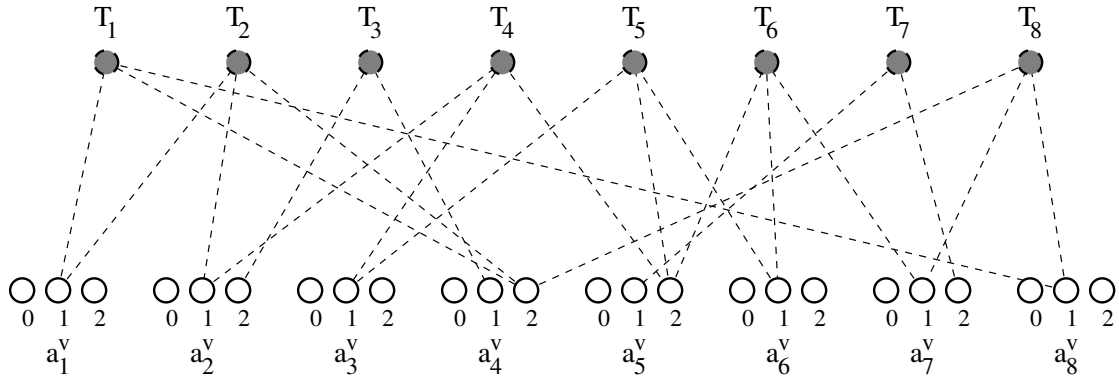

Figure 4-12: The initial graph representing the dependencies of the triggers $\{T_i\}$ on the measurement tokens $\{a_k^v\}_{v \in \mathcal{Z}_k, k \in \{1,\dots,n\}}$.

We begin with a graph consisting of a node corresponding to each trigger $\{T_i\}_{i=1}^8$ and a node for each of the measurement tokens $\{a_k^v\}_{v,k}$ as shown in Figure 4-12. As usual, we will abuse terminology and refer to the "node corresponding to a trigger T_i" by the "node T_i", and the "node corresponding to a measurement token a_k^v" by the "node a_k^v". Edges emanating downwards from each node T_i are connected to the measurement tokens that T_i is dependent. As discussed in Section 4.1.1, we wish to replace any node with down degree greater than 2 with a subgraph consisting only of nodes with down degree equal to 2 or 0. What we did not discuss in that section, however, was how to algorithmically construct a graph topology so that all nodes have down degree equal to 2 or 0. In Section 4.1.3 we discussed how the particular graph topology chosen determines the amount of communication that must occur. Thus, it is important that the algorithmic construction of the graph topology eventually leads to an efficient communication algorithm. This algorithmic construction of the graph topology is the first part of the algorithmic construction of the hierarchy, and the general treatment of this construction is given in Chapter 5 Section 5.5.

For this toy example, let us describe the general idea of algorithmically constructing the graph topology. Construction proceeds in three steps which iteratively updates the graph until all nodes of the graph have down degree equal to two.

Graph Topology Construction Algorithm

1. **Enumeration** We first enumerate a set of possible nodes B of down degree two which are candidates for reducing the down degree of at least one of the trigger nodes in the graph. Initially, the nodes T_1, T_2, T_4, T_5, T_6, and T_8 of the graph in Figure 4-12 are the nodes with down degree greater than two. For every pair of measurement tokens a_k^v and $a_{k'}^{v'}$, we can test whether there exists a T_i for $i \in \{1,2,4,5,6,8\}$ which is connected to both a_k^v and $a_{k'}^{v'}$. If there

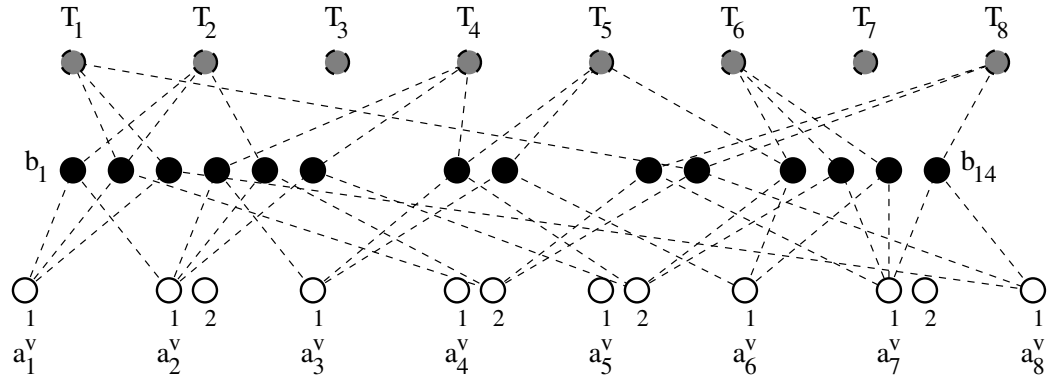

Figure 4-13: The black nodes, labelled b_1 to b_{14} from left to right, are the set of enumerated nodes from the enumeration step which are candidates for reducing the down degree of some node T_i. The number of dashed edges emanating above each enumerated node (black node) indicates the commonality heuristic. We have omitted those nodes a_k^v which were not connected to any node T_i.

does exist such a T_i, we add a node with down degree 2 with two children a_k^v and $a_{k'}^{v'}$ to the enumeration set B. The result of this enumeration results in B containing 14 nodes as represented by the black nodes in Figure 4-13. The interpretation of the set of enumerated nodes B is the set of possible one step data collections which is useful for the evaluation of at least one of the triggers. Certainly, some of the data collections represented by the nodes in B are more useful than others which is the point of the next step.

2. **Compute Heuristic and Rank** This step is the critical step which determines which graph topology will be constructed. For each of the enumerated nodes in B, we must compute a score based on a heuristical measure of its relative worth (e.g. some measure of information that it provides) weighed by the cost of the computational resources expended. A more general discussion of good heuristics to use will be presented in Section 4.4. To illustrate the hierarchy construction process in this toy example, let us use the *commonality heuristic* as a measure of worth. The commonality heuristic measures for each enumerated node $b \in B$, the number of nodes above which can be rewritten with this node. Graphically, this heuristic corresponds to the number of dashed edges emanating upwards from each of the black nodes in Figure 4-13. Labelling the black nodes from b_1 at the far left side to b_{14} at the far right side, we express the commonality heuristic formally by the function

$$\mathbf{common}(b) = \begin{cases} 2 & , \ b \in \{b_2, b_7, b_{10}, b_{11}\} \\ 1 & , \ \text{otherwise} \end{cases} .$$

The intuition behind using the commonality of an enumerated node $b \in B$ as a measure of worth is that if there are many functions which depend on collecting the data represented by b, then this communication is less wasteful than collecting data which few functions depend on. Thus, we wish to choose a node b which maximizes the commonality heuristic.

As for the cost of computational resources expended, we can measure the energy expended in communicating the data. For any node $b \in B$, which is connected to a_k^v and $a_{k'}^{v'}$, since a_k^v resides in sensor node k and $a_{k'}^{v'}$ resides in sensor node k', the distance of the communication is

$$d_b = d(x_k, x_{k'})$$

which is the Euclidean distance between the position, $x_k \in \mathbb{R}^2$, of sensor node k and the position, $x_{k'} \in \mathbb{R}^2$, of sensor node k'. In future iterations of this step, however, since a node $b \in B$ may be connected to nodes c and c' which were added to the hierarchy from previous iterations, we need an estimate of the distance of communication because we have not yet chosen an agent assignment to nodes c and c'. Let us use a worst case criterion for this example. Collect the set A of measurement tokens a_k^v which are descendants of b. Since we know the agent assignment of each a_k^v is k, we can compute the maximal distance between any two agents assigned to a token in A. This maximal distance will be denoted d_b.

The number of bits that are sent depends on the agent assignment of hierarchy node b and the agent assignments of its two children. However, since we have not yet chosen agent assignments, we must compute some estimate of the number of bits.[9] Again, let us use a worst case criterion so that the number of bits is the maximum of the number of bits that is contained in either child of b. Let us denote by

$$\mathbf{bits}_b$$

the number of bits of data contained in node b. For example, a measurement token a_k^v is considered to contain one bit of data. If b has j descendants that are measurement tokens, then b is considered to contain j bits. Then, for a node b, the maximum number of bits in either child of b is expressed by

$$\max\left\{\mathbf{bits}_{l(b)}, \mathbf{bits}_{r(b)}\right\}$$

[9]Rather than constructing the hierarchy completely and then choosing an agent assignment, uplinks, downlinks, and trigger sets as we are demonstrating in these four steps of the toy example, we could choose agent assignments, uplinks, downlinks, and trigger sets as we construct the graph topology which we will demonstrate in Section 4.5. However, a possible advantage of separating these steps is that we can choose agent assignments, uplinks, downlinks, and trigger sets in a more optimal fashion after the entire graph topology is known. Whether to separate the steps or to perform them all together is a choice that depends on the quality of the chosen heuristics used to construct the hierarchy and the details of the problem.

rank	b	children	**common**	**cost**
1	b_2	$\{a_1^1, a_4^2\}$	2	120.0
1	b_7	$\{a_3^1, a_5^2\}$	2	120.0
1	b_{10}	$\{a_4^2, a_8^1\}$	2	120.0
1	b_{11}	$\{a_5^2, a_6^1\}$	2	120.0
5	b_5	$\{a_2^1, a_4^2\}$	1	132.5
5	b_6	$\{a_2^1, a_5^2\}$	1	132.5
5	b_9	$\{a_4^2, a_7^1\}$	1	132.5
5	b_{12}	$\{a_5^2, a_7^1\}$	1	132.5
9	b_3	$\{a_1^1, a_8^1\}$	1	140.0
9	b_8	$\{a_3^1, a_6^1\}$	1	140.0
11	b_1	$\{a_1^1, a_2^1\}$	1	142.5
11	b_4	$\{a_2^1, a_3^1\}$	1	142.5
11	b_{13}	$\{a_6^1, a_7^1\}$	1	142.5
11	b_{14}	$\{a_7^1, a_8^1\}$	1	142.5

Table 4.4: Heuristic values for each enumerated node shown ranked according to larger **common** followed by smaller **cost** if there are ties. Rank numbers are repeated if there is no way to break a tie.

where $l(b)$ denotes the left child of b and $r(b)$ denotes the right child of b.

The cost of a node $b \in B$ is given by

$$\mathbf{cost}(b) = E_{Tx}(\max\left\{\mathbf{bits}_{l(b)}, \mathbf{bits}_{r(b)}\right\}, d_b) + E_{Rx}(\max\left\{\mathbf{bits}_{l(b)}, \mathbf{bits}_{r(b)}\right\})$$

where E_{Tx} and E_{Rx} were given by Eqn. 2.1 and Eqn. 2.2.

To rank the set of enumerated nodes, we will combine the commonality heuristic and cost of communication in a dictionary type order. In particular, we rank nodes according to larger value of **common**. Of the nodes which tie in **common**, we break the ties by ranking nodes according to smaller value of **cost**. If there are still ties, then we just choose an arbitrary order for the tied nodes. Table 4.4 shows the resulting ranking according to the commonality heuristic and cost. In computing **cost** for each node $b \in B$, we used the sensor positions of the nodes of the network as given in Table 4.2.

3. **Threshold and Update** The final step of this iteration is to choose the top best ranking nodes and add these nodes to the hierarchy. For this toy example, we will choose only one of the best ranking nodes. These are the node b_2, b_7, b_{10}, and b_{11}. We choose one of these arbitrarily, so say we choose b_2. Figure 4-14 shows the resulting hierarchy after node b_2 is added to the hierarchy. We update the edges emanating downwards from all nodes which can have b_2 as a child.

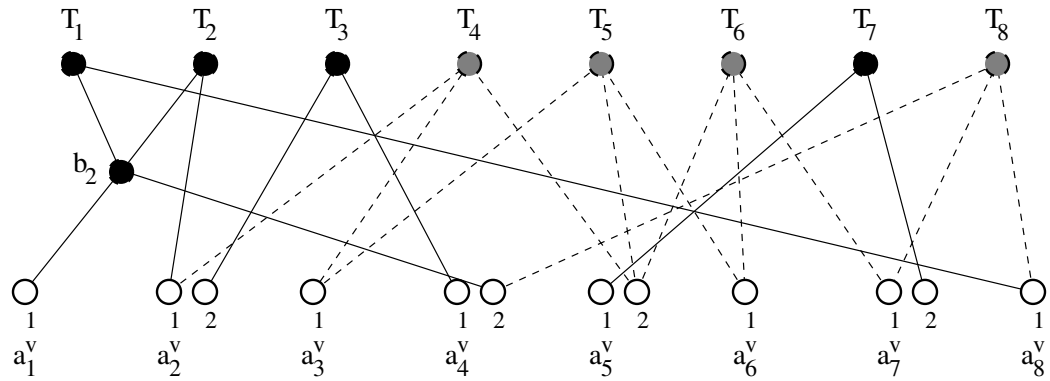

Figure 4-14: The updated hierarchy after adding enumerated node b_2 into the hierarchy. The gray nodes indicate nodes of down degree greater than 2 which we must replace with a subgraph of nodes.

This means that the nodes T_1 and T_2 are updated as shown in the figure. Note that T_1 and T_2 both have down degree equal to two now.

Figure 4-15 shows the rest of the iterations of the above three steps which results in the graph topology shown at the bottom of the figure. Note that all nodes of the hierarchy have down degree two. The set of steps above are a greedy method for constructing a hierarchy in the sense that the node added is the best such node to add at each iteration given the current partially constructed graph topology. As with all greedy algorithms, the heuristical construction of the hierarchy can only guarantee a locally optimal solution.

4.3.4 Step 3: Agent Assignments, Uplinks, Downlinks, and Trigger Sets

From step 2, we have constructed a particular graph topology of a hierarchy which should result in an efficient data collection scheme. Once we choose an agent assignment, we can label those edges which correspond to a real data transfer as an uplink or downlink. As discussed in Section 4.1.1, uplinks correspond to "push" type communications and downlinks correspond to "pull" type communications. There will certainly be a difference in cost for the different choices of uplinks and downlinks not only due to the extra query message needed in the "pull" type communication but also because the trigger set is tested against the local knowledge state of a different node. We will discuss this later in this section when we demonstrate how to choose an agent assignment, along with a choice of uplinks and downlinks, for the toy sensor network example.

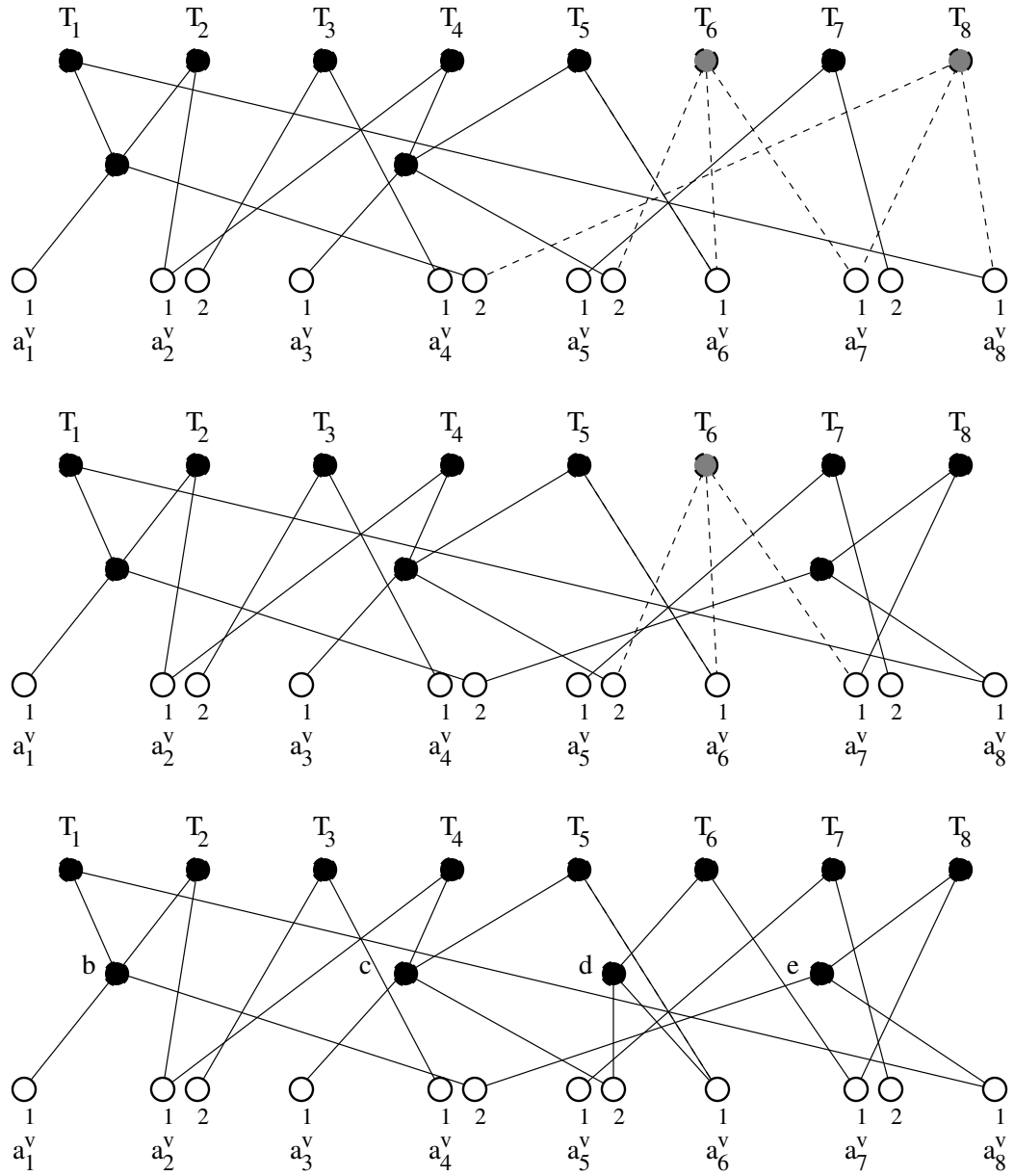

Figure 4-15: The progression of the rest of the iterations to form the hierarchy. The top two figures show the hierarchy after two more iterations. In the resulting graph topology of the hierarchy shown at the bottom of the figure, each black node is associated with a logical AND function. These AND functions are the partial computations for the triggers.

Let us discuss the trigger sets. In the resulting graph topology for the hierarchy, we associate a logical AND function with all nodes other than the set of nodes corresponding to the measurement tokens. This is the partial processing for evaluating the triggers. Note that if the AND function evaluates to 1 in a given node, then there is a possibility that nodes connected above can evaluate to a 1. This means that further communications may have to take place. On the other hand, if the AND function evaluates to 0 in a given node, then there is no possibility that any node connected above can evaluate to 1. Since a trigger T_i must evaluate to 1 before we evaluate the associated computation g_i, further communications of the local knowledge state at a node need not be communicated if the AND function evaluates to 0. Thus, the trigger set associated with any uplink or downlink is the subset $\{1\}$.

Let us now discuss in detail how to choose an agent assignment for each node. To formalize, let us denote the set of nodes in the hierarchy (bottom graph of Figure 4-15) by

$$N = \{a_1^1, a_2^1, a_2^2, a_3^1, a_4^1, a_4^2, a_5^1, a_5^2, a_6^1, a_7^1, a_7^2, a_8^1, b, c, d, e, T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\} \ .$$

To formalize, the idea is to choose an agent assignment

$$\mathcal{A} : N \to \{1, \ldots, 8\} \ ,$$

where N is the set of nodes in the hierarchy and $\{1, \ldots, 8\}$ represents the set of sensor nodes in the sensor network, such that some cost criterion is minimized. Not all agent assignments may correspond to a feasible assignment of hierarchy nodes to sensor nodes. For this toy example, the measurement tokens $a_k^v \in N$ must be assigned to sensor node k since that is the node where the value of a_k^v can be observed without any communication. Thus, we will call an agent assignment feasible if

$$\mathcal{A}(a_k^v) = k$$

for the measurement tokens $a_k^v \in N$. Furthermore, we denote the set of all feasible agent assignments by $\mathbf{A}$. Now, given some cost criterion

$$\mathbf{cost} : \mathbf{A} \to \mathbb{R}^+$$

where $\mathbb{R}^+$ refers to the non-negative reals, the optimal choice of agent assignment $\hat{\mathcal{A}}$ is given by

$$\hat{\mathcal{A}} = \arg_{\mathcal{A} \in \mathbf{A}} \min \mathbf{cost}(\mathcal{A}) \ .$$

Unfortunately, the set of feasible agent assignments $\mathbf{A}$ is far too large to solve the optimization above. Thus, the first idea will be to limit the set of possible assignments.

We will limit the set of agent assignments to those which only allow a node to be assigned to the agent assigned to one of its two children. For example, in the bottom graph of Figure 4-15, the node b can only be assigned to the agent assigned to a_1^1 or a_4^2, and the node T_5 can only be assigned to the agent assigned to node c or node

a_6^1. Thus, it can be seen that since we have two choices of agents for each of the 12 black nodes, there are a total of 2^{12} feasible agent assignments under this limitation. This is in contrast to the 8^{12} possible number of feasible agent assignments without this limitation since each of the twelve black nodes can be assigned to eight different sensor nodes. Although under this limitation there is a reduction in the size of the set of feasible agent assignments, the set still grows exponentially with the number of nodes in the hierarchy. Thus, some suboptimal way of picking an agent assignment is needed.

Under the limitation of the set of agent assignments, we can parameterize the set of agent assignments by elements of the space

$$\{0,1\}^{12} \ .$$

We associate each component of the space above with one of the black nodes in the hierarchy. For a parameter $A \in \{0,1\}^{12}$, if the kth component of A, denoted A_k, equals 0, then the corresponding node is assigned the agent of its left child. If A_k equals 1, then the corresponding node is assigned the agent of its right child. Since the measurement tokens a_k^v must be assigned to sensor node k, every parameter A corresponds to a feasible agent assignment $\mathcal{A}$ which we represent by a function

$$\mathcal{M} : \{0,1\}^{12} \to \mathbf{A} \ .$$

For example, if the parameter A is all zeros, meaning all black nodes are assigned the agent of its left child in the bottom graph of Figure 4-15, then the corresponding agent assignment

$$\mathcal{A} = \mathcal{M}(A)$$

is given by

$$\mathcal{A}(g) = \begin{cases} 1, \ g \in \{a_1^1, b, T_1, T_2\} \\ 2, \ g \in \{a_2^1, a_2^2, T_3, T_4\} \\ 3, \ g \in \{a_3^1, c, T_5\} \\ 4, \ g \in \{a_4^1, a_4^2, e, T_8\} \\ 5, \ g \in \{a_5^1, a_5^2, d, T_6, T_7\} \\ 6, \ g \in \{a_6^1\} \\ 7, \ g \in \{a_7^1, a_7^2\} \\ 8, \ g \in \{a_8^1\} \end{cases} \ .$$

The idea for suboptimally finding an agent assignment is to find an $A \in \{0,1\}^{12}$ which is locally optimal up to Hamming distance d for some given positive integer d. To be precise, we say that A is locally optimal up to Hamming distance d, if

$$\mathbf{cost}(\mathcal{M}(A)) \leq \mathbf{cost}(\mathcal{M}(B))$$

for all

$$B \in \{C \in \{0,1\}^{12} : d_H(A,C) \leq d\}$$

where $d_H(\cdot,\cdot)$ is the Hamming distance between two equal length binary strings. Such a locally optimal $A \in \{0,1\}^{12}$ can be computed by a straightforward greedy algorithm.

Greedy Algorithm for Agent Assignment
Choose at random some element of $\{0,1\}^{12}$ which we denote A_0. Then, given A_i, we will present the steps for computing A_{i+1}. Given A_i, compute the set of neighbors within Hamming distance d,

$$\mathcal{N}_{A_i}(d) = \{B \in \{0,1\}^{12} : d_H(A_i, B) \leq d\} \ .$$

If for all $B \in \mathcal{N}_{A_i}(d)$,

$$\mathbf{cost}(\mathcal{M}(A_i)) \leq \mathbf{cost}(\mathcal{M}(B)) \ ,$$

then A_i is a locally optimal point, and we are done. Otherwise, compute

$$A_{i+1} = \arg_{B \in \mathcal{N}_{A_i}(d)} \min \ \mathbf{cost}(\mathcal{M}(B))$$

and iterate.

Let us illustrate the steps of choosing a locally optimal agent assignment which minimizes the average communication cost in the toy example. For every node in the hierarchy which is not a measurement token, there are exactly two edges emanating downwards. Furthermore, at most one of these edges corresponds to a real communication. Thus, we have a choice of whether to make the edge an uplink or a downlink. Given a probability distribution P, which we will choose to be uniform over the elements in $\mathcal{Z}_{likely}$, the average communication cost of an agent assignment $\mathcal{A}$ is given by

$$\mathbf{cost}(\mathcal{A}) = \sum_{g \in N-I} \min\{C_{ul}^{\mathcal{A}}(g), C_{ur}^{\mathcal{A}}(g), C_{dl}^{\mathcal{A}}(g), C_{dr}^{\mathcal{A}}(g)\} \ .$$

The subset of nodes I is the set of input nodes corresponding to measurement tokens so that the subset of nodes $N - I$ is the set of black nodes in Figure 4-15. $C_{ul}^{\mathcal{A}}(g)$ is the contribution to the average communication cost if an uplink connects node g with its left child under the agent assignment $\mathcal{A}$. Similarly, $C_{ur}^{\mathcal{A}}(g)$ is the contribution if an uplink connects node g with its right child. $C_{dl}^{\mathcal{A}}(g)$ is the contribution if a downlink connects node g with its left child. And $C_{dr}^{\mathcal{A}}(g)$ is the contribution if a downlink connects node g with its right child. These costs are given by

$$\begin{aligned}
C_{ul}^{\mathcal{A}}(g) &= C(\mathcal{A}(g), \mathcal{A}(l(g)), \mathbf{bits}_{l(g)}) P([l(g) = 1]) \\
C_{ur}^{\mathcal{A}}(g) &= C(\mathcal{A}(g), \mathcal{A}(r(g)), \mathbf{bits}_{r(g)}) P([r(g) = 1]) \\
C_{dl}^{\mathcal{A}}(g) &= C(\mathcal{A}(g), \mathcal{A}(l(g)), \mathbf{bits}_{l(g)} + k_{query}) P([r(g) = 1]) \\
C_{dr}^{\mathcal{A}}(g) &= C(\mathcal{A}(g), \mathcal{A}(r(g)), \mathbf{bits}_{r(g)} + k_{query}) P([l(g) = 1])
\end{aligned}$$

where $l(g)$ is the left child of node g and $r(g)$ is the right child of node g. The function

$$C : \{1, \ldots, 8\}^2 \times \mathbb{Z}^+ \to \mathbb{R}^+ \ ,$$

where $\mathbb{Z}^+$ is the positive integers and $\mathbb{R}^+$ is the non-negative reals, is defined by

$$C(i,j,k) = \begin{cases} E_{Tx}(k, d_{i,j}) + E_{Rx}(k) & , \ i \neq j \\ 0 & . \ i = j \end{cases} .$$

$C(i,j,k)$ returns the cost of communicating k bits between sensor node i and sensor node j where $d_{i,j}$ is the distance between the two sensors. When $i = j$, there is really no communication occurring so that the cost is 0. By the notation $[g = 1]$ for any node $g \in N$, we mean the event that the AND function associated with node g evaluates to 1.

The difference between a "push" type communication and a "pull" type communication can now be seen explicitly in the difference between the expressions for $C_{ul}^{\mathcal{A}}$ and $C_{dl}^{\mathcal{A}}$ (and similarly, between $C_{ur}^{\mathcal{A}}$ and $C_{ur}^{\mathcal{A}}$). Assuming that the edge $(g, l(g))$ corresponds to a real communication so that the other edge $(g, r(g))$ is not a real communication, the sender of the data is $\mathcal{A}(l(g))$, and the receiver of the data is $\mathcal{A}(r(g))$. Then, with a "push" type communication, the event regarding the local knowledge state of the *sender* which initiates a communication is $[l(g) = 1]$. On the other hand, with a "pull" type communication, it is the event regarding the local knowledge state of the *receiver*, $[r(g) = 1]$, which initiates a communication. Furthermore, there are extra bits k_{query} in a "pull" type communication to communicate the query message as well as the data.

In particular to the toy example, we have the following specific values.

$$\begin{aligned} \mathbf{bits}_{a_k^v} &= 1 \text{ for all } v \in \{0,1,2\} \text{ and } k \in \{1 \ldots, 8\} \\ \mathbf{bits}_g &= 2 \text{ for all } g \in \{b, c, d, e, T_3, T_7\} \\ \mathbf{bits}_g &= 3 \text{ for all } g \in \{T_1, T_2, T_4, T_5, T_6, T_8\} . \end{aligned}$$

We will choose P to be uniform over elements of $\mathcal{Z}_{likely}$ so that $P([g = 1])$ for each $g \in N$ is given by the following table.

g	$P([g=1])$	g	$P([g=1])$
a_1^1	2/8	b	2/8
a_2^1	3/8	c	2/8
a_2^2	1/8	d	2/8
a_3^1	2/8	e	2/8
a_4^1	2/8	T_1	1/8
a_4^2	3/8	T_2	1/8
a_5^1	2/8	T_3	1/8
a_5^2	3/8	T_4	1/8
a_6^1	2/8	T_5	1/8
a_7^1	3/8	T_6	1/8
a_7^2	1/8	T_7	1/8
a_8^1	1/8	T_8	1/8

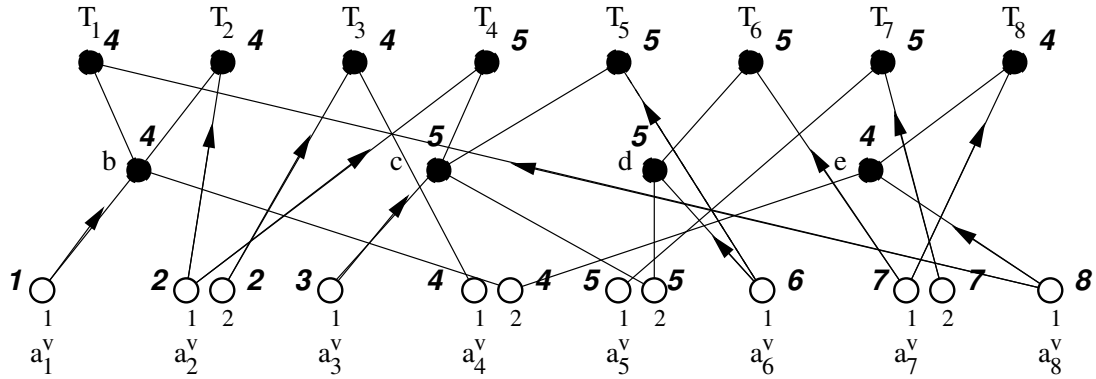

Figure 4-16: The agent assignment $\hat{\mathcal{A}}_1$, denoted as bold italic numbers next to each hierarchy node, minimizes the total communication distance when $k_{query} = 1$. All edges corresponding to a real communication are chosen to be uplinks.

We will compute a locally optimal agent assignment for two cases of the extra number of bits k_{query} for a query message.

- **Case 1** $k_{query} = 1$. This is the case when the number of bits in a query message is comparable to the number of bits in a data message.
- **Case 2** $k_{query} = 0$. This is the case when the number of bits in a query message is negligible compared to the number of bits in a data message.

Running the above greedy algorithm to find a locally optimal agent assignment up to Hamming distance 1 for various random values of the starting parameter A_0 results in the agent assignment $\hat{A}_1$ for case 1.

$$\hat{A}_1(g) = \begin{cases} 1 & , \ g \in \{a_1^1\} \\ 2 & , \ g \in \{a_2^1, a_2^2\} \\ 3 & , \ g \in \{a_3^1\} \\ 4 & , \ g \in \{a_4^1, a_4^2, b, e, T_1, T_2, T_3, T_8\} \\ 5 & , \ g \in \{a_5^1, a_5^2, c, d, T_4, T_5, T_6, T_7\} \\ 6 & , \ g \in \{a_6^1\} \\ 7 & , \ g \in \{a_7^1, a_7^2\} \\ 8 & , \ g \in \{a_8^1\} \end{cases} .$$

The resulting average communication cost of this assignment is

$$\mathbf{cost}(\hat{\mathcal{A}}_1) = 419.375 \ .$$

Figure 4-16 shows the agent assignment $\hat{\mathcal{A}}_1$ by a bold italic number next to each node of the hierarchy. Not surprisingly, the agent assignments of the black nodes were

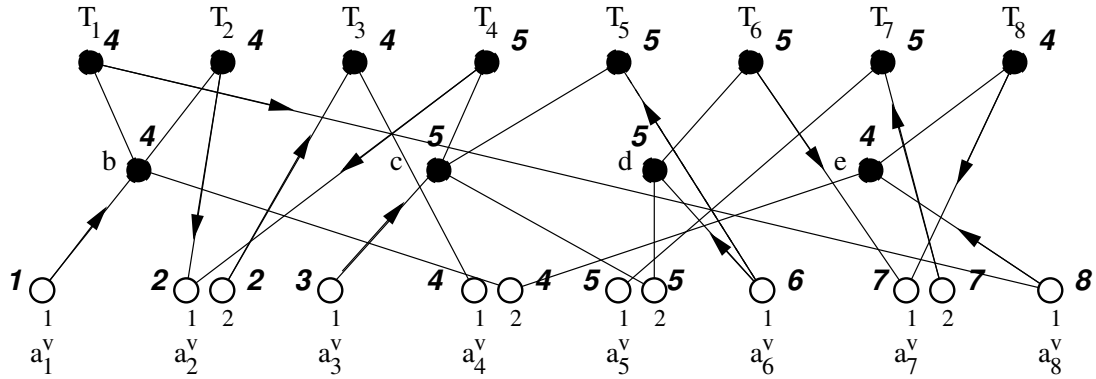

Figure 4-17: The agent assignment $\hat{\mathcal{A}}_2$, denoted as bold italic numbers next to each hierarchy node, minimizes the total communication distance when $k_{query} = 0$. Edges corresponding to a real communication are chosen to be both uplinks and downlinks.

either to sensor node 4 or sensor node 5 since these nodes are located in the middle of the network as shown in Figure 4-11. We expect that by passing data to the middle of the sensor network, less energy is expended by the network to collect the appropriate data. Furthermore, all edges which correspond to a real communication were chosen to be uplinks which is not surprising considering that a query message is expensive in this case.

For case 2, the resulting agent assignment $\hat{A}_2$ was the same as $\hat{A}_1$,

$$\hat{A}_2 = \hat{A}_1 \ .$$

The resulting average communication cost of this assignment is

$$\mathbf{cost}(\hat{\mathcal{A}}_2) = 335.625 \ .$$

This difference in cost is due to choosing some of the edges that correspond to real communications to be downlinks. Since there is no cost of a query message, the difference between choosing uplinks and downlinks is the difference in probability of the event initiating the communication. Figure 4-17 shows the chosen uplinks and downlinks which results in the average communication cost above.

As was discussed at the beginning of this section, the trigger sets associated with each uplink and downlink is the subset $\{1\}$ because the partial processing (for evaluating the triggers $\{T_i\}_{i=1}^{8}$) associated with each node is an AND function.

4.3.5 Step 4: Converting to Communication Algorithms

We will use the hierarchy of Figure 4-17 to demonstrate the conversion of the hierarchy into a set of communication algorithms which corresponds to a distributed algorithm

agent	link	B	R	Q
1	$a_1^1 \to b$	$[a_1^1 = 1]$	4	$\emptyset$
2	$a_2^2 \to T_3$	$[a_2^2 = 1]$	4	$\emptyset$
3	$a_3^1 \to c$	$[a_3^1 = 1]$	5	$\emptyset$
6	$a_6^1 \to d$	$[a_6^1 = 1]$	5	$\emptyset$
6	$a_6^1 \to T_5$	$[a_6^1 = 1]$	5	$\emptyset$
7	$a_7^2 \to T_7$	$[a_7^2 = 1]$	5	$\emptyset$
8	$a_8^1 \to e$	$[a_8^1 = 1]$	4	$\emptyset$
4	$T_1 \to a_8^1$	$[b = 1]$	$\emptyset$	8
4	$T_2 \to a_2^1$	$[b = 1]$	$\emptyset$	2
5	$T_4 \to a_2^1$	$[c = 1]$	$\emptyset$	2
5	$T_6 \to a_7^1$	$[d = 1]$	$\emptyset$	7
4	$T_8 \to a_7^1$	$[e = 1]$	$\emptyset$	7

Table 4.5: The set of local rules generated by the uplinks and downlinks of the hierarchy of Figure 4-17.

for information collection where each sensor node makes local decisions so that the global task of localization is achieved. Recall from the formulation of the single target localization problem in Section 2.3 that a communication algorithm can be viewed as a set of communication rules which is a triple (B, R, Q) of a local knowledge state B, a set of nodes R to send this local knowledge state, and a set of nodes Q to query from. Essentially, the local knowledge state B represents an event that triggers a "push" type communication to sensor nodes of R and a "pull" type communication with sensor nodes of Q. Uplinks correspond to the B and R parts of a communication rule, and downlinks correspond to B and Q parts of a communication rule.

Table 4.5 summarizes the communication rules generated by each uplink and downlinks in the hierarchy of Figure 4-17. We denote a link by

$$a \to b$$

where a and b are nodes. If a is above b, then the link is a downlink. If a is below b, then the link is an uplink. We also show the agent responsible for the communication. The agent is sensor node $\hat{A}_2(a)$ for the link $a \to b$. Let us unwrap the events $[\cdots]$ in the column for B in Table 4.5. Denoting the observed sensor measurement $z = (z_1, \ldots, z_8) \in \mathcal{Z}$, the events correspond to the following.

$$\begin{array}{rcl} [a_1^1 = 1] & = & [z_1 = 1] \\ [a_2^2 = 1] & = & [z_2 = 2] \\ [a_3^1 = 1] & = & [z_3 = 1] \\ [a_6^1 = 1] & = & [z_6 = 1] \\ [a_7^2 = 1] & = & [z_7 = 2] \end{array} \qquad \begin{array}{rcl} [a_8^1 = 1] & = & [z_8 = 1] \\ [b = 1] & = & [z_1 = 1] \cdot [z_4 = 2] \\ [c = 1] & = & [z_3 = 1] \cdot [z_5 = 2] \\ [d = 1] & = & [z_5 = 2] \cdot [z_6 = 1] \\ [e = 1] & = & [z_4 = 2] \cdot [z_8 = 1] \end{array}$$

By arranging the communication rules of Table 4.5 into the appropriate communication algorithm for each sensor node $i \in \{1, \ldots, 8\}$, we have the following communication algorithms C_i. Note we use the set representation as defined in Definition 2.3 for the communication algorithms. Denote the current time t.

$$\begin{array}{ll}
C_1 = & \{ \ (\ \{(t,1,1)\}, \ \{4\}, \ \emptyset \ ) \ \} \\
C_2 = & \{ \ (\ \{(t,2,2)\}, \ \{4\}, \ \emptyset \ ) \ \} \\
C_3 = & \{ \ (\ \{(t,3,1)\}, \ \{5\}, \ \emptyset \ ) \ \} \\
C_4 = & \left\{ \begin{array}{l} (\ \{(t,1,1),(t,4,2)\}, \ \emptyset, \ \{2,8\} \ ), \\ (\ \{(t,4,2),(t,8,1)\}, \ \emptyset, \ \{7\} \ ) \end{array} \right\} \\
C_5 = & \left\{ \begin{array}{l} (\ \{(t,3,1),(t,5,2)\}, \ \emptyset, \ \{2\} \ ), \\ (\ \{(t,5,2),(t,6,1)\}, \ \emptyset, \ \{7\} \ ) \end{array} \right\} \\
C_6 = & \{ \ (\ \{(t,6,1)\}, \ \{5\}, \ \emptyset \ ) \ \} \\
C_7 = & \{ \ (\ \{(t,7,2)\}, \ \{5\}, \ \emptyset \ ) \ \} \\
C_8 = & \{ \ (\ \{(t,8,1)\}, \ \{4\}, \ \emptyset \ ) \ \}
\end{array} \tag{4.16}$$

The communication algorithms C_1, C_2, C_3, C_6, C_7, and C_8 all have a single communication rule of the form

$$(B, R, Q) = (\ \{(t, i, z_i)\}, \ \{j\}, \ \emptyset \ ) \ .$$

The communication encoded is if the node knows that sensor i has observed the measurement z_i at time t, then implement a "push" type communication and send (t, i, z_i) to sensor node j.

On the other hand, the two communication algorithms C_4 and C_5 consist of two communication rules of the form

$$(B, R, Q) = (\ \{(t, i_1, z_{i_1}), (t, i_2, z_{i_2})\}, \ \emptyset, \ \{j\} \ )$$

or

$$(B, R, Q) = (\ \{(t, i_1, z_{i_1}), (t, i_2, z_{i_2})\}, \ \emptyset, \ \{j, k\} \ ) \ .$$

The communication encoded is if the node knows that sensor i_1 has observed the measurement z_{i_1} at time t *and* knows that sensor i_2 has observed the measurement z_{i_2} at time t, then we query either sensor node j (or sensor node j and sensor node k in the second case) for its local knowledge state.

The question may arise: what is the time t? The time t is the time that we wish to perform localization. Thus, if we wish to perform localization at all times, then we make copies of the communication rules in each communication algorithm for each time t.

We have not mentioned the stopping conditions that are to be associated with each node. The stopping conditions are exactly the condition when the triggers are equal to one,

$$T_i = 1 \ .$$

Due to the agent assignment, only certain nodes can evaluate certain stopping conditions. Looking at Figure 4-17, we see that the triggers $\{T_i\}_{i=1,2,3,8}$ have been assigned to sensor node 4, and the triggers $\{T_i\}_{i=4,5,6,7}$ have been assigned to sensor node 5. Thus, the stopping condition for sensor node 4 is if any of the triggers evaluate to 1 in which case sensor node 4 has enough data to compute an estimate of reasonable quality. Specifically, the stopping condition for sensor node 4 is

$$\begin{aligned} STOP_4 = \max\{ \quad & [z_1 = 1] \cdot [z_4 = 2] \cdot [z_8 = 1], \\ & [z_1 = 1] \cdot [z_2 = 1] \cdot [z_4 = 2], \\ & [z_2 = 2] \cdot [z_4 = 1], \\ & [z_4 = 2] \cdot [z_7 = 1] \cdot [z_8 = 1] \ \} \ . \end{aligned} \tag{4.17}$$

If a sensor measurement z_i is unavailable, then we use the convention that

$$[z_i = 1] = 0 \ .$$

Thus, $STOP_4$ returns 1 only if all sensor measurements in at least one element of max term above are known and equal to the specified values. Similarly, the stopping condition for sensor node 5 is

$$\begin{aligned} STOP_5 = \max\{ \quad & [z_2 = 1] \cdot [z_3 = 1] \cdot [z_5 = 2], \\ & [z_3 = 1] \cdot [z_5 = 2] \cdot [z_6 = 1], \\ & [z_5 = 2] \cdot [z_6 = 1] \cdot [z_7 = 1], \\ & [z_5 = 1] \cdot [z_7 = 2] \ \} \ . \end{aligned} \tag{4.18}$$

Note that all of the other nodes have no stopping condition because none of the original triggers $\{T_i\}_{i=1}^{8}$ were assigned to them. To be complete, we set the stopping conditions to be

$$STOP_i = 0$$

for all $i \in \{1, 2, 3, 6, 7, 8\}$.

If node 4 or 5 has determined that the data it has collected satisfies the stopping condition, then the node will compute the posterior density of the target location with the data in its local knowledge state and declare that it has computed the global task.

Walkthru of the Generated Communication Algorithms

We will demonstrate a walk thru of the generated communication algorithms derived in Eqn. 4.16 with stopping conditions given by Eqn. 4.17 and Eqn. 4.18. The walkthru described below is a high-level description of the formal **Sensor Node Operation** given on page 90 of Chapter 2.

Let us consider the case when the observations of the sensor nodes are

$$(z_1, \ldots, z_8) = (0, 1, 1, 0, 2, 0, 0, 0)$$

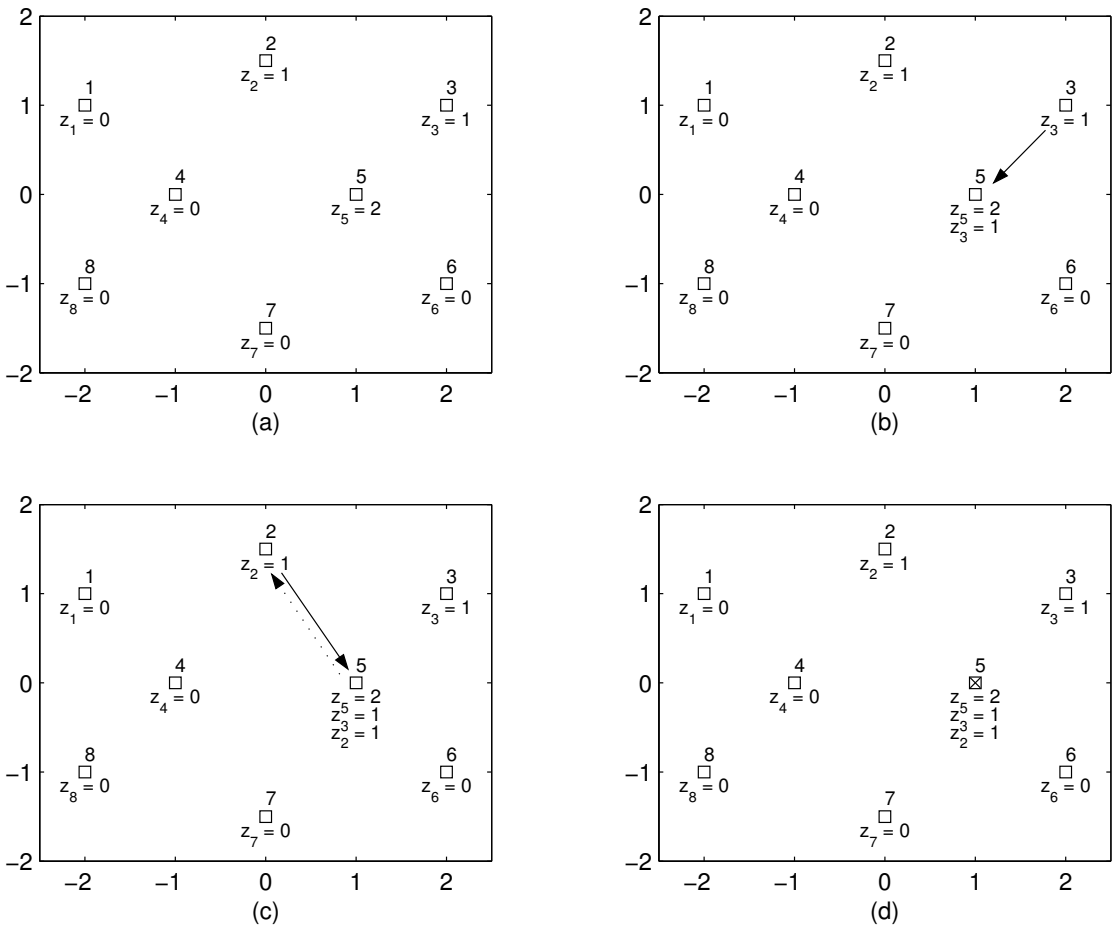

Figure 4-18: The known variable values of each node of the sensor network are shown underneath the respective node. (a) Before communications occur, each sensor node i only knows its own sensor measurement z_i. (b) Then, the communication algorithm in sensor node 3 sends z_3 to sensor node 5 as indicated by the black arrow. (c) The communication algorithm in sensor node 5 queries for the local knowledge state of sensor node 2 as shown by the dotted arrow. Then, sensor node 2 responds with the value of measurement z_2. (d) Sensor node 5 has enough data to satisfy its stopping condition $STOP_5$ as indicated by the $\times$ in sensor node 5.

at some specified time t. Each sensor node $i \in \{1, \ldots, 8\}$ has knowledge of its own sensor measurement z_i as indicated in Figure 4-18a. Using the notation of a local knowledge state as defined in Definition 2.1, the local knowledge state of $B_i \in \mathcal{B}$ for each sensor node $i \in \{1, \ldots, 8\}$ is given by

$$\begin{aligned}
B_1 &= \{(t,1,0)\} \\
B_2 &= \{(t,2,1)\} \\
B_3 &= \{(t,3,1)\} \\
B_4 &= \{(t,4,0)\} \\
B_5 &= \{(t,5,2)\} \\
B_6 &= \{(t,6,0)\} \\
B_7 &= \{(t,7,0)\} \\
B_8 &= \{(t,8,0)\} \ .
\end{aligned}$$

The first step each sensor node i performs is to check whether its current local knowledge state B_i satisfies its stopping condition $STOP_i$. Since none of them do, each sensor node i checks whether any communication rule $(B, R, Q) \in C_i$ in its communication algorithm C_i satisfies

$$B \in B_i \ .$$

If the condition above is satisfied, we say that the rule (B, R, Q) is *triggered* by the local knowledge state B_i. From the communication algorithms given in Eqn. 4.16, only the current local knowledge state of sensor node 3 triggers a rule in its respective communication algorithm C_3. The rule that is triggered in C_3 is

$$\{ \ (\ \{(t,3,1)\}, \ \{5\}, \ \emptyset \ ) \ \} \ .$$

This rule encodes that sensor node 3 should send its local knowledge state to sensor node 5. Note that this decision is completely local to sensor node 3.

Then, sensor node 3 initiates a "push" type communication and sends its local knowledge $\{(t,3,1)\}$ to sensor node 5 as shown in Figure 4-18b. Upon receiving the data, sensor node 5 incorporates the data into its local knowledge state resulting in

$$B_5 := \{(t,5,2),(t,3,1)\} \ .$$

Again, all sensor nodes check if their local knowledge state satisfies the stopping condition. Since none do, they all check whether their local knowledge state triggers any communication rules. At this point, the local knowledge state B_5 triggers the following rule in the communication algorithm C_5 for sensor node 5,

$$\{ \ (\ \{(t,3,1),(t,5,2)\}, \ \emptyset, \ \{2\} \ ), \ \} \ .$$

This rule encodes that sensor node 5 should query for the local knowledge state of sensor node 2. Again, note that this decision is completely local to sensor node 5.

Sensor node 5 then initiates a "pull" type communication and sends a query message to sensor node 2. Sensor node 2 then responds with its local knowledge state which is

$$\{(t, 2, 1)\}$$

as shown by the dotted and black arrows in Figure 4-18c. Upon receiving the data, sensor node 5 incorporates the data into its local knowledge state resulting in

$$B_5 := \{(t, 5, 2), (t, 3, 1), (t, 2, 1)\} \ .$$

At this point, when sensor node 5 checks whether its local knowledge state satisfies the stopping condition, it will find that it does. Namely, the first line of Eqn. 4.18 returns one because the local knowledge state B_5 above corresponds to knowledge that the observed measurements are

$$z_5 = 2, \ z_3 = 1, \ \text{and} \ z_2 = 1 \ .$$

Sensor node 5 then computes the global task since it now believes that it has enough data to compute the global task. Although the other sensor nodes have not reached a stopping condition, all communications in the sensor network have effectively ceased since no communication rules will be triggered by the respective local knowledge state. This completes the first walkthru.

To demonstrate that different observed measurements result in different communications, we show in Figure 4-19 the communications taking place when the observed sensor measurements at some time t are

$$(z_1, \ldots, z_8) = (1, 0, 0, 2, 0, 0, 0, 1) \ .$$

In particular, concurrent communications occur for these observed measurements.

Note that encoding the global computation into a hierarchy, choosing an agent assignment, converting the hierarchy into a set of communication rules, and finally, combining the appropriate rules to generate a communication algorithm for each sensor node results in a distributed information collection algorithm where each sensor node makes local decisions to communicate information based on its own local knowledge state. The power of this methodology is that these distributed algorithms can be automated by choosing appropriate heuristics and cost criterions. The idea we have for the task of localization in sensor networks is for these steps to be automated to construct distributed communication algorithms (DCA's) for the particular spatial arrangement of sensor nodes deployed in an ad hoc fashion.

4.3.6 Limitations of the Approach

Let us discuss some limitations of using the hierarchy to construct communication algorithms.

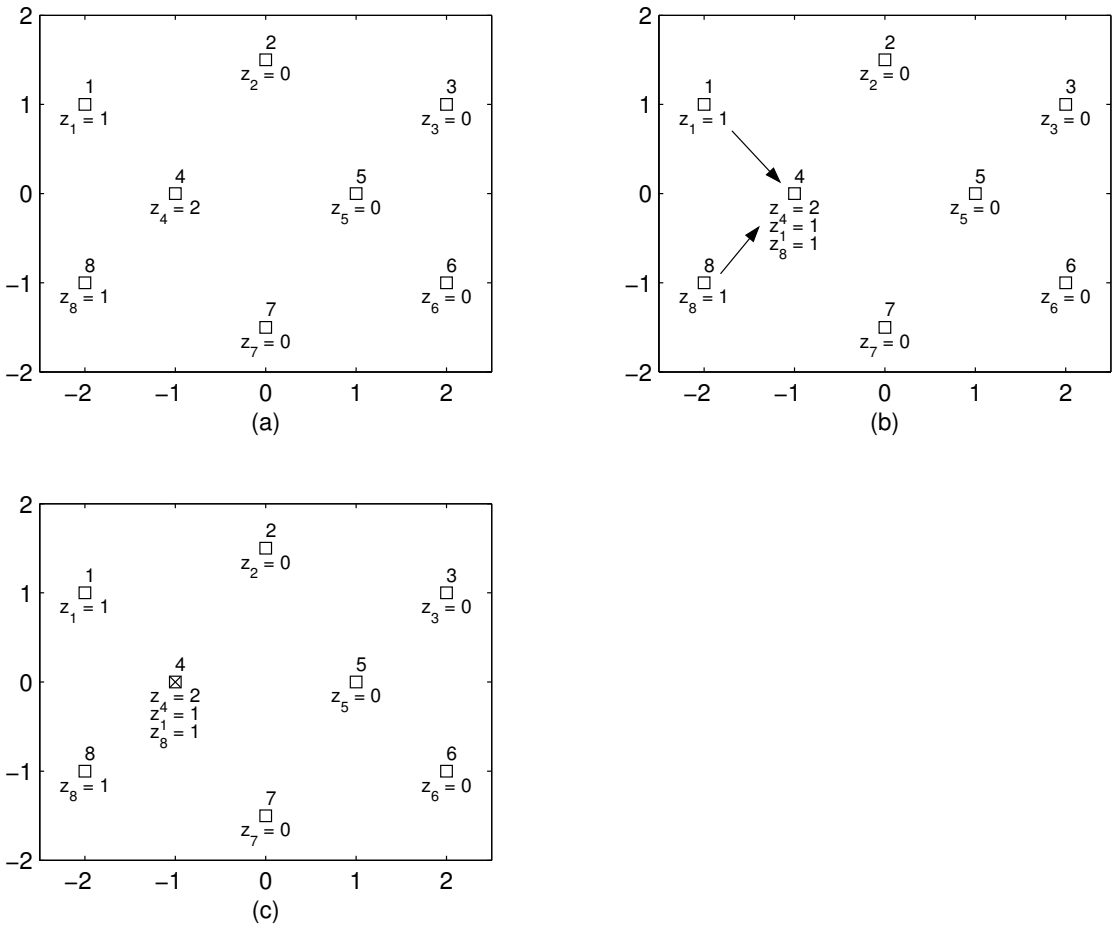

Figure 4-19: The known variable values of each node of the sensor network are shown underneath the respective node. (a) Before communications occur, each sensor node i only knows its own sensor measurement z_i. (b) Then, the communication algorithm in sensor node 1 sends z_1 to sensor node 4 and the communication algorithm in sensor node 8 sends z_8 to sensor node 4 as indicated by the black arrows. (c) Sensor node 4 has enough data to satisfy its stopping condition $STOP_4$ as indicated by the $\times$ in sensor node 4. Although there exist communication rules in C_4 which could be triggered by the local knowledge state of sensor node 4, the stopping condition takes precedence so that the communications encoded by these rules do not take place.

1. **Inference Task to Computations** The first step in determining the set of computations and associated triggers for the inference task is a time consuming batch computation. Computing the set of sensor measurements

$$\mathbf{S}_{[z]}$$

to collect for each equivalence class $\mathbf{Z}_{likely}$, which corresponds to the set of likely joint measurements, is a time consuming computation that requires knowledge of the sensor measurement models

$$p_i(z_i|x)$$

for all sensor nodes $i \in \{1, \ldots, n\}$ of the sensor network. This computation is certainly not scalable as the number of nodes in the sensor network grows. However, it is possible to address this issue of scalability by dividing the field into many local regions and constructing hierarchies for these local regions in a distributed manner. Section 4.6 addresses this issue.

2. **Enumeration** The algorithmic construction of the graph topology requires an enumeration step which enumerates nodes over pairs of existing nodes. For each of these enumerated nodes, we must test whether some trigger node can have this node as a child. If so, we must also calculate the heuristic for this node in order to rank it. Thus, for every pair of existing nodes, we must perform some computation. If there are m existing nodes over which we enumerate, then there are

$$\begin{pmatrix} m \\ 2 \end{pmatrix} = \frac{m(m-1)}{2}$$

pairs to consider. To be specific, consider the initial enumeration step over measurement tokens for a sensor network with n nodes and with the measurement space of each sensor discretized into k regions. Then, there are a total of nk measurement tokens, so that the number of pairs is

$$\frac{nk(nk-1)}{2} = \frac{n^2k^2 - nk}{2} .$$

For a sensor network with $n = 1000$ nodes with the measurement space of each sensor discretized into $k = 10$ regions, the number of pairs is about 10^8, so that there are 10^8 operations performed in this first iteration. With further iterations, the time it takes to construct the hierarchy can be substantial. Thus, using the hierarchy to construct communication algorithms must be performed off-line while the sensor network is not performing the inference task and has extra time. The idea in Section 4.6 also addresses this issue since a hierarchy is constructed only over nodes in a local region where the number of nodes in the local region is substantially smaller than the number of nodes in the entire sensor network.

3. **Dynamic Sensor Network** Another problem with using the hierarchy is that the set of computations and triggers to be encoded must be completely given before the process of constructing the hierarchy is begun. Thus, if a sensor node fails, then in order to update the communication algorithms to take this failure into account, we must recompute the set of computations and triggers taking into account the node failure and reconstruct a set of communication algorithms. Again, the idea in Section 4.6 can somewhat address this problem when node failures are confined to local regions since only the communication algorithms of those sensor nodes near the node that failed would need their communication algorithms updated. However, what would be a more desirable solution would be some way of encoding these failures into the original hierarchy so that sensor nodes can adjust to these dynamic changes of the sensor network. As the hierarchy currently stands now, dealing with a dynamic sensor network is not addressed.

4.3.7 Summary

We have presented the methodology of using the hierarchy to construct computationally efficient communication algorithms through a toy example of a sensor network. The first step is to convert the global inference task into a set of computations along with their triggers. It is the triggers which play the primary role in determining the graph topology of the hierarchy and the choice of agent assignment, uplinks, and downlinks. By discretizing the measurement space and defining the measurement tokens, all triggers can be expressed as a conjunction of measurement tokens. By converting everything into this Boolean domain, the partial processing for the evaluation of the triggers is simply a logical AND function. This simplifies the task of constructing a hierarchy immensely because the only choice left is how to implement multi-input logical AND functions in terms of two-input logical AND functions.

We presented a heuristical construction of the hierarchy which is essentially a greedy algorithm for constructing the graph topology of the hierarchy by using a heuristic measuring the worth of adding a node weighed by the computational cost of adding that node. This leads to a graph topology of the hierarchy which hopefully encodes an efficient communication algorithm pending a choice of agent assignment, uplinks, and downlinks. The choice of agent assignment, uplinks, and downlinks was then formulated as an optimization of the cost criterion of the problem.

Finally, once the hierarchy has been fully constructed, communication algorithms can be extracted in a straightforward manner. Furthermore, stopping conditions are also encoded in the hierarchy so that these should be extracted as well.

What has not been discussed is how partial processing of the computations are to be incorporated into the hierarchy. The only difference partial processing can make is in reducing the number of bits that needs to be transmitted by extracting the relevant information from the data and coding the output appropriately. Thus, taking

partial processing of the computations into account is merely a matter of adjusting the number of bits that will be transmitted. In particular, we mean adjusting the term $\mathbf{bits}_b$ for each node b of the hierarchy during the **Compute Heuristic and Rank** step of the graph topology construction algorithm in Section 4.3.3. Because the term $\mathbf{bits}_b$ may be different, the ranking of the enumerated nodes may be different so that the resulting graph topology is affected by this partial processing of the computations.

This completes the toy example of this section which has shown the basic methodology from inference task to the resulting communication algorithms on multiple distributed agents. In Section 4.4, we discuss possible heuristics to use for the algorithmic construction of the hierarchy. Note that the particular graph topology constructed is a function of the heuristics used. Thus, the success of using the hierarchy to construct efficient communication algorithms is limited by the ability to design good heuristics.

In Section 4.5, we present an alternative method for constructing hierarchies which chooses agent assignments, uplinks, downlinks, and trigger sets during the construction of the graph topology. The particular hierarchy constructed can be considered to be a generalization of the information-driven sensor querying (IDSQ) algorithm of [7]. The IDSQ algorithm is presented in the section as a particular communication algorithm encoded in a hierarchy which was constructed with a certain sequentiality heuristic and agent assignment constraint. On the other hand, the heuristic used for constructing the hierarchy corresponding to the IDSQ algorithm is a more sophisticated one than the commonality heuristic used in the toy example of this section. We leave a detailed presentation of the IDSQ algorithm for Appendix A.

Then, Section 4.6 addresses the issue of scalability of the algorithmic construction of the hierarchy and provides a scalable solution which is applicable for constructing communication algorithms for very large sensor networks.

4.4 Heuristics

The point of using heuristics is to construct a graph topology which encodes an efficient information collection mechanism. Designing good heuristics is certainly problem dependent, and a large part of the success and limitations of using the hierarchy as a tool for constructing computationally efficient algorithms depends on the ability of the human user to design good heuristics. We will present some general heuristics in this section to serve as a general guideline for designing heuristics.

Recall that the heuristics are used to rank each node from the set of enumerated nodes for possible inclusion into the hierarchy. Thus, a heuristic is a function of the *particular communication* and *collected information* that the newly enumerated node represents. Let us elaborate the above sentence. Each node in the hierarchy represents a possible local knowledge state. For example, node b in the hierarchy of Figure 4-17 represents the local knowledge state $\{(t,1,1),(t,4,2)\}$ for some time t which is interpreted as the local knowledge state where node 1 has observed that

$z_1 = 1$ *and* node 4 has observed that $z_4 = 2$. The left child of b represents the local knowledge state $\{(t, 1, 1)\}$, and the right child of b represents the local knowledge state $\{(t, 4, 2)\}$. The relationship of the local knowledge state of the parent and its children is given by a union operator. That is, if B_p is the local knowledge state of the parent and B_{c_1} and B_{c_2} are the local knowledge states of its children, then

$$B_p = B_{c_1} \cup B_{c_2} \ .$$

There is a possibility that B_p is inconsistent meaning that there exists a triple $(t, i, w) \in B_p$ and a triple $(t, i, y) \in B_p$ such that $w \neq y$. The local knowledge state B_p makes no sense because the interpretation would be that at time t the sensor measurement of sensor i was observed to be both w and y. In Definition 2.1, we disallowed such inconsistent local knowledge states.

The *collected information* of a newly enumerated node p connected to the pair of nodes c_1 and c_2 is the information contained in the local knowledge state

$$B_p = B_{c_1} \cup B_{c_2}$$

assuming that B_p is consistent. The *particular communication* of the newly enumerated node p refers to the communication needed to send the information in B_{c_1} to the agent assigned to node c_2 and/or to send the information in B_{c_2} to the agent assigned to node c_1. The heuristics measure various aspects about the enumerated nodes which includes the collected information and particular communication that the node represents.

The heuristics can be used in combination as was demonstrated in the toy example where we used a combination of the commonality and communication cost heuristics.

1. **Commonality** The idea of the commonality heuristic is that if the information collected by a newly enumerated node is one which is needed by many triggers, then this information should probably be collected. The nodes corresponding to a trigger T_i also represents a local knowledge state B_i for $i \in \{1, \ldots, M\}$ where M is the number of triggers. Then, the commonality heuristic of an enumerated node p representing local knowledge state B_p is given by

$$\textbf{common}(p) = |\{i \in \{1, \ldots, M\} : B_i \supset B_p\}| \tag{4.19}$$

   where $|\cdot|$ indicates the cardinality of the set.

2. **Layer (Sequential to Parallel)** There is a partial order on the nodes of a hierarchy. And thus, it is possible to define a layer number for each node of the hierarchy. In particular, we consider the layer number of all input nodes to be layer 0. For all other nodes, the layer number is the maximum number of edges downwards that must be traversed until we are at an input node.[10]

[10] For a formal definition, see Section 5.4.2

The layer number of a newly enumerated node can be a heuristic which controls how sequential or parallel the resulting communication algorithms will be. At the extreme of desiring only a completely sequential computation, we want the hierarchy to be tall, meaning that there are few nodes with the same layer number and the maximum layer number is large. At the extreme of desiring a completely parallel computation, we want the number of nodes in each layer to be as large as possible and the maximum layer number to be small. Thus, to design a sequential algorithm, we could penalize those nodes with low layer number and give lots of worth to those nodes with higher layer number. To design a parallel algorithm, we would do the opposite.

The effect of penalizing a low layer number and giving lots of worth for a high layer number results in an information collection mechanism which hands off information one at a time like the example of Figure 4-3. On the other hand, the effect of penalizing a high layer number and giving lots of worth to a low layer number results in an information collection mechanism where all nodes broadcast their data to their neighbors like the example of Figure 4-2. If there are real-time constraints, we would favor such a broadcast type of collection mechanism to reduce latency due to handoffs.

3. **Information Content** We can also use heuristics which measure some notion of information content. Intuitively, information content is how much information about the set of computations is contained in the information collected by the enumerated node. If we had an information measure

$$\mathcal{I} : \mathcal{B} \to \mathbb{R} \ ,$$

then for the enumerated node p with local knowledge state B_p, we wish to compute

$$\mathcal{I}(B_p) \ .$$

We will show in Section 4.5 that the IDSQ algorithm uses a measure of information content to determine what data to collect at the next iteration. Furthermore, Geman's 20 questions game is also an instance of a particular hierarchy construction with an information theoretic heuristic, namely the entropy, which measures the information gain from asking a certain question. The idea is that if we are limited in the number of questions we can ask, we should ask the questions with the greatest potential for information gain. We note that the general idea behind the IDSQ algorithm and Geman's 20 questions game is the principle of information-based computation.

4. **Communication Cost** This heuristic directly measures the amount of cost the communication represented by a node will incur as was demonstrated in the toy example of Section 4.3. This is certainly a quantity we would like to keep low. For an enumerated node p with children c_1 and c_2, the communication cost is

a function of the number of bits needed to encode the information in the local knowledge state B_{c_1} (or B_{c_2}) to send to the agent assigned to c_2 (or c_1).

5. **Processing Cost** If we incorporate partial processing of the computations in nodes, then the amount of time it takes to perform the partial processing may be a factor if there are real-time constraints. The partial processing required for an enumerated node p is the extraction of the information in the local knowledge state B_p. Thus, choosing nodes which require a lot of time for partial processing early may be desirable if there is no way to avoid these partial processing computations. On the other hand, if certain partial processing can be avoided entirely, then we can avoid choosing the associated node.

The above list of heuristics is not intended to be complete but only to serve as a general guideline for designing algorithms with some desired computational characteristics. The design of good heuristics for heuristical hierarchy construction is application specific and is critical for the success of using the hierarchy as a tool for constructing computationally efficient algorithms.

4.5 A Variation on Hierarchy Construction

We will consider a variation on the hierarchy construction which generalizes the particular hierarchy construction related to the IDSQ algorithm. Let us first present the IDSQ algorithm as a communication algorithm extracted from a particular constrained construction of a hierarchy in Section 4.5.1. Then, we will present a generalization of the IDSQ criterion and an alternative construction of the hierarchy in Section 4.5.2. We leave the detailed presentation of the IDSQ algorithm for Appendix A.

4.5.1 Hierarchy Representation of the IDSQ algorithm

The IDSQ algorithm is an active querying technique which incrementally decides what data to collect given the estimate computed from all previously collected data. To begin the IDSQ algorithm, some node of the sensor network is chosen to be the leader, denoted by $l \in \{1, \ldots, n\}$. The leader is assumed to know the measurement models of all sensor nodes in its neighborhood. That is, if the neighbors of the leader are $\mathcal{N}_l \subset \{1, \ldots, n\}$, then we assume that the leader knows the likelihood functions of its neighbors as well as its own likelihood function

$$\{p_i(z_i|x)\}_{i \in \mathcal{N}_l \cup \{l\}} \ .$$

As usual, we assume that the randomness in each sensor is independent of each other so that sensor fusion is, theoretically, a simple matter of multiplying likelihood

functions. For a given local knowledge state $B \in \mathcal{B}$, the posterior density of the target location at time t given the sensor measurements in B is given by

$$\tilde{p}_B(x) = p_0(x) \prod_{(t,i,z_i)\in B} p_i(z_i|x)$$

where $p_0(x)$ is the prior density of the target location assumed to be known by the leader.

The key idea in the IDSQ algorithm is to choose an information measure

$$\mathcal{I} : \mathcal{P}_d(\mathcal{X}) \to \mathbb{R}$$

where $\mathcal{P}_d(\mathcal{X})$ represents the set of probability densities on the space of target locations $\mathcal{X}$ and $\mathcal{X}$ is specifically some multi-dimensional Euclidean space. The information measure

$$\mathcal{I}(\tilde{p}_B)$$

returns a real number which measures the information content of the current posterior density $\tilde{p}_B$ of the target location given the local knowledge state B. One measure of the intuitive notion of information content could be based on some measure of the spread of the density. The more spread out the density is, then the less information content we assign to the density. For example, a measure of the spread could be the determinant of the covariance or the entropy. Appendix B will elaborate on some good candidates for an information measure and discuss the various limitations of them. Once we have chosen an appropriate information measure, we can run the IDSQ algorithm.

Recall the definition of $\mathbf{sensors}_t$ in Eqn. 2.9 which returns for any local knowledge state B, the subset of sensors $\{1, \ldots, n\}$ for which observations are known at time t.

Outline of IDSQ Algorithm

Let B be the current local knowledge state of the leader l. The leader first observes its own measurement $z_l \in \mathcal{Z}_l$, so that initially,

$$B = \{(t, l, z_l)\} \ .$$

Then, the leader can compute the posterior density with its own measurement

$$\tilde{p}_B(x) \ .$$

Initially,

$$\mathbf{sensors}_t(B) = \{l\} \ .$$

At this point, the leader must choose to query for the sensor measurement of some remote sensor node in $\{1, \ldots, n\} - \mathbf{sensors}_t(B)$ in an *intelligent* manner. the IDSQ algorithm chooses the next sensor $\hat{j}$ to query by maximizing the *expected information content* given by

$$\hat{j} = \arg_{j\in\{1,\ldots,n\}-U} \max E[\mathcal{I}(\tilde{P}(B, \{(t, j, Z_j)\})] \tag{4.20}$$

where

$$\tilde{P}(B_1, B_2) = \tilde{p}_{B_1 \cup B_2}$$

and Z_j is a random variable on $\mathcal{Z}_j$. The leader then queries sensor node $\hat{j}$, receives the sensor measurement $z_{\hat{j}}$, updates its local knowledge state

$$B' = B \cup \{(t, \hat{j}, z_{\hat{j}})\} \ .$$

Then, the leader updates its estimate of the target location and computes the updated posterior density

$$\tilde{p}_{B'} \ .$$

Then, with this new local knowledge state B', the leader chooses the next sensor node to query according to the criterion in Eqn. 4.20 again, and this process iterates until some stopping condition that an "estimate of reasonable quality" has been reached is satisfied.

To explain the IDSQ criterion (Eqn. 4.20) above, $\tilde{P}(B_1, B_2)$ is rewritten notation for the posterior density given all sensor measurements in both local knowledge states $B_1 \cup B_2$, assuming that $B_1 \cup B_2$ is consistent. The reason we use different notation is because the leader does not know the observation z_j of sensor node j until it has queried sensor node j. Thus, we must somehow marginalize out the unknown value of z_j. Although the leader does not know the value of z_j, the leader does know the distribution of sensor j's measurement which is represented by the random variable Z_j. Thus, we will compute the expected information content of the measurement from sensor node j, which is exactly the expression

$$E[\mathcal{I}(\tilde{P}(B, \{(t, j, Z_j)\}))] \ .$$

The distribution of Z_j used to compute the above expectation is the posterior distribution of sensor j's measurement given the sensor measurements of the local knowledge state B. Thus, the distribution of Z_j is a function of the local knowledge state B. This imples that the expected information content is a function of the local knowledge state B only, which is what we want because B is all the information the leader currently knows. Finally, the sensor node $\hat{j}$ is the sensor node with the maximum expected information content.

Let us now present a hierarchy construction algorithm which will result in a hierarchy encoding the IDSQ algorithm. We can immediately note a few constraints of the hierarchy construction imposed by the IDSQ algorithm.

1. All nodes except the input nodes of the hierarchy are assigned to the leader node.

2. All edges corresponding to real communications are downlinks.

Any hierarchy is constructed on top of a set of input nodes which are the nodes associated with the set of measurement tokens. For the IDSQ algorithm, the set of

measurement tokens $\{a_k^v\}_{v \in \mathcal{Z}_k, k \in \{1,\ldots,n\}}$ is a possibly uncountably infinite set since $\mathcal{Z}_k$ is usually a Euclidean space. Each measurement token a_k^v is equal to 1 if $z_k = v$ and 0 otherwise, where $z \in \mathcal{Z}$ is the observed joint measurement.

The IDSQ algorithm dictates the construction of the hierarchy which we will demonstrate below in regards to the three usual steps **Enumeration**, **Compute Heuristic and Rank**, and **Threshold and Update**.

1. **Enumeration** Initially, we begin with the measurement tokens associated with the leader l,
$$\{a_l^v\}_{v \in \mathcal{Z}_l} \ .$$
The set of enumerated nodes are over pairs of nodes with one of the pair being a measurement token of the leader l,
$$E = \{(a_l^{v_1}, a_k^{v_2})\}$$
for all $k \in \{1, \ldots, n\} - \{l\}$, $v_1 \in \mathcal{Z}_l$, and $v_2 \in \mathcal{Z}_k$.

2. **Compute Heuristic and Rank** The IDSQ criterion (Eqn. 4.20) is viewed as a heuristic which will be used to rank nodes in the enumerated set. For each enumerated node $b \in E$ with children $a_l^{v_1}$ and $a_k^{v_2}$, the heuristic assigned to b by the IDSQ criterion is
$$IDSQ(b) = E[\mathcal{I}(\tilde{P}(B_{a_l^{v_1}}, \{(t, k, Z_k)\}))]$$
where $B_{a_l^{v_1}}$ is the local knowledge state corresponding to node $a_l^{v_1}$ and Z_k is a random variable on $\mathcal{Z}_k$ whose distribution is computed from $B_{a_l^{v_1}}$. Thus, $IDSQ(b)$ is function of the local knowledge state $B_{a_l^{v_1}}$ and the label of the remote sensor k. Thus, the IDSQ criterion is computing the same heuristic value for the following group of enumerated nodes
$$E_{l,v_1}^k = \{(a_l^{v_1}, a_k^{v_2})\}_{v_2 \in \mathcal{Z}_k} \subset E\}$$
for $k \in \{1, \ldots, n\} - \{l\}$ and $v_1 \in \mathcal{Z}_l$.

3. **Threshold and Update** The IDSQ algorithm keeps a group of nodes at each iteration. In particular, for each $a_l^{v_1}$ for $v_1 \in \mathcal{Z}_l$, the IDSQ algorithm chooses one group of nodes from the set of groups
$$\{E_{l,v_1}^k : k \in \{1, \ldots, n\} - \{l\}\}$$
whose elements have the maximum heuristic value. The connection with the IDSQ algorithm is that given that the local knowledge state of the leader is $B_{a_l^{v_1}}$, the IDSQ criterion results in $\hat{j} \in \{1, \ldots, n\} - \{l\}$ which is the sensor node that maximizes the expected information content. To choose to query from sensor node $\hat{j}$ means to add the set of nodes $E_{l,v_1}^{\hat{j}}$ to the hierarchy. Certainly for different $v_1 \in \mathcal{Z}_l$, the resulting $\hat{j}$ from the IDSQ criterion can be different.

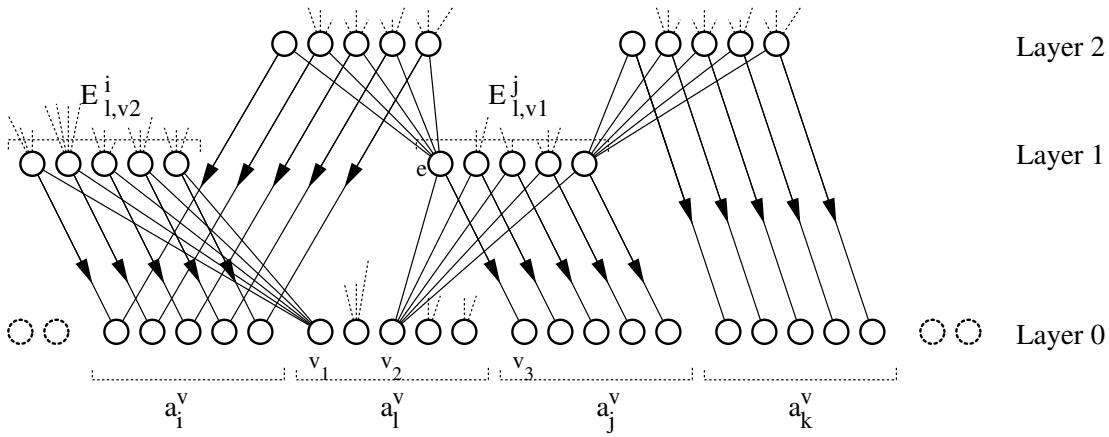

Figure 4-20: The hierarchy associated with the IDSQ algorithm. All nodes above the input layer are assigned to the leader node l and are associated with a two-input logical AND. Each node represents some local knowledge state, and downlinks represent a query from a remote node.

Figure 4-20 shows this first iteration of the hierarchy construction of the IDSQ algorithm. From each measurement token node of the leader a_l^v for any $v \in \mathcal{Z}_l$, we add the set of nodes $E_{l,v}^k$ if sensor node k maximizes the expected information content. For example, above the node labelled $a_l^{v_1}$, we have enumerated a set of nodes E_{l,v_1}^j since j is the sensor node with maximum expected information content. The other child of each node in E_{l,v_1}^j is a node corresponding to an element in the set of measurement tokens $\{a_j^v\}_{v \in \mathcal{Z}_j}$. Note that these measurement tokens correspond to the possible values that sensor node j could observe. All of these nodes are connected to a node in E_{l,v_1}^j by a downlink, and all nodes of E_{l,v_1}^j are assigned to the leader node l. The downlink indicates that sensor node l queries for the sensor measurement observed by sensor node j. The figure shows another example of the nodes added to the hierarchy over the node $a_l^{v_2}$.

After the first iteration, the possible local knowledge states of sensor node l is represented by the entire set of nodes in layer 1 of the hierarchy. For example, node e represents the local knowledge state

$$B_e = \{(t, l, v_1), (t, j, v_3)\} \ .$$

Continuing the iterations of the IDSQ algorithm is a matter of adding the set of nodes over each layer 1 node according to the IDSQ criterion. We do not wish to query from a sensor node which was already queried. For example, for node e in the hierarchy, the set of nodes to query from are now reduced to

$$\{1, \ldots, n\} - \mathbf{sensors}_t(B_e) \ .$$

Further iterations begin at the highest layer and add nodes to the next layer. If some stopping conditions is satisfied by the local knowledge state of a node, then no nodes are enumerated on top of this node.

This completes the relationship between the IDSQ algorithm and the particular hierarchy representation of it. Some of the limitations of the IDSQ algorithm is that the choice of the leader node is unspecified. Furthermore, communications may never occur concurrently. The next section will present a generalization of the IDSQ algorithm which can be used to construct communication algorithms where a leader node no longer needs to be specified and concurrent communications can occur.

4.5.2 Generalization of the IDSQ Criterion and an Alternative Hierarchy Construction

Let N be the set of nodes in the hierarchy currently, and let $\mathbf{d}(g)$ for any node $g \in N$ return the set of children of g.

The main idea we take away from the IDSQ algorithm is that at each iteration we add a set of nodes E to the hierarchy with the property that every node $e \in E$ has one child which is a particular node b and the other child is a node that is assigned to the some sensor node j. That is, for all $e \in E$, b is an element of $\mathbf{d}(e)$ and

$$\mathcal{A}(c) = j$$

where $\mathbf{d}(e) = \{b, c\}$ for some $j \in \{1, \ldots, n\}$. The function $\mathcal{A} : N \to \{1, \ldots, n\}$ is some agent assignment of the nodes N of the hierarchy. The set of other children of the nodes in E are denoted

$$F = \{c \in N : \{b, c\} \in \mathbf{d}(e) \text{ for all } e \in E\} \ .$$

We will choose an agent assignment while constructing the graph topology. The generalization of the IDSQ algorithm is that agent assignments are no longer constrained to the leader, edges that correspond to real communications no longer must be downlinks, and the heuristic can be any function of the node b and the set of nodes E.

Generalized IDSQ Criterion

In particular, let us elaborate on a particular heuristic which is a generalization of the IDSQ criterion (Eqn. 4.20). To recap, the IDSQ criterion uses a measure of the expected information content that the local knowledge state of the "leader" B combined with the unknown local knowledge state of sensor node j contains about the target location. The generalization is that the unknown local knowledge state of sensor node j no longer consists only of its own observation $\{(t, j, z_j)\}$ but can consist of other sensor observations due to the communications that may have already occurred.

Therefore, we wish to compute a heuristic for an enumerated set of nodes over a particular hierarchy node $b \in N$ and a subset of nodes $F_j^b \subset \mathcal{A}^{-1}(j)$ where $j \neq \mathcal{A}(b)$. The reason that F_j^b is not equal to $\mathcal{A}^{-1}(j)$ is due to possible inconsistencies of the local knowledge states. To be precise, for each node $a \in N$, denote the local knowledge state associated with a by B_a. Then,

$$F_j^b = \{a \in \mathcal{A}^{-1}(j) : B_a \cup B_b \in \mathcal{B}\}$$

where $\mathcal{B}$ is the set of local knowledge states which are consistent by definition (See Definition 2.1).

Let us now derive an expression for the expected information content in the combination of the local knowledge state B_b with the set of local knowledge states $\{B_a\}_{a \in F_j^b}$ corresponding to the set of unknown consistent local knowledge states of sensor node j. Assuming that we are in the middle of constructing a hierarchy, denote the current set of nodes in the hierarchy by N. For each node $a \in N$, denote by $[B_a]$ the subset of joint measurements $z \in \mathcal{Z}$ which are consistent with the local knowledge state B_a, which is given by

$$[B_a] = \{z \in \mathcal{Z} : z_i = v \text{ for all } (t, i, v) \in B_a\} \ .$$

Then, define the following function from joint measurements to local knowledge states

$$D_j^b : \mathcal{Z} \to \mathcal{B}$$

by

$$D_j^b(z) = \bigcup_{a \in F_j^b : z \in [B_a]} B_a \ .$$

The function D_j^b returns the local knowledge state of sensor j for each possible joint measurement observed by the sensor network. Now, we can write the generalized IDSQ criterion

$$\hat{j} = \arg_{j \in \{1,\dots,n\} - \mathbf{sensors}_t(B_b)} \max E[\mathcal{I}(\tilde{P}(B_b, D_j^b(Z)))] \tag{4.21}$$

where Z is a random variable on $\mathcal{Z}$ with distribution equal to the posterior distribution of a joint measurement given the local knowledge state B_b.

Alternative Hierarchy Construction

Let us now describe the alternative construction of the hierarchy. Again, the steps of the hierarchy construction are **Enumeration**, **Compute Heuristic and Rank**, and **Threshold and Update**. Let N denote the set of nodes currently in the hierarchy. Also, the agent assignment $\mathcal{A}$ has also been specified for the nodes N. Recall that we begin with a graph topology where the set of input nodes are measurement tokens and all other nodes correspond to trigger nodes.

2. **Enumeration** We enumerate groups of nodes for this variant of constructing a hierarchy. For each node $b \in N$ for which there is a trigger node T_i connected by an edge upwards from b, we enumerate a set of nodes E_b^j for each $j \in \{1, \ldots, n\} - \mathcal{A}(b)$ where E_b^j is the maximal set of nodes in N for which each node $e \in E_b^j$ satisfies
$$b \in \mathbf{d}(e)$$
and
$$c \in F_j^b$$
if $\mathbf{d}(e) = \{b, c\}$. Thus, we have constructed a collection of sets of nodes
$$\{E_b^j : j \in \{1, \ldots, n\} - \mathcal{A}(b), b \in N, (b, T_i) \text{ is an edge for some trigger } T_i\} .$$

3. **Compute Heuristic and Rank** For each set of nodes E_b^j, we can compute a heuristic measuring its worth weighed by the cost of the communication involved. We have a choice here of agent assignment and a choice of uplinks or downlinks. For each of the nodes in E_b^j, we choose all of these nodes to be assigned either to $\mathcal{A}(b)$ or j, and we choose all edges corresponding to a real communication to be uplinks or downlinks. The idea is that based on the local knowledge state B_b of sensor node $\mathcal{A}(b)$, this sensor node can either decide to send its information to sensor node j or query for the information in sensor node j. Thus, we have four possibilities for which we can compute the heuristic for each set of nodes E_b^j. We choose the best one according to the heuristic and rank all sets $\{E_b^j\}b, j$ accordingly. Note that the generalized IDSQ criterion of Eqn. 4.21 can be used here as part of the heuristic.

4. **Threshold and Update** We will choose some number of the best ranking sets of nodes E_b^j, add these nodes to the hierarchy, and update the edges emanating downwards from the trigger nodes. Note that in adding the nodes to the hierarchy we have already made agent assignments to the new nodes and have chosen edges that corresponding to real communications to be uplinks or downlinks.

In the simulations of Section 4.7, we will show that the hierarchies constructed in the above alternative manner as opposed to the construction in the toy example generally lead to better communication algorithms for data collection. The reason is that the alternative construction of the hierarchy is better matched to how real communications take place in the sensor network. That is, from a given local knowledge state of some sensor node, we either query for the unknown local knowledge state of a remote sensor or send this local knowledge state to the remote sensor. The exact information collected is unknown; however, the set of all possibilities of information collected after the communication takes place is enumerated by the above alternative construction of the hierarchy. In the hierarchy construction of the toy example, all possibilities of information collected from a single communication are not enumerated.

4.6 Distributed Construction of DCA's

One of the major problems with directly applying the technique presented for constructing DCA's with the hierarchy representation is that the construction of the hierarchy is a global computation which does not scale well with the number of sensor nodes in the sensor network. That is, some node of the sensor network must obtain knowledge of all nodes in the sensor network, construct some appropriate global hierarchy, and finally, generate the communication algorithms for every node. Hierarchy construction is not scalable because the enumeration step in the heuristical hierarchy construction algorithm takes time which grows with the square of the number of sensor nodes as was analyzed in Section 4.3.6. Thus, with thousands of sensor nodes in a sensor network, we soon reach the limits of using the hierarchy representation to generate DCA's in a practical application.

To control the poor scalability of the heuristical hierarchy construction algorithm, we will distribute the construction of the DCA's by constructing hierarchies over local subsets of target positions in the field $F \subset \mathcal{X}$. Because sensor nodes have a limited range where their measurement can provide significant information, the result of breaking up the field is that the number of sensor nodes involved in a local hierarchy is kept small. This means the time complexity of the enumeration step during hierarchy construction is kept under control. However, a possible problem with this idea is that the construction of local hierarchies completely oblivious of other local hierarchies will surely result in computational inefficiencies so that it will be important to construct local hierarchies which are mindful of what other local hierarchies are to be constructed. We will make explicit this notion of *cooperation* among local hierarchy constructions which will result in more efficient DCA's.

Let us formulate the distributed construction of DCA's. We have n sensor nodes of the sensor network labelled by an integer in $\{1, \ldots, n\}$. The set of target locations for the inference task is $\mathcal{X} = \mathcal{R}^2$. Each sensor node $i \in \{1, \ldots, n\}$ is responsible for generating communication algorithms for a subset $\mathcal{X}(i)$ of the field $F \subset \mathcal{X}$. Furthermore, it is desirable that the subsets are distinct,

$$\mathcal{X}(i) \cap \mathcal{X}(j) = \emptyset$$

for $i, j \in \{1, \ldots, n\}$ and $i \neq j$ and cover as much of F as possible. As an example of how these subsets $\{\mathcal{X}(i)\}_{i=1}^n$ are determined, each sensor node i is located at some position $x_i \in \mathcal{R}^2$. Then, we can ask each sensor node to be responsible for those positions in the field where it is closest sensor node which is

$$\tilde{\mathcal{X}}(i) = \{x \in F : d(x, x_i) \leq r\}$$

where $d(\cdot, \cdot)$ is the Euclidean distance and $r > 0$ is some chosen radius. The sets $\tilde{\mathcal{X}}(i)$ for $i \in \{1, \ldots, n\}$ are not necessarily distinct, so let us refine these sets.

After the configuration phase, each sensor node i is aware of a set of neighbors $\mathcal{N}(i) \subset \{1, \ldots, n\}$. This means sensor i knows the positions $\{x_j\}_{j \in \mathcal{N}(i)}$ of its neigh-

bors. As long as sensor node i knows all neighboring nodes within $2r$ units around itself, the subsets

$$\mathcal{X}(i) = \{x \in \tilde{\mathcal{X}}(i) : d(x, x_i) < d(x, x_j) \text{ for all } j \in \mathcal{N}(i)\}$$

for $i \in \{1, \ldots, n\}$ are distinct subsets.

Now, each sensor node i can construct a local hierarchy. The set of computations to be encoded in sensor node i's local hierarchy can be derived by discretizing the measurement spaces and sampling from sensor node i's local region $\mathcal{X}(i)$. Say that the resulting set of computations are

$$\{g_j^i\}_{j=1}^M$$

with associated triggers

$$\{T_j^i\}_{j=1}^M \ .$$

With an appropriate choice of heuristic, the hierarchy construction algorithm is invoked to encode the computations and triggers.

Let us discuss how we can incorporate *cooperation* between these local hierarchy constructions. Incorporating cooperation during the local hierarchy construction of sensor node i for the computations and triggers means that sensor node i also takes into account the computations and triggers to be encoded by its neighbors,

$$\{g_j^k\}_{j=1, k \in \mathcal{N}(i)}^M$$

and

$$\{T_j^k\}_{j=1, k \in \mathcal{N}(i)}^M \ .$$

The idea is that the set of enumerated nodes is to be constructed from the triggers $\{T_j^i\}_{j=1}^M$ of sensor node i only. On the other hand, the heuristics (like the commonality and information content heuristic) are to be computed based on all triggers

$$\{T_j^k\}_{j=1, k \in \mathcal{N}(i)}^M \ .$$

The heuristic for communication cost should be computed based on all computations

$$\{g_j^k\}_{j=1, k \in \mathcal{N}(i)}^M$$

as well so that a good estimate of the number of bits of information to be communicated can be computed. Since it is the heuristics which determine what the resulting graph topology of the hierarchy will be, it is in the heuristics that the effect from the computations and triggers of other local hierarchies should be incorporated.

Then, after the local hierarchy has been constructed, the communication algorithms (which we view as functions here) can be extracted. These communication algorithms are denoted

$$\{C_j^i\}_{j \in \{i\} \cup \mathcal{N}(i)}$$

and are the communication algorithms generated from sensor node i's local hierarchy. The communication algorithms for sensor nodes which are not i or a neighbor of i are the empty algorithms,

$$C_j^i(B) = \emptyset$$

for all $B \in \mathcal{B}$ for $j \in \{1, \ldots, n\} - (\{i\} \cup \mathcal{N}(i))$ where we remind the reader that all communication algorithms are a function

$$C : \mathcal{B} \to 2^{\{1,\ldots,n\}}$$

where $\mathcal{B}$ is the set of local knowledge states.

Every sensor node $i \in \{1, \ldots, n\}$ has generated a set of communication algorithms

$$\{C_j^i\}_{j=1}^n$$

so that what remains is to consolidate all of the communication algorithms for each sensor node j into a single communication algorithm. We consolidate the communication algorithms $\{C_j^i\}_{i=1}^n$ into a single communication algorithm C_j by

$$C_j(B) = \bigcup_{i=1}^n C_j^i(B)$$

for all $B \in \mathcal{B}$. Thus, we have described a method generating DCA's in a distributed fashion which addresses the issue of scalability of the algorithmic construction of the hierarchy.

As for the issue of faulty nodes, if we keep the different communication algorithms $\{C_j^i\}_{i,j\in\{1,\ldots,n\}}$ which were generated by the different local hierarchies, then if there is a single faulty node, say k, only the neighbors $\mathcal{N}(k)$ of k must update their local hierarchies. Thus, only the communication algorithms

$$\{C_j^i\}_{j\in\{1,\ldots,n\}}^{i\in\mathcal{N}(k)}$$

must be updated. This can be a significant factor in reducing the amount of computation occurring if the sensor network is very large and the number neighbors for each sensor node is small.

4.7 Simulations

In these simulations, we consider the DCA's constructed for two sensor networks. A small sensor network consisting of 10 sensor nodes placed randomly with a uniform distribution over the square area

$$\{(x, y) \in \mathbb{R}^2 : -50 \leq x \leq 50,\ -50 \leq y \leq 50\}$$

as shown in Figure 4-21. The field of this sensor network was chosen to be

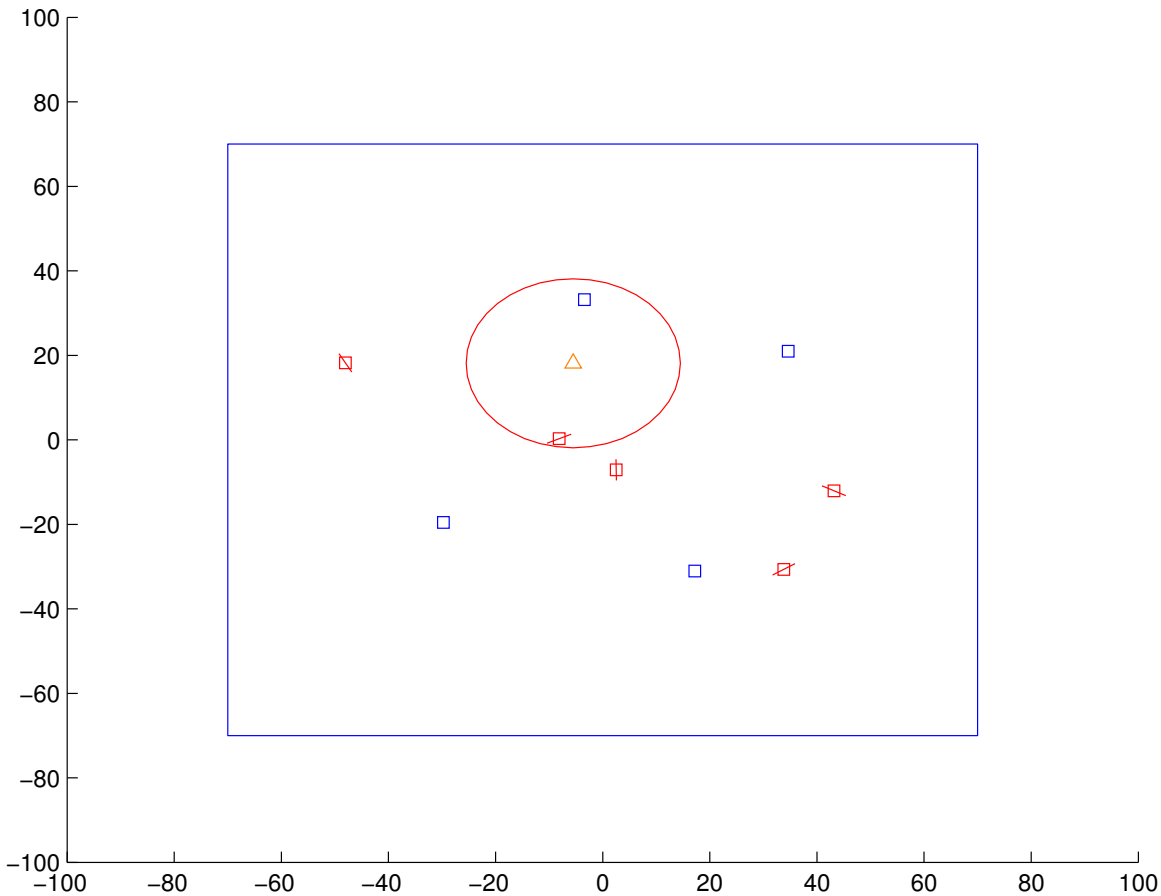

Figure 4-21: A small sensor network consisting 10 sensor nodes placed randomly within a square area of side length 100 centered at the origin. The outline box indicates the chosen field of this sensor network.

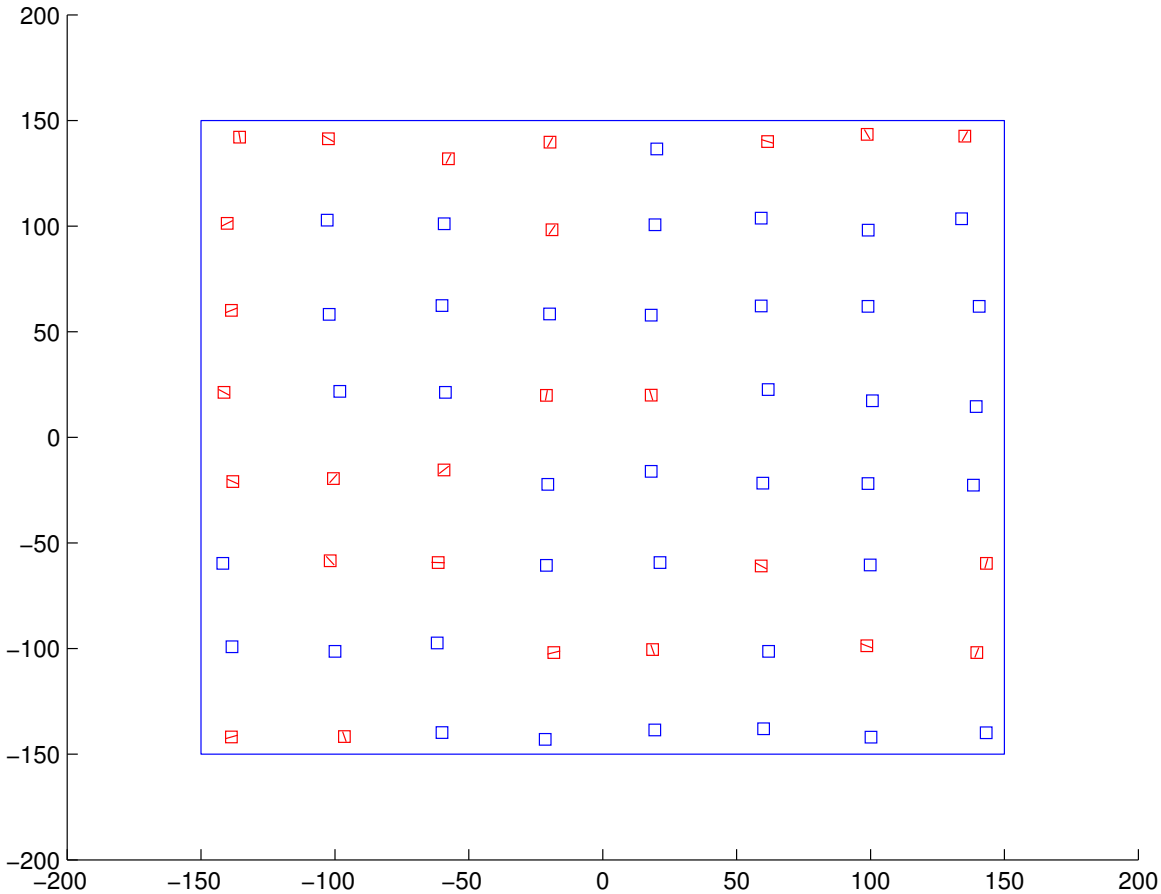

Figure 4-22: A large sensor network consisting of 64 sensor nodes placed approximately on a square grid. The outline box indicates the chosen field of this sensor network.

$$F_{small} = \{(x, y) \in \mathbb{R}^2 : -75 \leq x \leq 75,\ -75 \leq y \leq 75\} .$$

The other sensor network consists of 64 sensor nodes arranged approximately in a square grid, where the locations were generated by adding some noise to the positions of each node from the nominal location on the grid. Figure 4-22 shows this sensor network where each sensor node was chosen to be an intensity or TDOA sensor at random. The field of this sensor network was chosen to be

$$F_{big} = \{(x, y) \in \mathbb{R}^2 : -150 \leq x \leq 150,\ -150 \leq y \leq 150\} .$$

To generate the set of computations to encode into a hierarchy, we chose a set of samples of target locations in a square grid with spacing $d = 2.8$. The set of sample points corresponding to the discretization of the parameter space of target locations

is given by

$$\begin{aligned}\mathcal{F}_{small} &= F_{small} \cap G_{2.8} \\ \mathcal{F}_{big} &= F_{big} \cap G_{2.8}\end{aligned}$$

where G_d is a square grid of points as given in Eqn. 4.12. We will assume a uniform distribution over the discrete target locations, $\mathcal{F}_{small}$ and $\mathcal{F}_{big}$. Then, because we have a likelihood $p_i(z_i|x)$ of the measurements given the target location, the distribution of the measurement space for each sensor node i is given by

$$P_i(z_i) = \frac{1}{|\mathcal{F}|} \sum_{x \in \mathcal{F}} p_i(z_i|x)$$

where $\mathcal{F}$ is $\mathcal{F}_{small}$ or $\mathcal{F}_{big}$. The measurement space can then be quantized with this distribution P_i of each sensor node i. The set of equivalence classes constituting the likely measurements in $\mathbf{Z}_{likely}$ will be chosen to be those equivalence classes which contain the nominal joint measurement values observed by the sensor network when the target is at each sample location $\mathcal{F}$, just as in the toy example of Section 4.3.

At this point, the rest of the steps of constructing the graph topology of the hierarchy, choosing an agent assignment, uplinks, and downlinks, and generating DCA's follow exactly as in the toy example. We will choose an agent assignment, uplinks, and downlinks according to the energy consumed by transmission and reception of bits over a certain distance as demonstrated in the toy example in Section 4.3.4.

What we will vary in the construction will be the particular heuristic used to construct the graph topology of the hierarchy to demonstrate the different possible DCA's that can be generated using the hierarchy as a meta-representation. We will use the variation on the hierarchy construction as presented in Section 4.5.2. Recall, that in the alternative construction, a heuristic is computed over sets of nodes E. The heuristic used will be a linear combination of three specific heuristics.

1. **Information Content** In the alternative construction, a set of nodes E is enumerated over every node b currently in the hierarchy where the other child of the nodes in E are all assigned to the same agent. The information content heuristic is a normalized commonality heuristic given by

   $$\mathbf{info}(E) = \frac{\mathbf{common}(E)}{\mathbf{common}(b)}$$

   where $\mathbf{common}(\cdot)$ was defined in Eqn. 4.19 and we have extended the definition over sets of nodes by

   $$\mathbf{common}(E) = |\{i \in \{1, \ldots, M\} : B_i \supset B_p \text{ for some } p \in E\}| \ .$$ [11]

[11] See Eqn. 4.19 for the notation used here.

This choice of measuring the information content is essentially a measure of the number of triggers that need the information which results from the communications represented by the nodes E normalized by the number of triggers which considered b as needed information. If the information is equal to 1, this means that all triggers which needed the information in b also need the resulting information from the communications represented by E. A value of 0 indicates that the communications E are unneeded by any trigger.

2. **Communication Cost** The communication cost is given by the cost of energy expended by the sensor network to transmit and receive the k bits communicated over a certain distance d. We denote this heuristic by $\mathbf{cost}(E)$.

3. **Layer** We also use the layer heuristic to control how parallel or sequential the resulting DCA's are, which we denote by $\mathbf{layer}(E)$.

We will specify the linear combination of the heuristics used for generating a specific set of DCA's by a triple

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} \in \mathbb{R}^3$$

so that the resulting combined heuristic is

$$\alpha_1 \mathbf{info}(\cdot) - \alpha_2 \mathbf{cost}(\cdot) + \alpha_3 \mathbf{layer}(\cdot)$$

which we maximize to rank the set of enumerated nodes during graph topology construction.

We will also vary the number of quantization levels of the measurement space of each sensor node, which we denote by a positive integer q. Furthermore, we will construct DCA's according to the distributed construction method described in Section 4.6. Thus, we have a choice of whether to allow local hierarchies to cooperate, which we will denote by $c \in \{0, 1\}$ where $c = 1$ means to allow cooperation and $c = 0$ means to disallow cooperation.

Thus, by choosing the triple (α, q, c), we can construct many different DCA's. In particular, we generated DCA's for the large and small sensor networks with the

following set of choices of the triple (α, q, c)

Sim#	α	q	c
DCA1	$(1, 0, 0)$	10	1
DCA2	$(.75, .25, 0)$	10	1
DCA3	$(.5, .5, 0)$	10	1
DCA4	$(.25, .75, 0)$	10	1
DCA5	$(.5, .5, 10000)$	10	1
DCA6	$(.5, .5, -10000)$	10	1
DCA7	$(1, 0, 0)$	10	0
DCA8	$(.5, .5, 0)$	10	0
DCA9	$(.5, .5, 0)$	5	1
DCA10	$(.5, .5, 0)$	20	1

The first four DCA's explore the difference in trading off information for cost. DCA5 and DCA6 consider the effect of constructing for a maximally sequential or a maximally distributed DCA's respectively. DCA7 and DCA8 contrast how using cooperation can make the DCA more efficient. In particular, DCA7 is to be compared with DCA1, and DCA8 is to be compared with DCA3. Finally, DCA9 and DCA10 consider the effect of different quantization levels.

The statistics we can gather from these DCA's are

1. (AVE) the average energy drain by the sensor network as a whole,

2. (MD) the average maximal distance between nodes that send messages and the ground truth of the target location,

3. (P_{miss}) the probability of missed detection, and

4. (HANDOFF) the number of handoffs before a sensor node declares that if has computed the global localization task.

4.7.0.1 Small Sensor Network

For the small sensor network consisting of 10 nodes, we collected the statistics by running the DCA's over 300 runs by choosing a target location uniformly from the field F_{small} and simulating measurements at each sensor node according to the nominal choice of parameters for the sensor measurement models as described in Section 3.7.1.

The average statistics we have collected are given in the following.

Sim#	AVE($\times 10^6$)	MD	P_{miss}	HANDOFF
1	1.0201	198.6	.38	2.24
2	1.6493	197.6	.49	1.92
3	1.5931	202.2	.42	1.79
4	1.6664	205.7	.38	1.77
5	1.0006	197.4	.40	2.46
6	2.4513	200.4	.38	1.63
7	1.0497	201.1	.35	2.34
8	2.3584	207.7	.36	1.64
9	2.5308	205.9	.15	1.82
10	1.0585	198.8	.69	1.68

The probability of a missed detection is in the range .35 to .49 when the number of quantization levels is equal to 10 as in simulations DCA1–DCA8. The effect of more quantization, as in DCA10, is to increase the probability of a missed detection, P_{miss} whereas the effect of less quantization is to reduce P_{miss}. However, this comes at the cost of over double the average energy drain on the sensor network as seen between DCA9 and DCA10. The reason P_{miss} is so high for DCA10 is because the set of likely sensor measurements $\mathbf{Z}_{likely}$ in our simulations includes only the equivalence classes that contain the nominal joint measurement values observed the sensor network for each sample target position in $\mathcal{F}_{small}$. Since with more quantization levels the sizes of the equivalence classes are smaller, $\mathbf{Z}_{likely}$ covers less of the true set of likely joint measurements $\mathcal{Z}_{likely}$, which therefore results in larger P_{miss}.

The effect of choosing a high scaling factor for the layer heuristic results in more handoffs as exhibited in DCA5 as compared with DCA6. However, there is a tradeoff of more handoffs for less energy drain as seen between DCA5 and DCA6.

As for incorporating cooperation between the construction of local hierarchies, we see that the average energy drain in DCA1 is smaller than in DCA7. The difference in average energy drain between DCA3 and DCA8 shows the possibly significant effect that cooperation can have when constructing local hierarchies.

In Figure 4-23, we show the individual plots of AVE, MD, and HANDOFF for each DCA. The gray dots indicate the particular values of AVE, MD, and HANDOFF for each of the 100 runs, and black connected circles indicate the mean. Note that with this small sensor network, MD is about the same for all simulations. Simulations 1, 5, 7, and 10 seem to have the best characteristics with respect to average energy drain.

We desire DCA's with both low average energy drain and a low number of handoffs. Figure 4-24 shows the the number of handoffs versus average energy drain for each of the 10 DCA's. Clearly, DCA10 seems to be the best. However, DCA10 suffers from a high $P_m iss$. Thus, depending on how many handoffs and how much probability of

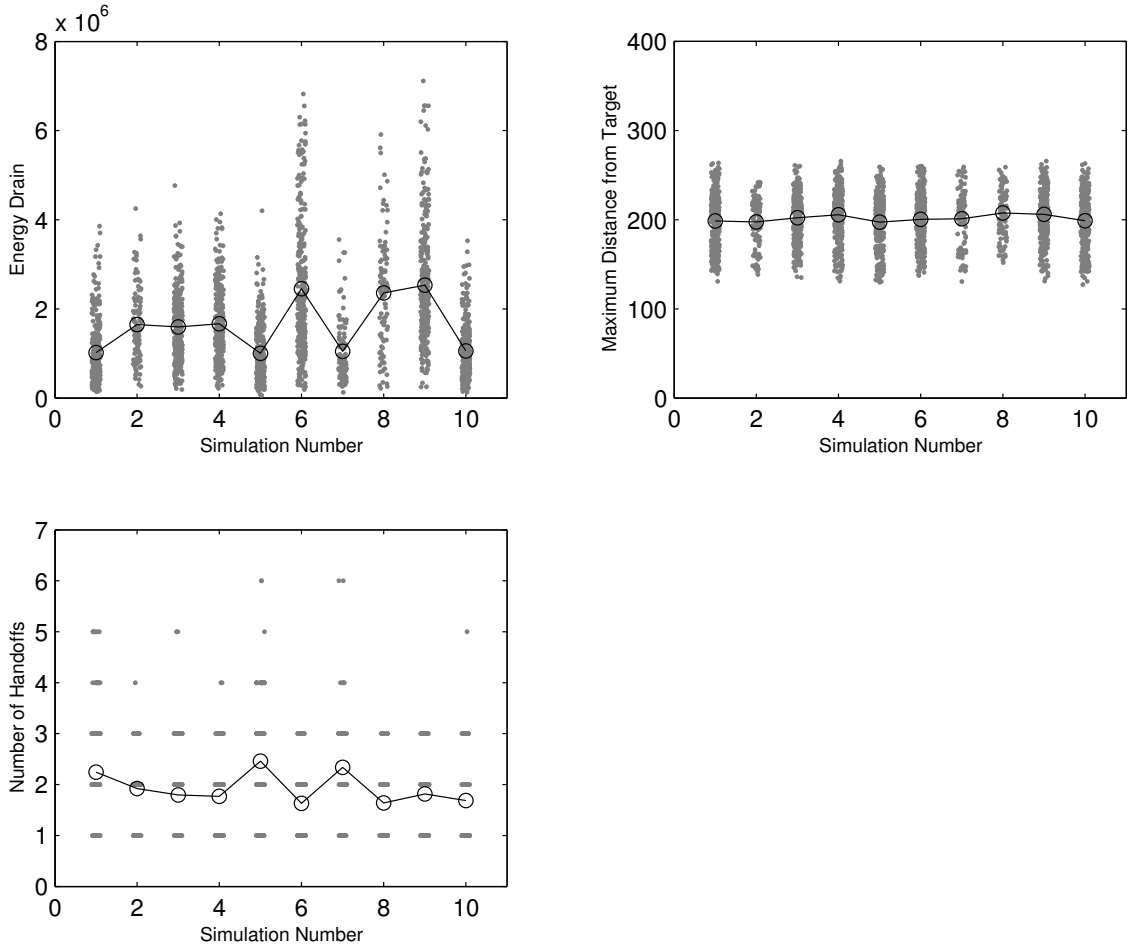

Figure 4-23: Individual plots of energy drain (AVE), maximal distance (MD) between target and a sensor node which transmits, and the number of handoffs (HANDOFF) before a sensor node declares that the global localization task has been computed.

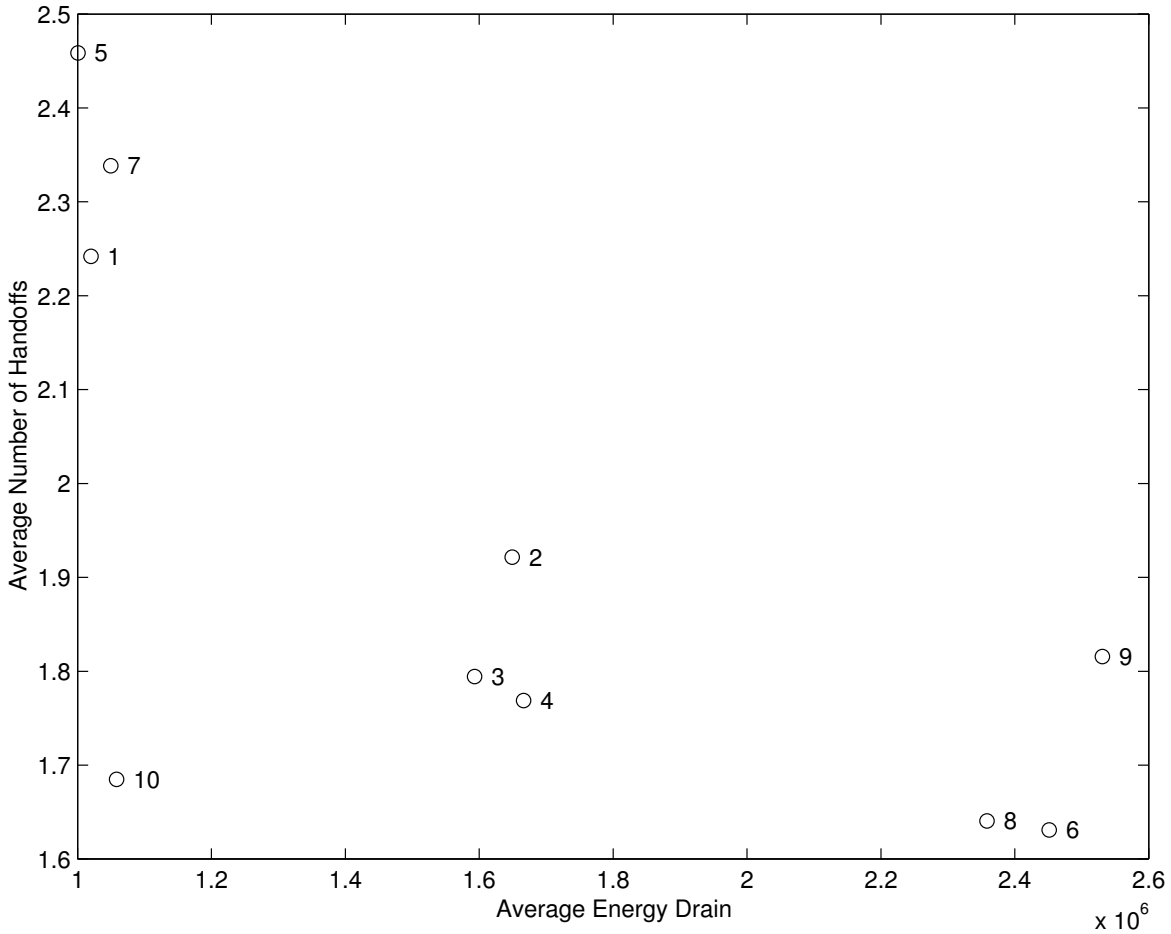

Figure 4-24: Plots comparing number of handoffs before a sensor node declares that the global localization task has been computed and the average energy drain.

missed detection we can tolerate, we can choose the DCA with the smallest average energy drain.

4.7.0.2 Large Sensor Network

For the large sensor network consisting of 64 nodes, we collected the statistics by running the DCA's over 100 runs by choosing a target location uniformly from the field F_{large} and simulating measurements at each sensor node according to the nominal choice of parameters for the sensor measurement models as described in Section 3.7.1. The average statistics we have collected are given in the following.

Sim#	AVE($\times 10^6$)	MD	P_{miss}	HANDOFF
1	.3188	72.3	.46	2.46
2	.7134	93.1	.41	1.93
3	1.0485	97.5	.44	2.05
4	1.0603	95.6	.42	2.12
5	.3792	73.8	.46	2.91
6	1.5760	115.5	.40	2.10
7	.8936	109.4	.48	2.44
8	.9905	92.1	.45	1.93
9	1.6630	201.7	.03	2.32
10	.7750	113.1	.91	2.00

As discussed in the previous section on the small sensor network, P_{miss} depends primarily on the number of quantization levels. DCA9 and DCA10 clearly show a large difference in probability of missed detection for a quantization level of 5 and 20 respectively.

Using a large scaling factor for the layer heuristic, the number of handoffs in DCA5 is only slightly larger than DCA6. However, the effect on the average energy drain is quite significant. Furthermore, notice that DCA6, which was intended to exhibit the least number of handoffs, does not have the smallest mean in the number of handoffs. This shows that designing desired behavior into the DCA's through the heuristics is very difficult, and thus, a limitation of using the hierarchy to construct distributed algorithms.

To compare the effect of incorporating cooperation when constructing local hierarchies, we clearly see less average energy expended in DCA1 than DCA7. However, the average energy expended in DCA8 is smaller than that in DCA3. This shows that choosing heuristics and varying the construction of the hierarchy in order to design some desired global behavior is very difficult and can lead to counterintuitive results.

In Figure 4-25, we show the individual plots of AVE, MD, and HANDOFF for each simulation. The minimal average energy drain is exhibited by DCA1 and DCA5. In this large sensor network, it is desirable that sensor nodes only communicate when there is a target near them. The statistic MD shows this. The sensor network spans

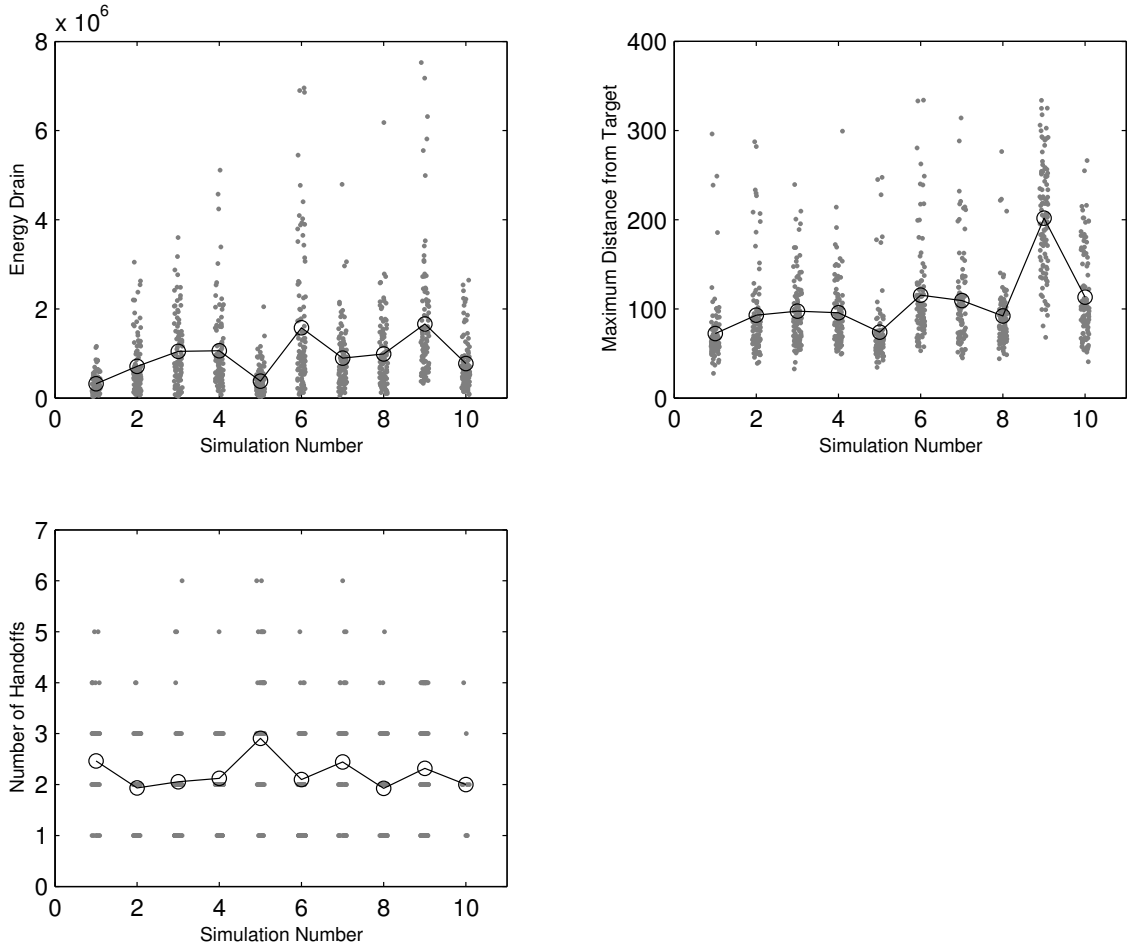

Figure 4-25: Individual plots of energy drain (AVE), maximal distance (MD) between the target and a sensor node which transmits, and the number of handoffs (HANDOFF) before a sensor node declares that the global localization task has been computed.

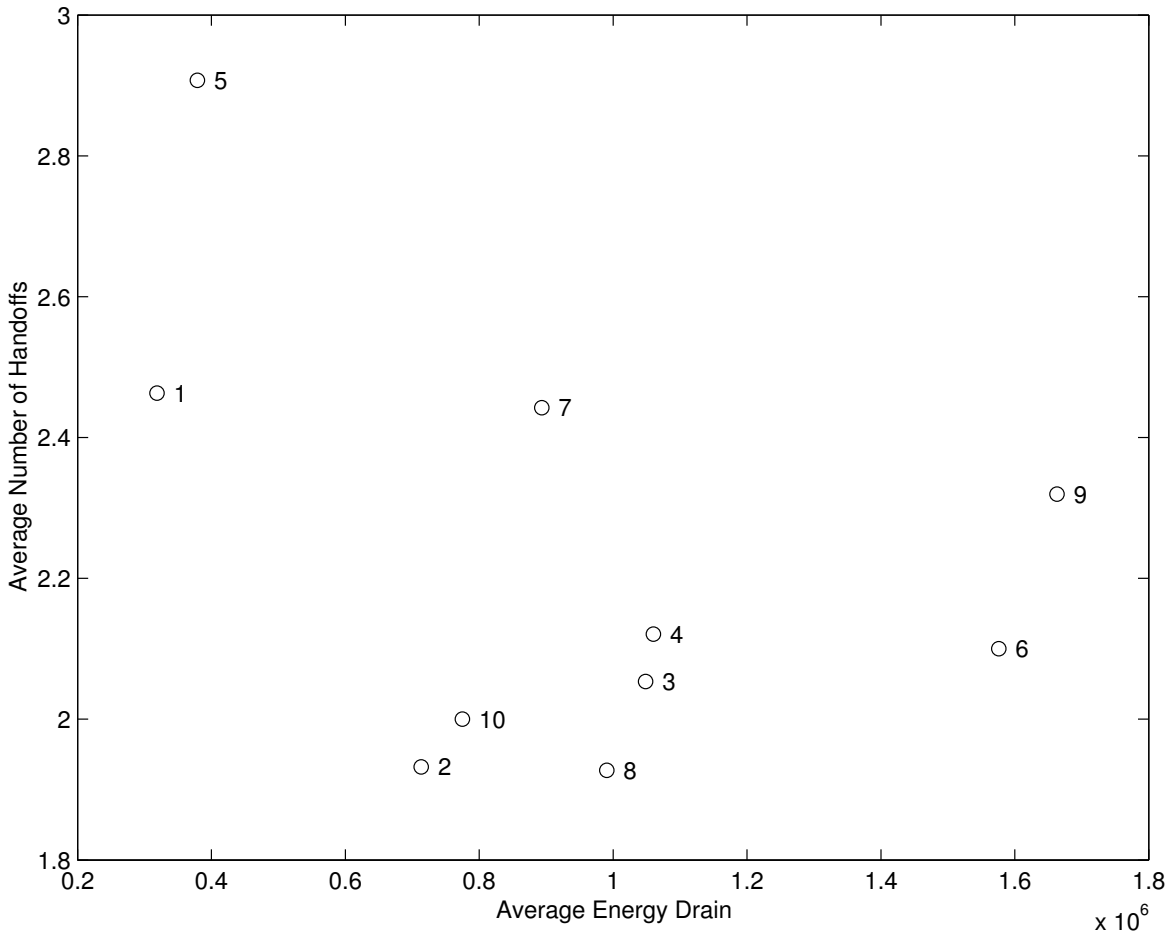

Figure 4-26: Plots comparing number of handoffs before a sensor node declares that the global localization task has been computed and the average energy drain.

a 300×300 unit area. Note that the mean of the sensor nodes which transmit are much less than 300 which shows that communications only occur when a target is near the sensor node.

Figure 4-26 shows the tradeoff between average energy drain and number of handoffs for the 10 DCA's constructed. The main tradeoff is that less average energy drain comes at the expense of more handoffs as shown by DCA1 and DCA5. This matches our intuition because when there are more handoffs, each sensor node has more of a chance to make local decisions incrementally. When there are few handoffs, each sensor node must make local decisions with less information which can lead to inefficiencies.

Chapter 5

Hierarchy Representation

In this chapter, we present the hierarchy in full generality which is a computation with communication model serving as a meta-representation from which efficient distributed algorithms that implement global tasks in system architectures with distributed autonomous agents can be generated. The generation of these local algorithms for each autonomous agent can be automated so that the particular local computations and communications occurring in and between agents are adapted to the particular characteristics of the system architecture. Thus, distributed algorithms implementing a specific global task are optimized for computational efficiency on a particular system architecture with the hierarchy serving as a meta-representation.

We abstract any global task to be a function evaluation

$$f : X_1 \times \cdots \times X_n \to Y$$

of n variables. The inputs $\{x_i\}_{i=1}^n$, where each x_i is an element of X_i, are the observables from the sensors of the system which can be physically distributed as in sensor networks. We will refer to this function evaluation as the *global function evaluation.* A further important characteristic we assume about this function is that the information needed to evaluate the function is contained in only a subset of the observed variables which depends on what the values of the observed variables are. For example, an instance of a function that represents a global task was given in Chapter 4. The global task of target localization on sensor networks is represented by the function $\underline{f}$ given in Eqn. 4.10. Note that $\underline{f}$ depends only on a subset $S_z \in \{1, \ldots, n\}$ of the n observed sensor measurements when the joint measurement observed is $z = (z_1, \ldots, z_n)$.

In order to distribute the evaluation of this global function, we must decompose the function into pieces. Thus, the first half of what we will call the hierarchy is a representation of the global function by a decomposition into a circuit of primitive functions. We will refer to this part of the hierarchy as the *graph topology.* The key idea for computational efficiency is that only a subset of the primitive functions of the circuit need to be explicitly evaluated in order to evaluate the global function. Thus, the second part of the hierarchy will be a control mechanism which selects

which of the primitive functions in the circuit representation of the global function should be explicitly evaluated. Note that this control mechanism is an application of the principle of information-based computation. We will refer to this part of the hierarchy as the *evaluation algorithm*.

The computational resources required are apparent in this representation which allows us to construct distributed algorithms with controlled usage of computational resources as was presented in Chapter 4. We begin this chapter with the relationship of this hierarchy representation to the computational models of circuits and branching programs in Section 5.1. Then, we present the potential application of the hierarchy representation to computer vision problems in Section 5.2. After motivating the desirable characteristics of the hierarchy and why it is potentially a good tool for constructing efficient algorithms in Section 5.3, Section 5.4 formally defines the hierarchy representation and the important characteristics of it which are useful for the presentation of how the hierarchy is to be used.

The use of the hierarchy as a meta-representation for constructing computationally efficient algorithms implementing information processes for a variety of system architectures proceeds in three conceptual steps. For particular applications, these steps can be combined and reordered as appropriate.

1. First, a graph topology is constructed for the set of computations of the problem. The construction of the graph topology of a hierarchy is presented in Section 5.5. The primary aim of this step is to encode the set of computations of the problem into a circuit of primitive computations so that there exists an efficient evaluation algorithm to be chosen in the next step. This step corresponds to the construction of the graph topology in Section 4.3.3.

2. Second, an evaluation algorithm is chosen over the constructed graph topology which determines what computations in the hierarchy need to be explicitly evaluated. The evaluation algorithm amounts to a choice of uplinks, downlinks, and trigger sets as presented in Section 4.3.4. The idea is that only a subset of the primitive computations in a hierarchy ever need to be explicitly evaluated depending on the observed measurements of the system. The evaluation algorithm will eventually be converted into an on-line algorithm to be run on a specific system architecture. Thus, the goal is to choose an evaluation algorithm on the hierarchy which corresponds to an efficient on-line algorithm on the system architecture. Section 5.6 will present the possible evaluation algorithms over a constructed hierarchy.

3. Finally, the evaluation algorithm is converted into a set of rules from which an on-line algorithm for a specific system architecture is generated. Section 5.7 presents the abstract conversion method.

Figure 5-1 shows the block diagram for these steps. The steps outlined above can be automated which can be important if the system architecture is dynamic or ad

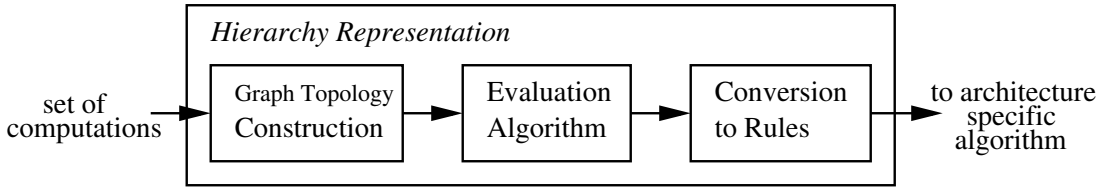

Figure 5-1: Steps in using the hierarchy for constructing architecture specific algorithms. The hierarchy bridges the gap between the abstract computations and the implementation of the computation by an algorithm.

hoc as in sensor networks so that the algorithms generated with the hierarchy can be adapted to the system architecture. For example, in sensor networks deployed for battlefield awareness, the positions of the sensor nodes are unknown a priori so that once the sensor network has self-configured, it must generate information collection algorithms for the task of target localization adapted to the current sensor network topology. Thus, the process of constructing a graph topology for the hierarchy, generating an evaluation algorithm, and converting the evaluation algorithm into an on-line algorithm for the specific system architecture must be automated and, hence, implementing this process must not overwhelm the computational resources available (like taking too long). However, since this process is essentially an off-line process, we will not concern ourselves too much with the computational complexity of the automation of these steps.

As for how we intend to automate the steps above, the graph topology of the hierarchy will be constructed by choosing a heuristic and running the heuristical graph topology construction algorithm of Section 5.5. We emphasize the need for using a heuristic and the difficulty in designing a good heuristic to construct a graph topology. This step is the construction of a circuit implementing a specific set of global computations we desire the system to evaluate. The process of choosing an evaluation algorithm (Section 5.6) and converting it to a set of rules (Section 5.7) is automated by optimizing a chosen cost criterion consistent with the costs of the resulting on-line algorithm running on the system architecture under the constraint that the set of global computations we desire the system to evaluate are evaluated correctly. We will define the notion of correct evaluation in Section 5.6. We end this chapter with a summary in Section 5.8.

Although we have used the hierarchy representation as a tool for constructing communication algorithms for target localization only in this thesis, we claim that the distributed implementation of the other individual information processing tasks of the layered information processing architecture presented in Chapter 3 could also have been generated by constructing an appropriate hierarchy. Furthermore, we also speculate that the design of the layered information processing architecture itself,

including the tasks within layers and the top-down priming and bottom-up spawning interactions, could also be considered to be the construction of a macro-hierarchy which could be automated. We have no idea how to make a concrete case for this speculation, but we mention it because we believe that the hierarchy representation, which is a representation embodying the principle of information-based computation, is applicable for constructing general complex systems with intelligent behavior.

5.1 Relationship to Circuits and Branching Programs

We will show the relationship of the hierarchy representation to circuits and branching programs in this section and give a preview of the hierarchy representation. The hierarchy can be viewed as a computational and communication model which is a combination of circuits and branching programs. In particular, it is a circuit with a selection mechanism for evaluation, and it is a branching program where there exist parallel processes which query and make local decisions to trigger computations of the hierarchy.

5.1.1 As a Circuit with a Selection Mechanism

A *circuit* is a directed acyclic graph as shown in Figure 5-2. Each node, or gate, is associated with a function which takes as input the values assigned to incoming edges. The output of the function is assigned to the outgoing edges incident on the node. Furthermore, we designate a set of nodes with no incoming edges to be the input nodes. Each input node is associated with a variable. The outgoing edges incident to an input node are assigned the value of the variable.

For example, the circuit in Figure 5-2 consists of 5 nodes associated with the functions $\{f_i\}_{i=1}^5$ and 6 input nodes associated with the variables $\{x_i\}_{i=1}^6$ which are elements of some set X. When the variables $\{x_i\}_{i=1}^6$ are assigned values, the circuit then defines a set of function evaluations. The outgoing edges incident to each f_i are assigned to the following.[1]

$$\begin{aligned} v_1 &= f_1(x_1, x_2, x_3) \\ v_2 &= f_2(x_2, x_4) \\ v_3 &= f_3(x_5, x_6) \\ v_4 &= f_4(v_1, v_2) \\ v_5 &= f_5(v_4, v_2, v_3) \end{aligned}$$

[1] To be completely precise, since the order of the inputs to f_i may matter, we must designate which incoming edge corresponds to which input of f_i. To keep the notation simple, let us keep this implicit.

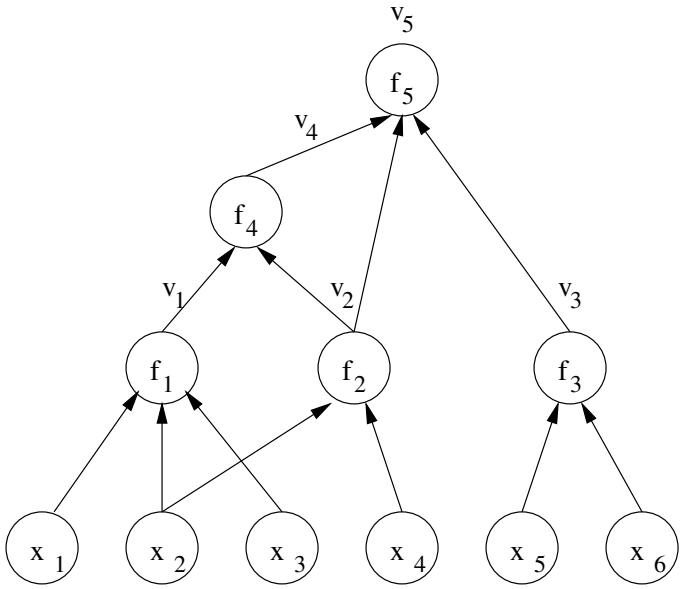

Figure 5-2: A simple circuit.

Although there is no outgoing edge incident on the node associated with f_5, v_5 represents the value assigned to such an outgoing edge if it existed. Notice that a circuit is a graphical representation of representing functions as compositions of one another. For example, v_4 and v_5 can be expressed as a function of the variables

$$\begin{aligned} v_4 &= h_4(x_1, x_2, x_3, x_4) \\ v_5 &= h_5(x_1, x_2, x_3, x_4, x_5, x_6) \end{aligned}$$

where

$$\begin{aligned} h_4(x_1, x_2, x_3, x_4) &= f_4(f_1(x_1, x_2, x_3), f_2(x_2, x_4)) \\ h_5(x_1, x_2, x_3, x_4, x_5, x_6) &= f_5(f_4(f_1(x_1, x_2, x_3), f_2(x_2, x_4)), f_2(x_2, x_4), f_3(x_5, x_6)) \ . \end{aligned}$$

The graph topology of the hierarchy is nothing more than a circuit in the sense that functions are represented as compositions of functions. However, we make the observation that it may not be necessary to explicitly evaluate all functions of the circuit depending on the assignment to the input variables. For example, let the set X that all variables x_i take value in, be $\{0, 1\}$, and let all functions in the circuit correspond to a logical AND function so that we are considering a Boolean circuit. Say that the input variables are assigned to be

$$(x_1, \ldots, x_6) = (1, 1, 1, 1, 1, 0) \ ,$$

and say that we only care about the value of v_5 given an assignment to the variables. If we explicitly evaluate f_3, we know that $v_3 = 0$. Then, because f_5 is a logical AND

function of its inputs, we know that the evaluation of f_5 will result in $v_5 = 0$. Thus, for this assignment to the input variables, there is no need to explicitly evaluate the functions f_1, f_2, or f_4. In general, different assignments to the input variables require different functions to be explicitly evaluated.

The hierarchy representation is a generalization of the circuit in the sense that a control mechanism for determining which functions of the circuit to explicitly evaluate is encoded along with the circuit representation of functions as compositions of other functions. Recall that in Section 4.3.3, we encoded a set of trigger functions, which were all multi-input AND functions, in terms of a network of two-input AND functions. We presented a heuristical algorithm for constructing such a graph topology in that section. In Section 5.5 of this chapter, we will consider the problem of constructing a graph topology for a general function f based on some given set of primitive functions.

5.1.2 As a Branching Program with Parallel Processes

A *branching program* is a directed acyclic graph where all nodes, referred to as query nodes, are labelled by variables except for two output nodes which are labelled with a 1 or 0 as in Figure 5-3. Each query node has two outgoing edges labelled with a 1 or 0. One node is designated the start node. The idea is to traverse this acyclic graph depending on the values of the variables associated with query nodes. If the variable is equal to 0, we proceed to the node along the edge labelled with 0. If the variable is equal to 1, we proceed to the node along the edge labelled with a 1. Figure 5-3 shows a simple branching program with four query nodes associated with variables $\{x_i\}_{i=1}^4$ where the start node is pictured with a double circle.

To demonstrate the traversal of the branching program, say the variables are assigned to be

$$(x_1, \ldots, x_4) = (0, 1, 1, 0) \ .$$

Then, starting at the query node labelled with x_1, we observe that x_1 is assigned to be 0. We then traverse to the node labelled with x_2. Observing that x_2 is equal 1, we traverse to the node labelled with x_4. Finally, observing that x_4 is equal to 0, we traverse to the node labelled with 1. Then, the output of this branching program is equal to 1. Note that, in this traversal of the branching program with this particular assignment of the variables, variable x_3 was never queried.

Any branching program encodes a Boolean function. In this example, the branching program encodes a Boolean function which is given by the expression

$$x_1 x_3 \vee x_1 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 \bar{x}_4 \vee \bar{x}_1 \bar{x}_2 x_3$$

where $\vee$ represents a logical OR and $\bar{x}$ represents the complement of the Boolean variable x. This simple branching program can be easily extended to compute functions on finite alphabets X by allowing each variable to take a value in X, each query

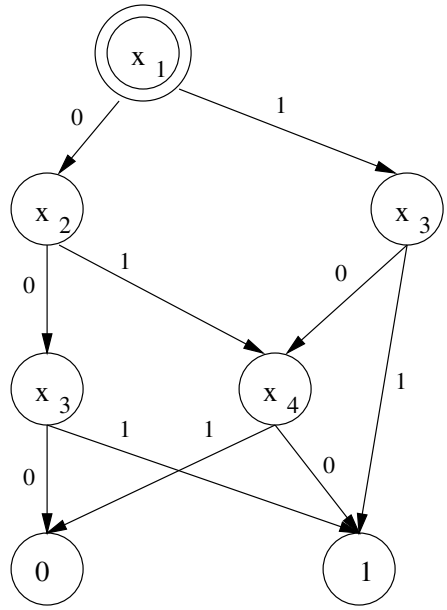

Figure 5-3: A simple branching program.

node to have $|X|$ outgoing edges, and there being $|X|$ output nodes corresponding to each distinct element of X. Thus, the branching program is a representation of functions with a mechanism which queries possibly only a subset of the variables.

By extending the query variables to correspond to querying the output of some function evaluation, we can consider constructing a branching program for traversing a circuit where the query variables correspond to the values assigned to edges of a circuit. For example for the circuit of Figure 5-2, we can imagine constructing a branching program which computes v_5 with the "query" variables

$$\{x_i\}_{i=1}^{6} \cup \{v_i\}_{i=1}^{4} \ .$$

Thus, it may seem that the branching program may be a good candidate for constructing a traversal mechanism for a circuit. However, what is limiting about the branching program is that the traversal of the branching program is sequential. That is, for any given assignment of the query variables, we traverse exactly one path from the start node of the branching program to one of the output nodes. This would mean traversal of the nodes of the circuit is sequential.

The hierarchy is considered to be a generalization of a branching program in the sense that "traversal", by which we mean the explicit evaluation, of the functions associated with nodes of the graph topology can occur in parallel. Recall from Section 4.3.4 that after constructing the graph topology of the hierarchy, we chose a subset of edges to be uplinks and downlinks. Furthermore, we chose a trigger set for

each uplink and downlink. The downlinks can be considered to be queries which are implicit in a branching program by the labelling of each node with a variable. The uplinks of a hierarchy are comparable to the directed edges in a branching program. The trigger sets which are associated with each uplink[2] correspond exactly to the labelling of each edge in a branching program by a 1 or 0. The difference is that trigger sets can take on a subset of values, and it is possible for there to be multiple or no nodes whose associated computations are triggered to be evaluated in the hierarchy.

This mechanism for explicitly evaluating the functions associated with nodes of the hierarchy is completely determined by the choice of uplinks, downlinks, and trigger sets, which we term the *evaluation algorithm* part of a hierarchy to be presented in Section 5.6. Comparable to the simple rule for traversing a branching program, we have the *canonical message passing algorithm* presented in Section 5.6.2 which is the rule that determines which nodes of the hierarchy are to evaluate their associated functions according to messages passed between nodes.

5.1.3 Preview of the Hierarchy

Let us give a preview of the hierarchy. The hierarchy consists of two parts.

1. The *graph topology* part of the hierarchy is the underlying circuit as shown in Figure 5-4. Being a circuit, the graph topology is a directed acyclic graph, and we arrange all nodes of the graph so that all directed edges are pointing upwards. The general heuristical construction of the graph topology is presented in Section 5.5 which is an algorithmic construction of the graph topology part of the hierarchy.

2. The *evaluation algorithm* part of the hierarchy is a choice of uplinks, downlinks, and trigger sets as shown in Figure 5-5. These added elements to the graph topology, which are an additional set of directed edges with a subset associated with some of them, define the possible flow of messages between nodes for the purpose of controlling the explicit evaluation of computations associated with nodes of the hierarchy. Choosing an evaluation algorithm is presented in Section 5.6.

Note that the evaluation algorithm is an additional part of the hierarchy which is added to the underlying circuit for the purpose controlling which functions in the hierarchy to explicitly evaluate. Our idea for implementing this control mechanism is to consider each node of the hierarchy to be an individual computational unit that runs the *canonical message passing algorithm*. When prompted to do so by neighboring connected computational units of the hierarchy, the function associated

[2]In Chapter 4, we presented the hierarchy where there is a trigger set associated with every uplink *and* downlink. However, in this general presentation of the hierarchy, we only need trigger sets for uplinks. See the footnote on page 328.

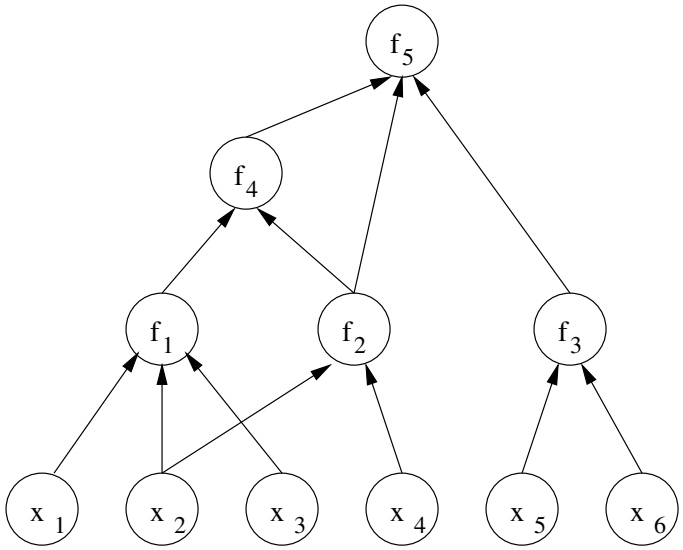

Figure 5-4: The *graph topology* of a hierarchy is a circuit. In the picture, nodes have been arranged so that all directed edges point upwards.

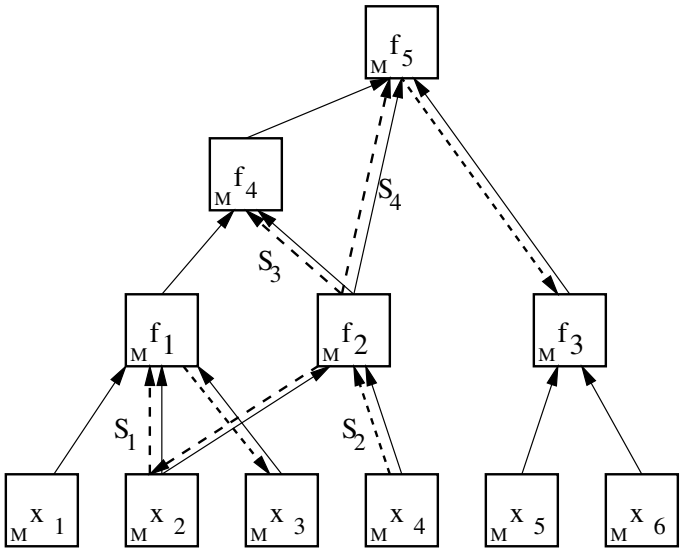

Figure 5-5: The choice of uplinks, downlinks, and trigger sets associated with uplinks, pictured as dashed arrows and subsets associated with dashed arrows, encompass the *evaluation algorithm* part of the hierarchy.

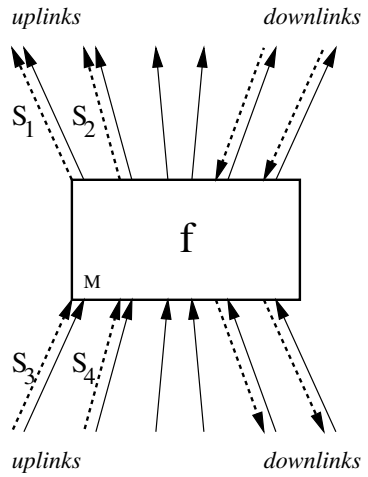

Figure 5-6: Each hierarchy node is associated with a function with two sets of directed edges: information edges and control edges. Each node is a computational unit that runs the *canonical message passing algorithm* which is a message passing scheme that determines when this node should evaluate its associated function and when this node should initiate the evaluation of functions of other nodes.

with the computational unit is explicitly evaluated. We will describe the canonical message passing algorithm in detail in Section 5.6.2. To indicate that each node of the hierarchy is an individual computational unit, we picture the nodes of the hierarchy with a square denoted by the associated function and an "M", referring to the canonical *m*essage passing algorithm, as shown in Figure 5-5.

Each node of the hierarchy is a computational unit with two sets of directed edges incident on it.

1. *Information edges* always point upwards and correspond to the directed edges of the underlying circuit part of the hierarchy. These edges are pictured as solid arrows in Figure 5-5 and Figure 5-6.

2. *Control edges* can point both upwards or downwards, referred to as uplinks and downlinks respectively, and can exist only if they are along an information edge. These edges are pictured as dashed arrows in Figure 5-5 and Figure 5-6.

All uplinks are associated with a trigger set whose role will be explained in the detailed discussion of the canonical message passing algorithm in Section 5.6.2.

The final key aspect of the hierarchy is to view it as a *meta-representation*. By assigning nodes of the hierarchy to agents, as indicated in Figure 5-7, some of the messages passed between hierarchy nodes running the canonical message passing algorithm correspond to real communications between agents. In particular, the messages passed along the information and control edges between nodes f_2 and x_2, f_2 and f_4,

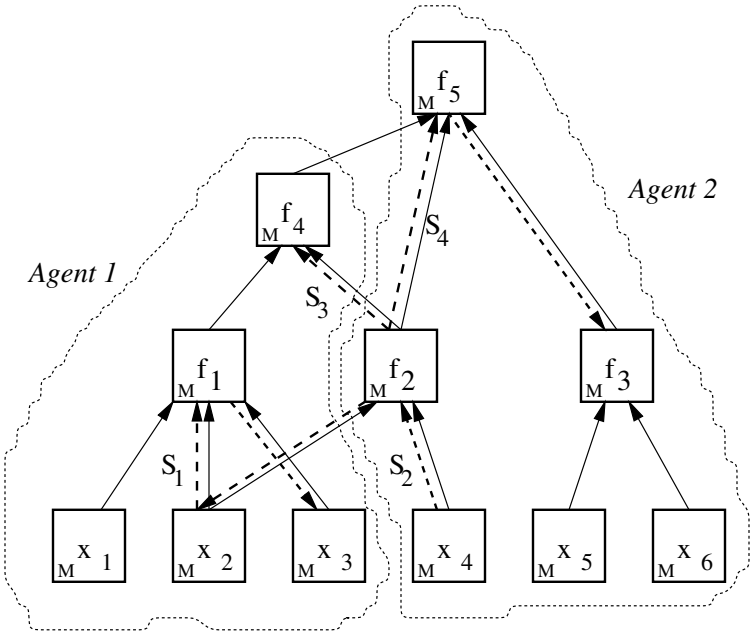

Figure 5-7: An example of an agent assignment.

and f_4 and f_5 correspond to real communications between agents 1 and 2. Thus, the hierarchy can represent distributed algorithms running on system architectures with distributed autonomous agents where agents evaluate local computations and communicate these results to each other until the desired global computation is evaluated. Using the hierarchy to construct distributed communication algorithms (DCA) for the problem of target localization was presented in Chapter 4. The more abstract conversion of the hierarchy into a set of rules for evaluating primitive computations and communicating the results is presented in Section 5.7.

5.2 Potential Application to Object Recognition in Computer Vision

Object recognition in computer vision based on images has been an active research field for several decades. Examples of object recognition include optical character recognition (OCR), handwriting recognition, face recognition, and bar-code scanning. It is a problem that we humans find quite natural, but it is a problem which has evaded practical implementation on computers. What the methodology of this thesis has to offer is a way of traversing an appropriately constructed layered information processing architecture efficiently so that recognition of a large class of objects is efficient. We envision the hierarchy representation as a tool which constructs the

appropriate bottom-up spawning and top-down priming actions between the layers of the layered architecture. In particular to object recognition, layers represent different representations of the information contained in the image ranging from the image itself to an abstract object space. This section is highly speculative since no details have been worked out; however, we include this section in this chapter to hint at the possible broad applicability of the methodology presented in this thesis.

5.2.1 Background

There are essentially two methodologies in the literature for attacking the problem of recognition in computer vision: the *top-down* approaches and the *bottom-up* approaches. Both approaches make use of a layered architecture which has been argued in the literature to be consistent with the structure of the brain. It has long been suggested that there must be a way of complementing bottom-up and top-down processes so that the result is an efficient algorithm for object recognition. The choice of an evaluation algorithm by viewing the layered architecture as a hierarchy is the potential application of the methodology presented in this thesis for complementing top-down and bottom-up processes.

Let us discuss some background on ideas for combining top-down and bottom-up processes. In [25], Ullman suggests a structure referred to as *counterstreams* for combining top-down and bottom-up processes which we had described in the introduction on page 46. Let us briefly summarize the ideas behind the counterstreams structure here again. We have a layered architecture where each layer represents one possible type of transformation like orientation or lighting from image to object. The idea is to start a search at both the bottom and top of this layered architecture. As the two processes search through highly probably transformations of objects to images (top-down process) and images to objects (bottom-up process), the two processes leave behind a "primed" trail. If the two processes happen to meet at some intermediary layer, then a path from object to image through the "primed" trail has been found which indicates a match between object and image. The counterstreams structure is given some physiological validity by comparing its structure to the structure of the neural pathways in the brain, the primary attribute being that there are complementary pathways both up and down the layers of the brain.

The main difficulties associated with object recognition problems include the following.

1. The images of a particular object are widely varying. For example, consider the variations of a single person's face which can have a variety of expressions and can be viewed from a variety of angles and lighting conditions. To achieve good recognition accuracy, all of these variations must be captured by the object model.

2. The space of objects is immense. The set of objects that we are able to recognize

in our everyday interactions with the world is huge. Somehow, we are storing these object models in a manner which allows us to retrieve highly evidenced objects efficiently.

The two difficulties amount to searching a huge object space for matches to some given input image. Top-down theories of vision assert that the only way to search such a huge space is to take advantage of context in order to cut down the space that must be searched. Although these top-down theories have much theoretical appeal, it is well-known that our brains actually do process most of the information contained in images to perform a near complete search of the entire space of objects. Our methodology for dealing with the huge space of objects will be to pick an evaluation algorithm which instantiates only those representations in each layer which are highly evidenced.

Compositionality of objects is an observation that objects in the real world are aggregates of parts which are further aggregates of subparts and so on. As an example, a face is composed of eyes, a nose, and a mouth. Each of these parts, say the nose, is further composed of subparts like the bridge and nostrils. Furthermore, there is an abstraction involved at each part. The specifics of the mouth, be it large or small, is irrelevant to whether we consider the mouth as a part of the face. The idea to take away from compositionality is that we have layers of representations from which more complex representations in higher layers can be formed.

Finally, an important notion in the study of object recognition in computer vision has focussed on *invariances* of objects. The invariants of an object can be used to characterize general properties of objects so that by taking advantage of these invariants, the computational complexity of searching through the object space is reduced. Some typical invariances for two-dimensional objects like handwritten characters include translation, rotation, scale, and skew. By devising appropriate normalization procedures on images like scaling the characters to some nominal size and adjusting for skew and rotation, the handwriting recognition system need only recognize characters in these normalized images. Thus, it is important that known invariances of images of objects be exploited when designing the recognition system.

5.2.2 Layers of Representation

We propose the following layers of representation to design an efficient object recognition system based on grayscale images. We list the layers from the lowest layer to the highest layer.

1. **Grayscale Images** This layer is the raw image data taken from a camera. We assume a rectangular lattice of pixels each taking a set of finite values representing the light intensity measured at that pixel.

2. **Edge Fragments** This layer is composed of small edge fragments based on 4×4 subimages. We assume only step edges.

3. **Short Line Segments** This layer is composed of aggregates of edge fragments which are collinear.

4. **Singularities** This layer is composed of aggregates of short line segments which denote the singularities of corners of various angles.

5. **To Object Models** Object models will consist of certain arrangements of singularities and short line segments.

This layered representation is essentially a handcrafted graph topology for a hierarchy since computations associated with higher layer representations are functions of computations of lower layer representations. That is, we design models between the representations of each layer as functions of the representations of layers below. We assume that these models are local in the sense that a particular edge depends only on a small subimage of the entire grayscale image, short line segments depends only on a local region of edge fragments, singularities depend on a small set of line segments, and so on up the layered architecture. The idea is that highly evidenced representations from the lower layers can spawn the computation of higher layer representations. Then, to compute the evidence for these higher layer representations, there may be priming interactions which initiate the computations of certain lower layer representations. Note that these interactions are exactly the bottom-up spawning and top-down priming interactions we have argued for in the layered information processing architecture. Furthermore, by designing models in each layer to range from generic to specific, we believe that the resulting recognition system can be made efficient.

5.3 Motivation

We wish to use the hierarchy as a meta-representation for constructing computationally efficient algorithms for various system architectures by complying with the principle of information-based computation. There are four desirable characteristics for such a meta-representation.

(M1) Correctness Constraints We desire a representation from which we can characterize a whole family of algorithms which can implement the desired task correctly.

(M2) Computational Resources We desire a representation from which the amount of resources required for computation can be measured.

(M3) Reuse of Partial Computations We desire a representation which reuses previous computations as much as possible so that resources are never wasted by recomputation.

(M4) Design for Various Architectures We desire a representation from which efficient algorithms can be designed for a variety of system architectures, including architectures with distributed autonomous agents.

The first three characteristics all lead to the same conclusion:

The representation should break down the computations of the problem into simple subcomputations.

Since there are several ways to break down the computations of a problem into subcomputations, we should be able to characterize a whole family of algorithms as desired in **(M1)**. Each of these subcomputations require some amount of computational resources to compute so that resource usage can be measured as desired in **(M2)**. Furthermore, by keeping track of what computations have already been performed, we can make maximal reuse of previous computations as desired in **(M3)**.

The characteristic **(M4)** suggests that the hierarchy should be considered a *meta-representation* from which algorithms for various system architectures are generated. For example, we should be able to generate algorithms on layered information processing architectures which exhibit the complementation of top-down and bottom-up processes and generate distributed algorithms over system architectures with distributed autonomous agents.

To further emphasize that the hierarchy is a meta-representation, we will consider the hierarchy to be a "computation-centric" representation where computations are partially ordered with no specification of the processors or agents responsible for them. The process of generating algorithms from the hierarchy to a specific system architecture converts the "computation-centric" representation into a "processor-centric" algorithm on the system architecture. It is in this conversion process that top-down and bottom-up interactions between processes of adjacent layers emerge in layered information processing architectures and distributed algorithms are generated for system architectures with distributed autonomous agents.

5.4 Mathematical Formalization

We formalize the hierarchy and characteristics of the hierarchy in this section to facilitate the discussion to follow on how to use the hierarchy as a meta-representation for constructing computationally efficient algorithms. We will define some terms we have been using more rigorously.

By a *computation*, we refer to the combination of

1. a function
$$f : X_1 \times \cdots \times X_n \to Y$$
with a set of inputs, each taking value in some space X_i, and with output in some space Y and

2. an associated algorithm for evaluating the function.

We say that a computation is *evaluated* when we calculate the output $f(x_1, \ldots, x_n)$ for some inputs $(x_1, \ldots, x_n) \in X_1^n$ using the associated algorithm. Any evaluation of a computation consumes some computational resources of time, memory, and energy since the algorithm must be run on some specified system architecture. Notice that there exist many algorithms which implement the evaluation of the same function with possibly varying requirements of computational resources. We consider one computation to be *simpler* than another if the computational resources needed for evaluation is less.

The starting point for constructing computationally efficient algorithms is a given set of functions

$$\{f_i\}_{i \in I} \tag{5.1}$$

for the system to evaluate, where the functions

$$f_i : V^n \to V$$

are a function of n variables taking value in a common set V with outputs in the set V. I is some finite index set. More generally, each of the n variables may take value in different sets and the output of the functions f_i may be different; however, these generalizations can be incorporated in a straightforward manner in the presentation of the hierarchy to follow.

The idea is to express these given functions f_i as functional compositions of primitive computations as was motivated by desirable characteristics **(M1)**, **(M2)**, and **(M3)** of Section 5.3. The resulting circuit of primitive computations is the graph topology part of the hierarchy, and it encodes an algorithm of how to evaluate the functions $\{f_i\}_{i \in I}$ according to evaluations of the primitive computations.

5.4.1 Hierarchy Definition

We will give a definition of the hierarchy in this section. Let us identify some standard notation to be used throughout this chapter.

V is the basic set of values for all functions. Examples include $V = \{0, 1\}$ for the boolean case and more generally, $V = [0, 1]$ for what we will later term as *evidence* functions. By "basic set of values" for all functions, we mean that all functions f in the given set of functions $\{f_i\}_{i \in I}$ are functions of the form

$$f : V^n \to V$$

where n is the number of variables that f depends on. Furthermore, the function part of all primitive computations (to be defined next) have domain and range which can be constructed from the set V.

$\mathcal{F}$ is some given set of *primitive computations*, which are a collection of functions, with domain V^m for possibly different values of a positive integer m, and range V along with a given associated algorithm for evaluating each. We assume that the set of functions $\{f_i\}_{i \in I}$ with domain V^n can be represented as a functional composition of the primitive computations in $\mathcal{F}$ so that there is a restriction on what sets $\mathcal{F}$ can be considered to be a set of primitive computations. For example, for the problem of constructing communication algorithms for target localization, the only primitive computation we needed for encoding the triggers was a two-input logical AND function.

We will eventually impose a probability distribution for various random variables on all of the spaces above so that we can construct hierarchies and evaluation algorithms which exhibit some minimal expected computational resource requirements. In particular, since the graph topology of a hierarchy is a circuit with nodes that are associated with variables and functions, by specifying a probability distribution over the variables, we have a graph topology constructed over random variables. Then, we can compute various statistics regarding the values that the variables or functions of them take. In the toy example, Section 4.3 of Chapter 4, we demonstrated the computation of the worst case communication cost during the construction of the graph topology in Section 4.3.3. By choosing a probability distribution over the values of the input tokens, we could have utilized a heuristic based on the expected communication cost. Furthermore, we demonstrated the computation of the expected communication cost for choosing an agent assignment, uplinks, downlinks, and trigger sets in Section 4.3.4 by choosing an appropriate probability distribution over the input tokens.

Let us now define the hierarchy.

Definition 5.1 *A* hierarchy $H = (N, F, E, U, D, \Upsilon)$ over n *variables* $x_i \in V$ *for* $i \in \{1, \ldots, n\}$ composed of primitive computations $\mathcal{F}$ *is a* 6*-tuple. The first three elements correspond to the* graph topology *of the hierarchy where*

(N, E) *is a directed acyclic graph with a finite number of nodes and*

$F : N \to \mathcal{F} \cup \{x_i\}_{i=1}^n$ *is a functional returning for each node in* N *either the primitive computation associated with it or a variable in* $\{x_i\}_{i=1}^n$

such that there exists exactly n *nodes of* N*, denoted by* N_0*, with no incoming edges in* E *and are each associated with a unique variable in* $\{x_i\}_{i=1}^n$.

The last three elements correspond to the evaluation algorithm *of the hierarchy where*

$U \subset E$ *is the set of uplinks where* $(a, b) \in U$ *means there is an uplink connecting the output of node* a *to the input of node* b*,*

$D \subset E$ is the set of downlinks where $(a, b) \in D$ means there is a downlink connecting the output of node a to the input of node b,[3]

$\Upsilon : U \to 2^V$ is a function returning for each uplink a subset of V, which is referred to as the trigger function*,*

such that $U \cap D = \emptyset$.

We will use the notation F_a to stand for the primitive computation or variable $F(a)$ for all nodes $a \in N$. Likewise, we use the notation $\Upsilon_{(a,b)}$ to stand for the subset of V returned by $\Upsilon((a, b))$ for all $(a, b) \in U$.[4]

The set of nodes N, the primitive computation or variable F_a associated with each node $a \in N$, and the set of edges E define a circuit. An edge $(a, b) \in E$ means that the input of the function associated with node a is connected to the output of the function associated with node b. That is, F_a is a function of F_b

$$F_a(\ldots, F_b, \ldots) \ .$$

The edges in E are the information edges of the hierarchy. The condition for the graph topology part of the hierarchy asks for the necessity of an input layer N_0 of nodes for which there exists no incoming edges and each is associated with a unique variable in $\{x_i\}_{i=1}^n$. The subject of Section 5.5 which we refer to as *graph topology construction* is about constructing N, F, and E given the set of functions (Eqn. 5.1) representing the global task of the problem in an algorithmic fashion. We emphasize that since an optimal construction of the graph topology is intractable, the idea of using a heuristic to construct the graph topology is of central importance.

An example of the graph topology of the hierarchy was given in Figure 5-5. The nodes N are the squares in the figure. For each node $a \in N$, the primitive computation or variable F_a associated with it is denoted by a function $\{f_i\}_{i=1}^5$ or a variable $\{x_i\}_{i=1}^6$. Note that the nodes associated with a variable are drawn at the bottom of the figure and have no incoming solid edges. These are the nodes N_0. The set of information

[3]Note that when we draw the downlink (a, b), we draw an arrow from node b to node a since a downlink points in the *opposite* direction of a directed edge $(a, b) \in E$ in the underlying circuit.

[4] It may seem puzzling that we have not defined the trigger function over the downlinks as well as the uplinks since we assigned a trigger set to each downlink as well as each uplink in the presentation of Section 4.1.2. The reason is that, in this general formulation of the hierarchy where nodes can have down degree greater than two, it is cleaner to add the effect of a trigger set on a downlink by choosing edges which were neither uplinks or downlinks in the examples of Section 4.1.2 to be an uplink and assigning a trigger set to the uplink. For example, in Figure 4-10, we add an uplink between the nodes labelled by g and h and set the trigger set to the same T as shown in the figure. Uplinks and downlinks no longer have the interpretation as corresponding to a "push" type or "pull" type communication in this general formulation. Instead, they indicate how computations associated with other nodes are to be initiated.

edges E are the solid edges which only point upwards. In particular,

$$E = \left\{ \begin{array}{l} (x_1, f_1), (x_2, f_1), (x_2, f_2), (x_3, f_1), \\ (x_4, f_2), (x_5, f_3), (x_6, f_3), (f_1, f_4), \\ (f_2, f_4), (f_2, f_5), (f_3, f_5), (f_4, f_5) \end{array} \right\} .$$

The second half of the definition is associating a subset of the edges E with uplinks U and downlinks D and choosing the trigger function Υ. The control edges, U and D, may only lie along one of the information edges E, although the direction of the control edge may be different than the information edge (as in the case for a downlink). The condition for this evaluation algorithm part of the hierarchy says that we can have an uplink, a downlink, or neither along each information edge, but never both. The choice of uplinks, downlinks, and trigger function essentially controls which primitive computations of the hierarchy should be evaluated depending on the results of what has already been evaluated. Section 5.6 will discuss this part, which is termed choosing an *evaluation algorithm*, in full detail.

For the example hierarchy in Figure 5-5, the uplinks are

$$U = \{(x_2, f_1), (x_4, f_2), (f_2, f_4), (f_2, f_5)\} \ ,$$

the downlinks are

$$D = \{(x_2, f_2), (x_3, f_1), (f_3, f_5)\} \ ,$$

and the trigger function is given by

$$\Upsilon(u) = \left\{ \begin{array}{lll} S_1 & , & u = (x_2, f_1) \\ S_2 & , & u = (x_4, f_2) \\ S_3 & , & u = (f_2, f_4) \\ S_4 & , & u = (f_2, f_5) \end{array} \right. .$$

Note that the downlink (x_2, f_2) is drawn with an arrow from node f_2 to node x_2 and not from node x_2 to node f_2 to indicate the direction that control messages will flow along the downlink.

Recalling Figure 5-1, we can now show in detail the different components of a hierarchy $H = (N, F, E, U, D, \Upsilon)$ and the steps of graph topology construction, choosing an evaluation algorithm, and converting to an algorithm on a specific system architecture as shown in Figure 5-8. When we say the *graph topology* part of a hierarchy $H = (N, F, E, U, D, \Upsilon)$, we mean the triple (N, F, E) of the 6-tuple. When we say the *evaluation algorithm* of a hierarchy, we mean the triple (U, D, Υ) of the 6-tuple. Note that evaluation algorithm (U, D, Υ) of a hierarchy cannot be defined unless we have defined the graph topology (N, F, E) of a hierarchy.

Characteristics of the graph topology of a hierarchy

For the rest of this section, let us define some characteristics of the graph topology part of the hierarchy, which is defined by the triple (N, F, E), which will aid in the presentation of subsequent sections.

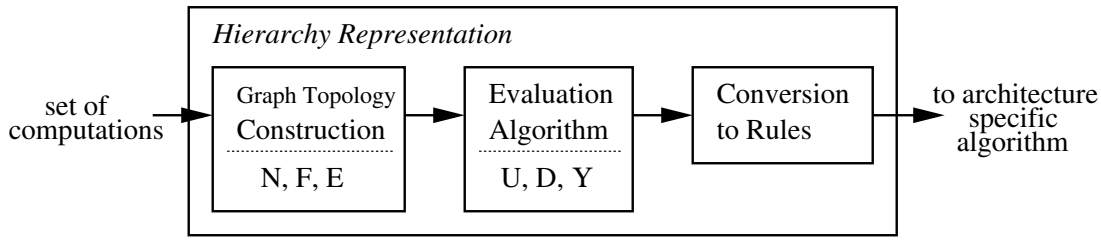

Figure 5-8: The steps of using the hierarchy representation for constructing computationally efficient algorithms. The process of graph topology construction constructs the N, F, and E parts of the hierarchy while the process of choosing an evaluation algorithm constructs the U, D, and Υ parts of the hierarchy.

Recall in Section 4.1 that the notion of a node being above or below one another was important. We formalize this notion here according to the directed information edges E.

Definition 5.2 *For a hierarchy with graph topology defined by (N, F, E), a node $a \in N$ is said to be* lower *than $b \in N$, denoted*

$$a \leq b \ ,$$

if $a \neq b$ and there exists a directed path starting from a and ending at b according to the directed edges E.

Definition 5.3 *For a hierarchy with graph topology defined by (N, F, E), a node $a \in N$ is said to be* higher *than $b \in N$ if b is lower than a.*

Lemma 5.1 *$\leq$ is a partial order.*

Proof *This follows from Def. 5.2, Def. 5.3, and the definition of the graph topology part of the hierarchy (Def. 5.1) as being a directed acyclic graph.* $\square$

Note that for the input layer of nodes N_0, there are no nodes in N which are lower than these nodes. The computations associated with the nodes

$$\{a \in N : a \leq b\}$$

are the only computations that can affect the output value of the computation associated with the node b. We mention this to allude to the characterization of $\{f_i\}_{i \in I}$-computable hierarchies in Section 5.5.1. Let us define the spawning function and decomposition function which will be useful for characterizing the neighboring nodes which are connected to each node by a directed information edge. The spawning function returns the parents of a node, and the decomposition returns the children of a node, where parents and children on the hierarchy were defined on page 208.

Definition 5.4 *The* spawning function $\mathbf{s} : N \to 2^N$ *is defined for every node* $a \in N$ *by*

$$\mathbf{s}(a) = \{b \in N : (a, b) \in E\} \ .$$

Definition 5.5 *The* decomposition function $\mathbf{d} : N \to 2^N$ *is defined for every node* $a \in N$ *by*

$$\mathbf{d}(a) = \{b \in N : (b, a) \in E\} \ .$$

The set of nodes $\mathbf{s}(a)$ is called the *spawn* of a node $a \in N$ and is the set of nodes which are connected to the output of node a. The set of nodes $\mathbf{d}(a)$ is called the *decomposition* of a node $a \in N$ and is the set of nodes which are connected to the inputs of node a. We also refer to the decomposition of a node a as the input nodes of node a. To be pedantic, for all $b \in \mathbf{s}(a)$, $a \leq b$, and for all $b \in \mathbf{d}(a)$, $a \geq b$. Thus, our view of the computations of the hierarchy are that higher nodes encode higher level computations.

Lemma 5.2 *(Inclusive Property) For* $a, b \in N$, $b \in \mathbf{s}(a)$ *if and only if* $a \in \mathbf{d}(b)$.

Proof *This is clear from the definitions of* $\mathbf{s}(\cdot)$ *and* $\mathbf{d}(\cdot)$. □

The inclusive property merely states that if a node is part of the spawn of another node, then the latter node is part of the decomposition of the former node.

5.4.2 Layers

Another property of the nodes determined completely by the graph topology of a hierarchy is the layer a node lies in, which will become important in designing algorithms with limited memory requirements and availability of multiple agents. Each node can lie in several layers in our definition of layers to follow. The reason we define layers in this manner is motivated by the desire to characterize the memory requirements and how sequential or parallel the explicit evaluations of computations in the hierarchy can occur.

We will assign nodes of a hierarchy to layers inductively. That is, we will determine the set of nodes in layer $i+1$, $N_{i+1} \subset N$, from the set of nodes in layer i, $N_i \subset N$. The input layer N_0 is already given in the definition of the hierarchy (Def. 5.1). To simplify notation, let

$$N_j^k = \bigcup_{l=j}^{k} N_l \ .$$

Figure 5-9 shows an example of the graph topology of a hierarchy (where all edges are directed upwards) denoted with the set of layers each node lies in.

The set of nodes in layer $i+1$ is defined as

$$N_{i+1} = Q_i \cup \{a \in N_i : \mathbf{s}(a) \not\subset Q_i\}$$

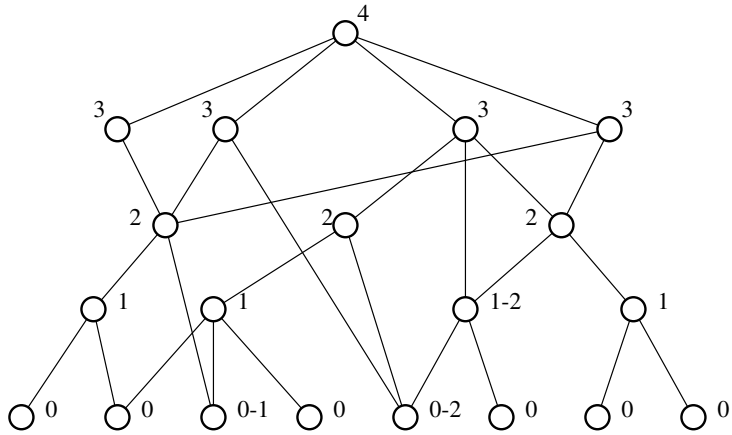

Figure 5-9: An example of the graph topology of a hierarchy with each node denoted by the set of layers that the node lies in. All edges are information edges directed upwards.

where

$$Q_i = \{a \in P_i : \mathbf{d}(a) \subset N_0^i\}$$

and

$$P_i = \left(\bigcup_{a \in N_i} \mathbf{s}(a) \right) - N_i \ .$$

P_i is the set of nodes which are in the spawn of any node of N_i but not in N_i. Q_i is a restriction of this set to those nodes which have all nodes of its decomposition lying in layers N_i or below. We also include some nodes of N_i in N_{i+1} whose outputs will be inputs to higher layers. The set of layers that node $a \in N$ lies in is the set

$$\Lambda_a = \{i \in \{0, 1, \ldots\} : a \in N_i\}$$

which are the numbers shown next to each node in Figure 5-9. It will be useful to define the lowest layer and highest layer that a node a lies in.

$$\begin{aligned} \mathbf{max}\Lambda(a) &= \max_{i \in \Lambda_a} i \\ \mathbf{min}\Lambda(a) &= \min_{i \in \Lambda_a} i \end{aligned}$$

The motivation for defining layers in this manner is to illustrate that the evaluation of all computations in the hierarchy can be considered unidirectionally going up one layer at a time, which is the layer abstraction result of Theorem 5.1 to follow. That is, a possible evaluation scheme of the computations in the hierarchy would be to first compute the computations associated with nodes in N_0, then compute the computations associated with nodes in N_1, and proceed on up the layers of the hierarchy.

This layer abstraction result has consequences on the memory requirements and how distributed evaluation of the computations encoded by the hierarchy can be. We must emphasize, however, that the point of the following discussion is not to say that evaluations of the computations in the hierarchy must proceed layer by layer because it is precisely because we do not evaluate computations layer by layer that we achieve efficiency in the resulting on-line algorithm on a specific system architecture.

Lemma 5.3 *(Highest Layer) For any node $b \in N$ with $\mathbf{max}\Lambda(b) = i$, $\mathbf{s}(b) \subset N_{i+1}$.*

Proof *We will prove the lemma by contradiction. If $\mathbf{max}\Lambda(b) = i$, then $b \in N_i - N_{i+1}$. Say there exists an element $c \in \mathbf{s}(b)$ such that $c \notin N_{i+1}$. Since $Q_i \subset N_{i+1}$, this implies $c \notin Q_i$. This implies $\mathbf{s}(b) \not\subset Q_i$. Since $b \in N_i$, the second term in the expression for N_{i+1} implies $b \in N_{i+1}$, which contradicts the fact that $b \in N_i - N_{i+1}$. Hence, $\mathbf{s}(b) \subset N_{i+1}$.* $\square$

This lemma says that if the highest layer of node b is

$$\mathbf{max}\Lambda(b) = i \ ,$$

the computation associated with node b is input to computations in nodes which reside *directly above* in layer $i+1$. The consequence of this is that once all computations associated with nodes in layer $i+1$ have been evaluated, the computation associated with b can be discarded because all subsequent computations of nodes in higher layers do not use the result of b.

Lemma 5.4 *(Lowest Layer) For a node $b \in N$ with $\mathbf{min}\Lambda(b) \geq i+1$,*

$$\mathbf{d}(b) \cap (N_0^{i-1} - N_i) = \emptyset \ .$$

Proof *We will prove the lemma by induction. If $\mathbf{min}\Lambda(b) \geq i+1$, then $b \in N - N_0^i$. By Lemma 5.3, for any $b \in N_0 - N_1$, $\mathbf{s}(b) \subset N_1$. This implies that the decomposition of any node in $N - N_0^1$ does not contain an element of $N_0 - N_1$ due to the inclusive property (Lemma 5.2) of $\mathbf{s}(\cdot)$ and $\mathbf{d}(\cdot)$.*

Assume that the decomposition of any node in $N - N_0^i$ does not contain an element of $N_0^{i-1} - N_i$. This implies that the spawn of any element of $N_0^{i-1} - N_i$ contains no elements of $N - N_0^i$ and, in particular, no elements of $N - N_0^{i+1}$. By Lemma 5.3, for any $b \in N_i - N_{i+1}$, $\mathbf{s}(b) \subset N_{i+1}$ which says that no element of $N_i - N_{i+1}$ has a spawn with elements in $N - N_0^{i+1}$. Since

$$N_0^i - N_{i+1} = \left(N_0^{i-1} - N_i\right) \cup (N_i - N_{i+1}) \ ,$$

the properties of the spawn of elements on both terms of the right hand side imply that the spawn of any element of $N_0^i - N_{i+1}$ contain no elements of $N - N_0^{i+1}$. By the

inclusive property of $\mathbf{s}(\cdot)$ *and* $\mathbf{d}(\cdot)$*, we find that the decomposition of any element of* $N - N_0^{i+1}$ *does not contain an element of* $N_0^i - N_{i+1}$*, which concludes the induction step.* $\square$

The above lemma says that for any node which is strictly above layer i, as expressed by

$$\mathbf{min}\Lambda(b) \geq i + 1 \ ,$$

the computation associated with these nodes is a function of nodes in layer i or above.

Theorem 5.1 *(Layer Abstraction) For every* $a \in N_{i+1}$*, either* $a \in N_i$ *or* $\mathbf{d}(a) \subset N_i$*.*

Proof *If* $a \in N_i$*, then we are done. Suppose* $a \notin N_i$*. Then,* $a \in Q_i \subset P_i$*. Since* $P_i \subset N - N_0^i$*, by Lemma 5.4,* $\mathbf{d}(a)$ *contains no elements* $N_0^{i-1} - N_i$*. By definition of* Q_i*, all elements of* $\mathbf{d}(a)$ *are in* N_0^i*. Thus,* $\mathbf{d}(a) \subset N_i$*.* $\square$

Theorem 5.1 uses Lemma 5.4 and says that to evaluate the computation in a node which is at layer $i+1$, either the computation was already evaluated at layer i or the computations needed to evaluate it are in layer i. The significance of this theorem is that since computations proceed unidirectionally from bottom to top, we only need to retain computations at layer i to evaluate all computations at layer $i+1$.

A consequence of Theorem 5.1 is a characterization of the computational resource requirements of the hierarchy if we evaluate computations layer by layer.

Cardinality of Layer The number of elements $|N_i|$ in each layer indicates how much concurrent processing is possible when there are multiple processors. For only a single processor, $|N_i|$ indicates how much memory is needed to store all the partial computations that will be needed for subsequent processing.

Number of Layers The number of layers in the hierarchy is an indication of the minimal amount of time it takes to evaluate all computations in the hierarchy since evaluation of a computation is possible only if all of its inputs have been evaluated.

Thus, to design algorithms which take advantage of the availability of multiple agents, we desire a hierarchy with a large number of elements in each layer. To design algorithms to be run on systems with little memory, we desire a hierarchy with a small number of elements in each layer. To design real-time algorithms, we desire a hierarchy with a small number of layers. The effect on computational resources based on the layer characteristics of the hierarchy can be designed into an algorithm by choosing an appropriate heuristic for graph topology construction which will be discussed in Section 5.5.2.

The definition of layers presented in this section is only one of several ways to define layers which have the layer abstraction property (Theorem 5.1). However, this

definition is perhaps the most natural since primitive computations in $\mathcal{F}$ will usually be chosen to consume about the same amount of computational resources.

In this section, we have not considered what computations we desire the hierarchy to encode. The application will give us a set of computations to encode, and the goal of *graph topology construction* will be to decompose these computations into functional compositions of primitive computations.

5.5 Graph Topology Construction

This section will present methods for constructing the graph topology for a hierarchy given some starting set of functions

$$\{f_i : V^n \to V\}_{i \in I}$$

that is to be encoded. The idea is to express these given computations in terms of functional compositions of a given set of simpler, primitive computations. More formally, we construct the graph topology (N, F, E) part of the hierarchy $H = (N, F, E, U, D, \Upsilon)$. In this section, we will ignore the evaluation algorithm part of the hierarchy which will be presented in Section 5.6

The presentation in this section is divided into two parts. First, Section 5.5.1 will characterize the set of all graph topologies which encode the given set of functions corresponding to the global task. The fact that there exist many graph topologies for any given set of functions implies that there are many algorithms which can evaluate the same functions but have different requirements of computational resources as desired by characteristics **(M1)** and **(M2)**. The enumeration of all of these graph topologies is intractable, so the key idea we employ for constructing the graph topology in a tractable manner is to use *heuristics* to guide the construction. We will present a heuristical algorithm for constructing graph topologies in Section 5.5.2 which is suitable for practical applications. For example, the construction of the graph topology in Section 4.3.3 is an instance of the more general algorithm to be presented in Section 5.5.2. The generalized aspect is that there is a choice of primitive computation for each node added to the hierarchy whereas in Section 4.3.3 the only computation associated with nodes was a two-input AND function.

Finally, Section 5.5.3 will discuss various aspects of the heuristical graph topology construction algorithm, including the time complexity of the enumeration step, computation of the minimal set of partial computations, issues in the threshold and update step, and distributing the construction of the graph topology. This distributed construction of the graph topology will be important for system architectures with distributed autonomous agents in order to address the issue of scalability. Such a distributed construction was presented for the application of target localization on sensor networks in Section 4.6.

The presentation in the subsections of this section is concerned only with the graph topology part of a hierarchy, so that we will denote a hierarchy by

$$H = (N, F, E, \cdot, \cdot, \cdot)$$

to emphasize we are only referring to the graph topology part.

5.5.1 Theory

To discuss hierarchies which encode certain computations, let us define the Φ-computation associated with each node of the hierarchy $H = (N, F, E, \cdot, \cdot, \cdot)$ over n variables $\{x_i\}_{i=1}^n$, each taking value in V.

Definition 5.6 *The* Φ-computation *associated with node $a \in N$, denoted*

$$\Phi_a : V^n \to V$$

is a function of the n variables $\{x_i\}_{i=1}^n$ with range in the set V defined recursively as

$$\Phi_a(x_1, \ldots, x_n) = F_a(\Phi_{b_1}(x_1, \ldots, x_n), \ldots, \Phi_{b_m}(x_1, \ldots, x_n))$$

for nodes $a \in N - N_0$, where $b_1, \ldots, b_m \in \mathbf{d}(a)$ are the m distinct nodes in the decomposition of a, and

$$\Phi_b(x_1, \ldots, x_n) = F_b = x_j$$

for a node $b \in N_0$ which is associated with the variable x_j.

The Φ-computation of a node in the hierarchy is just viewing the computation of the node as a function of all variables $\{x_i\}_{i=1}^n$.

Now, given a set of functions to encode

$$\{f_i : V^n \to V\}_{i \in I} \ ,$$

we can define the $\{f_i\}_{i \in I}$-computable hierarchies.

Definition 5.7 *A hierarchy $H = (N, F, E, \cdot, \cdot, \cdot)$ is said to be* $\{f_i\}_{i \in I}$-computable *if there exists a node $a_i \in N$ for each $i \in I$ such that*

$$\Phi_{a_i} = f_i \ .$$

What we desire from the graph topology construction process is a $\{f_i\}_{i \in I}$-computable hierarchy composed of computations from a given set of primitive computations $\mathcal{F}$ which after choosing an evaluation algorithm will result in a computationally efficient algorithm running on a specified system architecture. How efficient the resulting algorithm is depends on the computational resource costs assigned to the evaluation of the primitive computations in the hierarchy so that the choice of the most efficient $\{f_i\}_{i \in I}$-computable hierarchy is problem specific. However, what is not problem specific is the characterization of the set of all graph topologies composed of primitive computations from $\mathcal{F}$ which are $\{f_i\}_{i \in I}$-computable.

5.5.1.1 Lattice of Hierarchies

We will describe a lattice of hierarchies composed of nodes which are associated with computations from a set of primitive computations $\mathcal{F}$. More precisely, we could call this a lattice of graph topologies; however, it is understood that we only refer to the N, F, and E parts of the hierarchy. Let $H_1 = (N_1, F_1, E_1, \cdot, \cdot, \cdot)$ and $H_2 = (N_2, F_2, E_2, \cdot, \cdot, \cdot)$.

Definition 5.8 *We say that H_1 is equivalent to H_2, written*

$$H_1 \sim H_2$$

if there exists a bijection $g : N_1 \to N_2$ such that

1. $F_1(a) = F_2(g(a))$ *for all* $a \in N_1$ *and*
2. $(a, b) \in E_1 \iff (g(a), g(b)) \in E_2$.

H_1 and H_2 are equivalent if the only difference between the graph topology of the two is a re-labelling of the nodes. Uniqueness in the lattice is defined up to the equivalence relation $\sim$.

Definition 5.9 *We have a partial order $\subset$ on hierarchies where*

$$H_1 \subset H_2$$

if $|N_1| \leq |N_2|$ and there exists an injective function $g : N_1 \to N_2$ such that

1. $F_1(a) = F_2(g(a))$ *for all* $a \in N_1$ *and*
2. $(a, b) \in E_1 \Rightarrow (g(a), g(b)) \in E_2$.

Essentially, when $H_1 \subset H_2$, H_1 is a subhierarchy of H_2. We must now define the meet ($\cap$) and join ($\cup$) operators for the set of hierarchies.

Definition 5.10 *The meet* ($\cap$)

$$H = H_1 \cap H_2$$

is the maximal hierarchy H such that $H \subset H_1$ and $H \subset H_2$. By maximal, we mean that for all other hierarchies H' satisfying the above, $H' \subset H$.

Definition 5.11 *The join* ($\cup$)

$$H = H_1 \cup H_2$$

is the minimal hierarchy H such that $H_1 \subset H$ and $H_2 \subset H$. By minimal, we mean that for all other hierarchies H' satisfying the above, $H \subset H'$.

We will describe a construction of the *maximal hierarchy* composed of primitive computations in $\mathcal{F}$ over n variables $(v_1, \ldots, v_n) \in V^n$ which we denote

$$H_{max}(\mathcal{F}) = (N_{max}, F_{max}, E_{max}, \cdot, \cdot, \cdot)$$

We call this partially defined hierarchy *maximal* because for any other hierarchy $H = (N, F, E, \cdot, \cdot, \cdot)$,

$$H \subset H_{max} \ .$$

By choosing appropriate subhierarchies of the maximal hierarchy, we will be able to characterize the set of all $\{f_i\}_{i \in I}$-computable hierarchies.

The construction is bottom-up and begins with the n input layer nodes of N_0 which are associated with the variables

$$\{x_i\}_{i=1}^n \ .$$

Call these input layer nodes S_0 so that we do not confuse the definition of layers from Section 5.4.2 with the nodes constructed at each iteration of this process. The rest of the construction will be performed inductively for S_i given nodes in S_0^{i-1}.

For each primitive computation $f \in \mathcal{F}$, we create a node for each m length permutation of nodes in S_0^{i-1}, where m is the number of inputs of f and each permutation contains at least one element from S_{i-1}. We associate each of these newly created nodes with the primitive computation f. We require that the permutation contains an element of S_{i-1} so that we do not create redundant nodes. Iterating this step to infinity[5] results in the maximal hierarchy over $\mathcal{F}$. We ignore the evaluation algorithm $(U_{max}, D_{max}, \Upsilon_{max})$ part of the maximal hierarchy because these parts have no effect on characterizing the $\{f_i\}_{i \in I}$-computable hierarchies.

5.5.1.2 Characterization of the $\{f_i\}_{i \in I}$-computable Hierarchies

Now, we can characterize the set of $\{f_i\}_{i \in I}$-computable hierarchies composed of primitive computations $\mathcal{F}$. If it is true that the functions $\{f_i\}_{i \in I}$ of the problem can be expressed by functional compositions of primitive computations in $\mathcal{F}$, then there will exist a node $a \in N_{total}$ with

$$\Phi_a = f_i$$

for each $i \in I$. In fact, there will exist many such nodes in N_{total}. For each f_i, collect the set of nodes in N_{total} with Φ-computation equal to it,

$$A_i = \{a \in N_{max} : \Phi_a = f_i\}$$

[5]We defined the hierarchy with a finite number of nodes, but iterating to infinity will result in a countably infinite number of nodes. For the purposes of this discussion, we can just modify the definition of the hierarchy to allow for a countably infinite number of nodes.

for all $i \in I$. Now, we can characterize the set of all $\{f_i\}_{i \in I}$-computable hierarchies parameterized by elements from the space

$$(a_1, \ldots, a_k) \in A_1 \times \cdots \times A_k$$

where we assume the index set I is $\{1, \ldots, k\}$. The hierarchy corresponding to the "parameter" $(a_1, \ldots, a_k)$ is denoted

$$H_{max}(a_1, \ldots, a_k) \ .$$

$H_{max}(a_1, \ldots, a_k) \subset H_{max}$ is defined to be the subhierarchy $(\tilde{N}, \tilde{F}, \tilde{E}, \cdot, \cdot, \cdot)$ where

$$\tilde{N} = \{a \in N_{max} : a \leq a_i \text{ for some } i \in I\} \ ,$$

$$\tilde{F} : \tilde{N} \to \mathcal{F}$$

defined by

$$\tilde{F}(a) = F_{max}(a)$$

for all $a \in \tilde{N}$, and

$$\tilde{E} = \{(a, b) \in E_{max} : a, b \in \tilde{N}\} \ .$$

The set

$$A_1 \times \cdots \times A_k$$

can be considered parameters which parameterize the set of all $\{f_i\}_{i \in I}$-computable hierarchies composed of primitive computations $\mathcal{F}$.

5.5.1.3 Conclusions

The aim of the graph topology construction is to choose one of the $\{f_i\}_{i \in I}$-computable hierarchies parameterized by elements of $A_1 \times \cdots \times A_k$ which leads to a computationally efficient algorithm running on a specified system architecture. If we had a measure, **cost**, of the eventual computational cost of the algorithm constructed by these hierarchies, then the hierarchy construction process is to solve

$$\min_{(a_1, \ldots, a_k) \in A_1 \times \cdots \times A_k} \mathbf{cost}(H_{max}(a_1, \ldots, a_k)) \ .$$

Enumerating the entire set of $\{f_i\}_{i \in I}$-computable hierarchies is intractable. Thus, some tractable method is needed for picking out the graph topology of a hierarchy which will lead to a computationally efficient algorithm. The main idea will be to construct the graph topology in a bottom-up fashion like in the construction of the maximal hierarchy, except that we will employ heuristics to guide the construction which results in a hierarchy with suitable optimization of computational resources.

5.5.2 Heuristical Construction

Given the set of functions $\{f_i\}_{i\in I}$ to encode and the set of primitive computations $\mathcal{F}$, we will present an algorithm which constructs a $\{f_i\}_{i\in I}$-computable hierarchy. The construction follows the same basic steps as the construction algorithm presented in Section 4.3.3 which proceeds in a bottom-up fashion by enumerating possible nodes to add, keeping track of which nodes are partial computations of each f_i, and using heuristics to prune the set of nodes enumerated at each iteration.

We will define what we mean by a *partial computation* and present the heuristical graph topology construction algorithm for constructing the graph topology of a hierarchy. Specific heuristics for constructing graph topologies with various characteristics have been discussed in Section 4.4.

5.5.2.1 Algorithm

Let us define what we mean by *partial computations* and what it means to keep track of which nodes are partial computations of each f_i of the given set of functions to be encoded by the hierarchy.

Definition 5.12 *We say that a set of computations*

$$\{g_j : V^n \to V\}_{j=1}^m$$

are partial computations *of a computation*

$$f : V^n \to V$$

if there exists some function

$$h : V^m \to V$$

such that

$$f(\underline{(v)}) = h(g_1(\underline{v}), \ldots, g_m(\underline{v}))$$

for all $\underline{v} = (v_1, \ldots, v_n) \in V^n$.

This definition matches our intuition that with further deterministic processing h of the partial computations $\{g_j\}_{j=1}^m$, we can eventually evaluate the desired computation. What will be important for the heuristical graph topology construction algorithm is to compute the *minimal* set of partial computations for each of the functions in $\{f_i\}_{i\in I}$.

Definition 5.13 *Given a set of partial computations* $\{g_j : V^n \to V\}_{j=1}^m$ *for a computation* f*, the* minimal set of partial computations *for* f *is denoted* $\mathbf{minPC}(f, \{g_j\}_{j=1}^m)$ *and defined to be*

$$\mathbf{minPC}(f, \{g_j\}_{j=1}^m) = \arg_{G\in\mathcal{G}} \min |G|$$

where

$$\mathcal{G} = \{G \subset \{g_j\}_{j=1}^m : G \text{ is a set of partial computations of } f\}$$

and $|\cdot|$ *is the cardinality of the set.*

The minimal set of partial computations is not necessarily unique, so we only ask that **minPC** return an arbitrary choice of one of the minimal sets. We leave details of how to construct these minimal set of partial computations for Section 5.5.3.2.

We will now present the basic bottom-up construction of a $\{f_i\}_{i\in I}$-computable hierarchy composed of primitive computations in $\mathcal{F}$ which constructs the graph topology (N, F, E) part of a hierarchy $H = (N, F, E, \cdot, \cdot, \cdot)$. Each function f_i is assumed to be a function of the n variables $\{x_i\}_{i=1}^n$. For each step, we present the general quantities computed along with an example of the formalization of each step. The basic steps of the algorithm to be described in detail are: (1) initialization, (2) enumeration, (3) compute heuristics, (4) threshold and update construction, and (5) stopping condition or iterate.

Basic Heuristical Graph Topology Construction Algorithm

1. **Initialization** Construct the input layer nodes

$$N_0 = \{a_1, \ldots, a_n\}$$

where each node a_k is associated with one of the n variables

$$x_k \ .$$

- $N := N_0$
- $F_{a_k} := x_k$ for $k \in \{1, \ldots, n\}$ where $a_k \in N$
- $E := \emptyset$
- $S_i := < \mathbf{minPC}(f_i, \{\Phi_a\}_{a\in N}) >_N$ for each $i \in I$

N, F, and E are the nodes, primitive computations or variables associated with nodes, and information edges respectively currently in the hierarchy. Each S_i is a subset of nodes in N indicating the nodes which hold the minimal set of partial computations for the computation f_i. Because

$$\mathbf{minPC}(f_i, \{\Phi_a\}_{a\in N})$$

returns a subset of the Φ-computations $\{\Phi_a\}_{a\in N}$, we use the notation $< \cdot >_N$ to return the set of nodes associated with the Φ-computations. That is, for any subset of nodes $S \subset N$,

$$< \{\Phi_a\}_{a\in S} >_N = S \ .$$

Example In Figure 4-12 from Chapter 4, the set of computations $\{f_i\}_{i\in I}$ are the triggers $\{T_i\}_{i=1}^8$, N is the set of measurement token nodes $\{a_k^v\}_{k=1,v=0}^{8,2}$, F_{a_k} are the measurement token functions π_k^v (which we have abstracted to be a set of "variables" in this general discussion), E is empty, and S_i corresponds to the set of nodes connected to each trigger T_i downwards by a dotted line.

2. **Enumeration** Enumerate all possible distinct nodes with associated primitive computations from $\mathcal{F}$ over the set of nodes $\bigcup_{i \in I} S_i$, which is the subset of all nodes N currently in the hierarchy that are partial computations for at least one f_i.

   - $S = \bigcup_{i \in I} S_i$
   - $P :=$ { set of enumerated nodes over S with associated primitive computations in $\mathcal{F}$}

   For each enumerated node $p \in P$, we have an associated primitive computation $f_p \in \mathcal{F}$ as well as the decomposition $\mathbf{d}(p)$ of the set of nodes in S which are the input nodes of p.

   Example In Figure 4-13, the set of enumerated nodes P are the black nodes. For each enumerated node $p \in P$, the primitive computation f_p is a two-input AND function. The decomposition $\mathbf{d}(p)$ is the set of nodes connected to each enumerated node p by a dotted line emanating downwards from p. For example, for the left most enumerated node b_1,

   $$\mathbf{d}(b_1) = \{a_1^1, a_2^1\} \ ,$$

   and for the right most enumerated node b_{14},

   $$\mathbf{d}(b_{14}) = \{a_7^1, a_8^1\} \ .$$

3. **Compute Heuristics** For each computation f_i, calculate the new minimal set of partial computations with the inclusion of the new enumerated nodes from the previous step.

   - $S_i' :=< \mathbf{minPC}(f_i, \{\Phi_a\}_{a \in S \cup P}) >_{S \cup P}$ for each $i \in I$ [6]

   For each node $p \in P$, we calculate a score based on some set of heuristics about the currently constructed hierarchy (N, F, E) and the S_i', which is an indication of what the rest of the hierarchy can be.

   - $\mathbf{score}(p) := \mathbf{heuristic}(p, N, F, E, \{S_i'\}_{i \in I})$ for each $p \in P$

   We have discussed some useful heuristics and how they affect the computational resource requirements in Section 4.4.

   Example In Figure 4-13, the S_i' roughly correspond to the set of nodes which are connected by a dotted edge emanating downwards from each T_i. To illustrate what we mean by "roughly", consider the set of black nodes connected to T_2, which are b_1, b_2, and b_5. It should be clear

[6] The choice of S_i' can be generalized to represent any relationship between the set of computations f_i and the current graph topology with enumerated nodes $\{\Phi_a\}_{a \in S \cup P}$. The example shows a different choice for the S_i'.

that the minimal subset of partial computations which T_2 depends on consists of only two of these nodes. That is, the possible subsets for S_2' are

$$\{b_1, b_2\},\ \{b_1, b_5\},\ \{b_2, b_5\}\ .$$

To be precise, the choice for S_i' in this example was

$$S_i' = \{p \in P : \Phi_p \in \mathbf{minPC}(T_i, \{\Phi_a\}_{a \in S \cup \{p\}})\}\ .$$

That is, S_i' is the subset of enumerated nodes which could be an element of the minimal subset of partial computations for trigger T_i if p is the only enumerated node added to the hierarchy at this iteration.

In the toy example of Section 4.3, the example of the score **heuristic** was a dictionary combination of the commonality heuristic and cost of communication which is a function of the enumerated node p, the current graph topology of the hierarchy $(N, F, E, \cdot, \cdot, \cdot)$, and the relationship of the enumerated node p with the triggers, which is represented in this general formulation by S_i'.

4. **Threshold and Update Construction** Keep the best scoring nodes in P by thresholding the score by some suitably chosen γ. This is the pruning step.

   - $P^\gamma = \{p \in P : \mathbf{score}(p) > \gamma\}$

   Then, we add these nodes to the current hierarchy.

   - $N := N \cup P^\gamma$
   - Extend the domain of F to include P^γ and set

     $$F_p := f_p$$

     for all $p \in P^\gamma$
   - $E := E \cup \{(p, q) : p \in P^\gamma, q \in \mathbf{d}(p)\}$

   Finally, we update the minimal set of partial computations for each function f_i.

   - $S_i := < \mathbf{minPC}(f_i, \{\Phi_a\}_{a \in S \cup P^\gamma}) >_{S \cup P^\gamma}$ for each $i \in I$

   Example Figure 4-14 shows the result of adding a single enumerated node to the current graph topology where, after this first iteration, the resulting quantities above are

$$\begin{aligned} P^\gamma &= \{b_2\} \\ N &= \{a_k^v\}_{k=1,v=0}^{8,2} \cup \{b_2\} \\ F_{b_2} &= AND \\ E &= \{(b_2, a_1^1), (b_2, a_4^2)\} \end{aligned}$$

   Furthermore, the minimal set of partial computations S_i are different only for S_1 and S_2 after this first iteration which are given by

$$\begin{aligned} S_1 &= \{b_2, a_8^1\} \\ S_2 &= \{b_2, a_2^1\}\ . \end{aligned}$$

5. **Stopping Condition or Iterate** If each S_i for $i \in I$ has cardinality 1, then we are done and have constructed a $\{f_i\}_{i \in I}$-computable hierarchy. Otherwise, go back to step 2 and repeat.

   The single node in S_i has Φ-computation equal to the computation f_i we wished to encode. That is, if $S_i = \{a\}$,

   $$f_i = \Phi_a \ .$$

   Then, the result is a graph topology for a hierarchy which is $\{f_i\}_{i \in I}$-computable.

   Example In the bottom graph of Figure 4-15, the stopping condition has been reached because for all $i \in \{1, \ldots, 8\}$,

   $$S_i = \{T_i\} \ .$$

   Note that the "node T_i" has Φ-computation equal to the "trigger T_i" which is exactly the computation we wished to encode in the graph topology of the hierarchy.

The algorithm above can have many variants which include how we enumerate, compute heuristics, and threshold the set of enumerated nodes to keep for the next iteration. The best choice is application dependent; however, the basic steps are the same: enumerate, compute heuristic, and threshold and update the construction.

5.5.3 Discussion

Let us discuss some limitations and generalizations of the heuristical graph topology construction algorithm in this section.

5.5.3.1 Time Complexity of Enumeration Step

Let us derive the time complexity of the enumeration step. S is the set of nodes of the current graph topology of the hierarchy that are partial computations for at least one f_i. Let

$$n = |S| \ .$$

For each primitive computation $f \in \mathcal{F}$, let m_f be the number of inputs of f. Then, the number of nodes which must be enumerated by the enumeration step is given by

$$\begin{aligned} \sum_{f \in \mathcal{F}} \binom{n}{m_f} &\leq \sum_{f \in \mathcal{F}} \binom{n}{m^*} \\ &= |\mathcal{F}| \binom{n}{m^*} \\ &\leq |\mathcal{F}| n^{m^*} \end{aligned}$$

where

$$m^* = \max_{f \in \mathcal{F}} m_f \ .$$

The first inequality follows when $n \geq 2m^* + 1$ which is the regime we are interested in. Thus, asymptotically as n grows, the number of nodes enumerated by the enumeration step is linear in the number of primitive computations in $\mathcal{F}$ and grows with m^* power of n, where m^* is the maximum number of inputs any primitive computation in $\mathcal{F}$ has. For the applications for which the hierarchy is to be applied, n tends to be large so that a complete enumeration is intractable. Randomly enumerating certain nodes may be a better option for this step when n is large.

5.5.3.2 Computing Minimal Set of Partial Computations

Computing heuristics is rather quick because these are functions of the currently constructed hierarchy and the enumerated nodes. However, the real difficult computation is computing **minPC**. From a set of partial computations $\{g_j\}_{j=1}^m$ of f, we must determine a subset of the g_j's which is the minimal subset of partial computations that must be evaluated so that the output of f can be evaluated. A brute force algorithm which constructs a smaller subset of computations (not necessarily the minimal subset) is the following.

Algorithm for Computing Minimal Set of Partial Computations
Given a function

$$f : V^n \to V$$

and a set of partial computations

$$\{g_j : V^n \to V\}_{j=1}^m ,$$

we compute the subset $S \subset \{1, \ldots, m\}$ which corresponds to a subset of partial computations $\{g_j\}_{j \in S}$ of f.

1) $S := \{1, \ldots, m\}$
2) **for each** $k \in S$
3) Test if $\{g_j\}_{g \in S - \{k\}}$ is a set of partial computations for f
4) **if** true
5) **then** S := S - {k}
6) goto step 2
7) **else** continue looping

The difficult computation in the above algorithm is computing the test of whether a subset of computations $\{g_j\}_{j \in S}$ is a set of partial computations for f. Consider the vector function

$$\underline{g} : V^n \to V^{|S|}$$

which, when $S = \{j_1, \ldots, j_l\}$, is defined by

$$\underline{g}(\underline{v}) = \begin{bmatrix} g_{j_1}(\underline{v}) \\ \vdots \\ g_{j_l}(\underline{v}) \end{bmatrix}$$

where we use the shorthand notation $\underline{v} = (v_1, \ldots, v_n) \in V^n$. The condition that the subset of computations are a set of partial computations for f is that for all $v, v' \in f(V^n)$, where $f(V^n)$ is the image of f, if $v \neq v'$ then

$$\underline{g}(f^{-1}(v)) \cap \underline{g}(f^{-1}(v')) = \emptyset \ . \tag{5.2}$$

That is, the images of $\underline{g}$ of the inverse images of f of v and v' are non-overlapping. This means that the output of $\underline{g}$ captures all the information about $\underline{v} \in V^n$ needed to compute f. This test is difficult to compute without having to compute the images and inverse images of functions. By imposing a probability distribution P on V^n, we can extend the condition of Eqn. 5.2 above in terms of a condition on a conditional entropy,

$$H(A|B) = 0 \tag{5.3}$$

where the random variables A and B are defined in terms of a random variable C which has probability distribution P on V^n given by

$$\begin{aligned} A &= f(C) \\ B &= \underline{g}(C) \ . \end{aligned}$$

The interpretation of the condition in Eqn. 5.3 is that the function f is a function of $\underline{g}$ with probability 1 where the probability distribution is P. A nice aspect of the condition in Eqn. 5.3 is that $H(A|B)$, when non-zero, returns a value which can be considered to be a measure of how much f is a function of $\underline{g}$. This measure can be useful if we relax the condition that the subset of functions $\{f_i\}_{i \in I}$ must be encoded exactly in the hierarchy to encoding the subset of functions up to a certain quality.

The condition in Eqn. 5.3 also suggests an approximate test for whether a set of computations $\{g_j\}_{j \in S}$ is a set of partial computations for f. Sample from the distribution P, say n times, and using the empirical distribution over these samples, compute the condition in Eqn. 5.3. This test is only approximate since the only assurance we have is that f is a function of the g_j's for the set of samples.

5.5.3.3 Heuristics, Pruning, and Two-way Construction

In the threshold and update step where a subset of the best scoring enumerated nodes are added to the hierarchy, it is important to select the right threshold for pruning out the set of enumerated nodes. As with all pruning schemes, there is the danger of pruning out the "correct" enumerated nodes which would lead to the most efficient hierarchy. Which enumerated nodes are pruned is dependent on the scores given to them. Thus, the choice of heuristics ultimately determines which enumerated nodes are kept which results in the graph topology of the hierarchy.

In the basic heuristical graph topology construction algorithm, the entire construction proceeds in a bottom-up fashion. We speculate that some top-down refinement

procedure complementing the current bottom-up construction of the graph topology may make the construction of graph topologies tractable. Such a top-down refinement routine is comparable to a backwards filter which smooths estimates from a forward filter. The bottom-up construction of the graph topology is the forward filter, and the top-down procedure could be used to refine the current graph topology.

5.5.3.4 Distributed Construction

The basic algorithm presented in Section 5.5.2 is a global construction in the sense that it constructs a $\{f_i\}_{i\in I}$-computable hierarchy all at once. The problem with this global construction is that as the dimension n of the input space V^n grows, the time complexity to construct the graph topology of the hierarchy grows polynomially with degree m^*, where m^* is the maximum number of inputs for any primitive computation in $\mathcal{F}$. Although we do not care too much about the time complexity of the graph topology construction, we do want the graph topology construction process to finish in a reasonable amount of time.

For applications where the system architecture contains distributed autonomous agents, the computations $\{f_i\}_{i\in I}$ are usually functions of only a small subset of the components of V^n. Thus, it would be desirable to break up the global construction of the graph topology for the set of all computations $\{f_i\}_{i\in I}$ into a set of local constructions on subsets of those computations

$$\{f_i\}_{i\in S_j}$$

where each $S_j \subset I$,

$$\bigcup_{j=1}^{l} S_j = I \ ,$$

and

$$S_j \cap S_{j'} = \emptyset$$

for all $j, j' \in \{1, \ldots, l\}$ with $j \neq j'$. By partitioning the set of computations, we control the time complexity of the graph topology construction. Then, for the subset of computations $\{f_i\}_{i\in S_j}$, a the graph topology for a local hierarchy H_j is constructed. Then, merging these local hierarchies amounts to constructing the join of the local hierarchies,

$$H = \bigcup_{j=1}^{l} H_j \ .$$

The main question that arises from such a scheme is how to construct the graph topology of local hierarchies independently so that merging them together into a global one will not differ too much from the global construction. We suggest a notion of *cooperation* which interpolates between

- no cooperation where the graph topology of local hierarchies are constructed completely oblivious of other local hierarchies and
- full cooperation where the graph topology of local hierarchies are constructed with knowledge of the other local hierarchies.

The idea for incorporating cooperation into the construction of a local hierarchy in Section 4.6 was to include the effect of computations from neighboring local hierarchies when computing heuristics for the newly enumerated nodes. Certain heuristics, like the commonality heuristic, will then be more accurate during the construction of the graph topology for a local hierarchy.

5.6 Evaluation Algorithms

The graph topology construction, which represents the set of functions of the global task in terms of a circuit of primitive computations $\mathcal{F}$, builds the graph topology (N, F, E) part of a hierarchy $H = (N, F, E, U, D, \Upsilon)$. This section concerns itself with choosing a subset of the edges E to be uplinks U and downlinks D, and the specification of the trigger function Υ which defines an *evaluation algorithm* for evaluating computations of the hierarchy. Choosing uplinks, downlinks, and the trigger function is still an off-line process; however, this choice has a real impact on the resulting efficiency of the on-line algorithm for a specified system architecture.

The main idea for choosing the evaluation algorithm (U, D, Υ) part of the hierarchy comes from the observation that not all computations encoded in the hierarchy need to be evaluated for every input $\underline{v} \in V^n$. The choice of uplinks, downlinks, and trigger function will result in a selection mechanism which selects which computations are to be evaluated based on the output of computations which have already been evaluated. Thus, an evaluation algorithm embodies the principle of information-based computation where evaluations are initiated based on the results of previous evaluations.

Section 5.6.1 will describe the *typical evaluation requirements* we make for a hierarchy. We make the connection that the typical evaluation requirement on a hierarchy can be considered to be the partial evaluation of a posterior distribution in standard Bayesian estimation theory so that a tradeoff arises between estimation accuracy and computational complexity. Recall that our idea for a control mechanism that selects which computations to explicitly evaluate is to consider each node of the hierarchy as a computational unit that runs the *canonical message passing algorithm* concurrently. This message passing algorithm is presented in Section 5.6.2. It will be seen that the messages passed between nodes is dependent on the types of edges that are incident to each node. Furthermore, the relationship between the input and output of a computation will be critical to determine which computations should be evaluated and, hence, will drive the choice of uplinks U, downlinks D, and trigger function Υ

so that the evaluation requirement can be satisfied. Note that the choice of (U, D, Υ) parameterizes all evaluation algorithms once the graph topology (N, F, E) has been constructed. The section ends with a characterization of the choices of the evaluation algorithms (U, D, Υ) that lead to *correct evaluations*, which means that the evaluation requirement is satisfied. Section 5.6.3 will then describe the computational resource requirements for different choices of evaluation algorithm (U, D, Υ).

5.6.1 Typical Evaluations

We view inference and estimation as a recognition problem so that the typical choice of primitive computations $\mathcal{F}$ are evidence functions about certain patterns of data. To be more specific, we view each node of a hierarchy as an encoding of some pattern of data so that the associated Φ-computation is an evaluation of the amount of evidence that the input has about the encoded pattern. For example, the set of trigger functions from Section 4.3 can be considered to be the set of patterns of data for which we must evaluate the posterior density of the target location. Let us make more specific the computations of the hierarchy and the typical evaluation requirements.

5.6.1.1 Typical Evaluation Requirements

Definition 5.14 *We define* evidence *to be a value in the range* $[0, 1]$ *where* 1 *means that we have full evidence and* 0 *means we have no evidence.*

For the typical case when all computations are about computing evidence, we will consider the space V to be $[0, 1]$, and for all primitive computations $f \in \mathcal{F}$,

$$f : [0, 1]^m \to [0, 1]$$

where we interpret the range of values $[0, 1]$ as the degree of evidence and where m is the number of inputs to the primitive computation f.

In a $\{f_i\}_{i \in I}$-computable hierarchy, the nodes associated with Φ-computation equal to one of the f_i's are evidence computations about those patterns of data which we are interested in computing. If we wanted to know the evidence for every computation in $\{f_i\}_{i \in I}$, then there is no choice but to evaluate every computation encoded in the hierarchy. However, for practical applications, we do not care what the exact evidence value of f_i is if it is small. Thus, the typical evaluation requirement of the hierarchy is

- Given: a $\{f_i\}_{i \in I}$-computable hierarchy, thresholds $\{\gamma_i\}_{i \in I}$ where each $\gamma_i \in [0, 1]$, and an input $\underline{v} \in V^n$,
- Evaluate all f_i if
$$f_i(\underline{v}) \geq \gamma_i$$
for all $i \in I$.

The idea is that we only care to evaluate those computations f_i for which the evidence in the data is larger than γ_i. Evaluations for which evidence of f_i is smaller than γ_i is considered a waste of computational resources, and the goal is to minimize these evaluations. We consider γ_i to be a parameter which controls the *evaluation accuracy* of the evaluation algorithm. By evaluation accuracy, we mean how precisely we know the evaluation of the computations f_i. Thus, larger γ_i correspond to lower evaluation accuracy, and smaller γ_i correspond to higher evaluation accuracy. Furthermore, we consider more explicit evaluations of computations in the hierarchy to be more computationally complex and less explicit evaluations of computations in the hierarchy to be less computationally complex. The choice of the thresholds $\{\gamma_i\}_{i \in I}$ controls the tradeoff between evaluation accuracy and computational complexity where

1. best evaluation accuracy and most computational complexity is the extreme when all $\gamma_i = 0$ and

2. the worst evaluation accuracy and least computational complexity is the extreme when all $\gamma_i = 1$.

The hierarchy represents a set of computations as a network of simple computations so that a tradeoff between evaluation accuracy and computational complexity can emerge.

5.6.1.2 Connection to Standard Bayesian Estimation

Let us relate the evidence functions and the thresholds $\{\gamma_i\}_{i \in I}$ to the elements of standard Bayesian estimation theory. Evidence is a measure of what we can infer from the data, so that evidence is comparable to the likelihood function. The function associated with the set of evidence functions $\{f_i\}_{i \in I}$ which is comparable to a likelihood function is

$$L : V^n \times I \to [0, 1]$$

where

$$L(\underline{v}, i) = f_i(\underline{v}) \ .$$

We view the space V^n as the measurement space and the index set I as the parameter space. The function $L(\underline{v}, i)$ is comparable to the likelihood of measuring $\underline{v}$ when the parameter generating the measurement is i.

The other half of Bayesian estimation is the incorporation of prior beliefs which is represented as a prior probability distribution. This prior distribution is comparable to some function

$$P : I \to [0, 1] \ .$$

Inference in Bayesian estimation involves the fusion of prior beliefs with the evidence from the data to arrive at a posterior belief using Bayes' rule. For our case, this posterior distribution is comparable to a function

$$\tilde{P} : I \to [0, 1]$$

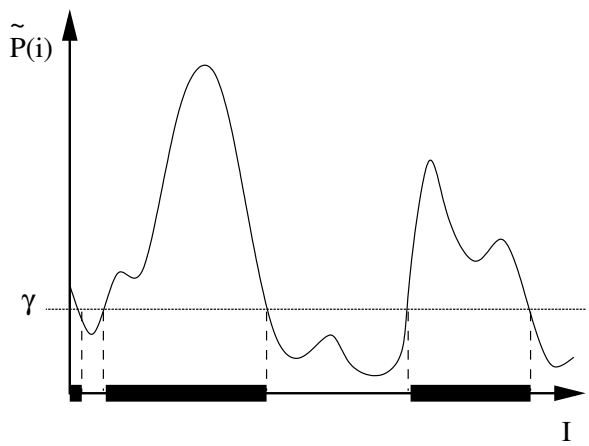

Figure 5-10: Graphical picture of the evaluation requirement. If it happened to be that $\tilde{P}$ is as shown in the figure, then for the given ρ, the evaluation requirement requires that we explicit evaluate $\tilde{P}(i)$ for those i which are highlighted on the x-axis of the graph. Note that $\tilde{P}(i)$ is different for different values of the input $\underline{v} \in V^n$. Thus, the subset of I for which we explicitly evaluate $\tilde{P}$ is different for different $\underline{v}$.

where

$$\tilde{P}(i) = L(\underline{v}, i)P(i)$$

for some given input $\underline{v} \in V^n$. The prior distribution biases the evidence we receive from the data with our prior beliefs which results in the posterior distribution.

What we will show is that the role of the thresholds γ_i in the typical evaluation requirement of the hierarchy is essentially about a partial evaluation of the "posterior" $\tilde{P}(i)$. In particular, our requirement in the partial evaluation of $\tilde{P}(i)$ is that

- given any $\underline{v} \in V^n$, the "prior" P, and some threshold $\rho \in [0, 1]$,
- we must evaluate $\tilde{P}$ with input $i \in I$ if it is true that $\tilde{P}(i) \geq \rho$.

Figure 5-10 shows a picture of the evaluation requirement. The condition

$$\tilde{P}(i) = L(\underline{v}, i)P(i) \geq \rho$$

is equivalent to

$$L(\underline{v}, i) \geq \frac{\rho}{P(i)} \ ,$$

which, since

$$L(\underline{v}, i) = f_i(z)$$

means that the thresholds in the typical evaluation requirements of the hierarchy are

$$\gamma_i = \frac{\rho}{P(i)} \ .$$

Note that when our prior beliefs for some $i \in I$ is high, the threshold γ_i is smaller which means that the computation f_i will be more likely to be evaluated. Thus, prior beliefs have a strong influence on how much and where computational resources will be consumed. The thresholds $\{\gamma_i\}_{i \in I}$ can be considered to be generated by context which has the effect of concentrating computation based on our expectations.

We have stated that the typical computations of the hierarchy are evidence functions, that the typical evaluation requirements ask only for partial evaluations of the hierarchy, and that which computations should be evaluated are determined by prior beliefs about what is in the data through the vehicle of thresholds $\{\gamma_i\}_{i \in I}$. We will demonstrate in the next section, after presenting the canonical message passing algorithm, how to choose uplinks and downlinks so that the evaluation requirements are satisfied.

5.6.2 Canonical Message Passing Algorithm

This section concerns itself with how the computations associated with all nodes of the hierarchy are to be evaluated after the evaluation algorithm (U, D, Υ) are chosen. Once we know how these choices affect which computations in the hierarchy are explicitly evaluated, we can turn to the problem of choosing the uplinks, downlinks, and trigger functions.

We refer to the algorithm run on each of the hierarchy nodes, each viewed as a separate computational unit, as the *canonical message passing algorithm* because a specific evaluation algorithm (U, D, Υ) determines what messages are passed between hierarchy nodes according to the canonical message passing algorithm. Recall that we have two kinds of edges in a hierarchy: the information edges E and the control edges U and D. Since all uplinks U and D lie along an information edge, uplinks always point in the same direction as an information edge, and downlinks always point in the opposite direction as an information edge, we effectively have three kinds of *links* between nodes of the hierarchy which affect the canonical message passing algorithm. Recall the hierarchy is $H = (N, F, E, U, D, \Upsilon)$.

- **Passive Links** The information edges which are neither uplinks or downlinks, which is the set

  $$\{e \in E : e \notin U \text{ and } e \notin D\} ,$$

  are referred to as the passive links because these links will not initiate an evaluation in some other node.

- **Uplinks** The edges from a node which are uplinks initiate the computation of nodes which are part of the spawn. The interpretation of uplinks is to generate hypotheses of what computations to evaluate next based on what computations have already been evaluated.

- **Downlinks** The edge from a node which are downlinks initiate the computation of nodes which are part of the decomposition. The interpretation of downlinks is as queries for extra information to refine the knowledge we have.

We will make some definitions useful for the presentation of the canonical message passing algorithm. Let us introduce some notation first. For any node $a \in N$ and a subset of nodes $A \in N$,

$$(a, A) = \{(a, b) \in E : b \in A\}$$

and

$$(A, a) = \{(b, a) \in E : b \in A\} \ .$$

Definition 5.15 *The* set of initiating links incident to a node $a \in N$ *is denoted* L_a *and defined to be*

$$L_a = ((\mathbf{d}(a), a) \cap U) \cup ((a, \mathbf{s}(a)) \cap D) \ .$$

These are the links *from* other nodes which can initiate the evaluation of the computation encoded in the node a. The next two definitions are about the links *to* other nodes to which a can initiate the computation.

Definition 5.16 *The* set of uplinks incident to a node $a \in N$ *is denoted* U_a *and defined to be*

$$U_a = (a, \mathbf{s}(a)) \cap U \ .$$

Definition 5.17 *The* set of downlinks incident to a node $a \in N$ *is denoted* D_a *and defined to be*

$$D_a = (\mathbf{d}(a), a) \cap D \ .$$

Figure 5-11 shows a picture of the three sets of links incident on a node a as defined above.

5.6.2.1 Algorithm

We are now ready to present the canonical message passing algorithm on a hierarchy $H = (N, F, E, U, D, \Upsilon)$ when the entire hierarchy has been specified. The algorithm assumes each node is an autonomous agent running the following algorithm concurrently. The specific interactions occurring between nodes of the hierarchy is by a message passing scheme based on the edges incident to the node. The messages passed between nodes of the hierarchy are of two types.

1. **Information messages** are the results from evaluating the computation associated with each node. Since the graph topology (N, F, E) part of a hierarchy defines a circuit, these messages only propagate upwards because these messages are the values to be used as inputs for evaluating the computation associated with higher nodes.

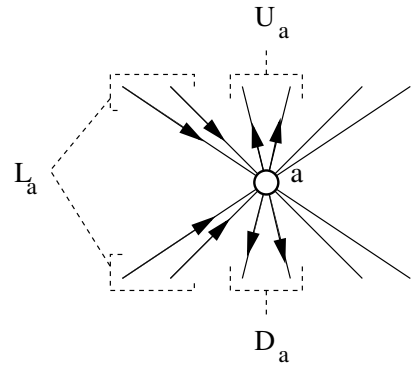

Figure 5-11: The three sets of links incident on a node a. Uplinks and downlinks pointing towards a are the edges in L_a. Uplinks connecting a to nodes above a are the edges in U_a. Downlinks connecting a to nodes below a are the edges in D_a.

2. **Control messages** are messages which control whether a node should initiate the evaluation of its computation or send an information message if queried. These are the messages associated with the evaluation algorithm (U, D, Υ) part of a hierarchy. Note that these messages can move both up and down the hierarchy in order to control which nodes should explicitly evaluate their computation.

Canonical Message Passing Algorithm Every node $a \in N$ of the hierarchy runs the following algorithm concurrently.

1. **Idle State** All nodes of the hierarchy are initially in this state and run the following steps repeatedly.

   (1) **if** $L_a = \emptyset$
   then goto step **Evaluation**
   (2) **if** a receives the control message "initiate computation"
   then goto step **Evaluation**
   (3) **if** a receives the control message "return value"
   then reply with the information message "no value".

2. **Evaluation** In this step, the node a will attempt to evaluate its computation F_a. In order to evaluate F_a, node a must first obtain the results of all evaluations of nodes in its decomposition $\mathbf{d}(a)$. The first step is to consider the downlinks D_a.

   (4) **for** each $(b, a) \in D_a$
   send control message "initiate computation" to node b

Then, the following steps are performed until this node a has retrieved the value from all of its inputs. We keep a value v_b for each element $b \in \mathbf{d}(a)$ in the decomposition of a.

(5) **for** all $b \in \mathbf{d}(a)$
  $v_b :=$ "no value"
(6) **while** any $v_b =$ "no value" for $b \in \mathbf{d}(a)$
  for each $b \in \mathbf{d}(a)$
    if $v_b =$ "no value"
    then send control message "return value" to node b
      assign the reply information message to v_b

Now that all v_b have been assigned a value which is not "no value", we can evaluate the computation in node a.

(7) $v_a := F_a(v_{b_1}, v_{b_2}, \ldots, v_{b_m})$ where $b_1, \ldots, b_m \in \mathbf{d}(a)$
(8) **goto** step **Initiate Other Computations**.

If, at any time during the while loop (6), this node a receives a control message from another node, perform the appropriate action below.

(9) **if** node a receives the control message "return value"
  then reply with the information message "no value"
(10) **if** node a receives the control message "initiate computation"
  then ignore this message

3. **Initiate Other Computations** This node a has evaluated its associated computation F_a, so this node must now initiate computations of other nodes according to the uplinks U_a incident to a.

(11) **for** each $(a, b) \in U_a$
  if $v_a \in \Upsilon_{(a,b)}$
  then send control message "initiate computation" to node b

At this point, since the computation of this node a has been evaluated, as stored in v_a, and the appropriate computations have been initiated, this node idles and only responds to control messages.

(12) **if** node a receives the control message "return value"
  then reply with the information message v_a
(13) **if** node a receives the control message "initiate computation"
  then ignore this message

The canonical message passing algorithm begins with all nodes in the idle state. Then, those nodes a which have no initiating links incident on it,

$$L_a = \emptyset \ ,$$

will proceed to the **Evaluation** step of the above algorithm. The result of these evaluations can initiate evaluations of other nodes, so that evaluations of nodes will propagate through the hierarchy as governed by the choice of uplinks, downlinks, and initiation constraint function.

The uplinks, downlinks, and trigger function is the mechanism which incorporates top-down and bottom-up processing into the hierarchy. At the extremes, a top-down evaluation of a hierarchy $H = (N, F, E, U, D, \Upsilon)$ would be when $U = \emptyset$, $D = E$, and the choice of Υ has no effect. At the other extreme, a bottom-up evaluation of a hierarchy would be when $U = E$, $D = \emptyset$, and $\Upsilon_u = V$ for all $u \in U$. When the hierarchy consists of both uplinks and downlinks and Υ outputs strict subsets of V, the resulting evaluation algorithm exhibits top-down and bottom-up processes interacting with each other which results in a selection mechanism that evaluates only a subset of all computations encoded in the hierarchy depending on the input $\underline{v} \in V^n$.

Note that the choice of uplinks, downlinks, and trigger function determines how the computations of the hierarchy will be evaluated, so that every choice is a particular *evaluation algorithm* on the hierarchy. The set of choices parameterizes all evaluation algorithms; however, we are only interested in those choices which satisfy the typical evaluation requirement which we will characterize in the last part of this section. We call this subset of choices the *correct evaluation algorithms*.

5.6.2.2 Characterization of Correct Evaluation Algorithms

We will now characterize the set of choices of uplinks, downlinks, and trigger function which result in evaluations which satisfy the typical evaluation requirement of a hierarchy. The characterization is based on the analysis of the canonical message passing algorithm of Section 5.6.2. In particular, we will derive the subsets of V^n for which the computation in each node of the hierarchy is "initiated" and "evaluated successfully".

Let us define these terms. By the computation in a node a being *initiated*, we mean that node a has entered step 2 (**Evaluation**) in the canonical message passing algorithm. To say the computation in node a has been *evaluated successfully*, we mean node a has entered step 3 (**Initiate Other Computations**) in the canonical message passing algorithm.

Recall that for a $\{f_i\}_{i \in I}$-computable hierarchy which are evidence functions, all functions are of the form

$$f_i : V^n \to [0, 1] \ ,$$

and the hierarchy H is the tuple $(N, F, E, U, D, \Upsilon)$. We specify the typical evaluation requirement by choosing thresholds $\{\gamma_i\}_{i \in I}$ so that we assume that these thresholds

are given. For each $i \in I$ when $f_i(z) \geq \gamma_i$, the evaluation requirement is that we must evaluate the computation associated with the node $a_i \in N$ that corresponds to f_i. By the node a_i being associated with f_i, we mean that

$$\Phi_{a_i} = f_i \ .$$

To restate the evaluation requirement, if $\Gamma_a \subset V^n$ for each node $a \in N$ of the hierarchy is the subset of inputs for which the computation in node a is evaluated successfully, then the requirement is that

$$f_i^{-1}([\gamma_i, 1]) \subset \Gamma_{a_i}$$

where $a_i \in N$ is the node of the hierarchy which corresponds to the computation f_i. $f_i^{-1}([\gamma_i, 1])$ is the subset of V^n for which the evaluation of f_i results in a value in $[\gamma_i, 1]$.

We will present an iterative algorithm which generates the subsets $\{\Gamma_a\}_{a \in N}$. We keep two sequences of subsets of V^n.

1. Γ_a^t, for $a \in N$ and for $t = 0, 1, \ldots$, corresponds to the subset of V^n by iteration t for which the computation in a is evaluated successfully.

2. τ_a^t, for $a \in N$ and for $t = 0, 1, \ldots$, corresponds to the subset of V^n by iteration t for which the computation in a is initiated, regardless of whether evaluation is successful.

The following construction is an analysis of the canonical message passing algorithm. The results, $\{\Gamma_a\}_{a \in N}$ and $\{\tau_a\}_{a \in N}$ characterize the set of inputs for which the computations in nodes are evaluated successfully and the set of inputs for which the computations in nodes are initiated. Certainly, for all $a \in N$,

$$\Gamma_a \subset \tau_a$$

since we must always initiate the computation before evaluating it.

Initially, we set

$$\Gamma_a^0 = \begin{cases} \emptyset & , \ a \in N - N_0 \\ V & , \ a \in N_0 \end{cases}$$

and

$$\tau_a^0 = \emptyset$$

for all $a \in N$. N_0 refers to the input layer of nodes in the hierarchy which we consider to be always evaluated since their values are set by the input $\underline{v} \in V^n$. We will now describe the computation of $\{\Gamma_a^{t+1}\}_{a \in N}$ and $\{\tau_a^{t+1}\}_{a \in N}$ from $\{\Gamma_a^t\}_{a \in N}$ and $\{\tau_a^t\}_{a \in N}$. For any node $a \in N$, we can divide the set of neighboring nodes of a into six groups.

1. Connected to the output of a by a downlink

$$AD(a) = \{b \in \mathbf{s}(a) : (a, b) \in D\}$$

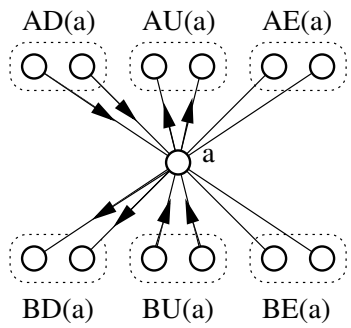

Figure 5-12: The six groups of neighbor nodes of any node a in the hierarchy. Uplinks are denoted by arrows that point upwards along an edge, and downlinks are arrows that point downwards along an edge.

2. Connected to the output of a by an uplink

$$AU(a) = \{b \in \mathbf{s}(a) : (a, b) \in U\}$$

3. Connected to the output of a by a passive link

$$AE(a) = \{b \in \mathbf{s}(a) : (a, b) \in E - (U \cup D)\}$$

4. Connected to the input of a by a downlink

$$BD(a) = \{b \in \mathbf{d}(a) : (b, a) \in D\}$$

5. Connected to the input of a by an uplink

$$BU(a) = \{b \in \mathbf{d}(a) : (b, a) \in U\}$$

6. Connected to the input of a by a passive link

$$BE(a) = \{b \in \mathbf{d}(a) : (b, a) \in E - (U \cup D)\}$$

In the labelling of the six groups of neighboring nodes, the first letter represents whether the nodes in the set are above (A) or below (B) the node a, and the second letter represents whether the edge between a and the set of nodes are an uplink (U), a downlink (D), or a passive link (E). These six groups of neighboring nodes of a is pictured in Figure 5-12. Of these six groups of nodes, the groups which affect when the computation in node a is initiated and evaluated successfully are $AD(a)$, $BU(a)$, $BD(a)$, and $BE(a)$. Let us examine the effect of these groups.

1. The effect of a node $b \in AD(a)$ is to initiate the computation in a whenever the computation in b is initiated. Thus, the subset of V^n for which the computation in a is initiated by nodes in $AD(a)$ is

$$\tau^t_{AD(a)} = \bigcup_{b \in AD(a)} \tau^t_b \ .$$

We take the union because any single node in $AD(a)$ can initiate the computation in a.

2. The effect of a node $b \in BU(a)$ is to initiate the computation in a whenever the computation in b results in a value in $\Upsilon_{(b,a)}$. Thus, the subset of V^n for which the computation in a is initiated by nodes in $BU(a)$ is

$$\tau^t_{BU(a)} = \bigcup_{b \in BU(a)} (\Phi_b^{-1}(\Upsilon_{(b,a)}) \cap \Gamma^t_b) \ .$$

The Φ part of the expression on the righthand side is the subset of V^n for which the computation in b returns a value in $\Upsilon_{(b,a)}$. This is intersected with Γ^t_b, the subset of V^n for which the computation in b is evaluated successfully. We take the union because any single node in $BU(a)$ can initiate the computation in a.

3. There is one special condition when $AD(a) \cup BU(a) = \emptyset$, which is exactly when $L_a = \emptyset$ (See Def. 5.15). This means that the computation in node a is always initiated. So, when $AD(a) \cup BU(a) = \emptyset$, set

$$\tau^t_{special} = V$$

otherwise if $AD(a) \cup BU(a) \neq \emptyset$, set

$$\tau^t_{special} = \emptyset \ .$$

4. Let $B(a) = BU(a) \cup BD(a) \cup BE(a)$. The effect of a node $b \in B(a)$ in regards to successful evaluation is to prevent successful evaluation of the computation in a if the computation in b is not successfully evaluated. Thus, the subset of V^n for which there is the possibility of successful evaluation in a is

$$\Gamma^t_{B(a)} = \bigcap_{b \in B(a)} \Gamma^t_b \ .$$

We take the intersection because all nodes in $B(a)$ must have evaluated successfully in order for the computation in node a to evaluate successfully.

Putting the pieces together, the subset of V^n corresponding to initiation of the computation in a is

$$\tau^{t+1}_a = \tau^t_{AD(a)} \cup \tau^t_{BU(a)} \cup \tau^t_{special} \ .$$

The subset of V^n corresponding to the successful evaluation of the computation in a is updated by

$$\Gamma_a^{t+1} = \begin{cases} \Gamma_a^t & , \ a \in N_0 \\ \tau_a^{t+1} \cap \Gamma_{B(a)}^t & , \ a \in N - N_0 \end{cases} .$$

We stop this iterative procedure when

$$\tau_a^t = \tau_a^{t+1}$$

$$\Gamma_a^t = \Gamma_a^{t+1}$$

for all $a \in N$ and refer to the final values as $\{\Gamma_a\}_{a \in N}$ and $\{\tau_a\}_{a \in N}$. If it is true that

$$\Gamma_{a_i} \supset f_i^{-1}([\gamma_i, 1])$$

for all $i \in I$, where a_i is the node associated with the computation f_i, then the evaluation requirement has been satisfied by this choice of uplinks U, downlinks D, and trigger function Υ.

Note that $\Gamma_a^{(U,D,\Upsilon)}$ is a function of the choice of uplinks U, downlinks D, and trigger function Υ. Let us denote by $\mathcal{S}$ the set of choices of uplinks, downlinks, and trigger function which generates a correct evaluation algorithm. That is, for all $(U, D, \Upsilon) \in \mathcal{S}$,

$$\Gamma_{a_i}^{(U,D,\Psi)} \supset f_i^{-1}([\gamma_i, 1]) \text{ for all } i \in I$$

where $a_i \in N$ is the node associated with f_i and γ_i is the given threshold in the typical evaluation requirement. Thus, $\mathcal{S}$ characterizes the set of all correct evaluation algorithms.

The characterization presented above is a bit unsatisfying because we must analyze each specific choice of uplinks, downlinks, and initiation constraint function in order to determine whether the typical evaluation requirement is satisfied. The difficulty in characterization is due to the passive links in the hierarchy because these can make the initiation subset τ_a differ from the evaluation subset Γ_a.

Because it is possible for τ_a and Γ_a to be strict subsets of the input space Z, we can now see that the input drives what computations are evaluated. This is a desirable characteristic because the computations evaluated will eventually be the computations of the on-line algorithm running on a specified architecture. The possibility that only a small set of computations encoded in the hierarchy are explicitly evaluated for any given input is the reason why the on-line algorithm can be made efficient.

5.6.3 Evaluation Resources

We can divide the cost of computational resource usage during evaluation of the hierarchy into two groups:

1. costs associated with nodes of the hierarchy, which include resources like time, memory, and energy,

2. and costs associated with the edges of the hierarchy, which include resources like time, communication, and energy.

For the costs associated with nodes, all computations of the hierarchy are associated with some cost of computational resources. Say that the evaluation of the computation in node $a \in N$ admits a cost of $\mathbf{cost}(a)$ where

$$\mathbf{cost} : N \to [0, \infty) \ .$$

If we are given a probability distribution P on V^n, then the expression for the average cost of evaluation is

$$\sum_{a \in N} \mathbf{cost}(a) \cdot P(\Gamma_a)$$

where $\Gamma_a \subset V^n$ is the set of inputs for which the computation in a is evaluated successfully. To choose an evaluation algorithm, we wish to solve the minimization problem

$$\min_{(U,D,\Upsilon) \in \mathcal{S}} \sum_{a \in N} \mathbf{cost}(a) \cdot P(\Gamma_a^{(U,D,\Upsilon)})$$

where $\mathcal{S}$ is the set of correct evaluation algorithms as defined in the end of Section 5.6.2.2. The particular choice of the cost depends on the application.

When we are dealing with system architectures with distributed autonomous agents, it is important to consider costs associated with edges of the hierarchy. The idea is that every node of the hierarchy must be assigned to some agent so that evaluation of the computation is possible. In order to evaluate a computation of any node $a \in N$, the results of the evaluations in the set of input nodes $\mathbf{d}(a)$ must be available to the agent which is responsible for a. To make the inputs available, the agent responsible for the nodes $\mathbf{d}(a)$ must communicate the results of their evaluation to a. Thus, the side effect of assigning agents to nodes is the necessity of communication which is a resource we would like to minimize. Again, the particular choice of the cost of communication is application dependent. In Section 4.3.4 of the toy example, we described how to choose uplinks, downlinks, and trigger sets so that the cost of communication is minimized.

5.7 Conversion to Rules

This section deals with the conversion of the chosen evaluation algorithm into a set of rules from which an algorithm on a specific system architecture can be generated. The generic aspects of converting an evaluation algorithm on a hierarchy into a set of rules is to

1. choose an agent assignment which specifies which agents are responsible for computations in the hierarchy and

2. determine what information needs to be passed between the agents as a result of the agent assignment.

By choosing an agent assignment, the result is that some of the messages passed between nodes of the hierarchy in the canonical message passing algorithm correspond to real physical communications between agents. Thus, this conversion process results in a set of rules for each agent to evaluate computations of nodes assigned to it and send or query for results from other agents. If we are to view the hierarchy as source code for some global program, the rules generated by this conversion process is about assigning parts of this global program to different agents along with the necessary communications that must occur between these agents if the global program is to be computed. The result is a distributed implementation of the global program.

These rules can then be converted into an on-line distributed algorithm using the basic operations of the system architecture. Let us discuss the assignment of hierarchy nodes to agents and the rules that are generated.

Definition 5.18 *Given a set of agents P, an* agent assignment *for a hierarchy $H = (N, F, E, U, D, \Upsilon)$ is a function*

$$\mathcal{A} : N \to P \ .$$

Depending on the problem, there may be a constraint on the set of feasible agent assignments which we will denote by $\mathbf{A}$. There are considerations of the amount of communications involved in the choice of $\mathcal{A} \in \mathbf{A}$ which we briefly mentioned in the end of Section 5.6.3 and described in detail in Section 4.3.4 of the toy example. What we wish to describe now is the set of rules generated by the hierarchy and the agent assignment.

Recall the notation of the elements of the hierarchy $H = (N, F, E, U, D, \Upsilon)$. Each agent $p \in P$ has associated with it some memory M^p_{eval} which stores subsets of

$$N \times V$$

which are hierarchy node/computation value pairs which have already been evaluated. For example, $(a, v) \in M^p_{eval}$ indicates that agent p has stored in its memory that the result of evaluating the computation of node a is v. There is also some memory $M^p_{pending}$ which stores subsets of N which are nodes whose computation needs to be evaluated. Each node $a \in N$ generates the following rules for its assigned agent $\mathcal{A}(a)$. Recall the definitions of the neighboring nodes of a in six groups: $AD(a)$, $AU(a)$, $AE(a)$, $BD(a)$, $BU(a)$, and $BE(a)$.

Condition If $a \in M^{\mathcal{A}(a)}_{pending}$,

Action Then if the results of the inputs to a are known by this agent $\mathcal{A}(a)$, meaning

$$\text{for all } b \in \mathbf{d}(a), (b, v_b) \in M_{eval}^{\mathcal{A}(a)}$$

perform the following steps.

- *(R1)* Evaluate the computation encoded by a

$$v_a := F_a(v_{b_1}, \ldots, v_{b_m}) \ ,$$

  where $b_1, \ldots, b_m \in \mathbf{d}(a)$. Note that $v_a = \Phi_a(z)$.
- *(R2)* Record v_a to memory by adding (a, v_a) to $M_{eval}^{\mathcal{A}(a)}$.
- *(R3)* Remove a from the pending set of evaluations $M_{pending}^{\mathcal{A}(a)}$.
- *(R4)* Send an "Add to Pending" message to agents assigned to nodes in the spawn $\mathbf{s}(a)$ of a, if appropriate, in order to add those nodes to the pending evaluations. That is, for each $c \in AU(a)$, if

$$v_a \in \Upsilon_{(a,c)} \ ,$$

  then send to agent $\mathcal{A}(c)$, if different from $\mathcal{A}(a)$, the message "Add to Pending: c" and "Add to Eval: (a, v_a)". If $\mathcal{A}(c) = \mathcal{A}(a)$, then add c to $M_{pending}^{\mathcal{A}(a)}$.

Otherwise if the inputs to a are not currently in the agent memory $M_{eval}^{\mathcal{A}(a)}$, agent $\mathcal{A}(a)$ performs the following steps.

- *(R5)* For each node $b \in BE(a)$ with $(b, v_b) \notin M_{eval}^{\mathcal{A}(a)}$, send to agent $\mathcal{A}(b)$ the message "Query for b".
- *(R6)* For each node $b \in BD(a)$ with $(b, v_b) \notin M_{eval}^{\mathcal{A}(a)}$, send to agents $\mathcal{A}(b)$, if different from $\mathcal{A}(a)$, the message "Initiate computation for b". If $\mathcal{A}(b) = \mathcal{A}(a)$, then add b to $M_{pending}^{\mathcal{A}(a)}$.

Furthermore, there are rules for responding to received messages. These response rules generated by node a are the following.

Condition If "Add to Pending: b" message received,

Action Add b to $M_{pending}^{\mathcal{A}(a)}$.

Condition If "Query for b" message received,

Action If there exists a pair $(b, v_b) \in M_{eval}^{\mathcal{A}(a)}$ in memory, then respond back with message "Add to Eval: (b, v_b)". Otherwise, do nothing.

Condition If "Initiate computation for b" message received,

Action Add b to $M^{\mathcal{A}(a)}_{pending}$. If b is ever evaluated, then a response message is generated "Add to Eval: (b, v_b)".

Condition If "Add to Eval: (b, v_b)" message received,

Action Add (b, v_b) to $M^{\mathcal{A}(a)}_{eval}$.

The set of rules generated for any agents $p \in P$ essentially implements a message passing scheme between agents so that the evaluation of the computations in nodes $\mathcal{A}^{-1}(p)$ which are assigned to p are initiated as dictated by the choice of uplinks, downlinks, and trigger function.

Let us discuss the various parts of the first rule above for an agent $p \in P$. M^p_{eval} is the local state of agent p which is essentially the evaluations of computations that are known by agent p. $M^p_{pending}$ is the set of computations that should be evaluated by the agent and represents the computations that have been initiated for evaluation. For each computation $a \in M^p_{pending}$ which is to be evaluated in agent p, we first check whether the inputs of a are known by agent p. If all inputs are known, then the computation of a is evaluated *(R1)*, stored into agent p's memory *(R2)*, removed from the pending set of evaluations *(R3)*, and finally, those agents responsible for computations in the spawn of a which are connected to a by an uplink are sent an appropriate message to initiate the appropriate computation. For the case when all inputs to a are not known by agent p, agent p either query for information *(R5)* or initiates the appropriate computations *(R6)*. Note that those nodes whose computations are queried are connected to a by a passive link and those nodes whose computations are initiated are connected to a by a downlink.

Let us interpret the top-down and bottom-up aspects of the generated rules. Top-down processing is represented in $(R5)$ and $(R6)$ which either queries for or initiates evaluation of lower layer computations. Bottom-up processing is represented in $(R4)$ which prompts computation of higher layer computations based on the evaluation result of this computation. Note that $(R6)$ of the rule is generated by downlinks and $(R4)$ is generated by uplinks which is how we have incorporated both top-down and bottom-up processing into the hierarchy. The important part of the two processes working together is to act as a selection mechanism so that an agent will only evaluate a subset of the computations assigned to it depending on the inputs to the system.

Finally, the rules generated for each agent need to be translated into an algorithm on a specific system architecture. For example, the messages passed between agents need to be converted into real transmissions on the communication devices of the agents. Furthermore, in the general rules presented above, the information in the messages that are passed between agents consist of the label for the hierarchy nodes and the results of the evaluation of those nodes. Thus, a representation for the nodes of the hierarchy also needs to be chosen for the conversion to an algorithm on the specific system architecture.

For example, Section 4.3.5 of the toy example showed how to convert the evaluation algorithm of the hierarchy into a set of communication rules for each agent. The representation of the hierarchy node was chosen to be the local knowledge state that it represented. Thus, the information passed around the sensor network were local knowledge states.

5.8 Summary

We have presented the hierarchy which is to be used as a meta-representation for constructing efficient algorithms for information processing tasks on various system architectures. The methodology for using the hierarchy as a meta-representation proceeds in the three steps as described at the beginning of this chapter. We will now refine those descriptions with the formalism that has been presented in this chapter.

From the specific application, a set of computations $\{f_i\}_{i \in I}$ must be generated whose evaluation solves the problem for the application. This set of computations is the starting point for using the hierarchy as a meta-representation for constructing computationally efficient algorithms.

1. Given a set of computations $\{f_i\}_{i \in I}$ for a problem, we select an appropriate heuristic from which a $\{f_i\}_{i \in I}$-computable hierarchy composed of primitive computations $\mathcal{F}$ can be constructed by the heuristical graph topology construction algorithm. The heuristics should be chosen so that the resulting graph topology will eventually result in an efficient evaluation algorithm. This part constructs the N, F, and E parts of the hierarchy and is the most computationally complex part of this entire process.

2. Next, we pick the thresholds $\{\gamma_i\}_{i \in I}$ which defines the typical evaluation requirement for the hierarchy. Then, by by choosing a cost criterion to measure the computational complexity of an evaluation algorithm, we can generate the uplinks U, downlinks D, and trigger function Υ which leads to correct evaluation and which minimizes the computational resource usage. Note that this is the part where the selection mechanism emerges.

3. Finally, we select a cost criterion for the set of possible agent assignments and find a near optimal choice. Then, we can convert the evaluation algorithm into a set of rules from which an algorithm to be run on a specific system architecture can be generated.

The hierarchy representation which can be thought of as a feedforward network with a modified evaluation scheme is used as a meta-representation for designing efficient algorithms on specific system architectures. It provides a systematic way of designing algorithms which minimize computational resource usage by embodying the principle of information-based computation.

The use of the hierarchy is intended to be automated where the settings that the user can adjust in this whole process are

1. the heuristics for hierarchy construction,
2. the cost criterion for choosing an evaluation algorithm, and
3. the cost criterion for choosing an agent assignment.

By choosing these settings appropriately, we can construct algorithms which range in parallel to sequential processing, communication cost, energy expended, memory requirements, and latency in time.

We have also speculated that algorithms which incorporate both top-down and bottom-up processing elements can be constructed for layered information processing architectures with the hierarchy. Furthermore, the conversion of an evaluation algorithm into a set of rules for several processors is the synthesis of local algorithms that exhibit certain desired global behavior for system architectures with distributed autonomous agents. The hierarchy allows us to design algorithms for these system architectures in a systematic way from which we can control the computational resource requirements of these algorithms.

Chapter 6

Conclusions and Future Work

Summary

In this thesis, we have presented a conceptual methodology for designing an efficient inference system for system architectures with possibly distributed autonomous agents and an application of the methodology to battlefield awareness with sensor networks. The main conceptual ideas for designing a computationally efficient inference system are embodied in the *layered information processing architecture* and include

1. taking a holistic view of the entire inference system and organizing the individual information processing tasks and their dependencies,

2. exploiting the information extracted from the various information processing tasks to facilitate the overall computation,

3. employing triggers to initiate algorithms implementing information processing tasks on-demand to expend computational resources only when necessary,

4. computing estimates of reasonable accuracy rather than optimal estimates to limit computational complexity, and

5. the integration of top-down and bottom-up processing.

Only after the individual information processing tasks and interactions between them have been designed and organized into a layered information processing architecture, which is a *global view* of the entire inference system, do we consider the problem of generating distributed algorithms for each individual information processing task. We introduced a meta-representation called the *hierarchy* which is a computation and communication structure from which computationally efficient distributed algorithms can be generated. The main conceptual ideas embodied by the hierarchy include the conceptual ideas listed above as well as the following.

1. A compositional view is taken of all information processing tasks, represented abstractly as a global function evaluation in terms of a family of primitive computations, so that a global task can be decomposed into subtasks.

2. By assigning subtasks to different agents and optimizing this agent assignment, we were able to generate distributed algorithms which were adapted to the constraints of the system architecture.

This methodology of using the layered information processing architecture and the hierarchy was applied to the application of sensor networks for battlefield awareness to demonstrate that the methodology can successfully generate computationally efficient distributed algorithms for an overall inference system. The incorporation of information from a variety of sensing modalities has been demonstrated by the ability to incorporate information from intensity, frequency bandlimited intensity, TDOA, and magnetometer sensors. Furthermore, we have demonstrated that the construction of the hierarchy and, thus, the generation of efficient distributed algorithms can be automated given only a description of the global task to be performed and the constraints on the system architecture.

Conclusions

Our conclusion is that the proposed framework of designing a layered information processing architecture and using the hierarchy to distribute algorithms can be applied to a broad range of applications for inference systems which extract a variety of information and are to be run on a variety of system architectures, including those with distributed autonomous agents. In Chapter 5, we briefly described in Section 5.2 the potential application of the hierarchy to construct computationally efficient algorithms for the problem of object recognition in computer vision based on gray scale images.

The broad applicability of the hierarchy is due to it being a representation of some function evaluation because all computations can be considered a function evaluation at an appropriate abstract level. For example, even the layered information processing architecture can be considered to be a function evaluation

$$H_t = \mathbf{T}(H_{t-1}, z(t)) \ ,$$

as described in Section 3.3.1, which updates the hypothesis of tracks from the previous time $t-1$ to the current time t based on the current measurements $z(t)$ observed by the sensor network. Thus, we consider the hierarchy to be the key contribution of this thesis because it is a representation of any function which embodies all the conceptual ideas we have emphasized in this thesis. However, there are several extensions to the current definition of the hierarchy which we will motivate by discussing some possible future applications of the hierarchy for other basic functions to be performed by a sensor network.

Future Work

Future work specific to sensor networks include the extension of the inference system to handle other basic functions to be performed on the sensor network such as self-configuration, other information transfer functions such as dissemination and retrieval, and network monitoring as discussed in Section 1.2. These other basic functions can be abstractly considered to be some global function evaluation by the sensor network for which a distributed algorithm can be generated by the hierarchy. In particular, just as the hierarchy was used to generate an information collection mechanism for collecting the appropriate sensor information to compute an estimate of reasonable quality, it seems the hierarchy can be used to generate routing protocols for the problem of efficiently retrieving information about all tracks which are represented distributedly in the sensor network. The main idea would be to determine a global function evaluation which corresponds to the routing of all tracks to some designated receiver, and then, an appropriate hierarchy can be constructed which is a meta-representation for a particular routing protocol. Thus, the hierarchy would serve as a representation which *indexes information* stored by the sensor network so that retrieval of the desired information is fast and efficient. Note, however, that the set of track holders changes over time, so that the hierarchy must be mutable to reflect this dynamic behavior.

Other extensions, which we have mentioned in Section 4.3.6, in generating DCA's for target localization is the possibility of a dynamically evolving sensor network topology. If sensor nodes fail or sensor nodes can move, then it will be necessary to update the DCA's for target localization to take these changes into account.

Currently, the hierarchy is a static representation in the sense that the global function evaluation must first be completely specified before a hierarchy is constructed. Furthermore, since the construction of the graph topology is bottom-up, the set of input variables to the global function must be fixed before a hierarchy can be constructed. Future work on the hierarchy will be to relax the knowledge of the global function and the fixed nature of the input variables. The future work specific for sensor networks suggest two broad categories of extending the hierarchy which relax knowledge of the global function and the fixed nature of the input variables.

1. **Adaptation to a Changing Global Function** If the global function that the hierarchy is to represent changes over time, a better solution than reconstructing a new hierarchy each time the global function changes is to incrementally update the hierarchy representation suitably. Since there is an underlying circuit part of the hierarchy, a change in the function represented at the top of the hierarchy would constitute a change in the graph topology. If the hierarchy can be updated properly and since a hierarchy is a meta-representation for a distributed algorithm, the result would be distributed autonomous agents which update their local algorithms to adapt to the changing global task. Note that this ability would allow us to deal with a dynamically evolving sensor network

arising from node failures and moving nodes. Furthermore, the problem of retrieving information about tracks distributed throughout the sensor network can then be solved with the hierarchy. In Section 5.5.3.3, we suggested a top-down refinement procedure as well as the bottom-up construction of the graph topology of a hierarchy which may be suitable for updating hierarchies to match an incrementally changing global function.

2. **Learning the Global Function** If the global function is only known partially or can be learned from training data as in supervised learning, then a significant extension of the hierarchy would be to construct hierarchies based on this partial knowledge of the global function. As more training data is acquired, the idea is to update the hierarchy so that the function represented converges to the true global function. Recalling that the hierarchy is a meta-representation for a distributed algorithm, we would then have the possibility of learning a global function with distributed autonomous agents. In particular, in Section 5.5.3.2 we suggested the use of the conditional entropy for constructing a hierarchy when a function is partially known.

With a learning component added to the hierarchy and the possibility of directly adapting the hierarchy to reflect changes of the global function, the hierarchy can be viewed as a sophisticated knowledge representation structure which can learn from observations, adapt to changing goals of the system, and efficiently retrieve information stored in it. We have shown how the hierarchy can be automated to generate efficient distributed algorithms for target localization adapted to the particular topology of an ad hoc sensor network by deriving an appropriate global function evaluation. In general, by specifying an appropriate abstract global function to evaluate, we believe that we can automate the use of the hierarchy to generate efficient algorithms for any given system architecture. Furthermore, we believe that the automation of learning and adaptation in the hierarchy is possible. Therefore, we believe that the hierarchy lays a strong foundation towards the goal of constructing complex inference systems with intelligent behavior.

Appendix A

Information-Driven Sensor Querying (IDSQ)

We present a highly constrained algorithm originally presented in [7] for collecting appropriate data to a single sensor for estimation tasks in sensor networks. The constraint imposed by this algorithm is to limit all communications to a single sensor node, referred to as the leader node. The IDSQ algorithm demonstrates how a sensor node can use its local information to determine what data to collect in an intelligent fashion. This appendix is divided into two parts. A.1 presents the basic IDSQ algorithm which is applicable for target localization. A.2 presents an extension of the IDSQ algorithm which is applicable for target tracking.

A.1 Basic IDSQ Algorithm

A.1.1 Overview

The sensor network consists of n nodes which we label with an integer from the set $\{1, \ldots, n\}$. To account for the possibility of nodes having different sensing modalities, we will refer to the measurement space of sensor node i by $\mathcal{Z}_i$. The target parameters we wish to estimate are elements of a given parameter space Θ. We are also given a model of the relationship between the target parameter and sensor node measurements.

We assume that an initial configuration phase has taken place so that all nodes have basic knowledge about each other. In particular, this knowledge is the model for each node i. We also choose a leader node, labelled by $l \in \{1, \ldots, n\}$, in the network with the special property that only this node may make estimates and collect data. How to choose the leader node will be important for a real implementation of the IDSQ algorithm on sensor networks, but we will ignore this issue for the present discussion.

Let $\mathcal{B}$ be the set of possible local knowledge states a node can be in as defined in

Definition 2.1. For the present discussion, we will consider the local knowledge states to be the set of probability distributions on the parameter space Θ because this is the information in the local knowledge state that the IDSQ algorithm uses to determine what data to collect next. The main idea in the IDSQ algorithm is to incrementally incorporate measurements into the local knowledge state by greedily choosing the next "best" sensor node measurement according to some measure of expected information content. Figure A-1 shows a block diagram of the IDSQ algorithm running in the leader node l.

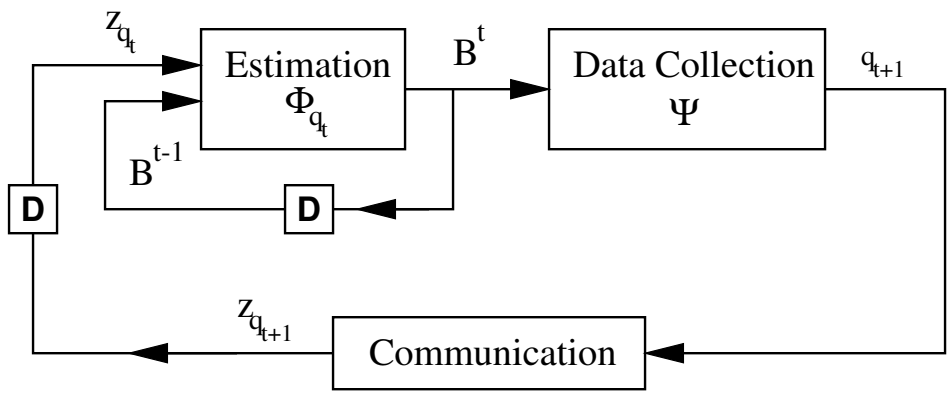

Figure A-1: The block diagram of the IDSQ algorithm, which is run in the leader node only, shows the feedback loop between the estimation process and the data collection process. The boxes labelled with "D" are delay elements.

The algorithm works by first updating the local knowledge state $B^{t-1} \in \mathcal{B}$ from the previous iteration by incorporating the new measurement $z_{q_t} \in \mathcal{Z}_{q_t}$ returned by the most recent communication. Formally, the estimation block updates the local knowledge state at iteration t by

$$B^t = \Phi_{q_t}(B^{t-1}, z_{q_t})$$

according to the local knowledge update function

$$\Phi_j : \mathcal{B} \times \mathcal{Z}_j \to \mathcal{B}$$

where $j \in \{1, \ldots, n\}$. Then, the local knowledge state B^t is fed into the data collection block which determines the next sensor $q_{t+1} \in \{1, \ldots, n\}$ to query. Formally, the data collection block computes the next sensor to query by

$$q_{t+1} = \Psi(B^t)$$

according to the querying function

$$\Psi : \mathcal{B} \to \{1, \ldots, n\} \ .$$

The leader node then queries sensor node q_{t+1} for its observed sensor measurement and awaits a reply message. Once the appropriate communications have taken place, the leader node has access to the new measurement $z_{q_{t+1}} \in \mathcal{Z}_{q_{t+1}}$ which is fed into the estimation block again to update the local knowledge state for iteration $t+1$. This process iterates until the local knowledge state is considered to be of good enough quality. That is, if the local knowledge state B^t is an element of some set, $STOP \subset \mathcal{B}$, of stopping knowledge states.

To summarize the above discussion, the IDSQ algorithm is as follows.

- Algorithm: Information-Driven Sensor Querying (IDSQ)
- Given:
  1. $l \in \{1, \ldots, n\}$ - leader node
  2. $B_{initial}$ - initial local knowledge state of the leader
  3. $\{\Phi_j\}_{j=1}^n$ - set of local knowledge update functions
  4. Ψ - querying function
  5. $STOP \subset \mathcal{B}$ - subset of stopping knowledge states
- Return: $B^T \in \mathcal{B}$ - the final knowledge state in l at final iteration T
- Implementation:
  1. *(Initialization)*
     $t := 0$
     $B^{-1} := B_{initial}$
     $q_0 := l$
  2. *(Estimation)*
     $B^t := \Phi_{q_t}(B^{t-1}, z_{q_t})$
  3. *(Stopping Condition)*
     if $B^t \in STOP$
        goto step 6
     else
        goto step 4
  4. *(Data Collection)*
     $q_{t+1} := \Psi(B^t)$
  5. *(Communication)*
     Query sensor node q_{t+1} for its measurement $z_{q_{t+1}}$
     $t := t+1$
     goto step 2

6. *(Termination)*
   $T := t$
   return B^T

Let us discuss in detail the estimation process and data collection process corresponding to the functions $\{\Phi_j\}_{j=1}^n$ and Ψ respectively.

A.1.2 Estimation

The estimation block computes the updated local knowledge state

$$B^t = \Phi_q(B^{t-1}, z_q)$$

based on the previous knowledge state and the new queried measurement z_q. Since the local knowledge states B^t are probability distributions on Θ, the local knowledge update function Φ_q can be derived from Bayes' rule. We will present an example of a typical local knowledge update function Φ_q.

In this example, the set of local knowledge states $\mathcal{B}$ is the set of probability distributions on Θ. We are given a prior distribution on Θ

$$p_{initial}(\theta)$$

which we referred to as $B_{initial}$ in the algorithm. Furthermore, the model between target parameter θ and measurement z_i of sensor node i will be given by a likelihood function

$$p_i(z_i|\theta) \ .$$

We assume that the sensor measurements are conditionally independent given the target parameter so that the combined likelihood of all the sensor node measurements is the product of the individual likelihoods

$$p_{1,\ldots,n}(z_1, \ldots, z_n|\theta) = \prod_{i=1}^{n} p_i(z_i|\theta) \ .$$

The previous local knowledge state B^{t-1} was constructed from some subset $U \subset \{1, \ldots, n\}$ of sensor measurements, and is given by

$$B^{t-1}(\theta) = p(\theta|\{z_i\}_{i\in U})$$

which is the posterior distribution of θ given the measurements $\{z_i\}_{i\in U}$. When a new measurement z_q arrives, the local knowledge update function needs to compute the new local knowledge state

$$B^t(\theta) = p(\theta|\{z_i\}_{i\in U} \cup \{z_q\})$$

which is the previous posterior distribution conditioned on z_q as well. Due to the conditional independence of sensor measurements, an application of Bayes' rule results in

$$B^t(\theta) = \frac{p_q(z_q|\theta)B^{t-1}(\theta)}{\int_\Theta p_q(z_q|\theta')B^{t-1}(\theta')d\theta'} \ .$$

Thus, the local knowledge update function is

$$\Phi_q(B(\theta), z) = \frac{p_q(z|\theta)B(\theta)}{\int_\Theta p_q(z|\theta')B(\theta')d\theta'} \ . \tag{A.1}$$

This equation is the product of two functions of θ normalized to integrate to one.

A.1.3 Data Collection

The data collection block computes the next sensor node to query, q, depending on the current local knowledge state B according to

$$q = \Psi(B)$$

where we drop the iteration number t for simplicity of notation. The form of this querying function Ψ is the key idea behind the IDSQ algorithm.

The intuition behind intelligent data collection in the IDSQ algorithm is to choose the sensor measurement which has the potential of providing the most "information" to the current local knowledge state. By choosing the sensor with maximum expected information content greedily, the idea is that the total number of sensor measurements chosen to reach a stopping knowledge state is minimized, thereby minimizing the amount of communication cost incurred. Thus, the efficiency of the IDSQ algorithm rests on the choice of an information measure from which we can compute the expected information content of a sensor measurement.

A possible way to define the notion of information content of a sensor measurement is to consider the difference in "spread" between the previous distribution B^t and the updated distribution B^{t+1} when a new sensor measurement z is incorporated. The smaller the spread of the resulting distribution B^{t+1} from incorporating a sensor measurement z, the more information content we consider z to have. One standard example of quantifying the notion of "spread" is the variance of a probability distribution, so that the information content is inversely proportional to the variance. We will elaborate on other possible measures of information in Appendix B. For now, let us continue the discussion of the data collection process.

We choose some measure of information

$$\mathcal{I} : \mathcal{B} \to \mathcal{R}$$

which is a real functional on the set of local knowledge states denoting the information content of the local knowledge state. To reiterate, smaller values of the information

measure indicate more spread out probability distributions while larger values of the information measure indicate tighter distributions.

Given the current local knowledge state B, we choose the next most informative sensor node q according to how much information the measurement z_q provides to the local knowledge state. If we could know the value of z_q before querying for it, then we would choose sensor node q according to

$$q = \arg_{j \in \{1,\ldots,n\}} \max \mathcal{I}(\Phi_j(B, z_j)) \ .$$

That is, for each sensor node, we predict how the local knowledge state B will be updated by z_j and optimize over the information content of these predicted local knowledge states. However, in a real situation, we do not know the value of z_j before querying for it, so we must marginalize out the value of z_j. One choice of this marginalization is to average out the value of z_j so that the querying function is

$$q = \arg_{j \in \{1,\ldots,n\}} \max E_{z_j} \left[\mathcal{I}(\Phi_j(B, z_j))\right] \ .$$

The interpretation of the above equation, referred to as the IDSQ criterion, is to compute for each sensor node j the *expected information content* over the set of possible predicted local knowledge states according to an appropriately chosen distribution for z_j. To be explicit, the querying function[1] is

$$\Psi(B) = \arg_{j \in \{1,\ldots,n\}} \max E_{z_j} \left[\mathcal{I}(\Phi_j(B, z_j))\right] \ . \tag{A.2}$$

A.1.4 Extensions

With the above form for the querying function Ψ, the IDSQ algorithm is purely an information-based criterion for sensor querying. A variant of this criterion is to take the communication cost into consideration explicitly as in

$$\tilde{\Psi}(B) = \arg_{j \in \{1,\ldots,n\}} \max \left(\beta \cdot E_{z_j} \left[\mathcal{I}(\Phi_j(B, z_j))\right] + (1-\beta) \cdot (-\mathbf{cost}(j, l))\right)$$

where $\mathbf{cost}(j, l)$ is the cost incurred for a communication between node j and the leader node l and $0 < \beta < 1$ is a parameter chosen to adjust the tradeoff between information content and communication cost. When β is 1, we are using the usual information-based querying function Ψ. For the case when β is 0, the criterion can be interpreted as a nearest neighbor query, where "nearest" means minimal communication cost.

The inclusion of the cost of communication into the querying function $\tilde{\Psi}$ was first considered in the constrained anisotropic diffusion routing (CADR) scheme also

[1] As stated here, it is possible for the querying function to choose the same sensor node repeatedly so that we run into the problem of incorporating the same measurement and, thus, unnecessarily bias our local knowledge state. In a real implementation, we also keep track of which sensor nodes have already been incorporated so that this overcounting phenomenon does not occur.

presented in [7]. The notion of querying for informative data in the IDSQ algorithm is comparable to the local decision of which sensor node to hop to next for routing data from some source to a destination. The idea behind adding a cost term is that even if some sensor can contribute a lot of information to the current local knowledge state, we may prefer to query information from a sensor which contributes less information but is a lot cheaper to query.

As presented in this section, the leader node is always the leader node in the IDSQ algorithm. We can extend the algorithm to include a criterion for switching leaders which can be important for extending the IDSQ algorithm for target tracking which we present in A.2.

A.1.5 Conclusions

This concludes the presentation of the basic IDSQ algorithm. The main idea was to break up the estimation and data collection processes into increments of belief update and sensor queries. Breaking up computations into pieces and allowing them to interact with each other is an application of the principle of information-based computation for the purpose of developing computationally efficient algorithms.

There is a deeper relationship implicit in the IDSQ algorithm, the interaction between estimation and communication. These two subjects are generally treated separately where the goal of estimation is to fully exploit information in data to construct optimal estimators whereas the goal of communication is to pump bits through a lossy channel at the fastest rate possible. Traditional estimation disregards the communication cost of accessing data by assuming all data is accessible, and traditional communication disregards whether the data being transmitted contains information relevant for estimation by the abstraction of information content into bits. It is clear that in applications like sensor networks the two disciplines can no longer be treated separately since interaction is key to developing an efficient system which trades off estimation accuracy for communication cost.

A.2 IDSQ Tracking Algorithm

The IDSQ algorithm presented in the previous section can be extended to track a single moving target through the sensor network. Just as in the basic setup of the IDSQ algorithm, the tracking task begins with a user chosen leader node. Again, there is only a single leader node throughout the entire sensor network at any time. Furthermore, the leader node is the only node which carries the current local knowledge state and can determine which node to query next. The difference between the single target tracking case and the single stationary target case is that measurements $z_i(t)$ at each sensor node i are functions of time t, and the target parameters θ_t can change over time. We have chosen time to proceed in uniform discrete steps so that what

we previously referred to as iteration number corresponds exactly to time. Extending the following discussion to non-uniform steps of time should be straightforward.

It is desirable for the leader node to be close to the location of the target to minimize communication latency and prevent track loss. Thus, the main modification of the single stationary target case is to add a criterion which allows the leader node to switch.

We will denote the current leader by l and its local knowledge state by B^t. Now that time is a factor, the estimation step requires a motion model for the target which we represent as a probability transition function

$$f(\theta_{t+1}|\theta_t) \ .$$

We will assume that the target parameters are Markov, and with the assumption of conditional independence of sensor measurements given the target parameters, we have a standard hidden Markov model. Thus, incorporating measurements into the local knowledge state can be computed recursively. In particular, at every time t we have the *predicted local knowledge state* from time $t-1$,

$$\tilde{B}^t(\theta_t) = \int_\Theta f(\theta_t|\theta_{t-1}) B^{t-1}(\theta_{t-1}) d\theta_{t-1}$$

and the *measurement updated local knowledge state*

$$B^t(\theta_t) = \frac{p_j(z_j(t)|\theta_t)\tilde{B}^t(\theta_t)}{\int_\Theta p_j(z_j(t)|\theta)\tilde{B}^t(\theta)d\theta}$$

where j is the sensor node whose measurement $z_j(t)$ we incorporate into the local knowledge state. For the data collection step, the leader will evaluate the querying function Ψ based on the current local knowledge state B^t to determine the next sensor q to query. Then, the leader l is faced with two choices:

1. either determine to merely query for measurement $z_q(t)$ in q and, thus, still remain the leader for the next iteration, or

2. decide that q should become the new leader and, thus, transmit the local knowledge state to node q and tell q to become the leader while deactivating itself as the leader.

Since the leader should be the node closest to the location of the target, the leader criterion is to make q the leader if it is closer to the mean $\tilde{\mu}_{t+1}$ of the predicted distribution $\tilde{B}^{t+1}$ than l is. Formally, the new leader $\tilde{l}$ is

$$\tilde{l} = \arg_{k\in\{l,q\}} \min \|\tilde{\mu}_{t+1} - x_k\|$$

where x_k is the location of sensor node k, and $\|\cdot\|$ is the usual norm in Euclidean space.

Discussion

Note that the above extension of the IDSQ algorithm to tracking is a distributed algorithm in the sense that the nodes take turns becoming the leader. However, it is not a distributed algorithm with concurrent processing and communication because only a single node can be processing and communicating at any given time. Let us elaborate on why the IDSQ algorithm is not a concurrent one.

All follower nodes are in an idle state while the leader is processing for the next sensor node to query or send the local knowledge state. Then, there are two cases.

1. When a query message is sent by the leader, the leader goes into an idle state waiting for a reply message. At this point, the sensor node that was queried becomes activated and replies back to the leader with the appropriate message. After replying to the leader, the queried sensor node then goes back to an idle state.

2. In the second case, when a local knowledge state message is sent, the old leader becomes a follower and goes into an idle state. The new leader becomes activated and begins the process of determining which sensor to query next.

We see that only a single sensor node is ever actively doing any kind of processing or communication so that there is no concurrency in the IDSQ tracking algorithm. Thus, we consider the IDSQ tracking algorithm to be a sequential algorithm on a virtual processor where the virtual processor jumps around from sensor node to sensor node.

Appendix B

Information Measures

The information measure

$$\mathcal{I} : \mathcal{B} \to \mathcal{R}$$

is a real functional on the set of local knowledge states $\mathcal{B}$. Due to the conceptual notion of local knowledge states being information about parameters, we will consider local knowledge states to be probability distributions on the parameter space Θ. Thus, the information measure is a statistic on the set of all probability distributions on Θ which captures the intuition of information content. For the discussion below, we will only consider the subset of probability distributions on Θ which have a probability density so that an element $B(\theta) \in \mathcal{B}$ is a probability density function.

B.1 Spread Type Measures

As discussed before, a notion of information content is the "spread" of the distribution. The information measure $\mathcal{I}$ should return lower values to those distributions which are more spread out and return higher values to those distributions which are tighter. For example, for a distribution $B(\theta)$, the subset of parameters

$$Y_\gamma^B = \{\theta \in \Theta : B(\theta) > \gamma\}$$

is the set of parameters for which the distribution is greater than some threshold γ. By choosing γ so that

$$\Pr(Y_\gamma^B) = \int_{Y_\gamma^B} B(\theta) d\theta = \beta$$

for $0 < \beta \leq 1$, the set Y_γ^B is the most probable set of probability β. The threshold $\gamma(\beta)$ is a function of β, and we will refer to the region $Y_{\gamma(\beta)}^B$ as the the most probable β-region of a distribution $B(\theta)$. The volume of this region

$$\mathbf{Vol}(Y_{\gamma(\beta)}^B) = \int_{Y_{\gamma(\beta)}^B} d\theta$$

for a given β would be a measure of how "spread out" this distribution is. The information measure for this construction would be the negative of the volume of $Y^B_{\gamma(\beta)}$,

$$\mathcal{I}_\beta(B) = -\mathbf{Vol}(Y^B_{\gamma(\beta)}) \ . \tag{B.1}$$

This measure captures the notion of "spread" for general multi-modal distributions. However, it suffers from the difficulty of computing $Y^B_{\gamma(\beta)}$ and choosing a β. Furthermore, as with any measure which measures the volume, if the region is singular, then the volume is zero. For example, the set of points corresponding to a line segment in three-dimensional Euclidean space has a volume of zero. This certainly does not match the intuition of the "spread" of the distribution since we would like to take into account the length of this line segment as well.

For $\Theta = \mathcal{R}^m$, another measure could be the negative of the determinant of the covariance matrix of a distribution

$$\mathcal{I}_{cov}(B) = -\mathbf{Det}(\mathbf{Cov}(B)) \tag{B.2}$$

where

$$\mathbf{Cov}(B) = \int_\Theta (\theta - \mu)(\theta - \mu)^T B(\theta) d\theta$$

and

$$\mu = \int_\Theta \theta \cdot B(\theta) d\theta \ .$$

This measure is good for distributions with a single mode but quite poor for multi-modal distributions. Furthermore, if the covariance matrix is singular, then the determinant is zero which means that the "spread" of the distribution is not being captured well.

A measure which is good for multi-modal distributions and easier to compute than the volume measure of Eqn. B.1 can be the negative of differential entropy

$$\mathcal{I}_{diff}(B) = \int_\Theta B(\theta) \log(B(\theta)) d\theta \tag{B.3}$$

since entropy can be related to the volume of the high probability region of the distribution.

All kinds of other statistics can be considered which measure the "spread" of a distribution like some measure of the perimeter enclosing the region of high probability. The best choice for matching the intuition of information content and maintaining computational tractability is application dependent.

There is one other fundamental flaw with measuring information content by the "spread" of a distribution. It is not necessarily true that as more information is gained the spread of the distribution becomes monotonically smaller. In fact, it is possible for some information to increase the spread of the distribution. Thus, we

must be careful when designing an information measure based on the spread of the distribution that the intuitive notion of information content for the application tends to be correlated with a smaller spread of the distribution.

B.2 Entropy Type Measures

The notion of information according to Shannon is of a different type, although the notion of "spread" is still present. Consider a binary random variable X taking values in $\{0, 1\}$ representing the outcome of a coin toss. Shannon's notion of information is equal to the uncertainty about X. For example, if X equals 1 with probability .999 and X equals 0 with probability .001, then if we toss the coin and find that X equals 1, we have gained very little information from our experiment since we believed that X would result in 1 with high probability. On the other hand, finding that X equals 0 results in a huge gain of information since we believed that X would rarely equal 0. Thus, the information gain from such an experiment is proportional to how different the outcome is than what we expected. Extending this idea to multi-valued random variables, the more spread out the possible outcomes of an experiment, the more potential information we can gain from it. This is opposite of how the "spread" was considered for spread type measures. For entropy type measures, the larger the "spread" is, the larger the information measure should be, and the tighter the "spread" is, the smaller the information measure should be.

In particular for application to the IDSQ algorithm, a relevant information measure for the potential information in sensor node j would be a measure of the "spread" of the distribution

$$P_j^B(z) = \int_\Theta p_j(z|\theta)B(\theta)d\theta$$

where $p_j(z|\theta)$ is the likelihood function of sensor node j and z takes values in $\mathcal{Z}_j$. This distribution is the conditional distribution of sensor j's measurement when we believe that the target parameter θ is distributed according to $B(\theta)$. At the extreme when the density is

$$P_j^B(z) = \delta_0(z - z_0)$$

where δ_0 is the Dirac delta function centered at 0, the potential information gain by querying for sensor j's observation is clearly none since querying for the sensor measurement will result in finding that the measurement is equal to z_0 exactly as expected.

Thus, we have the following three possible entropy type information measures about sensor node j according to how we wish to measure the "spread".

$$\tilde{\mathcal{I}}_\beta^j(B) = \mathbf{Vol}(Y_{\gamma(\beta)}^{\tilde{P}_j^B}) \tag{B.4}$$

$$\tilde{\mathcal{I}}_{cov}^j(B) = \mathbf{Det}(\mathbf{Cov}(\tilde{P}_j^B)) \tag{B.5}$$

$$\tilde{\mathcal{I}}^{j}_{diff}(B) = \int_{\mathcal{Z}_j} -\tilde{P}^{B}_{j}(z)\log(\tilde{P}^{B}_{j}(z))dz \tag{B.6}$$

The measures above, being based on a measure of "spread", have the same pathologies as presented in B.1.

B.3 Discussion

Spread type measures are functionals of probability distributions on the parameter space Θ whereas entropy type measures are measures on probability distributions on the measurement space $\mathcal{Z}_j$. The relationship of entropy type measures to spread type measures is that if $\tilde{P}^{B}_{j}(z)$ is tightly distributed, then for all likely values of θ according to $B(\theta)$, it happened that the likely measurement values z turned out to be nearly the same. This means that querying for the measurement value will not change the posterior distribution about which θ's are likely. On the other hand, if $\tilde{P}^{B}_{j}(z)$ is spread out, querying for the measurement value will result in cutting out those θ's which do not result in the measurement value of sensor node j. Thus, larger spread in entropy type measures correspond to smaller spread in spread type measures so that the two types of measures are equivalent. However, they can differ in terms of computation because the spaces Θ and $\mathcal{Z}_j$ can be of different dimensions.

Appendix C

Particle Filters

Let us present the particle filtering algorithm which is essentially a method which represents a filtered distribution by N independent samples from the distribution, called the *particle set*, with a set of steps which propagate these particles so that the result is an approximation of the filtered distribution for the next time step. Recall the set of motion models $\{q_t\}_{t=0}^{\infty}$ and initial distribution q_{-1} of the state and the set of likelihoods $\{L_t\}_{t=0}^{\infty}$ from Section 3.1.1. The state space is $\mathcal{X}$, and the measurement space is $\mathcal{Y}$.

A particle set of N particles is a set of state-weight pairs from

$$\mathcal{X} \times [0, 1]$$

written as

$$\mathbf{P} = \{(x^1, w^1), \ldots, (x^N, w^N)\}$$

such that the weights sum to one,

$$\sum_{i=1}^{N} w^i = 1 \ .$$

It will be useful to refer to the empirical probability mass function of a particle set $\mathbf{P}$ defined by

$$E_{\mathbf{P}}(x) = \sum_{(x',w)\in\mathbf{P}} w \cdot 1_{\{x'\}}(x)$$

for all $x \in \mathcal{X}$ where

$$1_S(x) = \begin{cases} 1 & , \ x \in S \\ 0 & , \ x \notin S \end{cases}$$

is the indicator function of the set $S \subset \mathcal{X}$.

With the usual assumptions of a Markov state process and conditional independence of measurements, the steps of the particle filtering algorithm are derived from Eqn. 3.2 and Eqn. 3.1. Eqn. 3.2 is essentially a prediction step which propagates the

particles ahead to the next time step. Eqn. 3.1 is essentially a reweight step which reweights those particles predicted ahead according to the likelihood. We will denote by $\tilde{\mathbf{P}}_t$ the particle set which represents the posterior distribution $\tilde{p}_t$, and by $\mathbf{P}_t$ the particle set which represents the predicted distribution p_t.

1. **Initialization** The initial step is to sample N times from the initial distribution q_{-1} which results in the samples

$$\{x_0^1, \ldots, x_0^N\} \ .$$

Then, the initial predicted particle set is

$$\mathbf{P}_0 = \{(x_0^i, \frac{1}{N})\}_{i=1}^N \ .$$

Initially, the measurement y_0 is given, so to construct $\tilde{\mathbf{P}}_0$ we need to reweight the particles of $\mathbf{P}_0$. In particular, the particles of $\tilde{\mathbf{P}}_0$ are

$$\tilde{\mathbf{P}}_0 = \{(x_0^i, \alpha \cdot L_0(y_0|x_0^i))\}_{i=1}^N$$

where α is a normalization factor chosen so that the weights of the particles in $\tilde{\mathbf{P}}_0$ sum to one.

2. **Induction Step** Given the particle set $\tilde{\mathbf{P}}_t$, we will construct the particle set $\tilde{\mathbf{P}}_{t+1}$ and, in the process, also construct $\mathbf{P}_{t+1}$. Note that we assume that the measurement y_{t+1} is given.

   - *(Prediction)* We sample N times with the possibility of repeating samples from the empirical probability mass function $E_{\tilde{\mathbf{P}}_t}$ which results in a set of samples

   $$\{z_t^1, \ldots, z_t^N\} \ .$$

   For each sample z_t^i, we sample once from the distribution

   $$q_t(\cdot|z_t^i)$$

   and call the sample x_{t+1}^i. Then, the next predicted particle set is

   $$\mathbf{P}_{t+1} = \{(x_{t+1}^i, \frac{1}{N})\}_{i=1}^N \ .$$

   - *(Reweight)* Each particle in $\mathbf{P}_{t+1}$ corresponds to a particle in $\tilde{\mathbf{P}}_{t+1}$. The only difference between the two particle sets is a difference in weight. In particular,

   $$\tilde{\mathbf{P}}_{t+1} = \{(x_{t+1}^i, \alpha \cdot L_{t+1}(y_{t+1}|x_{1+1}^i))\}_{i=1}^N$$

   where α is a normalization factor chosen so that the weights of the particles in $\tilde{\mathbf{P}}_{t+1}$ sum to one.

The above algorithm is the basic particle filtering algorithm. One of the problems with the above algorithm is that the samples of the particle set $\mathbf{P}_t$ are given by a random sampling algorithm based on the the motion model q_t without regard to the likelihood L_t. The problem is that sampling from q_t may propagate particles to those regions where the likelihood L_t is very small. Then, the major modes of the true posterior density are no longer populated by particles in the particle set. Thus, a variation to overcome this problem is to allow the likelihood L_t to have an effect on where particles are propagated. This variation is known as *particle importance sampling*, and we refer the reader to the literature on this idea.

If we use a particle filter for our track update algorithm, the corresponding track is computed from the sequence of particle sets

$$\{\tilde{\mathbf{P}}_t\}_{t=0}^{\infty} \ .$$

Particle sets are a simple representation of distributions which allow us to compute moments. For example, to compute the mean $\mu_{\mathbf{P}}$ corresponding to a particle set $\mathbf{P}$, we mean the expectation of the corresponding empirical probability mass function,

$$\mu_{\mathbf{P}} = \sum_{(x,w)\in\mathbf{P}} x \cdot E_{\mathbf{P}}(x) \ , \tag{C.1}$$

and the variance $\Sigma_{\mathbf{P}}$ is given by

$$\Sigma_{\mathbf{P}} = \sum_{(x,w)\in\mathbf{P}} (x-\mu_{\mathbf{P}})(x-\mu_{\mathbf{P}})^T E_{\mathbf{P}}(x) \ . \tag{C.2}$$

It is a trivial result that as number of particles N goes to infinity, the moments of the particle sets $\tilde{\mathbf{P}}_t$, generated by the particle filtering algorithm above, approach the moments of the true density $\tilde{p}_t$. However, for general distributions, the rate of convergence can be very slow, and it is unknown under what general conditions the rate of convergence of the moments is fast. For example, in [16], convergence results under certain conditions of the state process and measurement process are derived. On the other hand, the tradeoff between accuracy and computation is clear. The more particles we use, the better our approximation to the true distribution (in the sense of computing moments) at the expense of having to compute for more particles. Thus, there is a tradeoff between computational resource requirements and accuracy in estimation.

The idea we wish to take away is that when there is very little information about the particular motion models of a target, we must use a lot of particles to ensure that we do not lose track of the target. Once tracks have been established for a while, we can make the motion model more detailed so that less particles can be used without loss of the track. This idea allows us to optimize the computational resources expended under the requirement that we be able to track targets with reasonable accuracy.

The idea of using more detailed motion models to reduce the needed number of particles for successful tracking is not new. For instance, in [18], it is shown experimentally that the number of particles that are required for tracking the motion of a juggled ball can be quite small if the motion models used in the particle filtering algorithm are more detailed. The idea that is new, however, is the idea of dynamically switching to more detailed motion models depending on the information extracted from other inference tasks so that computational resources are optimized during the operation of the inference system. The proposed layered information processing architecture exploited this tradeoff in order to minimize the amount of computational resources consumed by choosing an appropriate number of particles for each particular motion model used for updating tracks as presented in Section 3.4.2.

Bibliography

[1] Y. Bar-Shalom. Extension of the probabilistic data association filter in multi-target tracking. In *Proc. 5th Symp. on Nonlinear Estimation*, pages 16–21, Sept. 1974.

[2] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*, volume 179 of *Mathematics in Science and Engineering*. Academic Press, Inc., 1988.

[3] E. Bienenstock, S. Geman, and D. Potter. Compositionality, mdl priors, and object recognition. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems*, 9, pages 838–844. MIT Press, 1997.

[4] S. Casadei and S. K. Mitter. A hierarchical approach to high resolution edge contour reconstruction. In *1996 Conference on Computer Vision and Pattern Recognition*, pages 149–154, San Francisco, CA, June 1996.

[5] S. Casadei and S. K. Mitter. Hierarchy image segmentation – part i: Detection of regular curves in a vector graph. *International Journal of Computer Vision*, 27(3):71–100, March 1998.

[6] S. Casadei and S. K. Mitter. A perceptual organization approach to contour estimation via composition, compression and pruning of contour hypotheses. Technical Report LIDS-P-2415, Massachusetts Institute of Technology, April 1998.

[7] M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks. *Int'l J. High Performance Computing Applications*, 16(3), August 2002.

[8] D. Geman and B. Jedynak. An active testing model for tracking roads from satellite images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18:1–14, 1996.

[9] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000.

[10] D. M. Howard and J. Angus. *Acoustics and Psychoacoustics*. Focal Press, Oxford, 2001.

[11] C. Hue, J. L. Cadre, and P. Perez. Sequential monte carlo methods for multiple target tracking and data fusion. *IEEE Transactions on Signal Processing*, 50(2), February 2002.

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, Boston, MA, 2000.

[13] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal for Computer Vision*, 29(1):5–28, 1998.

[14] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Mobile networking for smart dust. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Seattle, WA, August 1999.

[15] R. Duncan Luce. *Sound and Hearing: A Conceptual Introduction*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1993.

[16] P. Del Moral, J. Jacod, and P. Protter. The monte-carlo method for filtering with discrete-time observations. *submitted*, September 1999.

[17] D. Nguyen, J. Kay, B. Orchard, and R. Whiting. Classification and tracking of moving ground vehicles. *MIT Lincoln Laboratory Journal*, 13(2):275–304, 2002.

[18] B. North, A. Blake, M. Isard, and J. Rittscher. Learning and classification of complex dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(9):1016–1034, September 2000.

[19] D. Potter. *Compositional Pattern Recognition*. PhD thesis, Brown University, 1999.

[20] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[21] J. A. Ratches, C. P. Walters, R. G. Buser, and B.D. Guenther. Aided and automatic target recognition based upon sensory inputs from image forming systems. *IEEE Transactions on Patter Analysis and Machine Intelligence*, 19(9):1004–1019, September 1997.

[22] D. B. Reid. An algorithm for tracking multiple targets. *IEEE Trans. on Automat. Contr.*, AC–24(6):843–854, December 1979.

[23] T. D. Rossing, F. R. Moore, and P. A. Wheeler. *The Science of Sound.* Addison Wesley, San Francisco, California, 2002.

[24] J. Shin, L. Guibas, and F. Zhao. A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks. In *The Second International Workshop on Information Processing in Sensor Networks*, to appear in April 2003.

[25] Shimon Ullman. *High-level Vision: Object Recognition and Visual Cognition.* MIT Press, Cambridge, Massachusetts, 1996.

[26] M. Wainwright. *Stochastic processes on graphs with cycles: geometric and variational approaches.* PhD thesis, Massachusetts Institute of Technology, 2002.

[27] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *International Joint Conference on Artificial Intelligence*, August 2001.

[28] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department, May 2001.