Carnegie Mellon University
Research Showcase @ CMU

Dissertations | Theses and Dissertations

11-2010

Graphical Models and Overlay Networks for Reasoning about Large Distributed Systems

Stanislave Funiak
Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/dissertations

Recommended Citation
Funiak, Stanislave, "Graphical Models and Overlay Networks for Reasoning about Large Distributed Systems" (2010). *Dissertations.* Paper 38.

This Dissertation is brought to you for free and open access by the Theses and Dissertations at Research Showcase @ CMU. It has been accepted for inclusion in Dissertations by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Graphical Models and Overlay Networks for Reasoning about Large Distributed Systems

Stanislav Funiak

CMU-RI-TR-10-39

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Robotics

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

November 2010

Thesis Committee:
Carlos Guestrin, Chair
Geoffrey Gordon
Sanjiv Singh
Joseph Hellerstein, UC Berkeley

Copyright © 2010 by Stanislav Funiak. All rights reserved.

This research was sponsored by the National Science Foundation under grant numbers CNS-0428738, CNS-0625518, CNS-0721591, and IIS-0803333, the Office of Naval Research under grant number N000140710747, the Army Research Office under grant number W911NF0710287, by Intel Corporation, and by Carnegie Mellon University.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Abstract

This thesis examines reasoning under uncertainty in distributed systems. Unlike in centralized systems, where the observations reside in a single location, the observations in distributed systems are often scattered across the network. To reason accurately, a networked device often needs to incorporate observations from other nodes and must do so with limited computation and communication even for large problems. The reasoning is further complicated by unstable network conditions, characteristic to many real-world networks: the nodes may fail, communication links may become unreliable, and the entire network may get fragmented into several components that cannot communicate with each other. These aspects make distributed inference very challenging.

We consider one general problem of distributed filtering for estimating the state of a dynamical system and three independent applications: simultaneous localization and tracking, where a camera network localizes itself by observing a moving object, internal localization of large-scale modular robots, where a robot determines the relative poses of its internal parts, and collaborative filtering for providing recommendations in a peer-to-peer network. These problems share a common theme: each of these problems can be described by a graphical model that permits compact representation of and efficient reasoning about the problem. Using graphical models, we design algorithms that address challenges, such as inconsistency of node beliefs in fragmented networks and difficult local optima in modular robot localization. Due to the complexity of the reasoning tasks, it is not sufficient to coordinate the nodes locally within each node's immediate physical neighborhood. Instead, our algorithms employ overlay networks—distributed data structures built on top of the physical networks—to coordinate among distant nodes. The resulting algorithms obey the communication constraints imposed by the network, while solving the problems robustly.

We evaluate our algorithms on data from real sensor networks and on a realistic deployment on the PlanetLab network. We demonstrate robustness to network fluctuations and, in some cases, our distributed algorithms improve upon state-of-the-art centralized approaches.

Acknowledgments

I would like to thank my advisor Carlos Guestrin for all his guidance and support. Looking back, I feel that Carlos has always guided me in the right direction with his far-reaching vision. He has been very encouraging through the years, which made even the most difficult tasks seem manageable. I have been inspired by him to learn more and to become a better researcher.

I would also like to thank my committee members Joe Hellerstein, Geoff Gordon, and Sanjiv Singh, whose work and feedback have profoundly shaped my thinking. I would like to thank Joe for the collaboration and for many interesting discussions. I am also thankful to Geoff for his mathematical approach and pinpointed comments during the Select Lab meetings. I am grateful to Sanjiv for expanding my horizons in applications and keeping me grounded in reality.

During my times in graduate school, I had the opportunity to work with and learn from others. I would especially like to thank my coauthors Mark Paskin, Rahul Sukthankar, Babu Pillai, Michael Ashley-Rollman, Jason Campbell, Seth Copen Goldstein, Ashima Atul, and Kuang Chen for the fun times we had while solving the problems considered in this thesis. Their ideas and insights made the problems we worked on much more tangible.

While at CMU, I also had the privilege to interact with people from the Select Lab. I feel lucky for having been able to work with so many smart people around me. I am especially grateful to Joey Gonzalez, Joseph Bradley, Jon Huang, Khalid El-Arini, Andreas Krause, Yucheng Low, and Anton Chechetka for collaborating with me on both the theoretical and the practical aspects of research and for their feedback on this thesis. I would also like to thank Felix Duvallet and my newly rediscovered friend Jacob Eisenstein for their revisions to the thesis.

I am greatly indebted to all my friends and family for their help along the way. I would especially like to mention Ram Ravichandran—you are the best roommate ever! I would also like to mention Lucia Castellanos, Miro Dudík, and Max Van Kleek, whose company I truly enjoy. My parents Stanislav Funiak and Soňa Funiaková and my sister Monika Jačmeníková, though physically distant, were always with me when I needed them.

And Sanako, you are one of a kind, and you make my days bright.

Contents

1 Introduction **1**

1.1 Inference problems in situated distributed systems . . . 2
1.2 Graphical models for distributed inference . . . 5
1.2.1 Marginal queries in static probabilistic models . . . 5
1.2.2 Filtering in dynamic Bayesian networks . . . 10
1.2.3 Maximum-likelihood estimation for modular robot localization . . . 12
1.2.4 Matrix factorization for collaborative filtering . . . 13
1.3 Coordination with overlay networks . . . 14
1.3.1 Coordination for computing marginals . . . 15
1.3.2 Coordination for maximum-likelihood estimation . . . 16
1.3.3 Coordination for collaborative filtering . . . 17
1.4 Overview of the thesis . . . 17

2 Robust distributed probabilistic inference **21**

2.1 Centralized probabilistic inference . . . 21
2.1.1 Multivariate distributions . . . 22
2.1.2 Factorized probability models . . . 24
2.1.3 Variable elimination and junction trees . . . 28
2.1.4 Decomposable models . . . 33
2.1.5 Conditioning on observations . . . 35
2.2 Probabilistic inference in distributed systems . . . 39
2.2.1 The distributed probabilistic inference problem . . . 40
2.2.2 Distributed inference architecture . . . 41
2.2.3 Limitations of the sum–product algorithm in distributed systems . . . 44
2.3 Robust message passing . . . 46
2.3.1 Decomposable models revisited . . . 47
2.3.2 Probabilistic inference as distributed clique pruning . . . 48
2.3.3 Conditioning on observations . . . 54
2.3.4 Approximate partial correctness . . . 59

2.4 Related work . . . 63
2.5 Discussion . . . 63
Appendix 2.A Proofs . . . 64

3 Distributed inference in dynamical systems 67
3.1 Centralized inference in dynamical systems . . . 67
3.1.1 Dynamic Bayesian networks . . . 68
3.1.2 Exact filtering . . . 73
3.1.3 Assumed density filtering . . . 75
3.2 The distributed filtering problem . . . 79
3.3 Approximate distributed filtering . . . 80
3.3.1 Outline of the algorithm . . . 80
3.3.2 Estimation as robust message passing . . . 81
3.3.3 Prediction, roll-up, and projection . . . 82
3.3.4 Summary of the algorithm . . . 84
3.4 Robust distributed filtering . . . 85
3.4.1 The alignment problem . . . 86
3.4.2 Optimized conditional alignment . . . 88
3.4.3 Distributed optimized conditional alignment . . . 94
3.4.4 Jointly optimized alignment . . . 97
3.5 Experimental results . . . 100
3.5.1 Applications . . . 101
3.5.2 Convergence . . . 102
3.5.3 Alignment . . . 103
3.6 Related work . . . 104
3.7 Discussion . . . 108
Appendix 3.A Proofs . . . 109

4 Simultaneous localization and tracking for camera networks 113
4.1 Simultaneous localization and tracking . . . 113
4.2 Dynamic probabilistic SLAT . . . 114
4.2.1 Model . . . 114
4.2.2 Exact filtering for SLAT . . . 116
4.3 Addressing nonlinearities . . . 117
4.3.1 Gaussian representations in absolute parameters . . . 117
4.3.2 Relative over-parameterization . . . 119
4.3.3 Hybrid conditional linearization . . . 121
4.3.4 Approximate Kalman filter . . . 124
4.4 The BK approximation . . . 125

4.4.1 Selecting the BK structure . . . 126
4.4.2 Incorporating nonlinear observations . . . 127
4.5 Experimental results . . . 129
4.5.1 Convergence . . . 129
4.5.2 Number of observations . . . 130
4.5.3 BK approximation and comparison with off-line optimization . . . 130
4.6 Related work . . . 131
4.7 Discussion . . . 134

5 Localization in large-scale modular robot ensembles 135
5.1 Internal localization in modular robots . . . 135
5.2 Maximum-likelihood estimation for internal localization . . . 137
5.2.1 Probabilistic model for internal localization . . . 138
5.2.2 Maximum-likelihood estimation . . . 140
5.3 Localization with hierarchical graph partitioning . . . 141
5.3.1 Overview of the algorithm . . . 141
5.3.2 Determining an effective partition . . . 142
5.3.3 Merging partial solutions . . . 144
5.3.4 Scaling up the solution . . . 145
5.4 Distributed localization . . . 146
5.4.1 Network assumptions . . . 146
5.4.2 Individual layers . . . 146
5.4.3 Implementation using Meld . . . 149
5.4.4 On the role of group leaders . . . 152
5.5 Experimental results . . . 153
5.5.1 Scenarios . . . 154
5.5.2 Convergence . . . 154
5.5.3 Scalability . . . 156
5.5.4 Number of observations . . . 158
5.5.5 3D experiments . . . 158
5.5.6 Messaging costs . . . 159
5.6 Related work . . . 161
5.7 Discussion . . . 162

6 Collaborative filtering in peer-to-peer networks 165
6.1 Distributed collaborative filtering . . . 165
6.2 Latent variable models for collaborative filtering . . . 166
6.2.1 Matrix factorization . . . 167
6.2.2 Restricted Boltzmann Machines . . . 168

6.3 Parallel collaborative filtering . . . 171
6.4 Distributed collaborative filtering . . . 174
6.4.1 Super-peer architecture . . . 174
6.4.2 Outline of the algorithm . . . 179
6.4.3 User initialization . . . 179
6.4.4 Distributed averaging . . . 180
6.5 Experimental results . . . 182
6.5.1 Simulated experiments . . . 182
6.5.2 PlanetLab experiments . . . 184
6.5.3 Online experiments . . . 185
6.6 Related work . . . 187
6.7 Discussion . . . 189

7 Conclusions 191
7.1 Summary of the thesis . . . 191
7.2 Common themes . . . 193
7.3 Directions for future research . . . 196
7.3.1 Distributed filtering with minimal communication . . . 196
7.3.2 Adaptive DBN filtering . . . 197
7.3.3 Distributed generalized belief propagation . . . 197
7.3.4 Dynamic localization in modular robots . . . 199
7.3.5 Alignment for multi-robot SLAM . . . 200
7.4 Concluding remarks . . . 200

Bibliography 203

Chapter 1

Introduction

Networked devices often need to reason with incomplete, inaccurate information—that is, they must reason about uncertainty. For example, a camera network may wish to calibrate itself while tracking a moving object. Each camera is uncertain about its physical location and the trajectory of the object, and the cameras need to estimate their locations and the trajectory from the noisy images. Or, for example, a network of personal computers may wish to provide automatic movie recommendations. Each computer is uncertain about the preferences of its user and about the types of the songs, and the computers need to predict the user's interest in each movie by collecting opinions from a large number of users.

Reasoning under uncertainty is often reduced to one of two forms. In **probabilistic inference**, the system is modeled as a probability distribution over the hidden state, such as the object trajectory and the camera poses, as well as the observed images. Reasoning under uncertainty is then formulated as computing statistics, such as the mode or the marginals of the distribution, given the observations. In **parameter estimation** we wish to obtain a model that predicts aspects, such as the user's interest in a particular movie. The hidden parameters of the model are estimated from data. In distributed systems, these two forms of reasoning are complicated by the fact that a node often needs to incorporate measurements from other nodes. For example, measurements made at one camera improve the estimate of the object trajectory; a more accurate estimate of object trajectory can, in turn, help other cameras calibrate themselves better. Similarly, in a recommender system, accurate predictions can be only made by collecting data from other users; in the absence of any external knowledge of the movies, an individual user's opinions alone carry little information about the movies the user has not rated yet.

There are two fundamental challenges to nodes incorporating observations from across the network: scalability and robustness. The reasoning task may be difficult even in the centralized setting, and the nodes often need to exploit structure of the problem to obtain an efficient solution. The nodes themselves may have limited resources, and it is often prohibitive to communicate all the measurements into a single location. Furthermore, many of today's networks are not reliable: nodes may go down, communication may

get interrupted, or the whole network may become fragmented into components that cannot communicate with each other. The algorithm needs to degrade its performance gracefully under these conditions.

Graphical models are an established tool to represent and reason about large systems. A graphical model describes the interactions among variables that characterize the system that are often "local," in the sense that they relate small sets of variables. Increasingly, graphical models are being used to reason about distributed systems. Due to the limited resources, each node often has an incomplete view of the model, and the nodes need to coordinate to solve the inference task. In the simplest algorithms, such as loopy belief propagation (Pearl, 1988), the coordination is purely local, limited to pairs of nodes that are near each other. However, more complicated algorithms may require the algorithm to gather information from distant nodes. In this case, the graphical model needs to be complemented with an **overlay network**—a distributed data structure built on top of the physical network.

This dissertation focuses on three applications and one general problem, whose solutions leverage graphical models and overlay networks. Our algorithms scale to large networks, and we present novel techniques to address communication failures and nonlinearities that arise in many real-world systems. In some instances, our use of overlay networks is novel. We claim the following statement:

> **In many applications, devices need to integrate uncertain observations from across the network. By designing distributed algorithms that build upon graphical models and overlay networks, one can obtain scalable, robust, and accurate solutions.**

1.1 Inference problems in situated distributed systems

Many distributed systems are "situated" in their environment: they interact with the environment, by gathering data and carrying out actions. Situated distributed systems often need to solve difficult inference tasks that require the nodes to integrate observations from across the network. Here, we mention three examples that are considered in this thesis:

Calibration in sensor networks. A sensor network is a collection of autonomous devices, whose primary purpose is to monitor an environment. Sensor networks have been used for a variety of tasks, such as collecting data about the habitats of animals sensitive to human disturbances (Mainwaring et al., 2002) or cattle herding (Butler et al., 2004), and enhancing the quality of life with intelligent environments (Krumm et al., 2000).

Typically, the sensors need to be **calibrated** to interpret the measurements. For example, in a camera network (Figure 1.1(a)), measurements are only useful if we know from where the images were captured, that is, the real world locations of the cameras. Manually measuring the pose (location and orientation) of all cameras in the network is a very tedious and time consuming task; in some domains, such as emergency response systems, manual calibration is not even possible. Instead, the

cameras need to do the calibration automatically, by identifying common features in the scene, such as moving objects. Automatic calibration requires the cameras to integrate each other's measurements, because measurements from a single camera only relate the camera to the tracked objects, but not to the other cameras.

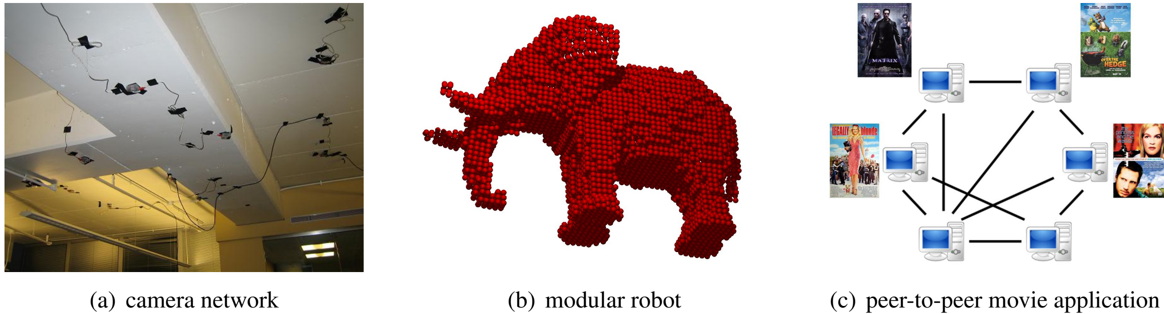
Localization in modular robots. A modular robot consists of interconnected components, called **modules**; the modules operate autonomously and rearrange their positions within the robot to adapt the robot to the task at hand. Large-scale modular robots with many fine-grained modules can be used for **rendering** of objects in tasks, such as product design and visualization (Goldstein and Mowry, 2004) or rapid prototyping (Gilpin et al., 2008). Figure 1.1(b) shows a simulated rendering of an elephant by a robot with more than 8000 modules; recent research suggests that real fine-grained modules can be manufactured in large quantities (Karagozler et al., 2009).

A key challenge in modular robots is determining the relative poses among the robot's individual modules. This task is important, because the modules typically need to know their poses, in order to plan for the shape adaptation and to determine their tasks within the robot. The modules can often sense their neighbors locally; however, such measurements are inherently noisy, so the modules need to take advantage of observations made in the loop to resolve a large rotational uncertainty. Reasoning under uncertainty in this system is substantially harder than reasoning with perfect information: if we were to assume perfect observations or lattice structure, simple constraint-based methods work well (Pillai et al., 2006; Reshko, 2004).

Recommendations in peer-to-peer networks. A peer-to-peer network is a collection of computers or devices that share each other's resources. The traditional use of peer-to-peer networks is file sharing, but more recently, peer-to-peer networks are being used for **content distribution**. For example, the music streaming service Spotify uses a peer-to-peer network to deliver the songs to each user. The music is streamed between the user's computers; in this manner, the service can operate with substantially less centralized infrastructure.

To help the users explore the catalogue, content providers like Spotify provide customized **recommendations** of items (songs, movies) that match the users' individual preferences. Typically, users are willing to provide opinions of items they know; a recommender system can leverage the opinions from a large number of users to identify hidden patterns in the data, such as the types of the songs or the individual user's preference. In order to lower the operating costs, such recommender system can, too, run in a distributed fashion on the peer-to-peer network, rather than on a dedicated centralized server. A distributed recommender system is also useful for those users of network-connected media centers, who do not have a subscription to a commercial services, such as Netflix.

Due to the scale of the systems described above, it is not feasible to collect the raw measurements to a single location for centralized processing. Instead, the nodes need to reason about the uncertainty in a



(a) camera network (b) modular robot (c) peer-to-peer movie application

Figure 1.1: Applications, considered in this thesis. (a) A camera network. (b) A simulated modular robot with 8,008 modules. (c) A peer-to-peer application for streaming movies.

distributed manner, communicating summary statistics of their observations. This reasoning must address the following challenges typical to situated distributed systems:

Scalability. In order to be most useful, the networks need to scale to large numbers of nodes. Current literature describes several sensor networks that have scaled to several hundreds of nodes; simulations with large-scale modular robots often reach tens of thousands to hundreds of thousands of modules. Current peer-to-peer networks are even larger; for example, the Gnutella network has scaled to more than three million nodes in 2006 (Rasti et al., 2006), and as of May 2010, Spotify had approximately seven million users. Thus, it is crucial that the algorithms are designed to scale to large number of nodes.

Constrained resources. The nodes in many situated distributed systems are computationally constrained. For example, a sensor network node TMote Sky by Sentilla has an 8MHz Texas Instruments MSP430 microcontroller with only 10kB of RAM and 48kB of Flash memory. The nodes in large-scale modular robots are similarly constrained. Furthermore, wireless sensor networks are often characterized by limited power they can harvest or store. The algorithms must be able to run under such tight computational constraints. Peer-to-peer networks often consist of personal computers with substantially more resources, but some nodes, such as Internet-enabled phones, may still be computationally constrained.

Unreliable communication. Many situated distributed systems form a wireless ad-hoc network, where nodes cooperate in a decentralized fashion using radio waves and do not rely on preexisting infrastructure to route the traffic. Wireless communication is a lossy form of communication, where often a large fraction of packets are lost between nodes. The network can experience serious communication failures, where the communication network gets fragmented into several parts. The algorithms for distributed reasoning in sensor networks need to be robust enough to handle these kinds of communication failures. Peer-to-peer networks can often leverage protocols, such as TCP, to achieve reliable communication, but the bandwidth can vary from one pair of nodes to another, and the algorithms need to be robust to these differences.

Unstable node membership. Finally, situated distributed systems have an unstable node membership. For example, in sensor networks, nodes often fail due to drained batteries and environmental factors, and new nodes may be added to increase the connectivity of the network. In modular robots, nodes can also fail, and new nodes can be introduced. In peer-to-peer networks, nodes rarely fail, but they often enter and leave the network.

Algorithms vary in how they address these challenges. In some algorithms, the challenges are addressed *implicitly*, by degrading the performance of the algorithm if it faces difficulties which the algorithm was not originally designed to handle. For example, loopy belief propagation (Pearl, 1988) computes approximate marginals of a distribution by passing messages between adjacent nodes. If a node fails, the algorithm ends up solving a perturbed problem that omits some of the variables from the model (Schiff et al., 2007). If the communication between a pair of nodes becomes unreliable, the messages will not be propagated, and the algorithm converges more slowly. By comparison, the algorithms in this thesis address the challenges *explicitly*: if a communication link becomes unreliable, the communication is rerouted through a different path. Addressing unreliable communication and unstable node membership explicitly is often better, since it permits the nodes use their resources more effectively.

1.2 Graphical models for distributed inference

In order to reason about large, complex systems, one needs to exploit problem structure to obtain a scalable solution. Graphical models are an established way to represent structure in systems. The exact type of graphical model varies from one problem to another, but they all share a common property: the system is represented compactly as functions over small sets of variables, and the dependencies among these variables are captured by a graph. In this section, we first review the graphical models for representing static systems and then give an overview of the graphical models and the algorithms in this thesis. Some of the work in this section is inspired by the work of Paskin (2004).

1.2.1 Marginal queries in static probabilistic models

Reasoning under uncertainty can be often formulated as computing a marginal of a probability distribution. Suppose that we wish to estimate the temperatures in an environment, and suppose that we can only afford to buy cheap, noisy sensors. One way to solve this task is to formulate a probabilistic model that captures the variations of both the true hidden temperatures and the observations and then compute the conditional distribution over each hidden temperature, given the observations. The expected value of the hidden temperature represents the estimate, and its variance represents the uncertainty.

To solve this task efficiently, we can leverage the structure in the probability distribution in the form of conditional independences. Informally, two random variables are **independent** if the knowledge of one

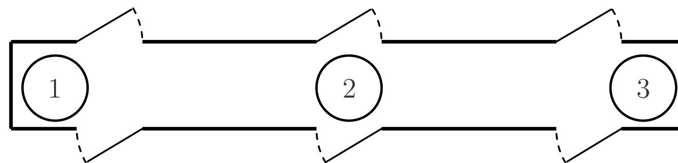


Figure 1.2: A hallway with three temperature locations.

gives us no information about the value of the other. On the other hand, two variables are **conditionally independent** if they are independent *once another set of variables is observed*. Conditional independence is a very common property that arises in many complex systems. For example, consider a narrow hallway shown in Figure 1.2. The temperatures at locations 1 and 3 are *not* independent, because they are related by the air flow in the hallway. Nevertheless, they are conditionally independent, given the temperature at location 2 in the middle of the hallway, because knowing this temperature tells us much about the air flow between locations 1 and 3.

In probabilistic models, conditional independences reveal themselves through **factorization structure**. Denote the temperatures in the above examples by X_1, X_2, and X_3.[1] Then we can use the chain rule and the conditional independence to write the distribution $p(x_1, x_2, x_3)$ as

$$\begin{aligned} p(x_1, x_2, x_3) &= p(x_1) \times p(x_2 \mid x_1) \times p(x_3 \mid x_1, x_2) \\ &= p(x_1) \times p(x_2 \mid x_1) \times p(x_3 \mid x_2). \end{aligned} \tag{1.1}$$

Thus, due to the conditional independences, we are able to express the distribution as a product of **factors**

$$p(x_1, x_2, x_3) = \psi_1(x_1) \times \psi_{12}(x_1, x_2) \times \psi_{23}(x_2, x_3), \tag{1.2}$$

where each factor ψ_i depends only on a subset of "nearby" random variables.

Factorized probabilistic models let us represent the distribution more compactly. For example, suppose that each X_i can have one of k discrete values. A naive representation that stores the probability for each triplet (x_1, x_2, x_3) of assignments to (X_1, X_2, X_3) as a table would require $O(k^3)$ parameters overall. On the other hand, the factorized representations in Equations 1.1 and 1.2 require us to store at most k^2 values for each factor and $O(k^2)$ parameters overall. Factorization structure can be also used to speed up inference. For example, suppose that we wish to compute the marginal distribution over the variable X_3. This task amounts to evaluating the sum

$$p(x_3) = \sum_{x_1} \sum_{x_2} p(x_1, x_2, x_3)$$

for each value of x_3. If we represent $p(x_1, x_2, x_3)$ as a table of probabilities, this sum requires $O(k^2)$

[1] We mentioned air flow as a mechanism through which uncertainty arises in this system, but for the purposes of this example, the temperatures are considered static.

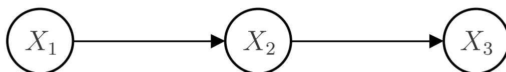


Figure 1.3: A Bayesian network.

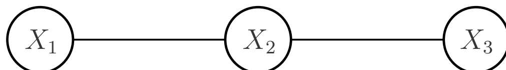


Figure 1.4: A Markov network.

addition operations for each value of x_3 (because we had to perform a double sum), or $O(k^3)$ addition operations overall. On the other hand, we can leverage the factorization structure (1.1) to "push" the summation over x_1 past the term that does not depend on it:

$$\begin{aligned} p(x_3) &= \sum_{x_2}\sum_{x_1} p(x_3 \mid x_2) \times p(x_2 \mid x_1) \times p(x_1) \\ &= \sum_{x_2} p(x_3 \mid x_2) \times \sum_{x_1} p(x_2 \mid x_1) \times p(x_1) \end{aligned} \tag{1.3}$$

The inner sum must be evaluated once for each value of x_2, and requires $O(k^2)$ multiplications and additions overall. The intermediate result is a table with k entries, one for each value of x_2. The outer sum then requires additional $O(k^2)$ operations. Thus, factorization structure can lead to substantial savings in computation.

The factorization structure of a distribution can be captured by a graph. Figure 1.3 shows an example a **Bayesian network** for the factorization (1.1). Each vertex corresponds to one variable in the distribution, and there is a directed edge from X_i to X_j if X_j conditions on X_i. Figure 1.4 shows a **Markov network** for the factorization (1.2). A Markov network is an undirected graph, where each vertex corresponds to one variable in the model, and there is an undirected edges between X_i and X_j if X_i and X_j appear together in some factor. Bayesian networks and Markov networks are useful, because they represent a set of conditional independence assumptions about the distribution and because they let us reason about the distribution graphically, without regard to the specific values of the factors.

So far, we have talked about a model that describes the environment. The distribution $p(x_1, x_2, x_3)$ is called the **prior distribution**, and $\mathbf{X} = (X_1, X_2, X_3)$ are called the **hidden variables**. Now suppose that we place a sensor at each location a, and each sensor makes a measurement $Z_a = \overline{z}_a$, where Z_a is called an **observed variable**, and $\overline{z}_a$ is a fixed assignment. By chain rule, the joint distribution over the hidden variables and the observed variables can be written as

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}) \times p(\mathbf{z} \mid \mathbf{x}),$$

where $p(\mathbf{z} \mid \mathbf{x})$ is the **observation model**. A common assumption is that the measurements are independent

of each other given the hidden variables. Furthermore, each observed variable is typically influenced only by a small number of hidden variables; for example the observed variable Z_a is influenced by the true temperature X_a at the same location and is independent of all other hidden variables, given X_a. The observation model can then be written as a product of factors

$$p(\mathbf{z} \mid \mathbf{x}) = \prod_a p(z_a \mid x_a).$$

Thus, the observation model itself has a significant factorization structure. This fact is very important for inference. Typically, in probabilistic inference, we are interested in the **posterior distribution** $p(\mathbf{x} \mid \overline{\mathbf{z}})$ that characterizes our knowledge of the system after the measurements $\mathbf{Z} = \overline{\mathbf{z}}$ have been made. In the temperature example in Figure 1.2, the posterior distribution can be written as

$$p(\mathbf{x} \mid \overline{\mathbf{z}}) \propto p(\mathbf{x}, \overline{\mathbf{z}}) = \big(p(x_1)p(\overline{z}_1 \mid x_1)\big) \times \big(p(x_2 \mid x_1)p(\overline{z}_2 \mid x_2)\big) \times \big(p(x_3 \mid x_2)p(\overline{z}_3 \mid x_3)\big).$$

Notice that this expression has the same structure as the prior distribution distribution (1.1), but we have replaced each factor of the prior distribution with a product of the factor and the **observation likelihood** $p(\overline{z}_a \mid x_a)$. Therefore, we can compute the marginal of the posterior distribution $p(x_3 \mid \overline{\mathbf{z}})$ as efficiently as computing the prior marginal $p(x_3)$ in Equation 1.3.

While factorized models, such as Markov networks and Bayesian networks, can lead to efficient inference, they an important drawback: it is difficult to interpret the factors in isolation. In a distributed system, each node is assigned a part of the model; for example, in a temperature sensor network, each sensor may carry a factor of the prior distribution (1.1). In the normal course of operation, some nodes may fail or be destroyed. When this happens, the factors held by the nodes are lost. We may hope that removing some of the factors in a factorized model, the remaining factors form a reasonable approximation of the original model. Unfortunately, this is not true. For example, if the factor $p(x_1)$ in the model (1.1) is lost, the remaining factors form a *conditional* distribution $p(x_2, x_3 \mid x_1)$. This conditional distribution only characterizes the variations of X_2 and X_3 relative to X_1, but not the marginal $p(x_2, x_3)$.

To address this challenge, Paskin and Guestrin (2004b) proposed an algorithm that relies on a special representation of probability distributions, called a **decomposable model**. A decomposable model can be described by a graph structure, called **junction tree**. A junction tree represents a distribution in terms of sets of tightly interaction variables, called the **cliques**. An example of a junction tree for a distribution with 5 variables is shown in Figure 1.5. Each vertex of a junction tree is associated with one clique, and each edge is associated with a **separator**, which is simply the intersection of the two cliques it connects. The cliques must satisfy the **running intersection property**: if a variable is present in two cliques, it must also be present in every clique on the unique path between them.

A key fact about decomposable models is that they represent the distribution as a ratio of clique marginals and separator marginals (Koller and Friedman, 2009, Section 10.2.3). For example, given the junction tree

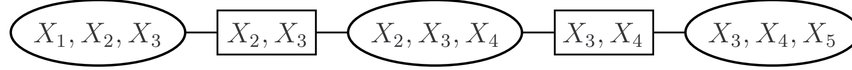


Figure 1.5: A junction tree for a distribution with 5 variables. Ellipses indicate the cliques; rectangles indicate the separators. The tree satisfies the running intersection property; for example, the variable X_3 is present in the two leaf cliques $\{X_1, X_2, X_3\}$ and $\{X_3, X_4, X_5\}$, so it is also present in the clique $\{X_2, X_3, X_4\}$.

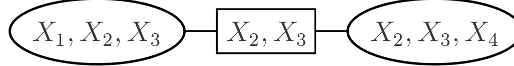


Figure 1.6: A junction tree that represents a marginal of the distribution in Figure 1.5.

in Figure 1.5, a decomposable model can be written as

$$p(x_1, x_2, x_3, x_4, x_5) = \frac{p(x_1, x_2, x_3) \times p(x_2, x_3, x_4) \times p(x_3, x_4, x_5)}{p(x_2, x_3) \times p(x_3, x_4)} \tag{1.4}$$

This representation has a very useful consequence: Any *subtree* of a decomposable model represents a marginal distribution. For example, suppose that we wish to marginalize out the variable X_5 from the distribution in (1.4):

$$p(x_1, x_2, x_3, x_4) = \sum_{x_5} \frac{p(x_1, x_2, x_3) \times p(x_2, x_3, x_4) \times p(x_3, x_4, x_5)}{p(x_2, x_3) \times p(x_3, x_4)}$$

We can push the summation past all the terms that do not depend on x_5, obtaining

$$\begin{aligned} p(x_1, x_2, x_3, x_4) &= \frac{p(x_1, x_2, x_3) \times p(x_2, x_3, x_4)}{p(x_2, x_3)} \times \frac{\sum_{x_5} p(x_3, x_4, x_5)}{p(x_3, x_4)} \\ &= \frac{p(x_1, x_2, x_3) \times p(x_2, x_3, x_4)}{p(x_2, x_3)}. \end{aligned}$$

But the last expression is simply a decomposable model with junction tree shown in Figure 1.6. Therefore, marginals in decomposable models can be computed by removing one or more leaf cliques and the corresponding separators from the model. Such an operation is called **pruning** a leaf clique. The robust message passing algorithm (Paskin and Guestrin, 2004b) executes such pruning operations in a distributed manner, with only a portion of a decomposable model visible to each node. Importantly, before convergence, the algorithm introduces principled approximations into the model. Therefore, the algorithm obtains accurate results even before all its messages have converged.

1.2.2 Filtering in dynamic Bayesian networks

Many applications require us to reason about a system, whose state changes over time. For example, in a camera network, the object may move from one time instant to another; the temperatures in an environment may also change over time. A **dynamic Bayesian network (DBN)** is a graphical model for describing dynamical systems whose state evolution obeys the Markov assumption. The state of the system at each discrete time step is characterized by a set of **state variables**; the observations at each time step are characterized by a set of **observed variables**. For example, in a camera network, the object is characterized by its position at each time step, each camera is characterized by its pose, and each camera observation is characterized by an observed variable. The relationship among these variables is described by a set of conditional probability distributions. The object motion is described by a **transition model** that describes the distribution of the object position at time t, conditioned on the object position at time $t-1$; this transition model captures the assumption that the object position at time t is independent of the previous positions and the camera poses, given the position at time $t-1$. Each camera measurement is described by a **observation model** that describes the distribution over the possible measurements, given the current object position and the pose of the camera.

A common task in DBNs is **filtering**. In filtering, we wish to compute the posterior distribution over the state variables at the current time step, given all the observations made so far. This posterior distribution is also called the **belief state**. The belief state is typically computed recursively: the filter maintains a "view" (a marginal distribution) over the current state and evolves the view from one time step to another. In each step, the filter starts with the prior distribution that has been conditioned on all the observations in the past, and conditions this distribution on the latest observations. The filter then predicts the distribution at the next time step using the transition model. This approach is similar to the Kalman Filter, but the filter works with a factorized representation of the transition model and the observation model.

Even though the DBN is represented compactly as a collection of conditional probability distributions, the belief state quickly loses conditional independences, and cannot be represented exactly using a compact graphical model. This means that exact filtering can be slow or intractable even in the centralized setting. A standard approach to address this problem is **assumed density filtering**. In assumed density filtering, the belief state is periodically approximated by a model, for which inference is cheaper. One way to approximate the belief state is to project it to a given family of distributions, by minimizing a measure of dissimilarity between the exact and the approximate distribution. A popular measure of dissimilarity between two distributions is the **Kullback-Leibler (KL) divergence**. When projecting a distribution p to a family of decomposable models with a fixed junction tree, minimizing the KL divergence has a particularly simple interpretation: the best approximate distribution $\tilde{p}$ sets the clique and the separator marginals equal to the marginals of the exact distribution p. For example, for the junction tree in Figure 1.5, the best

approximate distribution $\tilde{p}$ can be written as

$$\tilde{p}(x_1, x_2, x_3, x_4, x_5) = \frac{p(x_1, x_2, x_3) \times p(x_2, x_3, x_4) \times p(x_3, x_4, x_5)}{p(x_2, x_3) \times p(x_3, x_4)}$$

regardless of the structure of p. Thus, projection can be obtained simply by computing the marginals over the cliques (and separators) of the junction tree. This property is the basis of the Boyen–Koller algorithm (Boyen and Koller, 1998).

While assumed density filtering can often be carried out efficiently in the centralized case, distributed filtering is substantially more challenging. Since the observations are distributed across the network, nodes must coordinate to incorporate each others observations and propagate their estimates from one time step to the next. Online operation requires the algorithm to degrade gracefully when nodes run out of processing time before the observations propagate throughout the network. Furthermore, the algorithm needs to robustly address node failures and interference that may partition the communication network into several disconnected components. Since distributed filtering is so challenging, it has been typically only considered in the context of specific applications with important simplifying assumptions.

In Chapter 3, we present an efficient distributed algorithm for assumed density filtering in general DBNs. In our algorithm, each node maintains an approximate marginal distribution over one or more cliques, conditioned on the measurements made by the nodes in the network. At each time step, the nodes condition on the observations, using a modification of the robust message passing algorithm, outlined in the previous section, and then advance their estimates to the next time step locally. The algorithm guarantees that, with sufficient communication at each time step, the nodes obtain the same solution as the Boyen–Koller algorithm. Before convergence, the algorithm introduces principled approximations in the form of independence assertions in the node estimates and in the transition model.

In the presence of unreliable communication or high latency, the nodes may not be able to condition their estimates on all the observations in the network, for example, when interference causes a network partition, or when high latency prevents messages from reaching every node. Once the estimates are advanced to the next time step, it is difficult to condition on the observations made in the past. Hence, the beliefs at the nodes may be conditioned on different evidence and no longer form a consistent global probability distribution over the state space. We show that such inconsistencies can lead to poor results when nodes attempt to combine their estimates. Nevertheless, it is often possible to use the inconsistent estimates to form an informative globally consistent distribution; we refer to this task as **alignment**. We propose an online algorithm, **optimized conditional alignment** (OCA), that obtains the global distribution as a product of conditionals from local estimates and optimizes over different orderings to select a global distribution of minimal entropy. The algorithm integrates tightly with the pruning operations, performed by the robust message passing algorithm. We also propose an alternative, more global optimization approach that minimizes a KL divergence-based criterion and provides accurate solutions even when the communication network is highly fragmented.

1.2.3 Maximum-likelihood estimation for modular robot localization

In many cases, computing the exact marginals of a distribution is intractable even in static models. The reasons are twofold. First, the distribution may not be representable with a "thin" junction tree, that is, a decomposable model where each clique has only a small number of variables. This is an obstacle for exact inference, because the complexity of the marginalization operations increases substantially with the clique size. Second, the distribution may have a complicated shape, and may not be representable in a compact parametric form that permits marginals to be computed with simple formulas. In these models, a different inference method is needed.

Large modular robots are an example of systems that encounter this kind of distributions. To estimate the relative poses among its modules, the robot has access to internal readings between pairs of neighboring modules. Each reading is a noisy observation of the relative pose between two adjacent modules. The readings are conditionally independent given the module poses, and each reading is wholly determined by the relative pose of the two neighboring modules. Thus, the observation model can be written as a product of observation models for the individual observations $Z_{i,j}$:

$$p(\mathbf{z} \,|\, \mathbf{l}) = \prod_{i,j} p(z_{i,j} \,|\, l_i, l_j), \tag{1.5}$$

where l_i characterizes the pose of module i, and $\mathbf{l}$ is the vector of all the poses. Instantiating the readings $Z_{i,j} = \overline{z}_{i,j}$ in (1.5) yields the joint likelihood

$$p(\overline{\mathbf{z}} \,|\, \mathbf{l}) = \prod_{i,j} p(\overline{z}_{i,j} \,|\, l_i, l_j). \tag{1.6}$$

While (1.6) appears not much more complicated than the likelihoods we have seen above—each observation likelihood $p(\overline{z}_{i,j} \,|\, l_i, l_j)$ has two variables as its arguments, rather than one—the joint likelihood is substantially more difficult to work with. One reason is that the observation likelihoods form a tight mesh, as the one shown in Figure 1.7. Junction tree methods are not effective in this setting, because each clique needs to contain variables for *an entire cross-section* of the modular robot, which is very computationally intensive. Furthermore, the observation likelihood has non-linearities, because it contains a rotational component of the module pose. Therefore, a standard approach in these systems is to recover the **maximum-likelihood estimate**,

$$\mathbf{l}^* = \underset{\mathbf{l}}{\operatorname{argmax}}\, p(\overline{\mathbf{z}} \,|\, \mathbf{l}), \tag{1.7}$$

which represents the mode of the likelihood, but not the degree of uncertainty.

While computing a maximum-likelihood estimate is typically simpler than computing exact marginals of a distribution, the task is still not easy. Due to non-linearities, the optimization problem (1.7) is difficult to solve. Simple methods such as gradient descent perform very poorly, and even advanced methods (Grisetti et al., 2007b) may get stuck in local optima that arise due to the periodicity of the angle. Thus, in this

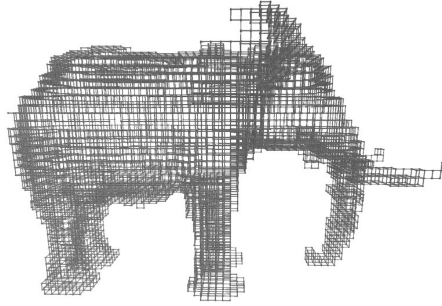


Figure 1.7: A Markov network for the elephant rendering in Figure 1.1(b).

case, even the centralized problem is difficult to solve.

To address this challenge, we propose a top-down approach that hierarchically partitions the connectivity graph into sub-problems and reduces global positioning errors. A key insight is that, in the case of a modular ensemble, the greatest error will tend to accumulate in regions with only a few inter-module observations. We call such regions **weak**. If we selectively incorporate the observations in densely connected regions first, the partial solution will be constrained and the error will be substantially reduced. We use this intuition to formulate a hierarchical algorithm, where we recursively split the graphical model into well-connected components using normalized cut (Shi and Malik, 2000). The partial solutions for the individual components are then merged using the observations between the components, by exploiting problem-specific structure of our observation model. Our distributed algorithm mirrors the operations of this centralized algorithm, by specifying a leader for each component. The leader performs certain key operations of the algorithm and distributes out the result.

1.2.4 Matrix factorization for collaborative filtering

We have mentioned recommender systems as one application where observations from multiple nodes need to be integrated. A popular approach to recommender systems is **collaborative filtering**. In collaborative filtering, the system provides recommendations based on opinions of other like-minded users. The user's opinion of an item is captured by a **rating**. The system observes the ratings for a subset of the user-item pairs, and needs to predict the ratings for the unobserved pairs. Thus, collaborative filtering is a form of a *missing value estimation* problem.

A standard approach to collaborative filtering is **matrix factorization**. In matrix factorization, each user u is associated with a vector of parameters ν_u, and each item i is associated with a vector of parameters θ_i. The rating $\hat{r}_{u,i}$ is predicted as a function $r(\nu_u, \theta_i)$ of the user and item parameters; the function is chosen in advance. The graphical model for this system is shown in Figure 1.8. Note that the graphical model has a very simple structure: it merely captures the fact that the prediction function r depends on the hidden parameters ν_u and θ_i. The parameters ν_u and θ_i for each user u and each movie i are typically

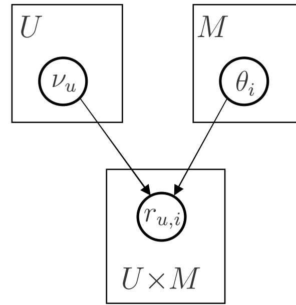


Figure 1.8: The graphical model for matrix factorization. The boxes indicate the repeated parts of the model, where U is the total number of users, and M is the total number of items.

estimated from data using stochastic gradient descent that minimizes the prediction error between the observed ratings $r_{u,i}$ and the predicted ratings $\hat{r}_{u,i}$.

In the distributed collaborative filtering problem, the ratings reside on the user nodes, rather than a centralized server. Together, the user nodes need to train the model parameters and predict the ratings for their respective users. The distributed collaborative filtering problem is substantially more challenging than the centralized one. The nodes need to train the model parameters without ever seeing the entire data at once. Furthermore, the stochastic gradient descent algorithm is a purely sequential algorithm that is hard to distribute or parallelize, and its parallel approximations work best for small node counts (Langford et al., 2009). Finally, the distributed algorithm should be robust to node fluctuations, prevalent in peer-to-peer networks, and should require only a modest amount of communication.

To address these challenges, we propose to use a two-level **super-peer architecture** (Yang and Garcia-Molina, 2003), where a subset of stable nodes with access to a high-bandwidth connection perform most of the computation and act as servers to the remaining nodes. Each super-peer collects the observed ratings from a subset of the clients, and executes the centralized stochastic gradient descent algorithm. The super-peer estimates all the item parameters, but only a subset of the parameters for the users, from whom it has collected the data. Periodically, the nodes synchronize their state, by averaging their item parameters. We demonstrate experimentally that our algorithm performs as well as a centralized algorithm, while requiring substantially less computation at each super-peer.

1.3 Coordination with overlay networks

While we have described the structure of algorithms in this dissertation and how they leverage graphical models, we have not discussed how they are implemented on real networks. All of our algorithms require some form of large-scale coordination among the nodes, because they rely on properties that are not local to the node and its immediate physical neighbors. To perform this coordination, the algorithms build distributed data structures, called overlay networks. Overlay networks are fully decentralized and respect

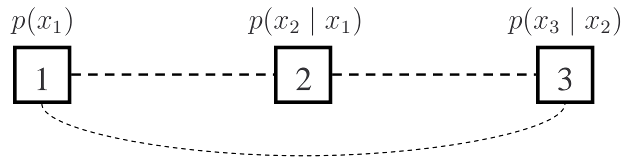


Figure 1.9: A small scenario, where node 3 wishes to compute the marginal over variable X_3.

the communication constraints of any given physical network they were built on top. The type of the overlay network and its function in the algorithm vary from one problem to another; we briefly outline them in the sections below.

1.3.1 Coordination for computing marginals

Coordination among the nodes is necessary when computing marginals of a distribution. Consider the scenario in Figure 1.9 with three nodes, where all three nodes are in communication range, but the link between nodes 1 and 3 is weaker than the other two links. The nodes carry the factors of the distribution in (1.1), and suppose that node 3 wishes to compute a marginal over the variable X_3. To compute this marginal, nodes 1 and 2 could communicate their factors to node 3 and let node 3 do all the work. However, doing so would not be efficient, as node 3 would need to receive all the marginals of the distribution and reason about the entire model on its own.

Instead, we can leverage the fact that the computation in Equation 1.3 is “tree-like”: the intermediate result from the inner summation contributes to exactly one outer summation, and there are no circular dependences among the results of different summations. Therefore, if we were to arrange the nodes into a spanning tree, as shown in Figure 1.10, the nodes would be able to implement (1.3) in a distributed fashion by passing messages along this tree: node 1 would send its factor $p(x_1)$ to node 2, node 2 would then multiply this incoming message with its local factor $p(x_2 \mid x_1)$ and marginalize out X_1 to compute the inner sum in (1.3) and send this intermediate result $p(x_2)$ to node 3, and finally, node 3 would compute the outer sum in (1.3). In order to determine which variables are “safe” to marginalize out, the nodes can annotate the vertices and the edges of the spanning tree with variable sets. Each node is associated with a **clique**—a set of variables that includes the arguments of its local factors, as well as any additional variables needed to satisfy the running intersection property. Each undirected edge is then associated with a **separator**, which is simply the intersection of the two adjacent cliques. The separators indicate which variables can be marginalized out: if a variable is not present in the separator, it is not relevant for the nodes on the other side of the tree, and can be safely marginalized out. The spanning tree, along with the cliques and the separators, is called a **network junction tree** (Paskin et al., 2005).

A network junction tree and an ordinary junction tree that describes a decomposable model are similar: they can both be used to perform operations on a probability distribution. However, unlike an ordinary junction tree, whose vertices and edges can be arbitrary (as long as they form a tree and as long as the

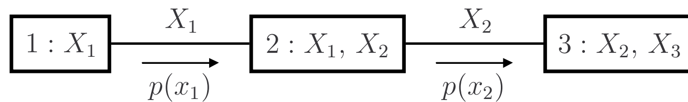


Figure 1.10: A network junction tree for the scenario in Figure 1.9. The rectangles specify the cliques, and the variables above the edges specify the separators. Note that the messages only contain the variables that are present in their respective separators; all the other variables have been marginalized out.

tree satisfies the running intersection property), a network junction tree is tied to the physical network it was built on top of: each vertex of the network junction tree corresponds to a node, and each edge corresponds to a communication link. The tree is adapted to changing network conditions. For example, if a communication link becomes unreliable, the nodes select an alternative link that requires fewer retries for a successful transmission. The computation and communication cost of the inference algorithm is determined by the clique and the separator sizes, respectively, and the tree can be automatically tuned to minimize the overall power usage, by trading the communication and computational complexity. Thus, a network junction tree explicitly handles unreliable communication and constrained resources of the distributed system.

While we have described network junction trees in the context of factorized probability models, network junction trees can also be used to compute marginals in decomposable models. The only difference is that, with decomposable models, multiple cliques may be pruned in a single message computation, and the rule for determining which cliques can be pruned is more complicated. We will discuss this rule when we review the algorithm of Paskin and Guestrin (2004b) in Chapter 2.

1.3.2 Coordination for maximum-likelihood estimation

Maximum-likelihood estimation may also require coordination among the nodes. Recall that our algorithm for modular robot localization performs *global* moves, by merging two partial solutions into a single one. A global move is implemented by computing the optimal rigid transform (translation and rotation) of one component relative to the other one. This transform can be computed by aggregating summary statistics of the observations between the two components to an arbitrarily chosen leader node. To compute this aggregate, the nodes construct a simple overlay network—a tree, rooted at the leader. For simplicity, the tree spans all the nodes in the two merged components, not just the nodes along the boundary between the two components. Once the tree has been constructed, the nodes compute the aggregate statistics incrementally, starting from the leaves of the tree.

There is one important difference between the overlay networks in maximum-likelihood estimation and the network junction tree. Typically, only a single network junction tree is constructed for the entire network, because the network reasons about one global model. On the other hand, due to the hierarchical nature of our maximum-likelihood estimation algorithm, multiple parallel spanning trees are constructed at any given time in the network. The parallel construction of these spanning trees and synchronization

across multiple levels of hierarchy poses a significant implementation challenge. In Chapter 5, we leverage recent advances in distributed declarative programming languages (Ashley-Rollman et al., 2009) to obtain a fully distributed implementation of our algorithm.

1.3.3 Coordination for collaborative filtering

In peer-to-peer networks, a key coordination problem is **lookup**, determining a node responsible for certain part of data storage or computation. Lookups takes on two forms in our algorithm. First, a super-peer may need to select a random neighbor for parameter averaging. Or, a client may need to determine which super-peer it should connect to. These problems are challenging, because no single node has a complete knowledge of all other nodes in the network to select the correct node.

Both these problems can be addressed using a **distributed hash table**. Conceptually, a distributed hash table provides a single operation: given a key, it determines the node responsible for handling this key. A distributed hash table is similar in function to a (centralized) hash table, but has two important differences: the hash table is represented as a set of pointers among the nodes in the network, rather than pointers among cells in memory. Furthermore, the hashing is **stable**: as nodes enter or leave the network, the data structure never requires a complete rehash. Rather, as nodes enter or leave the network, the mapping between keys and nodes is updated incrementally. This property lets distributed hash tables scale to very large networks and handle node arrivals, departures, and failures.

Distributed hash tables can also provide reliable data storage, by redundantly storing data (that is, observed ratings) on nodes with similar keys. Traditionally, redundant storage has been used to provide improved availability of data, for example, if a subset of super-peers leave the network. Perhaps surprisingly, we will demonstrate in Chapter 6 that redundant storage can be used to *speed up the convergence* of an inference algorithm.

1.4 Overview of the thesis

The organization of this thesis is outlined below:

Chapter 2: Robust distributed probabilistic inference. In this chapter, we review the probabilistic graphical models and some of the centralized methods, relevant for this thesis. We then formally define the distributed probabilistic inference problem and review the robust message passing algorithm (Paskin and Guestrin, 2004b) for solving this problem robustly, in face of communication delays and node failures.

Chapter 3: Distributed inference in dynamical systems. In this chapter, we consider the general problem of approximate distributed filtering in systems modeled with dynamic Bayesian networks. We

show that the problem can be reduced to a sequence of static inference steps, solved using the robust message passing algorithm. We then identify the belief inconsistency problem that arises in distributed systems, and propose two algorithms for selecting a set of consistent, informative beliefs. We demonstrate the convergence and the robustness of our algorithm on a novel application, simultaneous localization and tracking for networked cameras.

Chapter 4: Simultaneous localization and tracking for camera networks. In this chapter, we elaborate on the modeling aspects in localization of networked cameras. The camera observation model has non-linearities, which prevent the posterior distribution to be directly represented as Gaussian. We present two novel techniques to address the non-linearities and demonstrate that the resulting online, fully Bayesian approach is competitive with a state-of-the-art offline optimization approach.

Chapter 5: Localization in large-scale modular robot ensembles. In this chapter, we turn to our second application, localization in large-scale modular robot ensembles. We cast localization as a maximum-likelihood estimation problem. We demonstrate that state-of-the-art estimation algorithms can fail to recover a desirable solution, and we present a novel hierarchical algorithm that selects an effective ordering of observations using a normalized cut criterion. We present a fully distributed implementation of our algorithm, whose communication complexity scales logarithmically with the size of the ensemble.

Chapter 6: Collaborative filtering in peer-to-peer networks. This chapter presents our third application, collaborative filtering in peer-to-peer networks. We present a simple approach to parallelize an otherwise sequential matrix factorization algorithm. We then present a fully distributed implementation using a super-peer architecture. We demonstrate that our algorithm scales and is robust to network fluctuations.

We claim the following contributions.

Chapter 2: We present a new, simpler proof of a key theorem in (Paskin, 2004).

Chapter 3: We present a simple reduction for solving the distributed filtering problem approximately using the robust message passing algorithm. We identify the belief inconsistency problem that arises in distributed systems and address this problem using a distributed algorithm for optimized conditional alignment (OCA) and a centralized algorithm for joint alignment.

Chapter 4: The primary contribution of this chapter is relative over-parameterization (ROP), a novel parameterization of camera pose that permits camera poses to be represented with a simple Gaussian. A secondary contribution is a novel linearization method, hybrid conditional linearization, to condition on measurements with a non-linear observation model.

Chapter 5: We present a novel centralized algorithm that scales to large ensembles and obtains significantly more accurate solutions than state-of-the-art algorithms. Furthermore, we present a concise

distributed implementation using a declarative programming language.

Chapter 6: We present a distributed algorithm for collaborative filtering that scales, is robust to network fluctuations, and achieves the same accuracy as the corresponding centralized algorithm. We demonstrate the algorithm on the PlanetLab testbed.

In most chapters, the contributions lie in new distributed algorithms. Chapters 4 and 5 also present novel centralized approaches.

Most of the work described in this thesis has been previously published in conference articles and a journal publication. The distributed filtering algorithm and its application to camera localization was first published in (Funiak et al., 2006a,b). The work on localization in modular robot ensembles was published in (Funiak et al., 2008b, 2009).

Chapter 2

Robust distributed probabilistic inference

Many systems can be described by a probability distribution that relates the hidden and the observed phenomena. A central problem in such systems is computing one or more marginals of this distribution, conditioned on all the observations made in the network; this problem is called **distributed probabilistic inference**. In this chapter, we review an algorithm, called **robust message passing** (Paskin and Guestrin, 2004b), which addresses the distributed probabilistic inference problem robustly, in face of communication delays and node failures. We provide a simplified interpretation of the algorithm and a simpler proof of a key theorem. For brevity, we focus on the aspects of the algorithm that are important in this thesis; for a more complete presentation, see (Paskin, 2004, Chapter 6).

2.1 Centralized probabilistic inference

We begin by reviewing relevant concepts and algorithms in centralized probabilistic inference. We assume familiarity with basic probability theory and basic graph theory. The algorithms in this chapter work with distributions over multiple variables. We first review multivariate distributions and some of their properties. Then we describe a standard way of representing multivariate distributions compactly, using factorized probability models. Factorized probability models often permit efficient inference; we review the inference methods based on variable elimination. These inference methods, along with a representation of the distribution as decomposable models, form the foundation of the robust message passing algorithm. We conclude by discussing conditioning on observations. This section serves only as a brief introduction; for a complete treatment, see (Koller and Friedman, 2009).

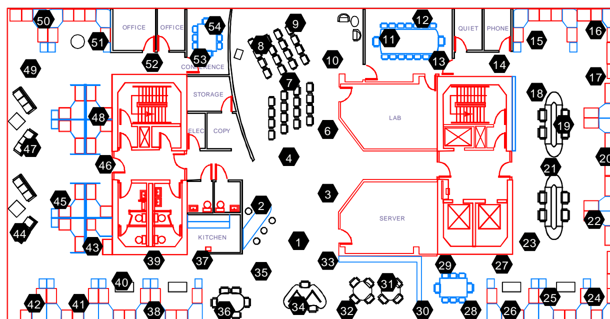


Figure 2.1: A deployment of a sensor network at Intel Research Lab, Berkeley. The temperatures variations in the environment are described by random variables at a set of fixed locations.

2.1.1 Multivariate distributions

Complex systems can often be described in terms of a collection of random variables that characterize the aspects of interest. For example, the temperature variations in an office environment, such as one shown in Figure 2.1, can be described by a collection of real-valued random variables that characterize the temperatures at some fixed set of locations. Likewise, the student's performance in a class can be described by several discrete variables, including one that characterizes the student's intelligence and one that represents his or her grade. In this chapter, we focus on systems that can be characterized by a finite number of random variables $\mathbf{X} = \{X_a : a \in V\}$, where V is a finite set of indices. We will denote a joint assignment to these variables by $\mathbf{x}$; this assignment specifies a value x_a for each index $a \in V$. For any subset of indices $S \subseteq V$, the set $\mathbf{X}_S = \{X_a : a \in S\}$ denotes a subset of the variables indexed by S, while $\mathbf{x}_S$ denotes a partial assignment to the variables $\mathbf{X}_S$.

Probabilistic models often take on a form of a **probability distribution** that specifies the probability for any (measurable) set of assignments $\mathbf{x}$ to the random variables $\mathbf{X}$. For discrete random variables, the distribution can be described by a probability mass function $p(\mathbf{x})$ that specifies the probability for each assignment $\mathbf{x}$; an example of a probability mass function for the student scenario is shown in Figure 2.2. When $\mathbf{X}$ are continuous, we will assume that the distribution has a probability density function $p(\mathbf{x})$ that specifies the relative likelihood for the variables $\mathbf{X}$ to take on the values $\mathbf{x}$. For any subset of indices $S \subseteq V$, the probability mass function of the marginal distribution

$$p(\mathbf{x}_S) = \sum_{\mathbf{x}_{V-S}} p(\mathbf{x}) \tag{2.1}$$

represents the probability of each joint assignment to the variables $\mathbf{X}_S$; here, $V - S$ are the indices of all the variables in $\mathbf{X}$, other than S. When $\mathbf{X}$ are continuous, the marginal density

$$p(\mathbf{x}_S) = \int p(\mathbf{x})\, \mathrm{d}\mathbf{x}_{V-S} \tag{2.2}$$

		Intelligence low	Intelligence high	
Grade	A	0.07	0.18	0.25
	B	0.28	0.09	0.37
	C	0.35	0.03	0.38
		0.7	0.3	1

Figure 2.2: Example of a joint distribution $p(grade, intelligence)$, taken from (Koller and Friedman, 2009). The last column and the last row indicate the marginal distribution over the *Grade* and *Intelligence*, respectively.

represents the relative likelihood among various assignments $\mathbf{x}_S$ to the variables $\mathbf{X}_S$. The concepts presented in this chapter apply equally to both discrete distributions and continuous distributions, and we will often call $p(\mathbf{x})$ simply as "the distribution" and $p(\mathbf{x}_S)$ as "the marginal distribution," with the understanding that we refer to the distribution's probability mass function or its density. Furthermore, to streamline the notation, we will use the discrete summation (2.1) even if the distribution is continuous.

It is often useful to reason about the distribution over some set of variables, given the knowledge of another set of variables. For example, we may be interested in describing the intelligence of the student, given that they received the grade A. For two disjoint sets of indices $S, T \subseteq V$, the **conditional distribution** over $\mathbf{X}_S$ given the assignment $\mathbf{x}_T$ is defined as

$$p(\mathbf{x}_S \,|\, \mathbf{x}_T) = \frac{p(\mathbf{x}_S, \mathbf{x}_T)}{p(\mathbf{x}_T)}. \tag{2.3}$$

For example, the conditional distribution $p(intelligence \,|\, Grade = A)$ is $[0.07\ 0.18]/0.25 = [0.28\ 0.72]$. Note that this distribution is different from the marginal distribution $p(intelligence)$ (shown as the last row in Figure 2.2), because the knowledge of the grade gave us some information about the student's intelligence. Nevertheless, in some cases, the knowledge of one random variable gives us *no* information about another random variable, such as when the variables represent the outcomes of two separate coin tosses. Formally, two disjoint sets of variables $\mathbf{X}_S$ and $\mathbf{X}_T$ are **marginally independent**, denoted by $\mathbf{X}_S \perp \mathbf{X}_T$, if the marginal distribution $p(\mathbf{x}_S)$ and the conditional distribution $p(\mathbf{x}_S \,|\, \mathbf{x}_T)$ are equal for all assignments $\mathbf{x}_S$ and $\mathbf{x}_T$. This condition is equivalent to saying that $p(\mathbf{x}_S, \mathbf{x}_T) = p(\mathbf{x}_S) \times p(\mathbf{x}_T)$; thus, marginal independence is a symmetric notion.

Marginal independence is a strong property that rarely occurs in practice. Nevertheless, sometimes the knowledge of one random variable gives us no information about another random variable *if we already know the value of a third one*. For example, suppose that the student takes an SAT test. Clearly, the SAT scores and the grade are not independent: if we condition on the fact that the student receives high SAT scores, his or her chances of getting a good grade increase. Nevertheless, if we know that the student is intelligent, the fact that he or she received high SAT scores gives us no information about the grade; the grade and the SAT scores are said to be **conditionally independent**, given the intelligence. In

general, for three pairwise-disjoint sets of indices $S, T, U \subseteq V$, the variables $\mathbf{X}_S$ and $\mathbf{X}_T$ are conditionally independent given $\mathbf{X}_U$, denoted by $\mathbf{X}_S \perp \mathbf{X}_T \mid \mathbf{X}_U$, if

$$p(\mathbf{x}_S \mid \mathbf{x}_T, \mathbf{x}_U) = p(\mathbf{x}_S \mid \mathbf{x}_U), \tag{2.4}$$

for all assignments $\mathbf{x}_S$, $\mathbf{x}_T$, and $\mathbf{x}_U$. When the set U is empty, conditional independence is equivalent to marginal independence.

2.1.2 Factorized probability models

In Figure 2.2, we were able to write the complete probability mass function as a table, because the table contained only six entries, one for each joint assignment $\mathbf{x} = (\textit{grade}, \textit{intelligence})$. Unfortunately, in general discrete distributions, each variable adds a dimension to the table of probabilities, so the representation size grows *exponentially* with the number of variables $|V|$, making it very costly to store a distribution over many variables. This is a problem for three reasons. First, the table may not fit to the disk or the memory. Second, each entry of the table is a free parameter of the distribution, and neither domain-specific knowledge nor automatic learning of the model may provide so many free parameters accurately. Finally, inference with such a large model may be intractable. Therefore, a different representation of the distribution is required.

A standard way to specify the distribution more compactly is to write it as a **factorized probability model**:

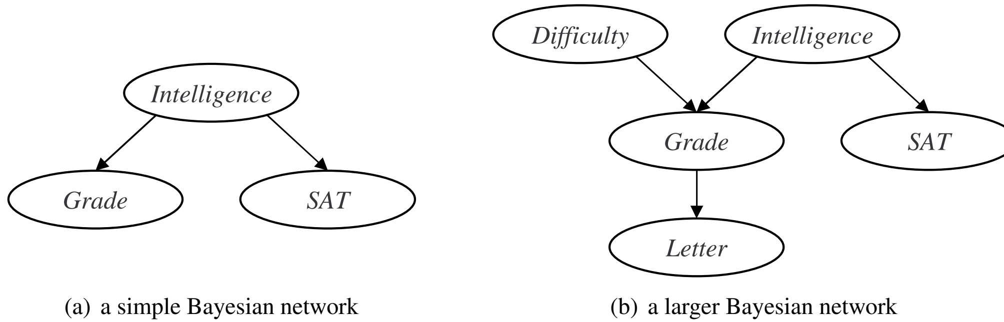
$$p(\mathbf{x}) = \frac{1}{Z} \prod_{k=1}^{K} \psi_k(\mathbf{x}_{A_k}). \tag{2.5}$$

Here, each **factor** ψ_k is typically a function over a small number of variables with indices $A_k \subseteq V$, and Z is a normalization constant ensuring that the distribution sums to 1. For example, when the distribution is discrete, each factor can itself be represented as a table of values; since the table has a small number of dimensions, its size remains bounded, and the representation requires only a small number of parameters overall.

One way to obtain a factorized probability model is to exploit the conditional independence assumptions in the distribution $p(\mathbf{x})$. Following the example from the previous section, we can write the joint distribution over *Intelligence*, *Grade*, *SAT* as

$$\begin{aligned} p(\textit{intelligence}, \textit{grade}, \textit{sat}) &= p(\textit{intelligence}) \times p(\textit{grade}, \textit{sat} \mid \textit{intelligence}) \\ &= p(\textit{intelligence}) \times p(\textit{grade} \mid \textit{intelligence}) \times p(\textit{sat} \mid \textit{intelligence}), \end{aligned}$$

where the first equality follows from the chain rule, and the second equality follows from the assumption that *Grade* and *SAT* are conditionally independent, given *Intelligence*. The dependencies can be captured



(a) a simple Bayesian network

(b) a larger Bayesian network

Figure 2.3: Two Bayesian networks for the Student example, taken from (Koller and Friedman, 2009). (a) A small network, where the variables *Grade* and *SAT* are conditionally independent, given *Intelligence*. (b) A larger network with two additional variables.

by a directed graph shown in Figure 2.3(a), where each variable corresponds to one vertex, and there is an edge from a to b if X_b conditions on X_a. The graph, together with the conditional probability distribution for each individual variable, form a **Bayesian network**. In general, a Bayesian network is a directed acyclic graph G that represents the distribution as a product of conditional probability distributions

$$p(\mathbf{x}) = \prod_{a \in V} p(x_a \mid \mathbf{x}_{\text{Pa}[X_a]}), \tag{2.6}$$

where $\text{Pa}[X_a]$ are the **parents** of the variable X_a in G. A Bayesian network for an extended version of the student example that also captures the difficulty of the course and the quality of a recommendation letter the professor may write for the student is shown in Figure 2.3(b). If the distribution can be written in the form (2.6) for a given graph structure G, we say that the distribution **factorizes according to** G.

A Bayesian network representation is useful, because it represents a set of independence assumptions about the distribution; these independence assumptions hold as long as the distribution factorizes according to G, regardless of the actual values of the conditional probability distributions $p(x_a \mid \mathbf{x}_{\text{Pa}[X_a]})$. The set of all independence represented by a Bayesian network can be identified using the notion of **d-separation**. Intuitively, two variables X_a and X_b are conditionally independent given $\mathbf{X}_U$ if information cannot flow along any path between X_a and X_b in G, given the variables in U. A path that does permit the flow of information is called an **active path**. In order to determine if a path p is active, one only needs to consider the subpaths consisting of two consecutive segments along p. For example, to determine if *SAT* and *Letter* are conditionally independent given *Grade* in Figure 2.3(b), we only need to determine if both subpaths *SAT–Intelligence–Grade* and *Intelligence–Grade–Letter* are active. Figure 2.4 shows the rules for determining if a subpath is active; whether or not a subpath is active depends on the directionality of the edges and on whether the middle variable is observed (that is, being conditioned on). By matching these rules against the scenario in Figure 2.3(b), we see that the subpath *Intelligence–Grade–Letter* is *not* active (because *Grade* is observed), so *SAT* and *Letter* are indeed conditionally independent given *Grade*.

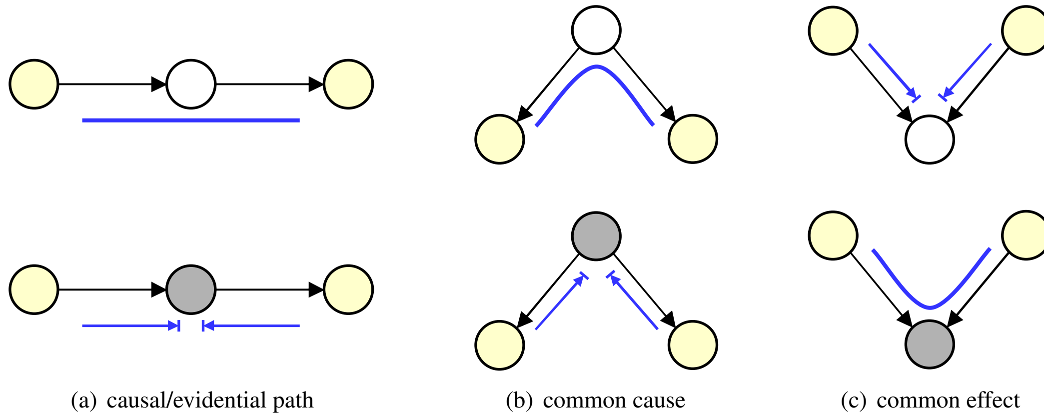


Figure 2.4: The rules for determining when a subpath is active. In the top row, the middle variable is not observed; in the bottom row, the middle variable is observed (whether or not the variables at the endpoints are observed, marked with yellow, is not important). The columns specify the different arrow directions. (a) A causal/evidential path is active if the middle variable is not observed. (b) Similarly, a path where the middle variable is a common parent (representing a common cause) is active, provided that the middle variable is not observed. (c) On the other hand, a path where the middle variable is a common child (representing a common effect) is active, provided that the middle variable or one of its descendants are observed.

An alternative way to obtain a factorized probability model is to use the factors to describe some local properties of the distribution. For example, we can assume that the temperatures at nearby locations in an environment tend to be similar. This property can be captured by a distribution that is written as a product of factors,

$$p(\mathbf{h}) = \frac{1}{Z} \prod_{\{i,j\} \in E} \psi_{i,j}(h_i, h_j), \tag{2.7}$$

where h_i is the temperature (heat) at location i, E is some set of neighboring locations, and each factor $\psi_{i,j}$ is a non-negative function that is high whenever the two temperatures h_i and h_j are similar. If the temperature variations among $\mathbf{h}$ are small, the value of each factor $\psi_{i,j}(h_i,h_j)$ will be high, hence the value of the probability density function for this assignment will be also high. On the other hand, if some of the temperatures are dissimilar, the value of the probability density function will be lower. Note that we only need to specify the factors $\psi_{i,j}$; the normalization constant Z can be represented implicitly.

The distribution (2.7) can be represented graphically, with an undirected graph G that contains a vertex X_a for each variable X_a and an edge X_a–X_b whenever X_a and X_b appear in the same factor. An example of such a graph for six temperature locations is shown in Figure 2.5. The graph, together with the factors in (2.7) form a **Markov network** for the distribution $p(\mathbf{h})$. In general, a factor may be defined over more than two variables, as long as the variables form a clique in the graph. Given a Markov network structure G, if the distribution $p(\mathbf{x})$ can be written as a product of factors over the cliques of G, then the distribution is said to **factorize over** G.

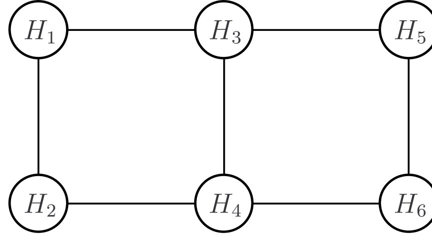


Figure 2.5: A Markov network for a distribution over temperatures at six locations.

Similarly to Bayesian networks, a Markov network graph G represents a set of independence assumptions about the distribution: if the distribution factorizes over G, then some independence assumptions hold, regardless of the values of the factors ψ. The procedure for establishing these independence assumptions is substantially easier than in Bayesian networks. Intuitively, information can flow between X_a and X_b along some path in G if the path is not "blocked" by one of the observed variables $\mathbf{X}_U$. Therefore, if removing the vertices U from G leaves a set of disconnected components, one of which contains X_a and another one which contains X_b, then X_a and X_b are conditionally independent given $\mathbf{X}_U$. For example, in Figure 2.5, the variables H_1 and H_5 are conditionally independent given H_3 and H_4, because removing H_3 and H_4 from the network leaves two disconnected components, $\{H_1, H_2\}$ and $\{H_5, H_6\}$.

An important special class of Markov networks are the Gaussian Markov networks. A Gaussian Markov network represents a multivariate Gaussian (normal) distribution $\mathcal{N}(\mu, \Sigma)$,

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right\}, \tag{2.8}$$

where μ is the **mean vector** that coincides with the mode of the distribution, Σ is a positive semi-definite **covariance matrix** that defines the shape of the distribution, and N is the dimensionality of the state vector $\mathbf{X}$. The parameterization (2.8) is called the **moment form** of a Gaussian, because it is specified using the first two moments $\mu = \mathbb{E}[\mathbf{X}]$ and $\Sigma = \mathbb{E}[(\mathbf{X}-\mu)(\mathbf{X}-\mu)^\top]$. Notice that the logarithm of the density $p(\mathbf{x})$ is a quadratic form, so the density can be equivalently written as

$$p(\mathbf{x}) \propto \exp\left\{-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \eta^\top \mathbf{x}\right\}, \tag{2.9}$$

where $\Lambda = \Sigma^{-1}$ is the **information matrix** (also called the precision matrix), and $\eta = \Sigma^{-1}\mu$ is the **information vector**. The parameterization (2.9) is called the **information form** of a multivariate Gaussian distribution and is denoted by $\mathcal{N}^{-1}(\eta, \Lambda)$.

The components of a Gaussian Markov network are **Gaussian factors**. A Gaussian factor over the variable indices $S \subseteq V$ with parameters η and Λ is the factor

$$\mathcal{G}(\mathbf{x}_S; \eta_S, \Lambda_S) \triangleq \exp\left\{-\frac{1}{2}\mathbf{x}_S^\top \Lambda \mathbf{x}_S + \eta^\top \mathbf{x}_S\right\},$$

where η is a $|S|\times 1$ vector and Λ is a $|S|\times|S|$ symmetric matrix. Unlike an information form of a Gaussian, the parameter matrix Λ does not need to be positive definite. This permits the factor to represent a broader range of functions, such as the unity (with Λ and η equal to zero). Because the logarithm of the factor is linear in the parameters η and Λ, the product of two Gaussian factors simply adds their corresponding parameters:

Lemma 2.1. *Let S, T, and U be disjoint set of variable indices. Then if*

$$\begin{aligned} \psi &= \mathcal{G}\left(\begin{bmatrix}\mathbf{x}_S\\ \mathbf{x}_T\end{bmatrix}; \begin{bmatrix}\eta_S\\ \eta_T\end{bmatrix}, \begin{bmatrix}\Lambda_{SS} & \Lambda_{ST}\\ \Lambda_{TS} & \Lambda_{TT}\end{bmatrix}\right) \\ \psi^* &= \mathcal{G}\left(\begin{bmatrix}\mathbf{x}_T\\ \mathbf{x}_U\end{bmatrix}; \begin{bmatrix}\eta_T^*\\ \eta_U^*\end{bmatrix}, \begin{bmatrix}\Lambda_{TT}^* & \Lambda_{TU}^*\\ \Lambda_{UT}^* & \Lambda_{UU}^*\end{bmatrix}\right), \end{aligned}$$

then the product $\psi \times \psi^*$ is

$$\mathcal{G}\left(\begin{bmatrix}\mathbf{x}_S\\ \mathbf{x}_T\\ \mathbf{x}_U\end{bmatrix}; \begin{bmatrix}\eta_S\\ \eta_T+\eta_T^*\\ \eta_U^*\end{bmatrix}, \begin{bmatrix}\Lambda_{SS} & \Lambda_{ST} & \mathbf{0}\\ \Lambda_{TS} & \Lambda_{TT}+\Lambda_{TT}^* & \Lambda_{TU}^*\\ \mathbf{0} & \Lambda_{UT}^* & \Lambda_{UU}^*\end{bmatrix}\right). \tag{2.10}$$

Gaussian factors also support efficient division, conditioning, and summation, see (Paskin, 2004, Appendix 2.B).

Note that in Lemma 2.1, the variables $\mathbf{X}_S$ and $\mathbf{X}_U$ did not appear together in either factor, and the corresponding entry of the information matrix in (2.10) for these two sets of variables was $\mathbf{0}$. This property holds in general: if we take a collection of Gaussian factors $\{\psi_k(\mathbf{x}_{A_k})\}$, then their product will have zeros in the information matrix for all pairs of variable indices (a, b) that do not appear together in any A_k. This property sheds some light on the relationship between a Gaussian Markov network G and the information form $\mathcal{N}^{-1}(\eta,\Lambda)$ of the same distribution $p(\mathbf{x})$: if there is no edge between a pair of variables X_a and X_b in G, then $\Lambda_{a,b}=0$. Thus, sparse Gaussian Markov networks correspond to Gaussian distributions with a sparse information matrix.

2.1.3 Variable elimination and junction trees

A joint distribution $p(\mathbf{x})$ can be used to answer queries about the system. A common query type is a **marginal probability query**, where we wish to compute the marginal distribution $p(\mathbf{x}_Q)$ over a set of **query variables** $Q \subseteq V$. For example, in a temperature network in Figure 2.5, we may wish to compute the marginal distribution $p(h_i)$, which represent our belief about the temperature at location i before any observations are made.

To compute the answer to such a query, a standard approach is to marginalize out all the other variables $V - Q$ from the distribution $p(\mathbf{x})$. For example, consider a simpler version of the model in Figure 2.5 that

only contains temperatures at locations $1, \ldots, 4$. The marginal $p(h_4)$ can be written as

$$p(h_4) \propto \sum_{h_3} \sum_{h_2} \sum_{h_1} \psi_{1,2} \times \psi_{1,3} \times \psi_{2,4} \times \psi_{3,4}. \tag{2.11}$$

If we were to perform this summation naively, its computational complexity would be cubic in the number of variables (because computing marginals in Gaussians in the information form requires an inversion). If p were a discrete distribution, computing the sum (2.11) naively would be even more expensive, as we would have exponentially many terms in the sum.

A key idea to computing a marginals of the distribution efficiently is to "push in" some of the summations, performing them only over a subset of the factors:

$$p(h_4) \propto \sum_{h_3} \psi_{3,4} \times \left(\sum_{h_2} \psi_{2,4} \times \left(\sum_{h_1} \psi_{1,2} \times \psi_{1,3} \right) \right). \tag{2.12}$$

Here, we were able to push the inner sum over h_1 past all the factors that did not depend on h_1, so that the sum was performed only over the factors $\psi_{1,2}(h_1,h_2)$ and $\psi_{1,3}(h_1,h_3)$. The result of this summation is a new factor over the variables H_2 and H_3 that is used in the next sum over h_2. Equation 2.12 is an example execution of the **variable elimination** algorithm. The variable elimination algorithm sums out one variable at a time. For each variable $a \in V - Q$, the algorithm first collects all the factors that depend on x_a and multiplies them together; the product is called the **pre-elimination factor** and is denoted by ξ_a. The algorithm then sums out x_a from the pre-elimination factor, which results in a **post-elimination factor** ξ_a^*. For example, in (2.12), the product $\psi_{1,2} \times \psi_{1,3}$ is the pre-elimination factor $\xi_1(h_1, h_2, h_3)$, while the result of the inner sum is the post-elimination factor $\xi_1^*(h_2, h_3)$. The post-elimination factor is added to the set of factors processed by the algorithm, and the algorithm proceeds with the next variable.

The computation in the variable elimination algorithm can be captured graphically by a procedure called **vertex elimination**. Vertex elimination starts with a Markov network G for the distribution $p(\mathbf{x})$, and performs a sequence of updates on this graph that mimic the operations of the variable elimination algorithm.[1] For example, when variable elimination computes the pre-elimination factor $\xi_1(h_1,h_2,h_3)$, it temporarily forms a dense representation that "connects" together the factor's arguments H_1, H_2, and H_3. This computation can be captured graphically by placing a clique over the three variables, as shown in Figure 2.6(a). Then, when variable elimination marginalizes out the variable H_1 from ξ_1, vertex elimination removes the variable H_1 (and all its incident edges) from the graph. Thus, vertex elimination interleaves two steps: placing a clique over the arguments of the pre-elimination factor, and removing the variable being summed out from the graph. Variable elimination performed its operations on the factors of the distribution, but we do not need to know the values of the factors to execute vertex elimination. We only need to know, which variables appear in the factors together with the eliminated variable. This information

[1]If the distribution $p(\mathbf{x})$ is specified as a Bayesian network, it can be reinterpreted as a Markov network by disregarding the edge directions and adding some edges.

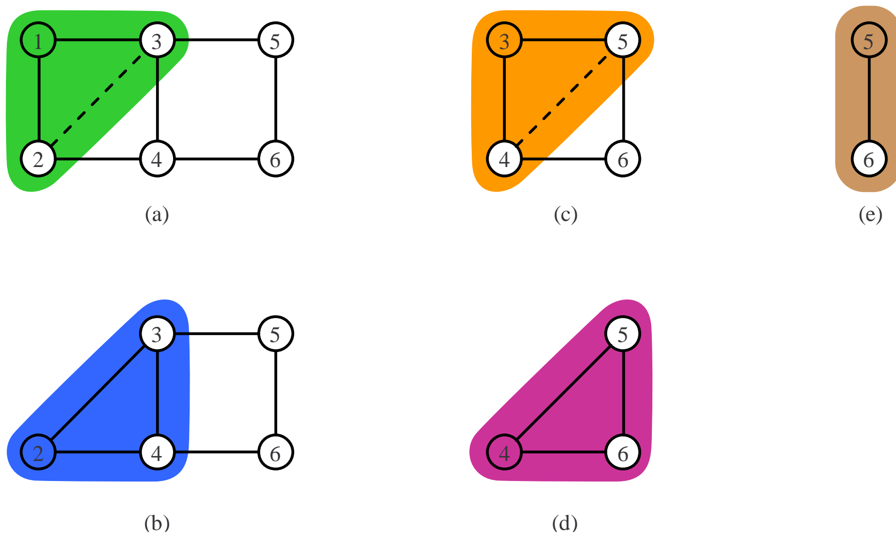


Figure 2.6: Vertex elimination on the Markov network in Figure 2.5. The transparent vertices indicate the variable eliminated in each step; the variables are eliminated in the order $1, \ldots, 5$. The dashed lines indicates the new edges added when the algorithm places a clique over the arguments of the pre-elimination factor.

can be "read off" directly from the graph: these variables are precisely the neighbors of the eliminated variable in G. Therefore, given a Markov network G for a distribution, we can execute vertex elimination on G, without ever looking at the factors of the distribution. This fact is very important, because it lets us cheaply construct an important data structure, discussed below.

The result of running the vertex elimination algorithm is a set of cliques $\mathbf{C}$ that were created during the execution of the algorithm; these cliques are exactly the shaded areas in Figure 2.6. The cliques $\mathbf{C}$ can be used to construct an important data structure, called the *junction tree*. A junction tree is a special kind of a *clique tree*:

Definition 2.1. *A* ***clique tree*** $(T, \mathbf{C})$ *is an undirected tree* T *with vertices* N_T *and edges* E_T*, where each vertex* $i \in N_T$ *is associated with a set of variable indices* $C_i \in \mathbf{C}$*, called the* ***clique*** *at* i*. For each undirected edge* $\{i, j\} \in E_T$*, the* ***separator between*** i ***and*** j *is*

$$S_{i,j} \triangleq C_i \cap C_j,$$

the variable indices that are common to the cliques at vertices i *and* j*.*

A junction tree is a clique tree that satisfies the following property:

Definition 2.2. *A clique tree* $(T, \mathbf{C})$ *satisfies the* ***running intersection property*** *if, for each pair of vertices* $i, j \in N_T$*,* $C_i \cap C_j \subseteq C_k$ *for all vertices* k *along the (unique) path between* i *and* j*. A clique tree that satisfies the running intersection property is called a* ***junction tree****.*

Intuitively, the running intersection property ensures the *flow of information*: if a variable is present in

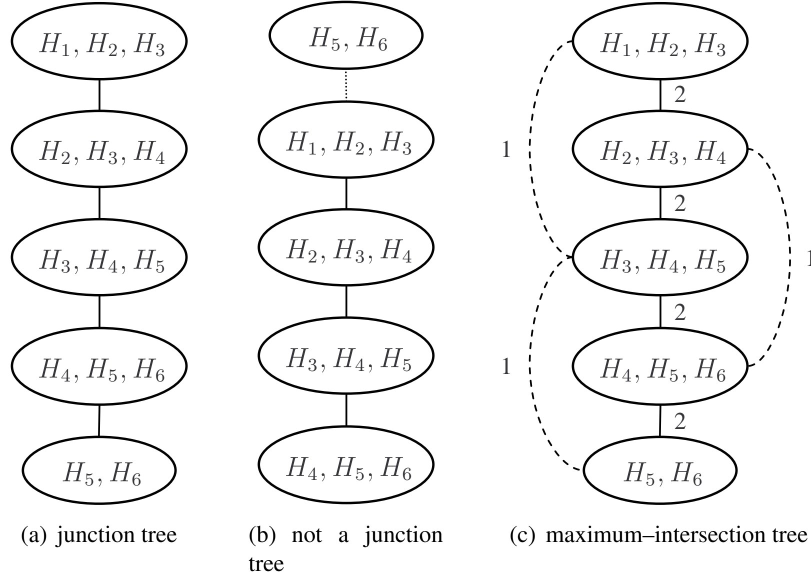


Figure 2.7: (a) A junction tree for the cliques obtained by the vertex elimination algorithm in Figure 2.6. Ellipses indicate the cliques (separators have been omitted for brevity). (b) A clique tree that is not a junction tree. The size of the separator between the cliques $\{H_5, H_6\}$ and $\{H_1, H_2, H_3\}$ is zero, indicating that the clique $\{H_5, H_6\}$ is far from the correct neighbor. (c) A complete graph, where each edge is associated with the weights equal to the cardinality of the intersection between the adjacent cliques (the edges with weight zero are not shown). The edges of the maximum intersection tree are shown as solid lines; the remaining edges of the graph are shown as dashed lines. Note that the maximum intersection tree is a junction tree.

two cliques C_i and C_j, then it is also present in all the cliques along the unique path between i and j. It follows then that the cliques that contain a variable $a \in V$ form a subtree of $(T, \mathbf{C})$. For example, in the junction tree in Figure 2.7(a), the cliques that contain the variable H_3 form a subtree $\{H_1, H_2, H_3\}$—$\{H_2, H_3, H_4\}$—$\{H_3, H_4, H_5\}$.

Junction trees are very important for algorithms that compute exact marginals of a distribution. Junction trees play a central role in the robust message passing algorithm, reviewed later in this chapter, and we will see examples of algorithms using junction trees throughout this chapter and in Chapter 3.

While we have formally defined a junction tree, we have not yet described how to construct one. Intuitively, a clique tree $(T, \mathbf{C})$ may only satisfy the running intersection property if its cliques are linked the right way. For example, if we were to take the clique $\{H_5, H_6\}$ and attach it to the clique $\{H_1, H_2, H_3\}$, as shown in Figure 2.7(b), then the resulting clique tree would not be a junction tree, because the clique $\{H_5, H_6\}$ would be, in some sense, far from other cliques that contain H_5 and H_6. One measure of how close the clique C_i is to a correct neighbor is the size of the separator $|S_{i,j}|$; the larger the separator, the closer the clique is to its correct neighbor. To identify the best neighbors for each clique, we can form a complete graph G over the cliques $\mathbf{C}$, where an edge between two cliques C_i and C_j is associated with a

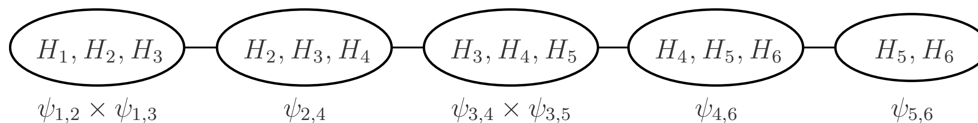


Figure 2.8: Assigning factors of the distribution in Figure 2.5 to the cliques of the junction tree. Each factor $\psi_{i,j}$ is assigned to a clique that includes the arguments H_i, H_j of the factor. The local factor at each clique is the product of the factors assigned to that clique.

weight equal to the size of the cliques' intersection, $w_{i,j} = |C_i \cap C_j|$. Figure 2.7(c) shows this graph for the cliques from the earlier example. Suppose that we form a maximum spanning tree T of G; the tree $(T, \mathbf{C})$ is called a **maximum–intersection tree** for cliques $\mathbf{C}$. Then this tree is a junction tree:

Theorem 2.1. *Let $\mathbf{C}$ be the cliques obtained by running the vertex elimination algorithm for any ordering, and let $(T, \mathbf{C})$ be a maximum–intersection tree for these cliques. Then $(T, \mathbf{C})$ is a junction tree.*

A proof of this theorem can be found in (Jensen and Jensen, 1994).

One benefit of junction trees is that they allow us to compute answers to multiple marginal probability queries at once. Given a junction tree $(T, \mathbf{C})$ for the distribution $p(\mathbf{x})$, we can compute the marginal $p(\mathbf{x}_{C_i})$ for each clique $C_i \in \mathbf{C}$ with the **sum–product algorithm**. The algorithm starts by assigning each factor $\psi_A(\mathbf{x}_A)$ of the distribution to some clique C_i of the tree that includes its argument, $A \subseteq C_i$.[2] Figure 2.8 shows an example assignment of the factors in the temperature network to the junction tree in Figure 2.7(a); the factors at each clique C_i are multiplied together to form the **local factor** $\psi_i(\mathbf{x}_{C_i})$.

The algorithm now proceeds to compute the clique marginals $p(\mathbf{x}_{C_i})$ using dynamic programming. The basic unit of dynamic programming in the sum–product algorithm is a **message** $\mu_{i \to j}$ between every pair of adjacent vertices in the junction tree $(i, j) \in T$. The message $\mu_{i \to j}$ is the product of all local factors on i's side of the tree, marginalized over the variables $S_{i,j}$. For example, the message from the clique $\{H_4, H_5, H_6\}$ to the clique $\{H_3, H_4, H_5\}$ in Figure 2.8 represents the marginal $\sum_{h_6} \psi_{4,6}(h_4, h_6) \times \psi_{5,6}(h_5, h_6)$; in this case, the separator is $\{H_4, H_5\}$, and the variable H_6 has been marginalized out. Formally, the message $\mu_{i \to j}$ is defined recursively as

$$\mu_{i \to j}(\mathbf{x}_{S_{i,j}}) \triangleq \sum_{\mathbf{x}_{C_i - S_{i,j}}} \psi_i(\mathbf{x}_{C_i}) \times \prod_{k \in N_T(i) \setminus j} \mu_{k \to i}(\mathbf{x}_{S_{k,i}}). \tag{2.13}$$

Here, $N_T(i) \setminus j$ are the neighbors of i in T, other than j. Thus, the message from i to j depends on the local factor ψ_i and on the messages from all the neighbors of i, other than j. The message dependencies are illustrated in Figure 2.9(a); since the computation is performed on a tree, the message definition is not circular.

Note that the computation in (2.13) looks very similar to one step of variable elimination: we multiply

[2] Such a clique must exist: let $a \in A$ be the variable that gets eliminated first among A, then the clique corresponding to the pre-elimination factor ξ_a contains A.

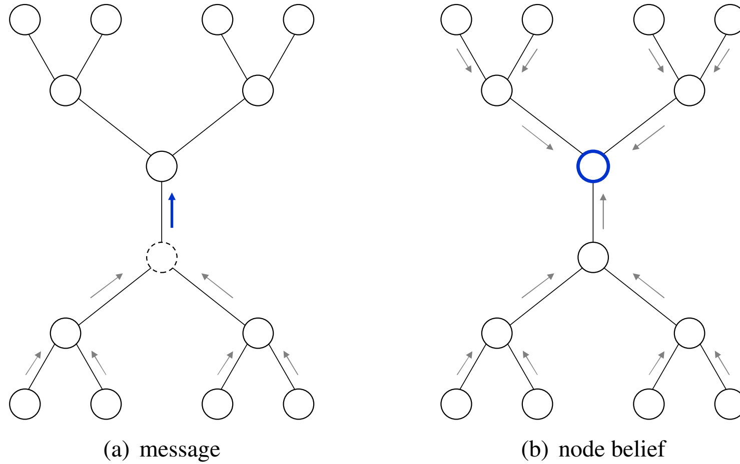


Figure 2.9: Direct and indirect dependencies in the sum–product algorithm, taken from (Paskin, 2004). (a) The dependencies (shown in gray) of the message (shown in blue) from the dashed vertex to its neighbor. The message directly depends on the incoming messages from other neighbors of the dashed vertex and indirectly on all the vertices in the bottom half of the figure. (b) The dependencies of the node belief, shown in blue. The belief directly depends on all the incoming messages towards the blue vertex and indirectly depends on all messages towards the vertex.

the local factor $\psi_i(\mathbf{x}_{C_i})$ together with a number of intermediate results $\mu_{k\to i}(\mathbf{x}_{S_{k,i}})$ and then sum out the nuisance variables $C_i - S_{i,j}$. In fact, if we consider the messages computed towards a specific vertex i, as shown in Figure 2.9(b), the messages of the sum–product correspond exactly to the post-elimination factors of variable elimination, and the messages sent to a specific vertex i represent the final stage of the computation, where all the variables other than C_i have been eliminated. Multiplying these messages together with the remaining factor $\psi_i(\mathbf{x}_{C_i})$ yields the **node belief**

$$\beta_i(\mathbf{x}_{C_i}) \triangleq \psi_i(\mathbf{x}_{C_i}) \times \prod_{j\in N(i)} \mu_{j\to i}(\mathbf{x}_{S_{i,j}}), \tag{2.14}$$

which is equal to the clique marginal $p(\mathbf{x}_{C_i})$ (up to the normalization constant Z).

2.1.4 Decomposable models

In the previous section, we have defined junction trees and discussed how they can be used to guide the computation in the sum–product algorithm. Junction trees can also be used to *parameterize a distribution*. Consider once again the distribution in Figure 2.3(a) that describes the student's performance in a class and on an SAT test. Recall that in this distribution, *Grade* and *SAT* are conditionally independent given

Figure 2.10: A junction tree for the distribution in Figure 2.3(a). The rectangle indicates the separator.

Intelligence. The distribution can thus be written as

$$\begin{aligned} p(\mathit{intelligence}, \mathit{grade}, \mathit{sat}) &= p(\mathit{intelligence}, \mathit{grade}) \times p(\mathit{sat} \,|\, \mathit{intelligence}) \\ &= \frac{p(\mathit{intelligence}, \mathit{grade}) \times p(\mathit{intelligence}, \mathit{sat})}{p(\mathit{intelligence})}. \end{aligned}$$

The first equality follows from the chain rule and the independence assumption $\mathit{Grade} \perp \mathit{SAT} \,|\, \mathit{Intelligence}$, while the second equality follows from the definition of a conditional distribution (2.3). Figure 2.10 shows a junction tree for this distribution. Note that we have expressed the distribution as a product of clique marginals, divided by the separator marginal.

In general, given a junction tree $(T, \mathbf{C})$ for a distribution, the distribution can be written as a ratio of clique and separator marginals:

Definition 2.3. *A **decomposable model** is a junction tree $(T, \mathbf{C})$, where each vertex $i \in N_T$ is associated with a **clique marginal** $p(\mathbf{x}_{C_i})$, and each edge $\{i, j\} \in E_T$ is associated with a **separator marginal** $p(\mathbf{x}_{S_{i,j}})$. A decomposable model represents the distribution $p(\mathbf{x})$ as a ratio*

$$p(\mathbf{x}) = \frac{\prod_{i \in N_T} p(\mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E_T} p(\mathbf{x}_{S_{i,j}})}. \tag{2.15}$$

The marginals $p(\mathbf{x}_{C_i})$ and $p(\mathbf{x}_{S_{i,j}})$ can be computed either using the sum–product algorithm or, more efficiently, using the **Lauritzen–Spiegelhalter algorithm** (Lauritzen and Spiegelhalter, 1988).

Decomposable models have two useful properties. The first property is that a decomposable model can be represented using the clique marginals alone: given a set of clique marginals, we can extract the cliques simply by examining the arguments of the clique marginals and link these cliques together using Theorem 2.1. The separator marginals can be then computed from the marginal at one of the adjacent cliques:

Property 2.1 (Implicit representation)**.** *Let $\{p(\mathbf{x}_{C_i})\}$ be a set of marginals over the cliques, obtained by the vertex elimination algorithm. The following procedure constructs a decomposable model for $p(\mathbf{x})$:*

1. *From the cliques $\mathbf{C} = \{C_i\}$, form a maximum-intersection tree $(T, \mathbf{C})$. By Theorem 2.1, $(T, \mathbf{C})$ is a junction tree for $\mathbf{C}$.*

2. *For each edge in the junction tree $\{i, j\} \in E_T$, compute the separator marginal $p(\mathbf{x}_{S_{i,j}})$ by further marginalizing either $p(\mathbf{x}_{C_i})$ or $p(\mathbf{x}_{C_j})$.*

Another useful property is that every subtree of a decomposable model is a decomposable model representing a marginal of the original distribution:

Property 2.2 (Marginalization by pruning)**.** *Let* $(T, \mathbf{C})$ *be a junction tree for a distribution* p*. Then for any subtree* T' *of* T*,*

$$p(\mathbf{x}_U) = \frac{\prod_{i \in N_{T'}} p(\mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E_{T'}} p(\mathbf{x}_{S_{i,j}})},$$

where $U = \bigcup_{i \in N_{T'}} C_i$ *are the variables in the cliques of* T'*.*

Property 2.2 is best understood by applying it to a subtree T' that consists of all the vertices of T, other than a single leaf vertex i. Suppose that we wish to marginalize out the variables that are present in a leaf clique C_i, but nowhere else in the tree; by the running intersection property, these are exactly the variables $C_i - S_{i,j}$, where j is the neighbor of i in T. Then we can push the summation past all the clique marginals in the numerator, other than $p(\mathbf{x}_{C_i})$, and past all the separator marginals in the denominator:

$$p(\mathbf{x}_U) = \sum_{\mathbf{x}_{C_i - S_{i,j}}} p(\mathbf{x}) = \frac{\prod_{k \in N_T \setminus i} p(\mathbf{x}_{C_k})}{\prod_{\{k,\ell\} \in E_T \setminus \{i,j\}} p(\mathbf{x}_{S_{k,\ell}})} \times \frac{\sum_{\mathbf{x}_{C_i - S_{i,j}}} p(\mathbf{x}_{C_i})}{p(\mathbf{x}_{S_{i,j}})}.$$

But $\sum_{\mathbf{x}_{C_i - S_{i,j}}} p(\mathbf{x}_{C_i})$ is simply $p(\mathbf{x}_{S_{i,j}})$, which cancels out the separator marginal between i and j in the denominator. The remaining terms are exactly the clique and separator marginals for a subtree T'. Therefore, marginalizing out the variables $C_i - S_{i,j}$ amounts to removing the clique C_i and the separator $S_{i,j}$ (and the corresponding marginals) from the decomposable model; such an operation is called **pruning a leaf clique**.

Property 2.2 gives us a simple procedure for computing a marginal $p(\mathbf{x}_Q)$ of a decomposable model. If the variables Q correspond to a subtree T' of the junction tree T, then we simply discard all the clique and separator marginals outside of T', keeping the rest. If the variables Q do not correspond to a subtree, we select a minimal subtree T', such that the union of the cliques in T' includes all the variables Q; we say that the subtree T' **covers** the variables Q. We apply Property 2.2 to obtain a marginal $p(\mathbf{x}_U)$ and then further marginalize out the variables $U - Q$ using variable elimination.

2.1.5 Conditioning on observations

In the examples we have discussed so far, the values of the random variables were unknown. For example, we typically do not know the true intelligence of the person or the difficulty of the class, nor do we know the true temperature at any given location. We will call $\mathbf{X}$ the **hidden variables**, to indicate that they are not directly observable. Nevertheless, we often do have access to some observations of the system. For example, we may place a number of sensors around the environment; these sensors provide noisy measurements of the true temperatures at the sensors' respective locations. We will assume that the observations can be characterized by a finite set of **observed variables** $\mathbf{Z} = \{Z_a : a \in E\}$, where E is

a finite set of indices. Before any observations are made, our knowledge of the system is captured by a joint probability distribution $p(\mathbf{x}, \mathbf{z})$ that characterizes the relationship among all the variables $(\mathbf{X}, \mathbf{Z})$. Suppose that we make observations $\mathbf{Z} = \overline{\mathbf{z}}$ (we will use the bar notation to emphasize that $\overline{\mathbf{z}}$ is a fixed, rather than a free assignment). The **posterior distribution** $p(\mathbf{x} \mid \overline{\mathbf{z}}) = p(\mathbf{x}, \overline{\mathbf{z}})/p(\overline{\mathbf{z}})$ characterizes our knowledge of the hidden variables after the observations have been made. This distribution can be used to answer queries about the system; in particular, in a marginal probability query, we wish to compute the marginal distribution $p(\mathbf{x}_Q \mid \overline{\mathbf{z}})$ over a set of query variables $Q \subseteq V$. In this section, we will show that the posterior distribution can itself be represented as a factorized probability model. Therefore, the inference techniques we have discussed so far can be used to answer marginal probability queries in this setting as well.

We begin by discussing our assumptions on the joint distribution $p(\mathbf{x}, \mathbf{z})$; these assumptions will prove to be crucial in the distributed methods reviewed later on. In many systems, the phenomena characterized by the hidden variables $\mathbf{X}$ exist on their own, without being influenced by the observations $\mathbf{Z}$. For example, the true temperatures in an environment are not affected by the sensor measurements; the sensors are passive devices that do not influence the true temperatures in their environment. In these systems, it is reasonable to separate out the system description from the observations. We therefore express the joint distribution as a product $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}) \times p(\mathbf{z} \mid \mathbf{x})$, where $p(\mathbf{x})$ is called the **prior distribution**, and $p(\mathbf{z} \mid \mathbf{x})$ is called the **observation model**.

We will assume that the prior distribution is specified as a factorized probability model. We have already seen examples of factorized probability models for the prior distribution; for example, the temperature network in (2.7) was a product of factors over the temperatures at neighboring locations. The observation model can also be written as a product of factors. In many systems, the measurements do not influence each other: they are conditionally independent, given the hidden state $\mathbf{X}$. The observation model in these systems can thus be expressed as a product

$$p(\mathbf{z} \mid \mathbf{x}) = \prod_{a \in E} p(z_a \mid \mathbf{x}). \tag{2.16}$$

Furthermore, the individual observations are often local: given a small number of hidden variables, the observation is conditionally independent of the rest. We will denote the variables that each observation Z_a directly depends on as $\mathrm{Pa}[Z_a]$; in other words, we assume that $Z_a \perp \mathbf{X}_{V - \mathrm{Pa}[Z_a]} \mid \mathbf{X}_{\mathrm{Pa}[Z_a]}$. Each factor in (2.16) then simplifies to $p(z_a \mid \mathbf{x}_{\mathrm{Pa}[Z_a]})$, which we call the **observation model for** Z_a. Combining the definition of the prior distribution $p(\mathbf{x})$ with the observation model (2.16), we see that the joint distribution $p(\mathbf{x}, \mathbf{z})$ can be expressed as a factorized probability model

$$p(\mathbf{x}, \mathbf{z}) = \underbrace{\left[\frac{1}{Z} \prod_{A} \psi_A(\mathbf{x}_A) \right]}_{\text{prior distribution}} \times \underbrace{\left[\prod_{a \in E} p(z_a \mid \mathbf{x}_{\mathrm{Pa}[Z_a]}) \right]}_{\text{measurement model}}. \tag{2.17}$$

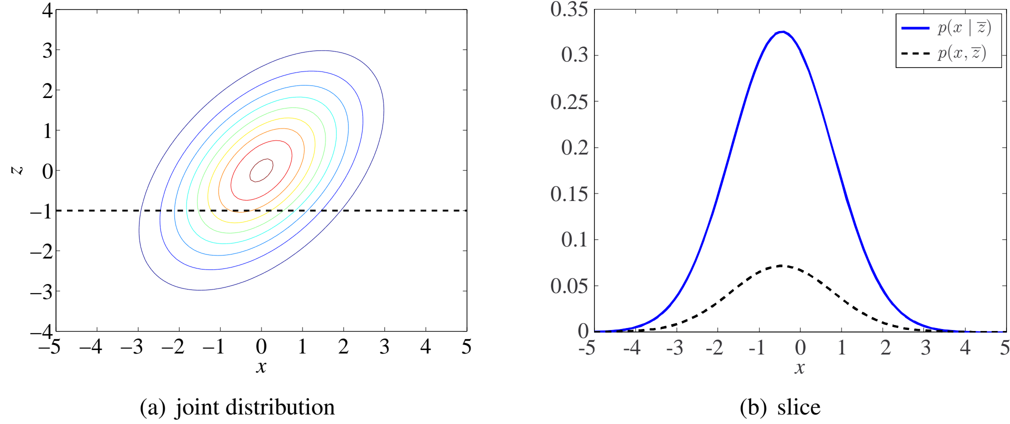


Figure 2.11: Conditioning in Gaussian distributions. (a) A Gaussian distribution over two variables, X and Z. The dashed line indicates the fixed value of Z. Fixing the value of Z effectively gives us an infinitely thin slice of the probability density function along the line $Z = -1$. (b) A plot of the slice $p(x, \bar{z})$. Normalizing this function yields the posterior distribution $p(x \mid \bar{z})$.

Example 2.1. ***Sensor calibration*** *is an example of a task that can be solved using probabilistic inference. After a sensor network is deployed, the sensors can be adversely affected by the environment, leading to biased measurements. The sensor calibration task involves automatic detection and removal of these biases (Bychkovskiy et al., 2003; Ihler et al., 2004). Automatic detection of biases is possible, because the quantities measured by nearby nodes are correlated but the biases of different nodes are independent.*

In the temperature sensor calibration problem, each observed variable $Z_i \in \mathbb{R}$ is a temperature measurement taken by one of the sensor nodes, and the hidden variables are the true temperatures $H_i \in \mathbb{R}$ and sensor biases $B_i \in \mathbb{R}$ of the network nodes. The temperature prior is a multivariate distribution characterized by a Markov network like the one in Figure 2.5; the biases of different nodes are independent. The complete model is a Gaussian distribution given by

$$p(\mathbf{h}, \mathbf{b}, \mathbf{z}) = \underbrace{\frac{1}{Z} \prod_{i,j} \psi_{i,j}(h_i, h_j)}_{\text{temperature prior}} \times \underbrace{\prod_i p(b_i)}_{\text{bias prior}} \times \underbrace{\prod_i p(z_i \mid h_i, b_i)}_{\text{observation model}}, \tag{2.18}$$

where each $\psi_{i,j}$ is a Gaussian factor and each bias prior $p(b_i)$ is a normal distribution $\mathcal{N}(0, 1)$. Each $p(z_i \mid h_i, b_i)$ is a ***conditional linear Gaussian*** *distribution $\mathcal{N}\left(h_i + b_i, \sigma^2\right)$, that is, the observation Z_i is a linear function of the true temperature and the bias, plus Gaussian noise $\epsilon \sim \mathcal{N}\left(0, \sigma^2\right)$.*

To estimate the bias at each sensor, we collect the measurements $Z_i = \bar{z}_i$ from all the sensors. The marginal posterior distribution $p(b_i \mid \bar{\mathbf{z}})$ then represents our belief about the bias at location i, given all the measurements made in the network. In particular, the expected value $\mathbb{E}[B_i \mid \bar{\mathbf{z}}]$ is an estimate of the bias.

		Job	
		offered	*not offered*
Letter, SAT	*good*, *high*	0.9	0.1
	good, *low*	0.5	0.5
	bad, *high*	0.5	0.5
	bad, *low*	0.1	0.9

Figure 2.12: Instantiating a variable in an observation model. The table shows the conditional probability distribution $p(job \,|\, letter, sat)$, indicating the probability of a student getting or not getting a job, given the quality of the recommendation letter and the student's SAT scores. Note that for each assignment to *Letter* and *SAT*, the distribution sums to 1. Suppose that we observe that the student is indeed offered a job. Then the column shown in bold represents the observation likelihood $p(Job = offered \,|\, letter, sat)$.

Conceptually, the posterior distribution $p(\mathbf{x} \,|\, \overline{\mathbf{z}})$ represents a "slice" of the joint distribution $p(\mathbf{x}, \mathbf{z})$ that has been normalized, so that the probabilities over the different assignments $\mathbf{x}$ sum to 1. Figure 2.11 illustrates this fact on a Gaussian distribution $p(x, z)$ over one hidden and one observed variable. Here, instantiating the hidden variable to $\overline{z}$ creates a bell curve over a single variable x; this curve is then scaled, so that the area under the curve is 1. In factorized models like (2.18), the slice $p(\mathbf{x} \,|\, \overline{\mathbf{z}})$ takes on a specific form: we instantiate the observation $Z_a = \overline{z}_a$ in the observation model for each observed variable Z_a, leaving all other factors of the joint distribution $p(\mathbf{x}, \mathbf{z})$ intact. For example, when the observation model $p(z_a \,|\, \mathbf{x}_{\mathrm{Pa}[Z_a]})$ is represented as a table of probabilities, instantiating the observation $Z_a = \overline{z}_a$ amounts to taking a part of the table, as shown in Figure 2.12. A similar procedure applies to Gaussian factors, see (Paskin, 2004, Lemma 2.17). The resulting factor $p(\overline{z}_a \,|\, \mathbf{x}_{\mathrm{Pa}[Z_a]})$ (which is a function of $\mathbf{x}_{\mathrm{Pa}[Z_a]}$ but not z_a) is called **observation likelihood**. Thus, the posterior distribution can be expressed as the product of the prior distribution and the observation likelihoods for all the observations, that is, a factorized probability model:

$$p(\mathbf{x} \,|\, \overline{\mathbf{z}}) \propto \underbrace{\left[\frac{1}{Z} \prod_{A} \psi_A(\mathbf{x}_A) \right]}_{\text{prior distribution}} \times \underbrace{\left[\prod_{a \in E} p(\overline{z}_a \,|\, \mathbf{x}_{\mathrm{Pa}[Z_a]}) \right]}_{\text{observation likelihoods}}. \tag{2.19}$$

Note that (2.19) is not a decomposable model: it is simply a product of factors that compactly specify the posterior distribution $p(\mathbf{x} \,|\, \overline{\mathbf{z}})$.

Since we have expressed the posterior distribution as a factorized probability model, we can now directly apply the techniques discussed in the previous sections. For example, we can construct a Markov network for the posterior distribution. Figure 2.13 shows an example Markov network for the posterior distribution in temperature sensor calibration. Note that some edges in the Markov network correspond to the factors of the prior distribution, while some edges correspond to the observation likelihoods $p(\overline{z}_i \,|\, h_i, b_i)$. In general, these two edge sets can overlap. Given a Markov network, we can construct a junction tree for the posterior distribution using vertex elimination, and then execute the sum–product algorithm to compute

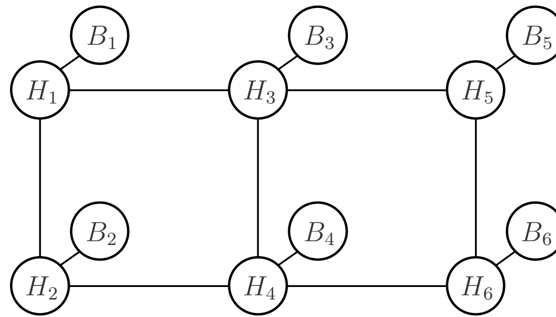


Figure 2.13: A Markov network for the posterior distribution in the temperature sensor calibration problem. The edges among the temperature variables $\mathbf{H}$ correspond to the factors of the prior distribution, while the edge between the temperature and the bias variables H_i and B_i correspond to the observation likelihoods.

marginals over the cliques. In initializing the sum–product algorithm, each clique may be assigned not only the factors of the prior distribution, but also one or more observation likelihoods. Since we did not assume anything about the factors assigned to each clique, other than that they support multiplication and marginalization, the algorithm can be executed without modifications. When the algorithm finishes its execution, each node belief $\beta_i(\mathbf{x}_{C_i})$ represents a marginal of the posterior distribution $p(\mathbf{x}_{C_i} \mid \bar{\mathbf{z}})$.

2.2 Probabilistic inference in distributed systems

We have seen that a core problem in probabilistic inference is computing marginals of a factorized probability model. To compute such marginals, the methods in the previous section required that the model and all the observations reside in a single, central location. Yet, in some applications, the observations are distributed, because they are made by nodes in a network, and it may not be feasible to collect them to a single location. Furthermore, even if we can collect the observations, the nodes may need to act based on the estimates extracted from the marginals. Therefore, it may be necessary to compute the marginals in a distributed manner, that is, the nodes may need to perform **distributed probabilistic inference**. In this section, we formally define the distributed probabilistic inference problem. In principle, the problem could be solved using the sum–product algorithm, described in Section 2.1.3, by sending messages along a distributed data structure called the **network junction tree** (Paskin et al., 2005). Unfortunately, the algorithm can yield very poor estimates, because the node beliefs before the convergence can be arbitrarily far from the correct posterior distribution. We will briefly discuss these limitations, which will motivate the robust message passing algorithm, reviewed later on in the chapter.

2.2.1 The distributed probabilistic inference problem

Throughout this chapter, we assume a network model where each node can perform local computations and communicate with other nodes over a lossy channel. The nodes of the network may change over time: existing nodes can fail, and new nodes may be introduced. We assume a message-level error model: messages are either received without error, or they are not received at all. The sender cannot distinguish a successful transmission from failure but is aware of the attempt; only the recipient is aware of a successful transmission. The probability of a successful transmission (called the link quality) is unknown and can change over time, and the link qualities of several node pairs may be correlated.

In the distributed probabilistic inference problem, we are given a global factorized probabilistic model (2.17) for the joint distribution $p(\mathbf{x}, \mathbf{z})$. Since the entire model may be too large to fit into the memory at each node, we assume that the node only has access to a portion of the model. Specifically, each node is given a partial description of the prior distribution $p(\mathbf{x})$; together, these partial descriptions must represent the prior distribution $p(\mathbf{x})$. For example, suppose that the factors of the prior distribution (2.5) are partitioned across the nodes, so that each node receives a subset of the factors, $\boldsymbol{\psi}_n = \{\psi_k(A_k) : k \in \mathcal{K}_n\}$, where $\mathcal{K}_n$ is a subset of the indices $\{1, \ldots, K\}$. Then the factor sets $\{\boldsymbol{\psi}_n\}$ together represent the prior distribution. In addition, each node has also access to the observation models for the variables $\mathbf{Z}_{E_n}$ it observes, where $E_n \subseteq E$ is a subset of the indices of $\mathbf{Z}$. We assume that each variable Z_i is observed at exactly one node n; in other words, we require that the index sets E_n form a partition of the indices E. This assumption is natural in many distributed systems, including sensor networks and mobile robot teams, where each node is equipped with local sensors that provide noisy measurements of the node's environment.

In addition to the partial view of the factorized probability model and the local observations, each node n is also associated with a set of **query variables** $Q_n \subseteq V$; these are the variables that the node wishes to reason about. The nodes need to collaborate, so that after the algorithm converges, each node can compute the posterior distribution over the query variables, given all the observations made in the network, $p(\mathbf{x}_{Q_n} \mid \overline{\mathbf{z}})$. If the algorithm can compute this distribution at convergence exactly, it is said to achieve **global correctness**. As strong as this property may seem, in many settings, it is of limited value: in large networks, it may take a long time to converge to the true posterior. If the network conditions are constantly changing or in the presence of network partitions, the algorithm may never converge. Finally, even if the algorithm converges, there is *no way to to know it*, as the nodes can never rule out the possibility that new factors will be entered in the network, forcing the messages to be recomputed. These challenges suggest that the algorithm should provide partial results that combine the local information and the messages received so far. If the algorithm is able to compute the **partial posterior**, the distribution over the node's query variables given the measurements that have been incorporated in the messages the node has received, the algorithm is said to achieve **partial correctness**.

Figure 2.14(a,c) shows an example of a distributed probabilistic inference problem for temperature sensor

calibration in Example 2.1. The probability model in Figure 2.14(a) consists of bias priors $p(b_i)$ and factors $\psi_{i,j}(h_i, h_j)$ for temperatures at four locations, as well as the observation models $p(z_i \mid h_i, b_i)$ for the variable Z_i observed at each node i. The dotted blue lines indicate which factors are assigned to each node. The query variables (not shown) are simply the temperature and the bias at the node, $Q_n = \{H_n, B_n\}$. Figure 2.14(c) shows the physical network; note that some of the links are stronger than others, and temperatures at two sensor locations may be connected by a factor even if the communication link between the corresponding pair of sensors is weak. This kind of discrepancy calls for a general-purpose approach that integrates both the modeling and the networking aspects of the problem.

2.2.2 Distributed inference architecture

The distributed probabilistic inference problem, outlined in the previous section, is an instance of a larger family of problems that seek to extract global conclusions in a distributed system based on local information known to each node. This class of problems includes optimal control, where the nodes can control the environment to maximize a reward function, and linear regression, where a sensor field is approximated by a weighted combination of basis functions. These problems may appear different, but they have the same algebraic structure (Paskin and Guestrin, 2004a): we first combine local pieces of information from the nodes to obtain a global model and then summarize the model to a subset of the variables. For example, in one formulation of the distributed probabilistic inference problem, the local pieces are the factors of the prior distribution and the local observation likelihoods; we combine (multiply) these factors together to form the global posterior distribution (2.19), and then summarize to (that is, compute a marginal over) a subset of the variables to answer a query. Because these problems share this structure, they can all be solved by passing messages on a junction tree; the sum–product algorithm was one specific instance of such an approach. The architecture of Paskin et al. (2005) provides a framework for solving these problems robustly, in presence of unreliable communication, by constructing an **overlay network**, a distributed data structure formed on top of the physical network.

We saw that, in one formulation of the distributed probabilistic inference problem, each node is assigned a subset $\mathcal{K}_n$ of the factors of the prior distribution. The node also makes local observations for a subset of the observed variables $E_n \subseteq E$, which can be instantiated in the respective observation models to obtain the observation likelihoods $p(\overline{z}_a \mid \mathbf{x}_{\mathrm{Pa}[Z_a]})$. If we were to organize the nodes into an undirected spanning tree, we would obtain a distributed data structure that is almost a junction tree: a tree where each node is associated with a local factor

$$\psi_n(\mathbf{x}_{L_n}) = \prod_{k \in \mathcal{K}_n} \psi_k(\mathbf{x}_{A_k}) \times \prod_{a \in E_n} p(\overline{z}_a \mid \mathbf{x}_{\mathrm{Pa}[Z_a]}),$$

where L_n are the indices of the **local variables** the node is initially aware of. All that would be missing are the clique C_n at each node and the separators $S_{m,n}$ between nodes adjacent in the spanning tree. This

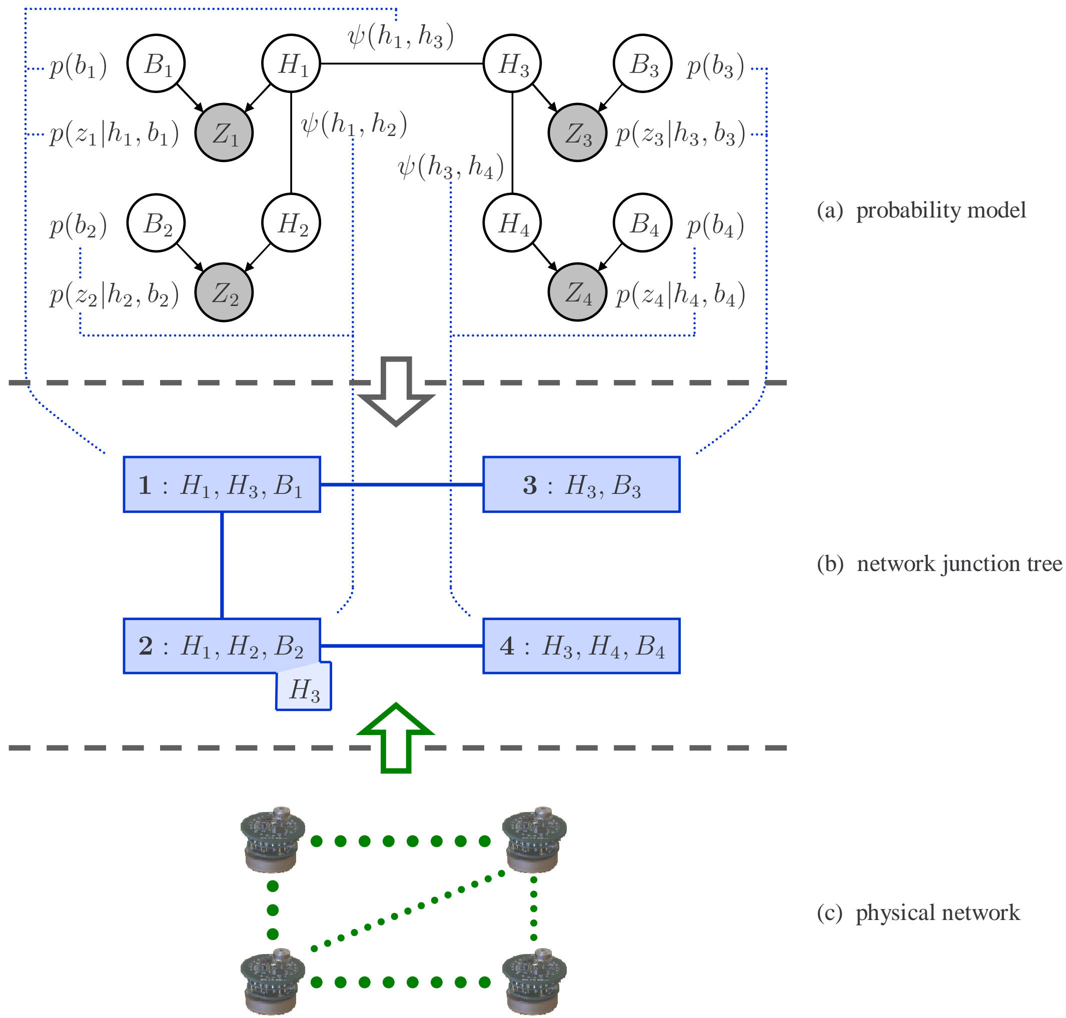


Figure 2.14: A small distributed probabilistic inference problem and the distributed inference architecture of Paskin et al. (2005). (a) A model over temperatures H_i at four sensor locations, the biases B_i of the corresponding sensors, and the biased measurements Z_i. Each factor of the prior distribution is assigned to one node; the node also carries the observation model for the its measurements. (b) A network junction tree, constructed for the model in (a). The tree spans the network over using strong communication links. The clique at each node includes the arguments of the hidden variables in the factors, assigned to that node, as well as any variables needed to satisfy the running intersection property (shown as a separate box). (c) A physical network; the strong links are indicated with large dots.

interpretation suggests the following three-layer architecture:

Spanning tree formation: Organize the nodes into a spanning tree, by selecting, at each node, a set of neighbors with strong communication links. For instance, based on the physical network in Figure 2.14(c), we create a spanning tree, shown in Figure 2.14(b). This spanning tree uses strong communication links among the nodes.

Junction tree formation: Associate each node with the variables in the factors that were assigned to it,

as well as any query variables. For example, in Figure 2.14(b), node 1 is associated with variables $\{H_1, H_3, B_1\}$, which are in the factors $\psi_{13}(h_1, h_3)$, $p(b_1)$, and $p(\bar{z}_1 \mid h_1, b_1)$ assigned to that node. The spanning tree and the variable sets at the nodes together form a clique tree, but this clique tree is not necessarily a junction tree, because it may not satisfy the running intersection property (Definition 2.2). To address this problem, we add a minimal set of variables at each node needed to satisfy the running intersection property. For example, in Figure 2.14(b), we have added variable H_3 to node 2, because H_3 is present at both nodes 1 and 4. This completes the description of a junction tree that is embedded in the network, called the **network junction tree**.

Inference: Compute the inference messages using the sum–product rule (2.13) over the edges of the network junction tree. Each node eventually converges to the global posterior over its clique, $p(\mathbf{x}_{C_n} \mid \bar{\mathbf{z}})$.

Each of these layers is implemented as a distributed algorithm that runs on every node of the network. The implementation addresses the following challenges, which are specific to distributed inference:

- The routing tree must be *stable*, so that its topology remains fixed whenever possible. Since the routing tree defines the topology of the network junction tree, it must be as stable as possible so that the inference algorithm can make progress.

- The junction tree can be *optimized*, in order to minimize the computation and communication required to solve the inference problem. The computational complexity of the sum–product algorithm is determined by the size of the cliques C_n. The communication complexity of the algorithm is determined by the quality of links as well as the size of the messages passed between neighboring nodes in the routing tree. The architecture (Paskin et al., 2005) employs local search to choose a junction tree that jointly optimizes the total computational and communication cost.

- Rather than running in sequence, the three layers run *concurrently*, responding to changes in each others states, so that the algorithm is robust to failure. For example, if communication along an edge of the spanning tree suddenly becomes unreliable, the spanning tree must be changed, which causes the junction tree layer to recompute the cliques, which in turn causes the inference layer to recompute new messages.

Since the spanning tree is chosen based on noisy estimates of the link qualities, it may change even if the true link qualities remain constant. Thus, temporarily, the constructed overlay network may not be connected and may not satisfy the running intersection property. Nevertheless, the architecture quickly recovers from these changes, and we will analyze the inference algorithms in this chapter in terms of a stable, singly connected network junction tree. Paskin et al. (2005) provide extensive experimental evaluation in a much richer setting that includes node and communication failures.

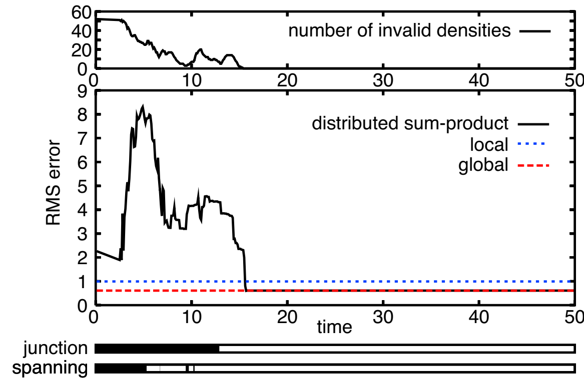


Figure 2.15: Convergence of the distributed sum–product algorithm on the application in Example 2.1, taken from (Paskin and Guestrin, 2004b). The middle part of the plot shows the root mean square (RMS) error of the distributed sum–product algorithm (black), the accuracy of the local estimate that incorporates complete prior but only the measurements local to each node (blue), and the accuracy of the centralized solution (red). The bottom part of the plot shows when the spanning tree and the junction tree were valid. We see that the results of the distributed sum–product algorithm are meaningful only some time after a valid junction tree is formed. The upper part of the plot shows that many of the beliefs are not even valid densities (in the sense that the precision matrix is not positive definite).

2.2.3 Limitations of the sum–product algorithm in distributed systems

In the sum–product algorithm in Section 2.1.3, the message from node m to node n can only be computed after all messages to m (except that from node n) have been computed. We will call this algorithm **synchronous**, because the messages among the nodes need to be synchronized to satisfy the dependencies in (2.13). We saw that the algorithm achieves global correctness: once all the messages have been propagated, the algorithm computes the exact posterior distribution over the clique C_n at each node. Nevertheless, as mentioned earlier, global correctness is a weak property: in large networks, the algorithm may take a long time to converge, and in volatile networks, the algorithm may never converge. This problem motivates an **asynchronous** version of the sum–product algorithm, where nodes periodically recompute their messages according to (2.13). At any point, a node may take all the incoming messages and compute the **partial belief** over its clique, using (2.14).

An asynchronous algorithm is useful if, at any point, the partial beliefs are meaningful. Some algorithms have meaningful partial beliefs; for example, partial results in the distributed linear regression algorithm amount to optimizing the coefficients from a subset of the measurements (Guestrin et al., 2004). Unfortunately, the partial beliefs in the sum–product algorithm are not meaningful at all. As illustrated in Figure 2.15, the algorithm is highly inaccurate before convergence, achieving errors that are much higher than if each node only took its local measurement into account. Furthermore, the improvements are not gradual; the algorithm converges only once all the messages are correctly propagated.

To explain the poor performance of the asynchronous sum–product algorithm, note that all the communi-

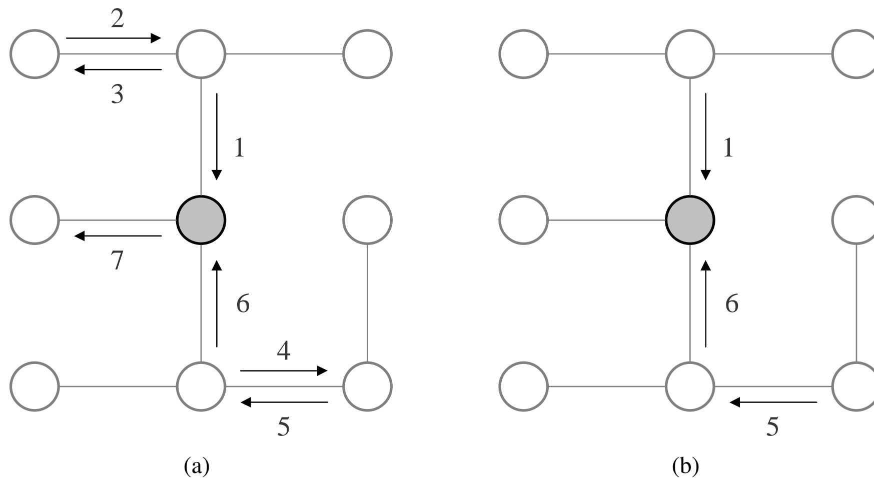


Figure 2.16: The partial belief in the sum–product algorithm at the middle node (shaded), taken from (Paskin, 2004). (a) One possible ordering of messages. (b) An equivalent set of messages from the perspective of the middle node. Since some of the messages are not used (directly nor indirectly) in computation of the belief at the middle node, they can be ignored. The partial belief is the exact marginal in a part of the model that includes factors stored at the four participating nodes.

cation among the nodes in this algorithm occurs along the edges of the junction tree. Then, assuming that a stable junction tree is formed, the partial belief at node n in the algorithm is equal to an exact belief in a subtree that contains n, see Figure 2.16. This structure makes it easy to reason about partial beliefs of the sum–product algorithm: partial belief is an exact posterior of a *different* probability model that is missing some factors.

In general, removing a factor from a factorized probability model does not yield a meaningful approximation to the original model. This fact can be seen by the following examples (Paskin, 2004):

Bayesian networks: Recall that a Bayesian network represents a distribution as a product of conditional probability distributions. Removing a conditional probability distribution from a Bayesian network can have a significant impact on the accuracy of the approximation. Consider a Bayesian network containing a variable that represents whether a reactor meltdown is imminent. The prior distribution for this variable would state that such meltdowns are unlikely. Now suppose that this prior is removed from the Bayesian network. Removing this factor effectively replaces it with a uniform prior distribution, where the probability of the meltdown is 50%. Clearly, this is a poor approximation to the exact model.

Multivariate Gaussians: In Gaussians, a missing factor performs worse than to bias the distribution: it can render the distribution unnormalizable. Suppose that V is partitioned into three non-empty sets,

S, T, and U, and suppose that the distribution is a Gaussian in the information form

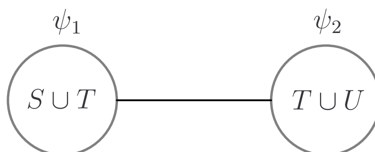
$$p(\mathbf{x}) = \mathcal{N}^{-1}\left(\begin{bmatrix}\mathbf{x}_S\\ \mathbf{x}_T\\ \mathbf{x}_U\end{bmatrix};\begin{bmatrix}\eta_S\\ \eta_T\\ \eta_U\end{bmatrix},\begin{bmatrix}\Lambda_{SS} & \Lambda_{ST} & \mathbf{0}\\ \Lambda_{TS} & \Lambda_{TT} & \Lambda_{TU}\\ \mathbf{0} & \Lambda_{UT} & \Lambda_{UU}\end{bmatrix}\right).$$

Then $p(\mathbf{x})$ can be written as a product of two factors

$$\psi_1 = \mathcal{G}\left(\begin{bmatrix}\mathbf{x}_S\\ \mathbf{x}_T\end{bmatrix};\begin{bmatrix}\eta_S\\ \eta_T\end{bmatrix},\begin{bmatrix}\Lambda_{SS} & \Lambda_{ST}\\ \Lambda_{TS} & \Lambda_{TT}\end{bmatrix}\right) \tag{2.20}$$

$$\psi_2 = \mathcal{G}\left(\begin{bmatrix}\mathbf{x}_T\\ \mathbf{x}_U\end{bmatrix};\begin{bmatrix}\mathbf{0}\\ \eta_U\end{bmatrix},\begin{bmatrix}\mathbf{0} & \Lambda_{TU}\\ \Lambda_{UT} & \Lambda_{UU}\end{bmatrix}\right) \tag{2.21}$$

These two factors can be assigned to two distinct nodes, in order to form the following junction tree:



The factors ψ_1 and ψ_2 represent the partial beliefs before any messages are exchanged between the two nodes. By the rule of conditioning in Gaussians (Paskin, 2004, Appendix 2.B), (2.20) represents the conditional distribution of $\mathbf{X}_{S\cup T}$, given the evidence $\mathbf{X}_U = \mathbf{0}$. Thus, removing ψ_2 from the distribution hallucinates evidence on $\mathbf{X}_U$. On the other hand, (2.21) does not represent a normalizable distribution, since the information matrix is not positive definite. This fact explains why in Figure 2.15, the sum–product algorithm performed so poorly.

These examples illustrate that factorized models lack a key property, called **locality**, which would allow us to interpret a factor outside of the context of other factors. There are, nevertheless, some specific instances, where factorized models exhibit locality. For example, in Bayesian networks, removing factors associated with **barren vertices**—that is, leaf vertices with no evidence—preserves the marginal over the remaining variables. Decomposable models, which lay the foundation for the robust message passing algorithm reviewed in the next section, generalize this property to be independent of the edge direction.

2.3 Robust message passing

In the previous sections, we defined the distributed probabilistic inference problem and outlined a distributed implementation of the standard sum–product algorithm, based on the distributed inference architecture (Paskin et al., 2005). We saw that while the algorithm converges to a marginal of the posterior distribution at each node, it does not produce accurate partial beliefs. Furthermore, the algorithm is not robust to node loss: if a node fails, all the factors stored at this node are lost, which may prevent the algo-

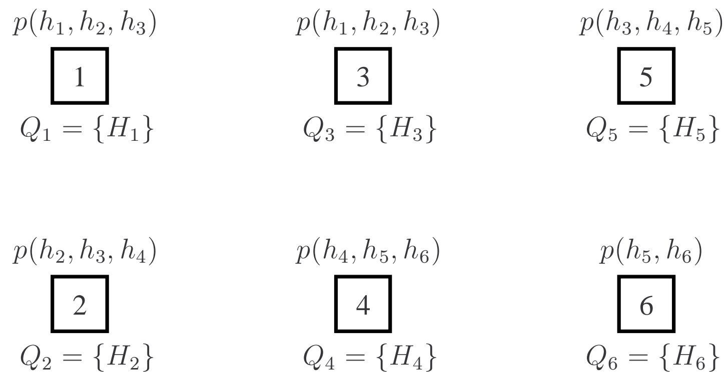


Figure 2.17: Assigning clique marginals of the global decomposable model to the individual nodes. Each node receives a subset of the marginals (for simplicity, each node received only one marginal in this figure). Note that the prior marginal $p(h_1, h_2, h_3)$ has been assigned to multiple nodes for robustness. Each node is also associated with a set of query variables; these are the variables that the node wishes to reason about.

rithm from ever reaching a meaningful solution. Both these limitations stem from the fact that removing a factor from a factorized probability model does not, in general, yield a meaningful approximation to the original model. In this section, we review an algorithm, called **robust message passing**, that employs a different representation of the posterior distribution—a variant of a decomposable model. This representation leads to accurate partial beliefs and ensures robustness to node loss. A key component of the algorithm is once again an overlay network—a network junction tree—that guides the marginalization operations. We first describe the operations of the algorithm in the absence of observations. We then discuss how to incorporate observations in the algorithm and state its partial correctness guarantees.

2.3.1 Decomposable models revisited

In Section 2.1.4, we have introduced decomposable models. Recall that a decomposable model (Definition 2.3) represents a distribution as a ratio of clique and separator marginals:

$$p(\mathbf{x}) = \frac{\prod_{i \in N_T} p(\mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E_T} p(\mathbf{x}_{S_{i,j}})}, \tag{2.22}$$

where $(T, \mathbf{C})$ is a junction tree for the distribution $p(\mathbf{x})$. In the context of distributed systems, the tree $(T, \mathbf{C})$ is called the **external junction tree**, indicating that it pertains to the external environment of the distributed system.

Suppose that we construct a decomposable model for the prior distribution $p(\mathbf{x})$ in a separate pre-processing step. Property 2.1 tells us that we can represent the decomposable model (2.22) using the clique marginals $\{p(\mathbf{x}_{C_i}) : i \in N_T\}$ alone; the junction tree $(T, \mathbf{C})$ and the separator marginals $\{p(\mathbf{x}_{S_{i,j}}) : \{i, j\} \in E_T\}$ can always be recovered. This property is very important in distributed systems, because it lets us easily

distribute the prior probability model: if we assign each clique marginal to a network node, the clique marginals from all the nodes form an implicit representation of the decomposable model for the prior distribution. Figure 2.17 shows one possible assignment of the cliques of the external junction tree in Figure 2.7(a). Note that the clique marginal $p(h_1, h_2, h_3)$ has been assigned to multiple nodes. Assigning a clique marginal to multiple nodes does not alter the prior distribution: if a clique C_i is included multiple times in $\mathbf{C}$, each duplicate copy will be connected to the the rest of the tree through a separator $S_{i,j} = C_i$, hence the corresponding prior and separator marginals in (2.22) cancel out. This duplicity provides some robustness against node loss: if a node carrying $p(\mathbf{x}_{C_i})$ fails, the clique marginal will still be carried by other nodes in the network.

Once the nodes obtain the designated clique marginals, they begin inference. For now, we will only consider computing a marginal of the prior distribution $p(\mathbf{x}_{Q_n})$ over the query variables Q_n at each node n; we will discuss conditioning on observations in Section 2.3.3. Property 2.2 (marginalization by pruning) leads to a simple (albeit inefficient) algorithm for computing $p(\mathbf{x}_{Q_n})$ at each node n: the node collects all clique marginals of the prior distribution, forms a decomposable model (2.22), and prunes any cliques outside a minimal tree that covers Q_n. One way to collect the clique marginals is to form a spanning tree $\mathcal{T}$ over the nodes of the network, as shown in Figure 2.18. The nodes communicate along the edges of this tree, passing clique marginals to each other. The message $\mu_{m \to n}$ from node m to node n is a set that contains all the clique marginals on node m's side of the tree; for example, the message $\mu_{4 \to 3}$ in Figure 2.18 contains the marginal $p(h_4, h_5, h_6)$ stored at node 4 and the marginal $p(h_5, h_6)$ stored at node 6. To compute this message, node m takes the union of its locally stored clique marginals $\boldsymbol{\pi}_m$ and the marginals in all the message coming from its other neighbors in $\mathcal{T}$:

$$\mu_{m \to n} = \boldsymbol{\pi}_m \cup \bigcup_{\ell \in N_{\mathcal{T}}(m) \setminus n} \mu_{\ell \to m}. \tag{2.23}$$

At convergence, the messages incoming to a node carry all the clique marginals stored elsewhere in the tree. Then the belief at node n

$$\beta_n = \boldsymbol{\pi}_n \cup \bigcup_{m \in N_{\mathcal{T}}(n)} \mu_{m \to n} \tag{2.24}$$

is an implicit representation of a decomposable model for the entire prior distribution $p(\mathbf{x})$.

2.3.2 Probabilistic inference as distributed clique pruning

The algorithm described in the previous section is not practical, because it has a high communication complexity: some messages contain almost all the clique marginals in the network, and each node obtains a decomposable model for the entire prior distribution $p(\mathbf{x})$. Clearly, obtaining a decomposable model for the entire distribution is unnecessary; a node only needs to obtain a subtree of the external junction tree that covers its query variables. For example, if node 3 knew that nodes 1 and 2 are only interested

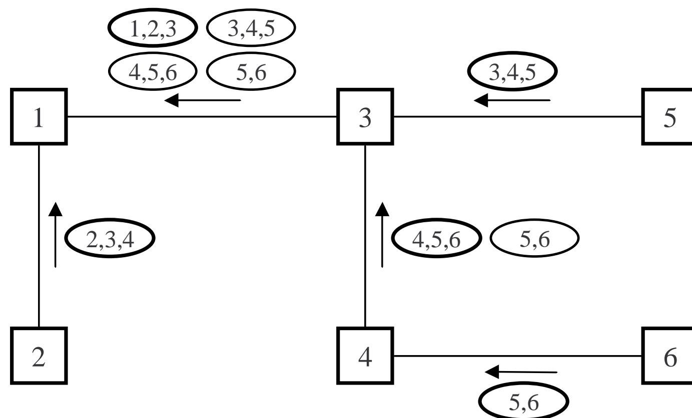


Figure 2.18: The messages of a simple algorithm that collects all the clique marginals at node 1. Each message is the union of the local marginals (shown in bold) and the incoming messages from the other nodes.

in the query variables H_1 and H_2, and if node 3 had access to the structure of the external junction tree, it could conclude that the cliques $\{H_3, H_4, H_5\}$, $\{H_4, H_5, H_6\}$, and $\{H_5, H_6\}$ are outside the minimal subtrees that cover the query variables and remove these cliques from the inference message $\mu_{3\to 1}$. Thus, we could prune the cliques from the inference messages along the way, rather than pruning them from the node belief when the nodes have already communicated the entire probability model.

Unfortunately, it may not be practical to disseminate the structure of the prior distribution to all the nodes; the external junction tree may itself be computed with a distributed algorithm that never stores a complete external junction tree at any given location. Nevertheless, the nodes may be able to determine a *partial* structure of the external junction tree given the cliques they have collected. Recall (Theorem 2.1) that a junction tree $(T, \mathbf{C})$ for a set of cliques $\mathbf{C}$ can be obtained by forming a maximum–intersection tree for $\mathbf{C}$. Suppose that a node has obtained the marginals over a subset of the cliques $\mathbf{C}' \subseteq \mathbf{C}$ and forms a maximum–intersection tree $(T', \mathbf{C}')$ for these cliques. Intuitively, if two cliques $C_i, C_j \in \mathbf{C}'$ are neighbors in T, their intersection $C_i \cap C_j$ is large, and we would expect them to be also neighbors in T'. For example, Figure 2.19 shows a maximum–intersection tree $(T', \mathbf{C}')$ for the cliques in the inference message from node 3 to node 1. We see that the cliques $\{H_3, H_4, H_5\}$ and $\{H_4, H_5, H_6\}$ that are neighbors in the external junction tree in Figure 2.7(a) are also neighbors in the maximum–intersection tree $(T', \mathbf{C}')$, and similarly for cliques $\{H_4, H_5, H_5\}$, $\{H_5, H_6\}$. This observation can be generalized to the following statement:

Lemma 2.2. *Let* $\mathbf{C}$ *be the cliques obtained by the vertex elimination algorithm and let* $\mathbf{C}' = \{C_i : i \in M\}$ *be a subset of cliques* $\mathbf{C}$ *for some index set* M. *If* $(T', \mathbf{C}')$ *is a maximum–intersection clique tree for* $\mathbf{C}'$, *then there exists a junction tree* $(T, \mathbf{C})$ *such that for any* $i, j \in M$, $\{i, j\} \in E_T \implies \{i, j\} \in E_{T'}$.

In other words, we can always find a junction tree $(T, \mathbf{C})$ such that whenever two cliques among $\mathbf{C}'$ are neighbors in T, they are also neighbors in T'. Lemma 2.2 requires some freedom in selecting the junction tree for $\mathbf{C}$, because in general, there may be several equivalent junction trees for a set of cliques (two trees

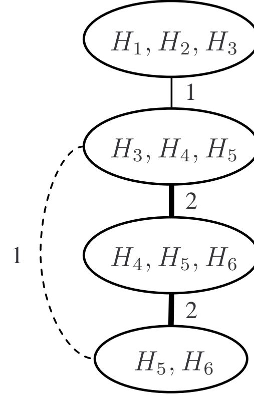


Figure 2.19: A maximum–intersection tree T' for a subset of cliques $\mathbf{C}'$; the weight at each edge is the size of the intersection of the adjacent cliques. The edges of the tree are shown as solid lines; a lower-weight edge that does not belong to the tree is shown as a dashed line. Note that some of the edges among $\mathbf{C}$ (shown in bold) are present both in T' and in the external junction tree in Figure 2.7(a).

are equivalent if they have the same set of cliques and the same set of separators). For example, if there was an additional clique $\{H_6, H_7\} \in \mathbf{C}'$ in the model in Figure 2.7(a), this clique could be attached to either $\{H_4, H_5, H_6\}$ or $\{H_5, H_6\}$. Lemma 2.2 ensures that this clique is attached to the correct neighbor that matches the maximum–intersection tree $(T', \mathbf{C}')$. The lemma has not been previously published and is proved in Appendix 2.A.

Naturally, for distributed inference, we need the converse result: starting from a maximum–intersection tree $(T', \mathbf{C}')$, we would like to determine which edges of T' are also present in an external junction tree $(T, \mathbf{C})$. We will first consider a special case where the leafs of the maximum–intersection tree are determined to be also the leafs of the external junction tree. We will discuss this case in the context of a simplified version of the robust message passing algorithm that only computes a marginal over the prior distribution at each node, disregarding the observation likelihoods.

The algorithm begins by constructing a network junction tree over the nodes. Similarly to the network junction tree, described in Section 2.2.2, the tree spans the nodes over strong communication links. Each clique C_n of the network junction tree is set to include the variables in the priors $p(\mathbf{x}_{C_i})$, assigned to node n, as well as the query variables Q_n and any additional variables needed to satisfy the running intersection property. For example, in Figure 2.20, node 1 is associated with the variables $\{H_1, H_2, H_3\}$, as well as the variable H_4, which was added, so that the network junction tree satisfies the running intersection property. Each separator $S_{m,n} = C_m \cap C_n$ is the set of variables that are common to the nodes on the different sides of the edge $\{m, n\}$. For example, the separator $S_{3,4}$ in Figure 2.20 are the variables $\{H_4, H_5\}$ that are found among the cliques at both nodes $\{1, 2, 3, 5\}$ and nodes $\{4, 6\}$.

By now, we have seen two junction trees in the algorithm: the external junction tree and the network junction tree. These two trees are two separate data structures with two radically different purposes: the external junction tree specifies the global decomposable model for the prior distribution $p(\mathbf{x})$, whereas

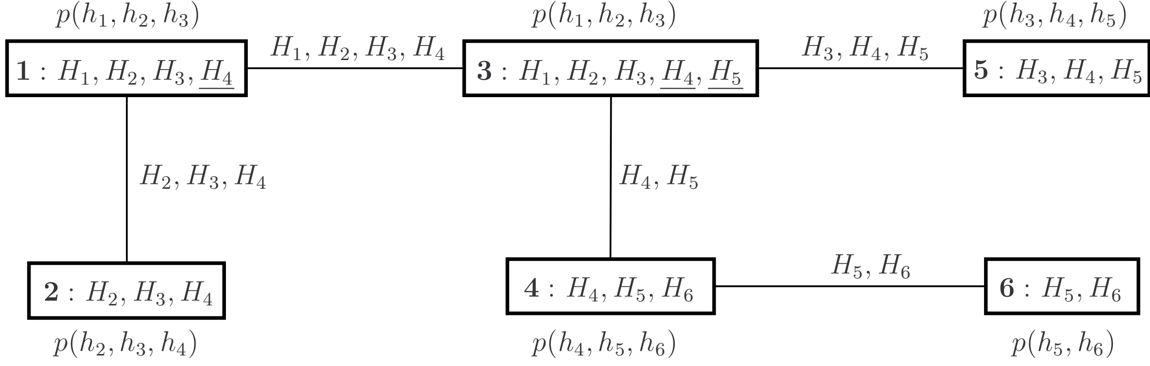


Figure 2.20: A network junction tree for the distributed inference problem in Figure 2.17. Each clique C_n includes the variables in the marginals assigned to node n, as well as the query variables Q_n. The variables added to satisfy the running intersection property are underlined.

the network junction tree specifies the communication pattern of the distributed inference algorithm. Nevertheless, the two trees are related: each clique C_n in the network junction tree includes the subset of the cliques of the external junction tree assigned to node n, and the network junction tree helps each node locally determine a partial structure of the external junction tree, as we will now show.

Continuing the description of the algorithm, the nodes send messages over the edges of the network junction tree, pruning cliques that are deemed unnecessary for other nodes. In computing the message $\mu_{m \to n}$, node m takes a union of its local priors $\boldsymbol{\pi}_m$ and the priors in the incoming messages from nodes other than n:

$$\boldsymbol{\pi}_m \cup \bigcup_{\ell \in N_{\mathcal{T}}(m) \setminus n} \mu_{\ell \to m}. \tag{2.25}$$

For example, in computing the message $\mu_{4 \to 3}$, node 4 takes a union of its local marginal $\{p(h_4, h_5, h_6)\}$ with the incoming message $\mu_{6 \to 4} = \{p(h_5, h_6)\}$. Node m now forms a maximum–intersection tree $(T', \mathbf{C}')$ for the priors in (2.25), such as the one shown in Figure 2.21(a), and repeatedly prunes any leaf clique C_i with neighbor C_j that satisfies the following condition:

$$C_i \cap S_{m,n} \subseteq C_j. \tag{2.26}$$

This condition is very intuitive: it states that if $C_i \cap S_{m,n} \subseteq C_j$, then any information about $S_{m,n}$ represented by the clique C_i is also represented by the clique C_j, and so C_i is redundant. For example, in Figure 2.21(a), the leaf clique $\{H_5, H_6\}$ carries information about H_5 that is relevant for nodes 3 and 5 on node 3's side of the network tree; however, this information is also represented by the neighboring clique $\{H_4, H_5, H_6\}$. Thus, clique $\{H_5, H_6\}$ can be safely pruned. Similar reasoning shows that when computing the message $\mu_{3 \to 1}$ in Figure 2.21(b), the leaf clique $\{H_4, H_5, H_6\}$ can be pruned.

A remarkable fact about the pruning rule (2.26) is that when considering the messages sent towards a given node n at convergence, the pruning operations in the algorithm trace the branches of some external

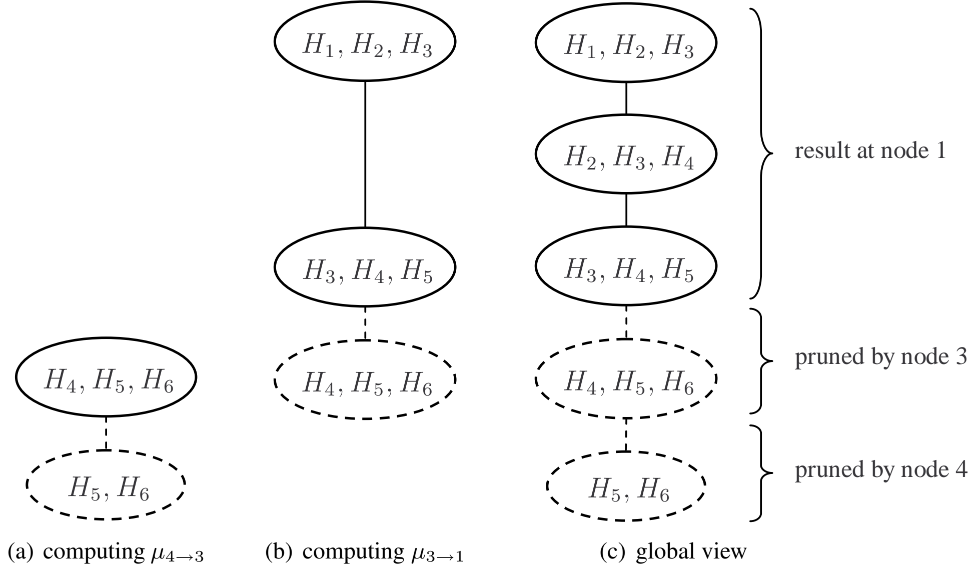


Figure 2.21: (a,b) Maximum–intersection trees for the cliques in messages $\mu_{4\to3}$ and $\mu_{3\to1}$. Dashed ellipses indicates the pruned cliques. (c) A global view of the algorithm from the perspective of node 1.

junction tree $(T, \mathbf{C})$ for $p(\mathbf{x})$. For example, Figure 2.21(c) shows that the clique $\{H_5, H_6\}$, pruned by node 4, is a leaf of the external junction tree for $p(\mathbf{h})$, while the clique $\{H_4, H_5, H_6\}$, pruned by node 3, is the next adjacent clique. Thus, node 4 correctly identified a leaf clique of the external junction tree (and its neighbor), while node 3 correctly identified a leaf clique in the remaining tree (that is, a tree containing all the cliques of the external junction tree, other than the previously pruned clique $\{H_5, H_6\}$). The pruning operations always happen in order, from the leafs of the external junction tree inwards. Then, at convergence, the node belief

$$\beta_n = \boldsymbol{\pi}_n \cup \bigcup_{m \in N_{\mathcal{T}}(()n)} \mu_{m\to n}$$

contains all the cliques that form a subtree of the external junction tree:

Theorem 2.2. *For any node $n \in N_{\mathcal{T}}$, let $\mathbf{C}' \subseteq \mathbf{C}$ be the cliques obtained by the algorithm at node n at convergence. Then the maximum intersection tree $(T', \mathbf{C}')$ is a subtree of some junction tree $(T, \mathbf{C})$.*

This theorem is obtained as a corollary to the proof of Theorem 6.1 in (Paskin, 2004).

The subtree $(T', \mathbf{C}')$ is guaranteed to include all the variables in the network clique C_n. For example, in Figure 2.21(c), the subtree obtained at node 1 contains all the variables $C_1 = \{H_1, H_2, H_3, H_4\}$. This fact is important, because node n can now answer its query Q_n: it forms a decomposable model for its belief β_n using Property 2.1 (implicit representation) and locally marginalizes out all variables but $\mathbf{X}_{Q_n}$ from this model using variable elimination.

To provide a theoretical justification for the algorithm, we will discuss two key theorems from (Paskin, 2004); we provide a simplified proof of one of these theorems in Appendix 2.A. Neither of these theorems

is important for understanding of the material later in the thesis.

Recall that in computing a message $\mu_{m\to n}$, node m forms a maximum–intersection tree $(T', \mathbf{C}')$ using the cliques in its local priors $\boldsymbol{\pi}_m$ and in the incoming messages from all the nodes, other than n. In order to relate this tree to the external junction tree, it is useful to define a notion of partial equality between two clique trees:

Definition 2.4. *Let $\mathbf{C}$ be a set of cliques, and let $\mathbf{C}' \subseteq \mathbf{C}$ be a subset of the cliques. Two clique trees $(T, \mathbf{C})$ and $(T', \mathbf{C}')$ are U-**subgraph consistent** if and only if for every pair of cliques $C_i, C_j \in \mathbf{C}$ whose intersection meets U (that is, $C_i \cap C_j \cap U \neq \emptyset$),*

$$\{i,j\} \in E_T \iff \{i,j\} \in E_{T'},$$

that is, $\{i,j\}$ is an edge of T if and only if it is also an edge of T'.

Thus, the clique trees $(T, \mathbf{C})$ and $(T', \mathbf{C}')$ are U-subgraph consistent if they agree on all the edges $\{i,j\}$, whose separator $S_{i,j}$ meets (that is, has a non-empty intersection with) U. For example, the external junction tree in Figure 2.7(a) and the maximum–intersection tree in Figure 2.19 are $\{H_5, H_6\}$-subgraph consistent, because the cliques $\{H_3, H_4, H_5\}$, $\{H_4, H_5, H_6\}$, and $\{H_5, H_6\}$ form the same subtree in both clique trees.

In general, a maximum–intersection tree $(T', \mathbf{C}')$ is U-subgraph consistent with an external junction tree $(T, \mathbf{C})$, provided that it contains enough cliques:

Theorem 2.3 (Theorem 6.6 in (Paskin, 2004))**.** *Let $\mathbf{C}$ be the cliques obtained by the vertex elimination algorithm and let $\mathbf{C}^U \subseteq \mathbf{C}$ be a subset that includes all cliques that meet U. Let $(T^U, \mathbf{C}^U)$ be a maximum–intersection tree for $\mathbf{C}^U$. Then there exists a junction tree $(T, \mathbf{C})$ that is U-subgraph consistent with $(T^U, \mathbf{C}^U)$.*

A simple proof of Theorem 2.3 is provided in Appendix 2.A.

To illustrate this theorem, we apply it to the set of cliques, collected by node 3 in computing its message to node 1 in the network junction tree in Figure 2.20. These cliques include the local clique $\{H_1, H_2, H_3\}$ at node 3, as well as the cliques $\{H_3, H_4, H_5\}$, $\{H_4, H_5, H_6\}$ in the incoming messages from nodes 5 and 4. Suppose that we disregard the leaf clique $\{H_5, H_6\}$ from the global model; this clique has already been pruned by node 4, so by Property 2.2 (marginalization by pruning), the remaining cliques in the network form a marginal model, shown in Figure 2.22. Since the separator between node 3 and node 1 is $\{H_1, H_2, H_3, H_4\}$, node 3 knows that there are no cliques that contain H_5 or H_6 on node 1's side of the tree. Therefore, node 3 must have collected all the cliques of the marginal model in Figure 2.22 that meet $\{H_5, H_6\}$. Node 3 can now conclude that its maximum–intersection tree in Figure 2.21(b) is $\{H_5, H_6\}$-subgraph consistent with the tree in Figure 2.22, and, in particular, the edge $\{H_3, H_4, H_5\}$—$\{H_4, H_5, H_6\}$ is present in both trees. In general, in applying Theorem 2.3 to the algorithm discussed earlier, U is the set of variables in the priors collected to form the message $\mu_{m\to n}$, other than the separator

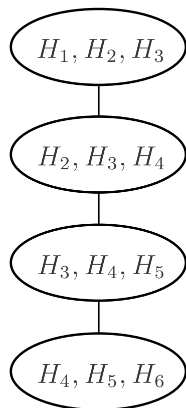


Figure 2.22: A subtree of the external junction tree in Figure 2.7(a), where the leaf clique $\{H_5, H_6\}$ has been pruned.

$S_{m,n}$, while $(T, \mathbf{C})$ is a subtree of the external junction tree for the marginal model.

Theorem 2.3 is useful in identifying edges of the external junction tree; this result is powerful, because it allows us to merge neighboring cliques in decomposable models, as discussed in (Paskin, 2004, Section 3.4.1). However, we have seen from Property 2.2 that it is often useful to identify the *leafs* of the external junction tree. The final link between an external junction tree and maximum–intersection trees is given by the following theorem:

Theorem 2.4 (Theorem 6.5 in (Paskin, 2004))**.** *Let* $\mathbf{C}$ *be the cliques obtained by the vertex elimination algorithm, and let* $\mathbf{C}^U \subseteq \mathbf{C}$ *contain all cliques that meet a subset* U *of variables. If* $(T^U, \mathbf{C}^U)$ *is a maximum–intersection tree for* $\mathbf{C}^U$*, and if* C_i *is a leaf clique of* T^U *with neighbor* C_j *such that*

$$C_i - C_j \subseteq U, \tag{2.27}$$

then there exists a junction tree $(T, \mathbf{C})$ *in which* i *is a leaf and* j *is its neighbor.*

For the distributed algorithm discussed above, the set U and the junction tree $(T, \mathbf{C})$ play exactly the same role in both Theorems 2.3 and 2.4. In this setting, the condition (2.27) can be shown to be equivalent to the pruning condition $C_i \cap S_{m,n} \subseteq C_j$ we have seen earlier. Thus, the algorithm repeatedly applies Theorem 2.4 to progressively smaller subtrees $(T, \mathbf{C})$ of the external junction tree and repeatedly prunes the leaf cliques of this tree.

2.3.3 Conditioning on observations

In the previous section, we outlined an algorithm that, starting with set of prior clique marginals at each node, computes the prior distribution over the query variables $p(\mathbf{x}_{Q_n})$ at each node. Similarly to the distributed sum–product algorithm, described in Section 2.2.3, the algorithm sends messages along the edges of network junction tree. However, rather sending "raw" factors, each message is a set of prior

marginals, pruned as permitted by the global structure, identified by the node. At this point, however, the algorithm is of limited use—we wish to solve a problem, where each node computes the *posterior* distribution $p(\mathbf{x}_{Q_n} \mid \overline{\mathbf{z}})$ that incorporates all observations made in the network. The robust message passing algorithm (Paskin and Guestrin, 2004b) can be viewed as an extension of the algorithm from the previous section to this setting. The key idea is to extend the definition of the decomposable model, so that it includes some information about the observations at each node.

Let $(T, \mathbf{C})$ be a junction tree for the prior distribution $p(\mathbf{x})$. Suppose that we assign each observed variable Z_a to some clique that covers its parents, $\text{Pa}[Z_a] \subseteq C_i$.[3] For example, in the temperature sensor calibration problem in Example 2.1, the parents of each variable Z_a are $\{H_a, B_a\}$, so we could assign the variable to the clique $\{H_a, B_a\}$. If the measurements are unbiased, the measurement model for the variable Z_a is simply $p(z_a \mid h_a)$, and we can assign the observed variable Z_a to any clique that includes H_a. Let $E_i \subseteq E$ denote the indices of the observed variables assigned to clique C_i. Then we can group the observation models for variables $\mathbf{Z}_{E_i}$ together and write the observation model $p(\mathbf{z} \mid \mathbf{x})$ as

$$p(\mathbf{z} \mid \mathbf{x}) = \prod_{i \in N_T} p(\mathbf{z}_{E_i} \mid \mathbf{x}_{C_i}), \tag{2.28}$$

where $p(\mathbf{z}_{E_i} \mid \mathbf{x}_{C_i}) = \prod_{a \in E_i} p(z_a \mid \mathbf{x}_{\text{Pa}[Z_a]})$. Instantiating the observed variables $\mathbf{Z}$ in (2.28) to the measurements $\overline{\mathbf{z}}$, we obtain the following factorization of the observation likelihood $p(\overline{\mathbf{z}} \mid \mathbf{x})$:

$$p(\overline{\mathbf{z}} \mid \mathbf{x}) = \prod_{i \in N_T} p(\overline{\mathbf{z}}_{E_i} \mid \mathbf{x}_{C_i}), \tag{2.29}$$

where each **likelihood factor** $p(\overline{\mathbf{z}}_{E_i} \mid \mathbf{x}_{C_i}) = \prod_{a \in E_i} p(\overline{z}_a \mid \mathbf{x}_{\text{Pa}[Z_a]})$ is a product of observation likelihoods for the variables Z_a assigned to C_i.

So far, we have not done much: we have simply grouped the observation likelihoods by the cliques they were assigned to. Nevertheless, the representation (2.29) is useful, because it ties into the definition of a decomposable model:

$$p(\mathbf{x}, \overline{\mathbf{z}}) = p(\mathbf{x}) \times p(\overline{\mathbf{z}} \mid \mathbf{x}) = \frac{\prod_{i \in N_T} p(\mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E_T} p(\mathbf{x}_{S_{i,j}})} \times \prod_{i \in N_T} p(\overline{\mathbf{z}}_{E_i} \mid \mathbf{x}_{C_i}). \tag{2.30}$$

Thus, we represent $p(\mathbf{x}, \overline{\mathbf{z}})$ as a collection of prior marginals that define the shape of the prior distribution $p(\mathbf{x})$ and a collection of likelihood factors that "modify" the shape to account for the observations. As usual, normalizing $p(\mathbf{x}, \overline{\mathbf{z}})$ yields the posterior distribution $p(\mathbf{x} \mid \overline{\mathbf{z}})$. The representation of the posterior distribution (2.30) is called a **prior/likelihood (P/L) decomposable model**.

The properties of decomposable models, discussed in Section 2.1.4, generalize to P/L decomposable models:

[3] Here, we assume that such a clique exists; the junction tree can typically be adjusted to ensure that this condition is satisfied.

Implicit representation: We have discussed that a decomposable model is described completely using clique marginals. Similarly, a P/L decomposable model is described completely using the prior clique marginals $\{p(\mathbf{x}_{C_i}) : i \in N_T\}$ and the likelihoods $\{p(\overline{\mathbf{z}}_{E_i} \,|\, \mathbf{x}_{C_i}) : i \in N_T\}$:

$$p(\mathbf{x}, \overline{\mathbf{z}}) = \frac{\prod_{i \in N_T} p(\mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E_T} \sum_{\mathbf{x}_{C_i - C_j}} p(\mathbf{x}_{C_i})} \times \prod_{i \in N_T} p(\overline{\mathbf{z}}_{E_i} \,|\, \mathbf{x}_{C_i}). \tag{2.31}$$

As before, the edges E_T can be recovered by forming a maximum–intersection tree for the cliques $\mathbf{C} = \{C_i : i \in N_T\}$. Each separator marginal $p(\mathbf{x}_{S_{i,j}})$ is recovered by further marginalizing the prior for clique C_i or C_j.

Marginalization as pruning: We have discussed that certain marginalization operations in decomposable models can be performed efficiently: pruning a leaf clique of a decomposable model yields another decomposable model over a subset of variables included in the remaining cliques. This property generalizes to P/L decomposable models. Let i be a leaf of the junction tree $(T, \mathbf{C})$, with neighbor j. As before, we can group the terms related to the clique C_i and the separator $S_{i,j}$ to efficiently sum out the variables $C_i - S_{i,j}$:

$$\sum_{\mathbf{x}_{C_i - C_j}} p(\mathbf{x}, \overline{\mathbf{z}}) = \frac{\prod_{k \in N_T \setminus i} p(\mathbf{x}_{C_k}) \times p(\overline{\mathbf{z}}_{E_k} \,|\, \mathbf{x}_{C_k})}{\prod_{\{k,\ell\} \in E_T \setminus \{i,j\}} p(\mathbf{x}_{S_{k,\ell}})} \times \frac{\sum_{\mathbf{x}_{C_i - C_j}} p(\mathbf{x}_{C_i}) \times p(\overline{\mathbf{z}}_{E_i} \,|\, \mathbf{x}_{C_i})}{p(\mathbf{x}_{S_{i,j}})}.$$

Thus, marginalizing out $\mathbf{x}_{C_i - C_j}$ yields a new decomposable model for the remaining variables and a new likelihood factor

$$p(\overline{\mathbf{z}}_{E_i} \,|\, \mathbf{x}_{S_{i,j}}) = \frac{\sum_{\mathbf{x}_{C_i - C_j}} p(\mathbf{x}_{C_i}) \times p(\overline{\mathbf{z}}_{E_i} \,|\, \mathbf{x}_{C_i})}{p(\mathbf{x}_{S_{i,j}})}. \tag{2.32}$$

This factor can be multiplied into any likelihood for a clique that includes $S_{i,j}$, such as the neighbor C_j. As before, a marginal over an arbitrary set of variables Q can be computed by repeatedly pruning leaf cliques, until we reach a minimal junction tree, whose cliques cover Q. Then we perform variable elimination on the remaining factors to marginalize out all the variables, other than Q.

We are now ready to describe the robust message passing algorithm. In the algorithm in the previous section, the local factors, the messages, and the node beliefs were collections of prior marginals over a subset of the cliques of the external junction tree. In the robust message passing algorithm, these priors are augmented with the likelihoods for the corresponding cliques:

Definition 2.5. *A **robust factor** is a tuple $(\mathbf{C}, \boldsymbol{\pi}, \boldsymbol{\lambda})$, where $\mathbf{C} = \{C_i : i \in N\}$ is a collection of cliques, $\boldsymbol{\pi} = \{\pi_i(\mathbf{x}_{C_i}) : i \in N\}$ is a collection of clique priors, one for each clique C_i, and $\boldsymbol{\lambda} = \{\lambda_i(\mathbf{x}_{C_i}) : i \in N\}$ is a collection of clique likelihoods, one for each clique C_i. The **scope** of a robust factor $\psi = (\mathbf{C}, \boldsymbol{\pi}, \boldsymbol{\lambda})$ is the union of its cliques, Scope$[\psi] \triangleq \cup_{i \in N} C_i$.*

Each node starts by constructing its local factor $\psi_n = (\mathbf{C}, \boldsymbol{\pi}, \boldsymbol{\lambda})$. This factor includes the clique priors $\pi_i(\mathbf{x}_{C_i}) = p(\mathbf{x}_{C_i})$ that were assigned to the node and the corresponding set of cliques $\mathbf{C}$. The node polls its sensors for the measurements $\overline{z}_a$, $a \in E_n$, and instantiates the observed variables $Z_a = \overline{z}_a$ to obtain the observation likelihoods $p(\overline{z}_a \mid \mathbf{x}_{\mathrm{Pa}[Z_a]})$ for $a \in E_n$. The node then pairs each likelihood with a clique that cover its arguments (here, we require that the parents of each measurement are covered by some clique assigned to node n). The likelihood λ_i is the product of the likelihoods assigned to clique C_i. The cliques that are not assigned any observation likelihoods have λ_i set to the unity factor 1.

Similarly to other message passing algorithms, the robust message passing algorithm collects information by combining factors. The combination in the robust message passing algorithm is essentially a union operation that multiplies the likelihoods for the identical cliques:

Definition 2.6. *The **combination** of two robust factors $\psi_1 = (\mathbf{C}^1, \boldsymbol{\pi}^1, \boldsymbol{\lambda}^1)$ and $\psi_2 = (\mathbf{C}^2, \boldsymbol{\pi}^2, \boldsymbol{\lambda}^2)$, written $\psi_1 \otimes \psi_2$, is the robust factor with $\mathbf{C} = \mathbf{C}^1 \cup \mathbf{C}^2$, $\boldsymbol{\pi} = \boldsymbol{\pi}^1 \cup \boldsymbol{\pi}^2$ and*

$$\lambda_i = \begin{cases} \lambda_i^1 \times \lambda_i^2 & \text{if } C_i \in \mathbf{C}^1 \cap \mathbf{C}^2 \\ \lambda_i^1 & \text{if } C_i \in \mathbf{C}^1 - \mathbf{C}^2 \\ \lambda_i^2 & \text{if } C_i \in \mathbf{C}^2 - \mathbf{C}^1. \end{cases} \tag{2.33}$$

Because each clique prior $p(\mathbf{x}_{C_i})$ is assigned to at least one node, the combination of local factors from all the nodes $\bigotimes_n \psi_n$ is an implicit representation of the P/L decomposable model for the posterior distribution $p(\mathbf{x} \mid \overline{\mathbf{z}})$. This fact suggests an efficient algorithm analogous to the algorithm in the previous section: to compute its message to node n, node m combines its local factor with the incoming messages from $\ell \neq n$, forms a maximum–intersection tree and repeatedly prunes leaf cliques to reduce the message size. In order to mirror the marginalization operations in P/L decomposable models, each such pruning operation needs to transfer the likelihood to a neighboring clique:

Definition 2.7. *Let $(\mathbf{C}, \boldsymbol{\pi}, \boldsymbol{\lambda})$ be a robust factor and let $(T, \mathbf{C})$ be a maximum–intersection tree for $\mathbf{C}$ with a leaf i. The factor $(\mathbf{C}^*, \boldsymbol{\pi}^*, \boldsymbol{\lambda}^*)$ is obtained from $(\mathbf{C}, \boldsymbol{\pi}, \boldsymbol{\lambda})$ by **pruning** C_i **from** C_j if $\mathbf{C}^* = \mathbf{C} \setminus C_i$, $\boldsymbol{\pi}^* = \boldsymbol{\pi} \setminus \pi_i$, and*

$$\lambda_k^* = \begin{cases} \lambda_j \times \dfrac{\sum_{\mathbf{x}_{C_i - C_j}} \pi_i \times \lambda_i}{\sum_{\mathbf{x}_{C_i - C_j}} \pi_i} & \text{if } k = j \\ \lambda_k & \text{otherwise} \end{cases}.$$

The message computation of the robust message passing algorithm is summarized in Algorithm 1. As before, the algorithm forms a maximum–intersection tree for all the cliques in its local factor and in the incoming messages from $\ell \neq n$, and repeatedly prunes the leaf clique C_i, as long as the information about the separator $S_{m,n}$ is also present in the neighboring clique C_j. Each such pruning operation transfers the likelihood to a neighboring clique C_j, using Definition 2.7.

To illustrate the robust message passing algorithm, we will step through the computations needed to send

Algorithm 1 Computing the message $\mu_{m\to n}$ in the robust message passing algorithm

1: $\mu_{m\to n} \leftarrow \psi_m \otimes \bigotimes_{\ell \in N_{\mathcal{T}}(m)\setminus n} \mu_{\ell\to m}$
2: $T \leftarrow$ a maximum–intersection tree for $\mathbf{C}$
3: **while** T has a leaf i with neighbor j such that $C_i \cap S_{m,n} \subseteq C_j$ **do**
4: Prune C_i from C_j in $\mu_{m\to n}$ and prune i from T.

the message $\mu_{3\to 1}$ in the network junction tree in Figure 2.20. Suppose that each node n makes a local observation $Z_n = \overline{z}_n$. The messages from nodes 5 and 6 are simply their local clique priors and observation likelihoods:

$$\begin{aligned}\mu_{5\to 3} &= (\{\{H_3, H_4, H_5\}\}, \{p(h_3, h_4, h_5)\}, \{p(\overline{z}_5 \mid h_5)\})\\ \mu_{6\to 4} &= (\{\{H_5, H_6\}\}, \{p(h_5, h_6)\}, \{p(\overline{z}_6 \mid h_6)\}).\end{aligned}$$

To compute its message to node 3, node 4 combines $\mu_{6\to 4}$ with its local factor

$$\psi_4 = (\{\{H_4, H_5, H_6\}\}, \{p(h_4, h_5, h_6)\}, \{p(\overline{z}_4 \mid h_4)\}).$$

The maximum–intersection tree for the two cliques $\{H_4, H_5, H_6\}$ and $\{H_5, H_6\}$ is shown in Figure 2.21(a). The leaf clique $\{H_5, H_6\}$ is pruned, transferring the likelihood

$$\frac{p(h_5, h_6) \times p(\overline{z}_6 \mid h_6)}{p(h_5, h_6)} = p(\overline{z}_6 \mid h_6)$$

to the neighboring clique $\{H_4, H_5, H_6\}$. The resulting message is

$$\mu_{4\to 3} = (\{\{H_4, H_5, H_6\}\}, \{p(h_4, h_5, h_6)\}, \{p(\overline{z}_4, \overline{z}_6 \mid h_4, h_6)\}).$$

Node 3 now combines its local factor

$$\psi_3 = (\{\{H_1, H_2, H_3\}\}, \{p(h_1, h_2, h_3)\}, \{p(\overline{z}_3 \mid h_3)\})$$

with the incoming messages $\mu_{4\to 3}$ and $\mu_{5\to 3}$, forming the maximum spanning tree, shown in Figure 2.21(b). The clique $\{H_4, H_5, H_6\}$ is pruned, transferring the likelihood

$$\frac{\sum_{h_6} p(h_4, h_5, h_6) \times p(\overline{z}_4, \overline{z}_6 \mid h_4, h_6)}{p(h_4, h_5)} = p(\overline{z}_4, \overline{z}_6 \mid h_4, h_5)$$

to $\{H_3, H_4, H_5\}$. The resulting message $\mu_{3\to 1}$ is thus

$$\mu_{3\to 1} = \left(\left\{ \begin{array}{c} \{H_1, H_2, H_3\}, \\ \{H_3, H_4, H_5\} \end{array} \right\}, \left\{ \begin{array}{c} p(h_1, h_2, h_3), \\ p(h_3, h_4, h_5) \end{array} \right\}, \left\{ \begin{array}{c} p(\overline{z}_3 \mid h_3), \\ p(\overline{z}_4, \overline{z}_5, \overline{z}_6 \mid h_4, h_5) \end{array} \right\} \right).$$

In this message, the clique $\{H_3, H_4, H_5\}$ has collected the likelihoods for the observations made at nodes

4, 5, and 6.

Consider the messages sent towards a specific node and suppose that the messages are computed in an order that follows the message dependencies. Theorem 2.4 then shows that whenever we prune a dangling leaf C_i from its neighbor C_j, we are effectively pruning C_i in a global P/L decomposable model for the posterior distribution. The priors and likelihoods for the remaining cliques form an implicit representation of a marginal of the posterior distribution:

Theorem 2.5 (Global convergence, Theorem 6.1 in (Paskin, 2004))**.** *Let $\mathcal{T}$ be a network junction tree with cliques $\{C_n : n \in N_{\mathcal{T}}\}$ and let $\{\psi_n : n \in N_{\mathcal{T}}\}$ be a collection of local factors such that $\bigotimes_{n\in N_{\mathcal{T}}} \psi_n$ represents $p(\mathbf{x} \,|\, \overline{\mathbf{z}})$ and Scope$[\psi_n] \subseteq C_n$. Then if all the messages are computed in an order that follows the message dependencies, the belief*

$$\beta_n = \psi_n \otimes \bigotimes_{m\in N_{\mathcal{T}}(n)} \mu_{m\to n} \tag{2.34}$$

at each node $n \in N_{\mathcal{T}}$ represents $p(\mathbf{x}_U \,|\, \overline{\mathbf{z}})$ for some $U \supseteq C_n$.

In other words, at convergence, the algorithm obtains a marginal of the posterior distribution over a set of variables that includes the clique C_n of the network junction tree. Since the cliques are chosen so that each clique C_n includes the query variables Q_n, the beliefs can be then further marginalized to answer the queries.

2.3.4 Approximate partial correctness

In the previous section, we described the robust message passing algorithm. We studied the behavior of the algorithm at convergence, when all the messages are computed in an order that satisfies the message dependencies; we showed that at convergence, each node obtains a marginal of the posterior distribution over the query variables. While this result is important, in time-critical systems, the nodes cannot wait for the convergence to interpret the results and need to instead rely on the partial beliefs. Unlike the sum–product algorithm, the partial beliefs in the robust message passing algorithm are often accurate. In this section, we illustrate some of the partial correctness guarantees of the algorithm. For a more extensive description with proofs, we refer the reader to (Paskin, 2004, Section 6.3.5).

Consider the scenario, shown in Figure 2.23, where only some of the messages have been successfully transmitted. Suppose that node 3 receives the messages $\mu_{4\to3}$ and $\mu_{5\to3}$ before the node computes its message to node 1. Then the message $\mu_{3\to1}$ still consists two cliques $\{H_1, H_2, H_3\}$ and $\{H_3, H_4, H_5\}$, but it does not capture the information about the observation $\overline{z}_6$:

$$\mu_{3\to1} = \left(\left\{ \begin{array}{c} \{H_1, H_2, H_3\}, \\ \{H_3, H_4, H_5\} \end{array} \right\}, \left\{ \begin{array}{c} p(h_1, h_2, h_3), \\ p(h_3, h_4, h_5) \end{array} \right\}, \left\{ \begin{array}{c} p(\overline{z}_3 \,|\, h_3), \\ p(\overline{z}_4, \overline{z}_5 \,|\, h_4, h_5) \end{array} \right\} \right). \tag{2.35}$$

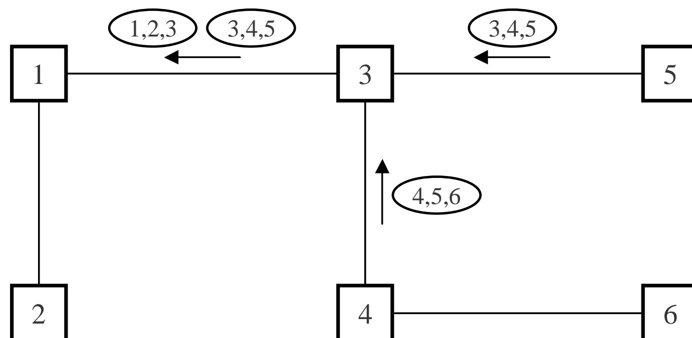


Figure 2.23: A scenario, where only a subset of the messages have been successfully transmitted.

Furthermore, suppose that the link between node 1 and node 2 is weak, so that no inference messages have been successfully transmitted between these two nodes. Then the partial belief at node 1 is the combination of the message $\mu_{3\to 1}$ (2.35) and the node's local factor

$$\psi_1 = (\{\{H_1, H_2, H_3\}\}, \{p(h_1, h_2, h_3)\}, \{p(\overline{z}_1 \mid h_1)\}).$$

This belief consists of two cliques, $\{H_1, H_2, H_3\}$ and $\{H_3, H_4, H_5\}$, which can be linked together to form a junction tree. We can thus interpret the partial belief at node 1 as a P/L decomposable model over the variables $H_1, \ldots, H_5$:

$$\frac{p(h_1, h_2, h_3) \times p(h_3, h_4, h_5)}{p(h_3)} \times p(\overline{z}_1, \overline{z}_3 \mid h_1, h_3) \times p(\overline{z}_4, \overline{z}_5 \mid h_4, h_5). \tag{2.36}$$

While the distribution (2.36) was obtained by a complicated process that involved combinations of robust factors and pruning of leaf cliques, it could have been equivalently obtained by instantiating the observations $\mathbf{Z} = \overline{\mathbf{z}}$ in the following model:

$$\tilde{p} = \frac{p(h_1, h_2, h_3) \times p(h_3, h_4, h_5)}{p(h_3)} \times p(z_1, z_3 \mid h_1, h_3) \times p(z_4, z_5 \mid h_4, h_5). \tag{2.37}$$

The distribution $\tilde{p}$ is *not* the joint distribution $p(\mathbf{x}, \mathbf{z})$, because it leaves out some of the variables. It is not even a marginal of the joint distribution $p(\mathbf{x}, \mathbf{z})$, because it makes the conditional independence assumption

$$H_1, H_2 \perp H_4, H_5 \mid H_3.$$

Nevertheless, it is a principled approximation to a marginal of the joint distribution $p(\mathbf{x}, \mathbf{z})$, as we will now show.

One way to find a principled approximation to a distribution p is to define a measure of "dissimilarity" between two distributions and then find a distribution among some family of distributions that minimizes the dissimilarity from p. A standard measure is the **Kullback-Leibler divergence** (also called the **relative**

entropy):

Definition 2.8. *Given two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ over the same set of variables, the* ***Kullback-Leibler (KL) divergence*** *from p to q is the sum*

$$D(p \,\|\, q) \triangleq \sum_{\mathbf{x}} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

KL divergence is asymmetric (in general, $D(p \,\|\, q) \neq D(q \,\|\, p)$), so it is not a distance function. Nevertheless, KL divergence is a natural notion of dissimilarity between two distributions: $D(p \,\|\, q)$ is always non-negative, $D(p \,\|\, q) = 0$ if and only if the two distributions p and q are same, and in coding theory, $D(p \,\|\, q)$ represents the inefficiency in the best code that is based on the distribution q, when the data comes from the distribution p.

When approximating one distribution with another, the first argument of KL-divergence is typically the exact distribution, and the second argument is the approximation. Given a family of possible approximate distributions, the best approximation in this family is called the **Kullback-Leibler projection**:

Definition 2.9. *Given a distribution p and a family of distributions $\mathcal{F}$ over the same set of variables, a* ***Kullback-Leibler (KL) projection*** *of p to $\mathcal{F}$ is the distribution*

$$q^*(\mathbf{x}) = \arg\min_{q \in \mathcal{F}} D(p \,\|\, q).$$

If p happens to be a member of $\mathcal{F}$, then the KL projection of p to $\mathcal{F}$ is p itself, since $D(p \,\|\, p) = 0$.

The difficulty of finding a KL projection of p to $\mathcal{F}$ depends on the family $\mathcal{F}$. When $\mathcal{F}$ is the family of decomposable models for a given junction tree, computing the KL projection is particularly simple:

Theorem 2.6. *Let $p(\mathbf{x})$ be a distribution over a set of variables $\{X_i : i \in V\}$ with indices V, and let $(T, \mathbf{C})$ be a junction tree over the same indices. Then the KL projection of $p(\mathbf{x})$ to the family of decomposable models with the junction tree $(T, \mathbf{C})$ is*

$$q^*(\mathbf{x}) = \frac{\prod_{i \in N_T} p(\mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E_T} p(\mathbf{x}_{S_{i,j}})}.$$

In other words, when projecting p to a family of decomposable models with a fixed junction tree $(T, \mathbf{C})$, the best approximation is the one that sets the clique marginals for the approximation equal to the marginals of the exact distribution, $q^*(\mathbf{x}_{C_i}) = p(\mathbf{x}_{C_i})$ and similarly for the separators.

Going back to the example earlier in this section, we see that the approximate model (2.37) can be expressed as a decomposable model

$$\tilde{p} = \frac{p(h_1, h_2, h_3, z_1, z_3) \times p(h_3, h_4, h_5, z_4, z_5)}{p(h_3)},$$

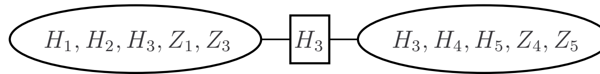


Figure 2.24: The junction tree for the distribution in (2.37).

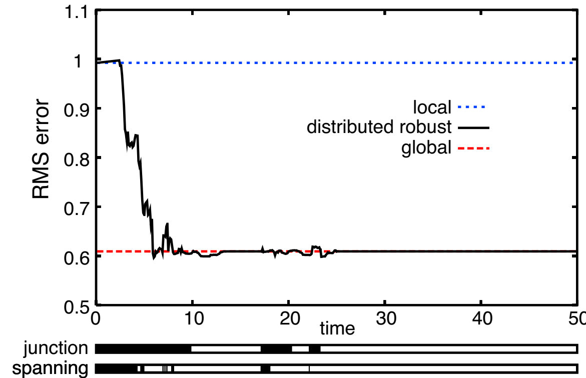


Figure 2.25: Convergence of the robust message passing algorithm on the sensor calibration problem in Example 2.1, taken from (Paskin and Guestrin, 2004b). Unlike the sum–product algorithm in Figure 2.15, whose partial beliefs were inaccurate, the partial beliefs of the robust message passing algorithm are very close in accuracy to the exact marginals (marked as "global"). Remarkably, the algorithm obtains accurate results even before a valid junction tree is formed.

with the junction tree shown in Figure 2.24. This decomposable model carries both hidden and observed variables in its cliques, but it is conceptually no different from the decomposable models we have seen so far: it describes a distribution as a ratio of clique and separator marginals. By Theorem 2.6, the distribution $\tilde{p}$ is a KL projection of a marginal of the joint distribution $p(\mathbf{x}, \mathbf{z})$ to the family of decomposable models with the junction tree in Figure 2.24. All distributions in this family satisfy the conditional independence assumption $H_1, H_2 \perp H_4, H_5 \mid H_3$; of these distributions, $\tilde{p}$ is the one most similar to (a marginal of) the joint distribution $p(\mathbf{x}, \mathbf{z})$. Thus, in computing the partial belief (2.36), the robust message passing algorithm introduces independence assumptions to the model in a principled manner.

In general, the partial correctness guarantees of the robust message passing algorithm are more complicated, because a node may not be able to form a junction tree for the cliques in its partial belief. In this case, the algorithm performs an additional projection step that "shrinks" the cliques, so that they do form a P/L decomposable model. Furthermore, the likelihood information available to a node may not be exact, because some of its terms may have been computed by the nodes before they obtained a converged versions of their messages. Nevertheless, it can be shown that the partial belief at a node is obtained by a sequence of projection and marginalization operations (Paskin, 2004). Since the algorithm makes principled approximations in its partial beliefs, it obtains accurate results very quickly, as illustrated in Figure 2.25.

2.4 Related work

Some centralized algorithms for probabilistic inference have a message-passing flavor and can be distributed easily. A popular algorithm for approximate inference is **loopy belief propagation** (LBP) (Pearl, 1988). LBP can be viewed as an extension of the sum–product algorithm to graphs with cycles. However, unlike the sum–product algorithm, which relied on a junction tree to define its messages, LBP sends its messages directly between the variables adjacent in the Markov network (assuming that factors have at most two arguments). In a distributed setting, each variable is assigned to a node, and whenever the algorithm computes a message between a pair of variables, the message is transmitted between the corresponding pair of nodes. The message computation is very simple, requiring only local information about the variable and its neighbors. Crick and Pfeffer (2003) used this observation to argue that LBP is an "ideal computational and communication framework for sensor networks". Indeed, there have been some successful applications of loopy belief propagation in the context of sensor network localization (Ihler et al., 2004). Unfortunately, LBP is not guaranteed to converge, and it can be only applied to a restricted class of models.[4] If LBP converges, it produces overconfident estimates. On the other hand, assuming a stable network junction tree can be formed, the robust message passing algorithm is guaranteed to converge to the exact solution and its partial beliefs make principled approximations.

In the inference algorithms considered in this chapter, the network junction tree specifies the communication topology for the network. We have briefly mentioned that this tree can be optimized to lower the communication and computational complexity of the inference algorithms. Similar optimizations can be performed for loopy belief propagation. In loopy belief propagation, the algorithm incurs communication cost whenever it needs to send messages between variables that reside on two different nodes. Schmidt and Aberer (2006) proposed a spring relaxation algorithm that optimizes the placement of variables onto the network nodes, placing tightly connected clusters of variables onto the same node. The algorithm ensures that the resulting placement is load-balanced, using the information provided by the P-Grid overlay network (Aberer, 2001). The inference architecture (Paskin et al., 2005) does not explicitly provide load-balancing, but it may be possible to extend its optimization layer to only permit local moves that do not introduce unbalanced cliques.

2.5 Discussion

In this chapter, we discussed the distributed probabilistic inference problem, in which nodes wish to compute marginal distributions over query variables, given all the observations made in the network. We reviewed the robust message passing algorithm for solving this problem robustly, in face of communication delays and node failures. The algorithm represents the posterior distribution as a graphical model—a

[4] For Gaussian models, LBP can be only applied if all the pairwise factors are normalizable; otherwise, the messages and beliefs may not be well-defined.

P/L decomposable model—and computes the marginals by pruning leaf cliques of this model at convergence. Before convergence, the algorithm makes principled approximations by introducing independence assumptions into the joint model. A key component of the algorithm is an overlay network—a network junction tree—that helped the nodes identify redundant parts of the model. Interestingly, in robust message passing, the overlay network and the inference algorithm are integrated: the inference algorithm specifies the local variables at each node; these variables, together with the tree topology, define the cliques and the separators of the network junction tree. The separators then, in turn, determine which parts of the model can be pruned. We will see another instance of such an integration later in Chapter 5.

A potential concern is how much one gains by shifting the work from online inference to initial reparameterization of the prior distribution as a decomposable model. Initial reparameterization is meaningful when the inference task is executed several times. In this case, the cost of computing the structure of the decomposable model and the clique priors is amortized over several executions of the inference algorithm. Furthermore, some algorithms naturally generate prior distributions in the form of a decomposable model, which makes the robust message passing algorithm a prime candidate for inference. We will see an example of such a setting in the next chapter when we discuss inference in dynamical systems.

Appendix 2.A Proofs

Proof of Lemma 2.2. The lemma is proved by starting from an arbitrary junction tree $(T, \mathbf{C})$ and iteratively replacing any disagreeing edges $\{i, j\} \in E_T : i, j \in M$ with equivalent edges in T'.

Let $\{i, j\}$ be a pair of vertices s.t. $i, j \in M$, $\{i, j\} \in E_T$, but $\{i, j\} \notin E_{T'}$. Let (X, Y) be a cut of T that is obtained by removing $\{i, j\}$ from T, and let $(X', Y') = (X \cap M, Y \cap M)$ be the corresponding cut in T'. Let $\{k, \ell\}$ be any edge on the path from i to j in T' that crosses the cut (X', Y'); such an edge must exist, since i and j are on opposite sides of the cut. Furthermore, since the edge $\{i, j\}$ connects the two subtrees of T over the vertex sets X and Y, $\{i, j\}$ must be on the unique path between k and ℓ in T. Therefore, by the running intersection property

$$S_{k,\ell} = C_k \cap C_\ell \subseteq C_i \cap C_j = S_{i,j}.$$

However, since $(T', \mathbf{C}')$ is a maximum–intersection tree, $|S_{k,\ell}| \geq |S_{i,j}|$. Therefore, $S_{i,j} = S_{k,\ell}$, and if we replace the edge $\{i, j\}$ with $\{k, \ell\}$ in T, then $(T, \mathbf{C})$ is still a junction tree. This process is monotonic: we never remove an edge that is in $E_{T'}$. Therefore, after finitely many such swaps, all edges among M in T are also present in T'. □

Proof of Theorem 2.3. Let $(T^U, \mathbf{C}^U)$ be a maximum–intersection tree for $\mathbf{C}^U$, and let $(T, \mathbf{C})$ be the tree constructed by Lemma 2.2 with $\mathbf{C}' = \mathbf{C}^U$. Since $(T, \mathbf{C})$ is a tree, any subset of edges $E \subseteq E_T$ forms a collection of disjoint subtrees $\{T_u : u \in M\}$, where $\{N_{T_u} : u \in M\}$ is a partition of the vertices N_T. Let

$\{T_u : u \in M\}$ be the collection of subtrees, formed by the edges whose clique intersection meets U,

$$E = \{\{i, j\} \in E_T : C_i \cap C_j \cap U \neq \emptyset\}.$$

We can show that the subtrees $\{T_u : u \in M\}$ have a property that the intersection of cliques at two different subtrees does not meet U: if $i \in N_{T_u}$ and $j \in N_{T_v}$ with $u \neq v$, then $C_i \cap C_j \cap U = \emptyset$; we say that the subtrees are **disjoint with respect to** U. This property is a direct consequence of the running intersection property of $(T, \mathbf{C})$: let $\{k, \ell\} \in E_T$ be some edge on the unique path between i and j in T that crosses between the subtrees, that is, $\{k, \ell\} \notin E_{T_w}, \forall w \in M$. Then

$$C_i \cap C_j \cap U \subseteq C_k \cap C_\ell \cap U = \emptyset.$$

Since the subtrees $\{T_u : u \in M\}$ are disjoint with respect to U, they can be treated in isolation: we only need to show the condition

$$\{i, j\} \in E_T \iff \{i, j\} \in E_{T^U} \tag{2.38}$$

in the definition of U-subgraph consistency for pairs of vertices $i, j \in N_{T_u}$ within each subtree T_u. Let T_u be any subtree with more than one vertex. Since the separators of T_u meet U, so do its cliques. But $\mathbf{C}^U$ contains all the cliques that meet U; thus, $N_{T_u} \subseteq N_{T^U}$. By Lemma 2.2, $E_{T_u} \subseteq E_{T^U}$, which proves the forward direction of (2.38). But since T^U is acyclic, there are no additional edges among N_{T_u} in T^U, which proves the reverse direction of (2.38). $\square$

Chapter 3

Distributed inference in dynamical systems

In the previous chapter, we considered a static inference problem, where nodes make noisy measurements and wish to estimate the hidden state of a system, given all the measurements made by the network. While this problem is useful in some settings, the state in many systems changes over time. To describe such dynamical systems, the probabilistic models need to capture the entire evolution of the state and observations. A frequent inference task is then to compute a marginal distribution over the state at the latest time, given all the observations made by the network so far. In this chapter, we show that this task can be reduced to a sequence of (static) estimation problems, solved using the robust message passing algorithm, reviewed in the previous chapter. While simple, this reduction does not guarantee robustness: if the communication network is partitioned, the estimates in different parts of the network may diverge. We show that these inconsistencies can lead to poor solutions when the nodes attempt to combine their estimates once the communication is restored. We describe a distributed algorithm that resolves such inconsistencies, while retaining as much information in the estimates as possible.

3.1 Centralized inference in dynamical systems

Our distributed algorithm builds upon standard formalisms and algorithms in centralized inference. The dynamical system is once again described with a graphical model that represents the distribution in a factorized form. However, despite the factorized representation, exact inference is intractable in this model, but the task can be solved approximately by periodically projecting the distribution to a tractable representation. We briefly describe one instance of such an approach, the Boyen-Koller algorithm (Boyen and Koller, 1998). This algorithm lays the foundation for our distributed inference algorithm, presented later in this chapter.

3.1.1 Dynamic Bayesian networks

To describe a dynamical system, we need a model that characterizes the entire temporal evolution of the system. We model the system as a **dynamic Bayesian network** (DBN) (Dean and Kanazawa, 1990). A DBN consists of a set of **state processes** $\mathbf{X} = \{X_a : a \in V\}$; these random processes characterize the state of the system environment. Each state process X_a represents the evolution of one quantity over discrete time and can be viewed a sequence of **state variables** $X_a^{(t)}$, indexed by discrete time $t \geq 0$. In addition, a DBN contains a set of **observation processes** $\mathbf{Z} = \{Z_a : a \in E\}$; each observation process can be viewed as a sequence of **observed variables** $Z_a^{(t)}$. Each observation process Z_a corresponds to one of the sensors on one of the nodes; state processes are not necessarily associated with unique nodes.

A DBN enforces a certain structure among the the state and the observation variables. For example, consider the scenario in Figure 3.1(a) that contains temperature sensors at four locations in a hallway, with a fan blowing the air downwards. We can assume that initially, before any observations are made, the temperatures $H_a^{(0)}$ are independent, so the distribution over the temperatures at time step 0 can be written as a product of marginals:

$$p(\mathbf{h}^{(0)}) = \prod_{a=1}^{4} p(h_a^{(0)}).$$

In the context of DBNs, the distribution $p(\mathbf{h}^{(0)})$ is called the **initial prior**. In general, the initial prior is represented as a Bayesian network:

$$p(\mathbf{x}^{(0)}) = \prod_{a \in V} p(x_a^{(0)} \mid \mathbf{x}^{(0)}_{\mathrm{Pa}[X_a^{(0)}]}),$$

where $\mathrm{Pa}[X_a^{(0)}]$ are the parents of $X_a^{(0)}$ in the initial time step.

As the time progresses, the heat gets diffused in the environment, with the fan driving the diffusion process downwards. Therefore, the temperature $H_a^{(t)}$ at time step t is influenced only by the temperatures at locations a and $a-1$ at the previous time step. This assumption is captured by a **transition model** that factorizes as a product of conditional probability distributions for each location:

$$p(\mathbf{h}^{(t)} \mid \mathbf{h}^{(t-1)}) = p(h_1^{(t)} \mid h_1^{(t-1)}) \times \prod_{a=2}^{4} p(h_a^{(t)} \mid h_a^{(t-1)}, h_{a-1}^{(t-1)})$$

This transition model represents the assumption that the temperature $H_a^{(t)}$ is independent of all other temperatures at time steps t and $t-1$, given the $H_a^{(t-1)}$ and $H_{a-1}^{(t-1)}$. In general, the transition model describes the evolution of the system separately for each process X_a as a conditional probability distribution that relates the variable $X_a^{(t)}$ at time t to a subset of the variables at the previous time step $t-1$, called the

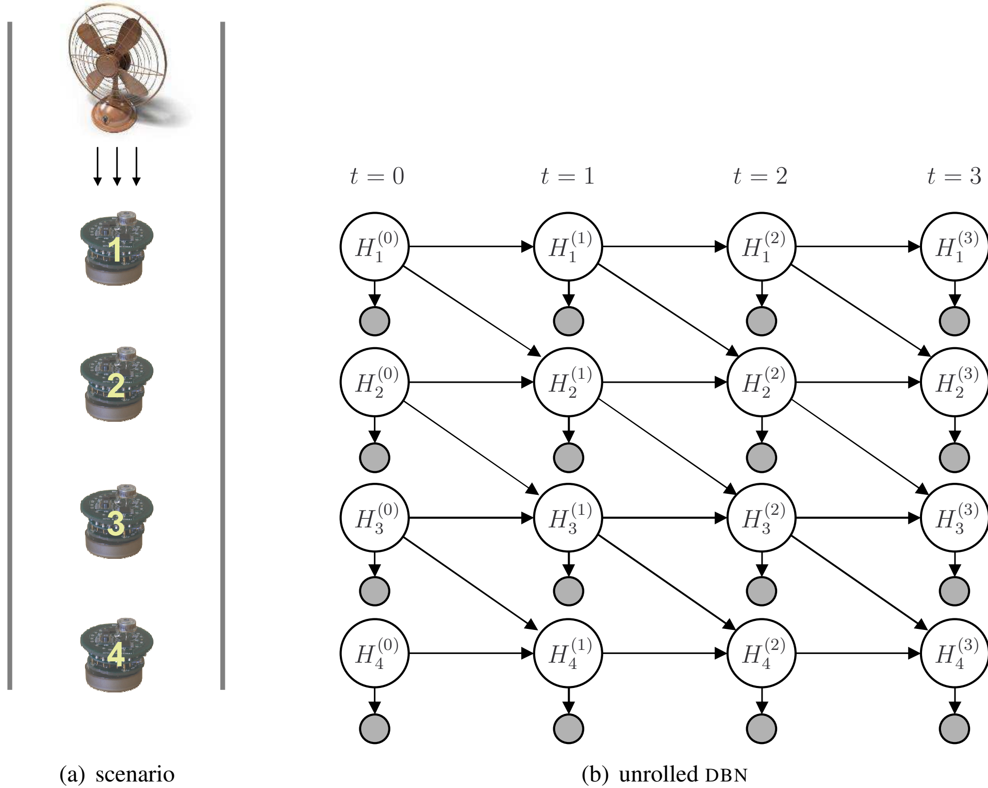


Figure 3.1: (a) A scenario with temperature sensors at four locations in a hallway. A fan is driving the heat diffusion process downwards. (b) The corresponding unrolled DBN. The temperature at location a depends on the temperatures at locations a and $a-1$ at the previous time step. The observed variables are shaded.

parents of $X_a^{(t)}$. The transition model is a product of these conditional probability distributions:

$$p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) = \prod_{a \in V} p(x_a^{(t)} \mid \mathbf{x}_{\text{Pa}[X_a]}^{(t-1)}),$$

where $\text{Pa}[X_a] \subseteq V$ are the indices of parents of $X_a^{(t)}$.[1]

So far, we have discussed the system in the absence of observations. Typically, the sensors measure the temperature *instantaneously* and *locally*: the temperature reading $Z_a^{(t)}$ is determined only by the true (hidden) temperature $H_a^{(t)}$ at the same time step t and at the same location a. Furthermore, the temperature measurement at one sensor does not affect the measurements at other sensors. These assumptions can be captured by a homogeneous **observation model** that specifies the distribution of the observed variables at

[1]The indices of the parents are the same in all time steps $t \geq 1$, so we omit the time designation.

each time step, given the state variables at that time step:

$$p(\mathbf{z}^{(t)} \mid \mathbf{h}^{(t)}) = \prod_{a=1}^{4} p(z_a^{(t)} \mid h_a^{(t)}).$$

In general, the observation model can be written as a product

$$p(\mathbf{z}^{(t)} \mid \mathbf{x}^{(t)}) = \prod_{a \in E} p(z_a^{(t)} \mid \mathbf{x}_{\text{Pa}[Z_a]}^{(t)}),$$

where $\text{Pa}[Z_a] \subseteq V$ are the indices of parents of $Z_a^{(t)}$ in the current time step.

Together, the initial prior, the transition model, and the observation model define a joint distribution over the hidden variables and the observed variables for an arbitrary number of time steps:

$$p(\mathbf{x}^{(0:T)}, \mathbf{z}^{(0:T)}) = \underbrace{p(\mathbf{x}^{(0)})}_{\text{initial prior}} \times \prod_{t=1}^{T} \underbrace{p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)})}_{\text{transition model}} \times \prod_{t=0}^{T} \underbrace{p(\mathbf{z}^{(t)} \mid \mathbf{x}^{(t)})}_{\text{observation model}} . \tag{3.1}$$

Here, we write $0 : T$ to denote the time indices $0, \ldots, T$. Thus, the joint distribution is the product of the initial prior at time step 0, the transition model for each time step $t \geq 1$, and the observation model for each time step $t \geq 0$. This expansion can be represented graphically as a Bayesian network, where each vertex is either a state variable or an observed variable. As usual, the edges in this Bayesian network represent the dependences among the variables. The procedure of obtaining the Bayesian network for the joint distribution (3.1) is called **unrolling the DBN**. An example of an unrolled DBN for the temperature example is shown in Figure 3.1(b).

We illustrate the DBN formalism with the following three additional examples that will be important later in this chapter:

Example 3.1. *A frequent task in sensor networks is **tracking**, where the network estimates the trajectory of a moving object. A number of sensors are placed in an environment at known locations. Whenever the object is in the range of a sensor, an observation is generated that indicates a noisy measurement of the position of the object relative to the sensor's own location. For example, for wireless sensor networks, the observation may be the perceived distance of the object to the sensor; for camera networks, the observation is represented by a point in the image plane. These relative observations are combined with the known locations of the sensors and with a model of the object motion to generate an estimate of the position of the object at each time step.*

*The position of the object is represented as a real vector $M^{(t)}$. The initial prior is simply $p(m^{(0)})$, the initial estimate of the object position. The transition model $p(m^{(t)} \mid m^{(t-1)})$ (also called the **motion model**) represents our knowledge of the object motion before any observations are made. Typically, the motion of the object is modeled as a conditional linear Gaussian distribution $M^{(t)} \sim \mathcal{N}\left(AM^{(t-1)} + b, \Sigma\right)$. For*

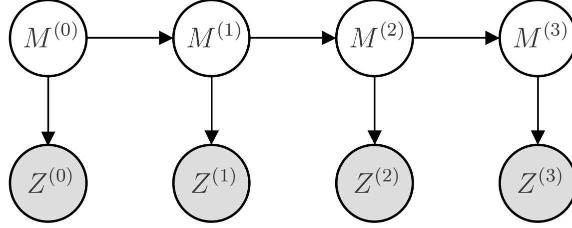


Figure 3.2: The unrolled DBN for tracking with a single sensor. The model consists of a single hidden process M and a single observation process Z.

example, if nothing is known about the motion of the object beyond smoothness, we set $A = I$, $b = \mathbf{0}$, *and* $\Sigma = \sigma^2 I$, *which corresponds to the Brownian motion*

$$M^{(t)} = M^{(t-1)} + \epsilon,$$

where $\epsilon \sim \mathcal{N}\left(0, \sigma^2 I\right)$ *is white Gaussian noise. Finally, each sensor observation* $\overline{z}_a^{(t)}$ *is drawn from the conditional distribution* $p(z_a^{(t)} \mid m^{(t)})$. *This distribution depends on the type of the sensor measurements. For example, the model for a camera network is described in Chapter 4. Together, these parts define the joint model over time steps* $0 : T$ *as*

$$p(m^{(0:T)}, \mathbf{z}^{(0:T)}) = \underbrace{p(m^{(0)})}_{\textit{initial prior}} \times \prod_{t=1}^{T} \underbrace{p(m^{(t)} \mid m^{(t-1)})}_{\textit{transition model}} \times \underbrace{\prod_{t=0}^{T} \prod_{a} p(z_a^{(t)} \mid m^{(t)})}_{\textit{observation model}} .$$

The unrolled DBN for this model is shown in Figure 3.2.

Tracking is a simple example of an inference task that can be modeled with a DBN. The following example is an extension of tracking to the case with unknown sensor locations:

Example 3.2. *Manually measuring the location of each sensor in a sensor network is an error-prone and time-consuming task; in some settings, such as emergency response systems, manually measuring the location of each sensor is outright prohibitive. Suppose that a moving object is in the range of a sensor and, a few moments later, the same object is observed by another sensor. If we knew the trajectory of this object, we could infer information about the relative position of the two sensors automatically. Yet, without an independent positioning system like GPS, the trajectory of the object is unknown. Naturally, if we knew the locations of the sensors, we could infer the trajectory of the object by tracking. Therefore, we can address the sensor localization task by solving a* ***simultaneous localization and tracking*** *(SLAT) problem, where we estimate both the trajectory of the object and the locations of the sensors.*

The trajectory of the object is modeled as a random process M *that captures the position of the object* $M^{(t)}$ *at each time step* t. *In addition, the model contains one random variable* L_a *for each sensor* a *that represents the location of the sensor. Note that* L_a *is not time-dependent; the location of the sensor is assumed to be fixed throughout the experiment. The initial prior over* L_a *is independent of other sensor*

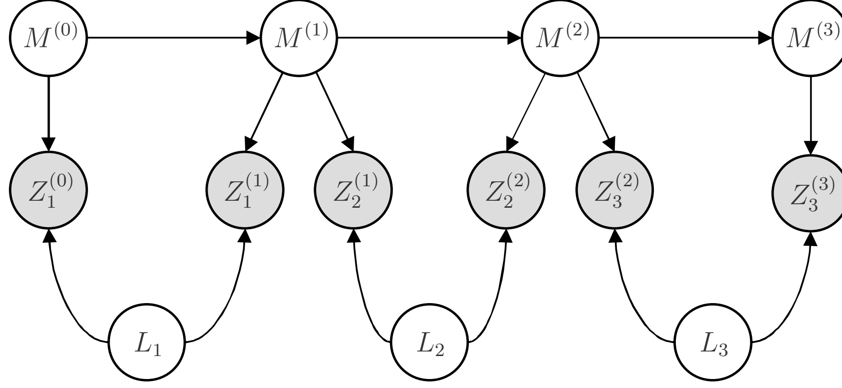


Figure 3.3: The unrolled DBN for simultaneous localization and tracking. The model contains variables for three sensors; each sensor observes the object at two consecutive time steps. Here, we disregard negative observations (observations when an object appears to not be in the range of the sensor).

locations and is uninformative, except to set the global reference frame, as discussed in Section 4.2.1. Whenever the object is in the range of a sensor, an observation $\overline{z}_a^{(t)}$ *is generated by the sensor. The observed variable* $Z_a^{(t)}$ *depends on the location of the sensor and the current position of the object, so its distribution can be described by the observation model* $p(z_a^{(t)} \mid l_a, m^{(t)})$*. Thus, the joint model over time steps* $0 : T$ *is*

$$p(\mathbf{l}, m^{(0:T)}, \mathbf{z}^{(0:T)}) = \underbrace{\prod_a p(l_a) \times p(m^{(0)})}_{\text{initial prior}} \times \prod_{t=1}^{T} \underbrace{p(m^{(t)} \mid m^{(t-1)})}_{\text{transition model}} \times \prod_{t=0}^{T} \underbrace{\prod_a p(z_a^{(t)} \mid l_a, m^{(t)})}_{\text{observation model}} .$$

The unrolled DBN for this model is shown in Figure 3.3.

In the previous example, the model contained a static component—the locations of the sensors. The methods presented in this chapter can be adjusted to explicitly handle such a static component. However, to fit the example into the DBN framework, each static variable can be treated as a random process X, whose value does not change over time. Such a random process has an identity transition model

$$p(x^{(t)} \mid x^{(t-1)}) = \begin{cases} 1 & \text{if } x^{(t)} = x^{(t-1)}, \\ 0 & \text{otherwise} \end{cases}$$

when the process is discrete and similarly for the continuous case.

Our last example is the extension of Example 2.1 to the dynamic setting:

Example 3.3. *In Example 2.1, we examined the task of automatically removing biases from a temperature sensor network. We considered the* ***static calibration*** *task, where the temperatures were treated as fixed quantities. In practice, it is often possible to measure the temperatures over a period of time, that is, we may wish to solve the* ***dynamic calibration*** *task. By gathering more observations of the temperatures, we*

may hope to eliminate more of the measurement bias.

The state processes in this problem are the true temperatures H_a, one at each location a. At each time step t, the temperature at location a depends on the temperature at the nearby locations $Pa[H_a]$ at the previous time step $t-1$. This dependency is captured in the transition model $p(h_i^{(t)} \mid h_{Pa[H_a]}^{(t-1)})$. In addition, each sensor a is associated with measurement bias B_a; this bias is assumed to be fixed throughout the experiment. The measurement processes are the observed temperatures Z_a at each location a, with the observation model $p(z_a^{(t)} \mid h_a^{(t)}, b_a)$. Together, these parts define the joint model over T time steps as

$$p(\mathbf{h}^{(0:T)}, \mathbf{b}, \mathbf{z}^{(0:T)}) = \underbrace{\prod_a p(h_a^{(0)}) \times \prod_a p(b_a)}_{\textit{initial prior}} \times \prod_{t=1}^{T} \underbrace{\prod_a p(h_a^{(t)} \mid h_{Pa[H_a]}^{(t-1)})}_{\textit{transition model}} \times \prod_{t=0}^{T} \underbrace{\prod_a p(z_a^{(t)} \mid h_a^{(t)}, b_a)}_{\textit{observation model}}.$$

3.1.2 Exact filtering

A frequent inference task in DBNs is **filtering**. In filtering, we wish to compute the **posterior distribution** $p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ for $t = 1, 2, \ldots$ as the observations $\overline{\mathbf{z}}^{(0)}, \overline{\mathbf{z}}^{(1)}, \ldots$ arrive. This distribution represents our knowledge of the hidden state (such as the sensor locations and the latest object position), given all the observations made so far. Conceptually, filtering requires expanding the DBN model for t time steps, instantiating the variables $\mathbf{Z}^{(0:t)}$ to $\overline{\mathbf{z}}^{(0:t)}$, and computing the marginal over the state variables at the latest time step $\mathbf{X}^{(t)}$. The posterior distribution $p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ can be further summed up to compute the marginal over a set of query variables, $p(\mathbf{x}_Q^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$.

For any time step t, we will call $p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$ the **prior distribution** at time step t. This is the distribution over the state at time t that incorporates all observations up to, but not including, time step t. The the prior distribution at time step 1 is simply the initial prior $p(\mathbf{x}^{(0)})$. The prior distribution at time step $t+1$ can be expressed in terms of the prior distribution at time step t:

$$\underbrace{p(\mathbf{x}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})}_{\text{prior at } t+1} \propto \sum_{\mathbf{x}^{(t)}} \underbrace{p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})}_{\text{prior at } t} \underbrace{p(\overline{\mathbf{z}}^{(t)} \mid \mathbf{x}^{(t)})}_{\text{observation likelihood}} \underbrace{p(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)})}_{\text{transition model}}.$$

This recursion leads to the following three-step procedure that computes the prior distribution at time step $t+1$ from the prior at time step t:

1. **Estimation**: Condition on the latest observations, by computing

$$\begin{aligned} p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) &\propto p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) \times p(\overline{\mathbf{z}}^{(t)} \mid \mathbf{x}^{(t)}) \\ &= p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) \times \prod_a p(\overline{z}_a^{(t)} \mid \mathbf{x}_{\text{Pa}[Z_a]}^{(t)}). \end{aligned}$$

Therefore, in order to condition on the latest observations, we multiply in the observation likelihood

for each observed variable $Z_a^{(t)}$ and renormalize. The result is the posterior distribution at time step t, which we also call the **belief** at time step t.

2. **Prediction**: Extend the belief to include the state at the next time step,

$$\begin{aligned} p(\mathbf{x}^{(t)}, \mathbf{x}^{(t+1)} \,|\, \overline{\mathbf{z}}^{(0:t)}) &= p(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t)}) \times p(\mathbf{x}^{(t+1)} \,|\, \mathbf{x}^{(t)}) \\ &= p(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t)}) \times \prod_a p(x_a^{(t+1)} \,|\, \mathbf{x}_{\mathrm{Pa}[X_a]}^{(t)}) \end{aligned}$$

   That is, we multiply in the transition model for each state variable $X_a^{(t)}$. The result is a distribution over the state at two consecutive time steps, conditioned on all observations up to time t.

3. **Roll-up**: Eliminate the state variables at time step t from the belief:

$$p(\mathbf{x}^{(t+1)} \,|\, \overline{\mathbf{z}}^{(0:t)}) = \sum_{\mathbf{x}^{(t)}} p(\mathbf{x}^{(t)}, \mathbf{x}^{(t+1)} \,|\, \overline{\mathbf{z}}^{(0:t)}).$$

   This is the prior distribution at time step $t + 1$.

At each time step, the prior distribution and the belief are represented as a factorized probability model. The filtering steps manipulate these models by multiplying in factors (in estimation and prediction) and by summing out variables (in roll-up). Thus, exact filtering is similar in spirit to variable elimination, discussed in Section 2.1.3. Since exact filtering works with a factorized representation of the posterior distribution, we may hope that it is efficient. Unfortunately, this is not the case, as we will now show.

The complexity of exact reasoning about a distribution is affected by the **conditional independence structure** of the distribution, that is, the set of conditional independence assumptions, satisfied by the distribution. The models in Chapter 2 had a significant conditional independence structure, and variable elimination was able to compute marginals without ever creating "large" factors, that is, factors with many arguments. In DBNs, the initial prior typically also has significant conditional independent structure. In Example 3.3, the temperatures and the biases are independent in the initial prior; Figure 3.4(a) shows that there are no active paths among the temperatures at time step 0. However, as the time progresses, the belief quickly loses conditional independence structure, because influence can flow through variables in the past (common cause) and through common observed effect, as illustrated in Figures 3.4(b) and 3.4(c). After a while, none of the variables in the belief are independent, or even conditionally independent given other variables in the same time step.

Since the belief does not satisfy any conditional independence assumptions, it can be only represented as a dense Markov network that forms a complete graph over the variables at the current time step. This fact is critical for inference: even if the belief can be represented in a factorized form, any marginalization operations on this distribution will immediately create a large factor that includes all the variables at that

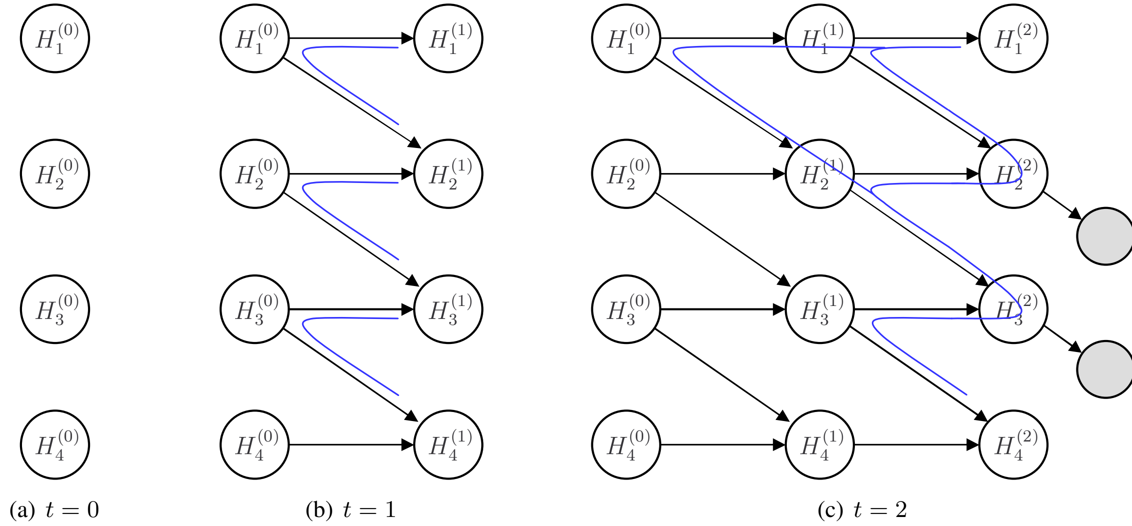


Figure 3.4: Active paths in the temperature network. After a few time steps, there are no conditional independences among the variables at the latest time step.

time step. Working with such a large factor is very expensive or typically intractable.

3.1.3 Assumed density filtering

We have seen that exact filtering is not efficient, because the belief quickly loses conditional independence structure. Nevertheless, while none of the conditional independences hold in the belief exactly, some may hold approximately. For example, the variables $H_1^{(2)}$ and $H_3^{(2)}$ are approximately conditionally independent, given $H_2^{(2)}$, because the active paths between these two variables pass through several transition models and common effects, each of which represents only a noisy relationship between an adjacent pair of variables. Therefore, if we were to enforce this independence assumption in the belief, we would expect that the resulting approximate belief would be in some sense close to the exact posterior distribution.

How do we enforce a set of independence assumptions in a belief? Recall that KL projection (Definition 2.9) is one mechanism to choose a distribution $\tilde{p}$ that is most similar to the exact distribution p, as measured by the KL divergence from p to $\tilde{p}$. In order to enforce a set of independence assumptions in the belief, we choose a junction tree $(T, \mathbf{C})$ with small cliques C_i; one such junction tree for the temperature example is shown in Figure 3.5(a). We then compute a KL projection from the belief to the family of decomposable models with junction tree $(T, \mathbf{C})$. Theorem 2.6 shows that computing the KL projection is easy: we simply need to compute the marginal of p over the cliques and the separators of $(T, \mathbf{C})$. This observation motivates the algorithm of Boyen and Koller (1998), hereby denoted "B&K98". At each time step, the algorithm projects the belief to a tractable distribution, represented by a fixed junction tree. Because the algorithm "assumes" an approximate representation for the belief, it is an instance of **assumed**

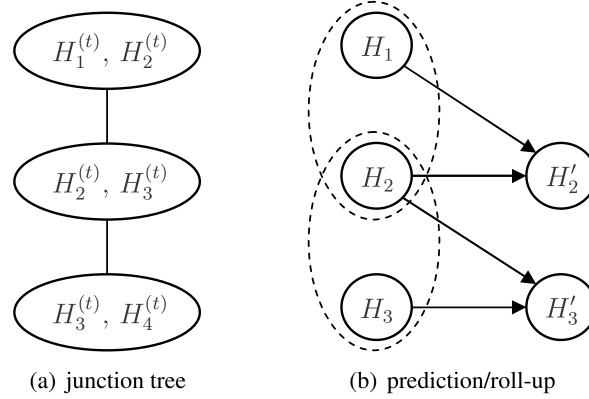


Figure 3.5: (a) A junction tree that enforces a natural set of independence assumptions in the belief. (b) Prediction/roll-up step in the B&K98 algorithm. The dashed ellipses indicate the cliques of the minimal subtree that covers the parents of H'_2 and H'_3.

density filtering, and we will sometimes refer to the approximate density as the **assumed density**.

In the B&K98 algorithm, the approximate prior distribution and the belief are maintained as a decomposable model. The algorithm approximates the true prior $p(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)})$ as a decomposable model $\tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)})$ with junction tree $(T, \mathbf{C})$:

$$p(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)}) \approx \tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)}) = \frac{\prod_{i \in N_T} \tilde{p}(\mathbf{x}_{C_i}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)})}{\prod_{\{i,j\} \in E_T} \tilde{p}(\mathbf{x}_{S_{i,j}}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)})}. \tag{3.2}$$

The algorithm begins by reparameterizing the initial prior $p(\mathbf{x}^{(0)})$ as a decomposable model with junction tree $(T, \mathbf{C})$. This is performed by setting $\tilde{p}(\mathbf{x}_{C_i}^{(0)}) \triangleq p(\mathbf{x}_{C_i}^{(0)})$. The algorithm then recursively computes the approximate prior at step $t+1$ from the approximate prior at step t.

Recall that the recursive formulation of exact filtering begins with the estimation step that conditions on the observations at time t. In exact filtering, this step was accomplished by multiplying in the observation likelihood into the prior distribution at time t. In the B&K98 algorithm, we similarly condition on the observations, by multiplying the exact observation likelihoods into the approximate prior:

$$\tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t)}) \propto \tilde{p}(\mathbf{x}^{(t)}, \overline{\mathbf{z}}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)}) = \tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)}) \times \prod_a p(\overline{z}_a^{(t)} \,|\, \mathbf{x}_{\mathrm{Pa}[Z_a]}^{(t)}). \tag{3.3}$$

This procedure yields an approximate posterior distribution $\tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t)})$. However, (3.3) is not yet a decomposable model: it is a product of a decomposable model and one or more observation likelihoods. It is desirable to reparameterize the $\tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t)})$ as a decomposable model, to leverage properties, such as marginalization by pruning (Property 2.2). Fortunately, the distribution can be reparameterized easily: we multiply each observation likelihood $p(\overline{z}_a^{(t)} \,|\, \mathbf{x}_{\mathrm{Pa}[Z_a]}^{(t)})$ into some clique that covers the likelihood arguments, $\mathrm{Pa}[Z_a] \subseteq C_i$ and re-run the Lauritzen–Spiegelhalter algorithm (Lauritzen and Spiegelhalter, 1988).

The result is a representation of the belief where the clique and separator marginals have been conditioned on the observations:

$$\tilde{p}(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) = \frac{\prod_{i \in N_T} \tilde{p}(\mathbf{x}_{C_i}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})}{\prod_{\{i,j\} \in E_T} \tilde{p}(\mathbf{x}_{S_{i,j}}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})}. \tag{3.4}$$

The second step of the B&K98 is a variation of the prediction and roll-up steps of the exact filtering procedure. Recall that in exact filtering, we multiply the posterior distribution $p(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ by the transition model $p(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)})$ and marginalize out $\mathbf{X}^{(t)}$ to obtain the prior distribution at time $t+1$. We could perform the same computation, starting with the approximate posterior distribution $\tilde{p}$:

$$\tilde{p}_1(\mathbf{x}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)}) = \sum_{\mathbf{x}^{(t)}} \tilde{p}(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) \times p(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)}).$$

To keep the representation tractable, we would then project $\tilde{p}_1$ to the family of decomposable models with the junction tree $(T, \mathbf{C})$ using Theorem 2.6. Unfortunately, $\tilde{p}_1$ can be a dense distribution: after just one roll-up, the distribution can lose much of its independence structure. Therefore, we need to compute the projection of $\tilde{p}_1$ to $(T, \mathbf{C})$ without ever constructing the complete distribution $\tilde{p}_1$.

Luckily, computing a projection of $\tilde{p}_1$ to $(T, \mathbf{C})$ is not difficult, as we only need to compute the clique and separator marginals $\tilde{p}_1(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$ and $\tilde{p}_1(\mathbf{x}_{S_{i,j}}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$. We will illustrate this computation on the clique $\{H_2, H_3\}$ in Figure 3.5(a); for brevity, we will omit conditioning on observations, and we will refer to the variables at the current time step as simply X and the variables at the next time step as X'. We can leverage the structure of the DBN to compute the clique marginal at the next time step as

$$\tilde{p}(h_2', h_3') = \sum_{h_1} \sum_{h_2} \sum_{h_3} \tilde{p}(h_1, h_2, h_3) \times p(h_2' \mid h_1, h_2) \times p(h_3' \mid h_2, h_3).$$

Thus, we multiply a marginal belief $\tilde{p}(h_1, h_2, h_3)$ over the parents of H_2' and H_3' with the transition models for H_2' and H_3', as illustrated in Figure 3.5(b), and then eliminate all the variables at the previous time step. The marginal $p(h_1, h_2, h_3)$ can itself be computed very efficiently, because the variables H_1, H_2, and H_3 form a subtree of the junction tree $(T, \mathbf{C})$, so we can apply Property 2.2 to marginalize the belief (3.4) by pruning.

In general, when computing the marginal $\tilde{p}_1(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$ over an arbitrary clique C_i, we need a marginal over the parents $\mathrm{Pa}[\mathbf{X}_{C_i}]$ of this clique at the previous time step. However, the parents may not form a subtree of the junction tree $(T, \mathbf{C})$. In this case, we take a minimal subtree $(T', \mathbf{C}')$, whose clique cover the parents:

$$\mathrm{Pa}[\mathbf{X}_{C_i}] \subseteq \bigcup_{C_i \in \mathbf{C}'} C_i = U,$$

where U is the union of the cliques of T'. By Property 2.2, the subtree represents a decomposable model for the marginal posterior distribution $\tilde{p}(\mathbf{x}_U^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ at time step t. To compute the marginal for the clique

C_i at the next time step, we multiply this decomposable model by the transition models for all variables in C_i and eliminate all the variables at the current time step:

$$\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)}) = \sum_{\mathbf{x}_U^{(t)}} \tilde{p}(\mathbf{x}_U^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) \times \prod_{a \in C_i} p(x_a^{(t+1)} \mid \mathbf{x}_{\text{Pa}[X_a]}^{(t)}). \tag{3.5}$$

We conclude by illustrating the B&K98 algorithm on the SLAT application from Example 3.2. The SLAT model has a structure that greatly simplifies some of the operations in the algorithm. An example scenario is shown in Figure 3.6(a). This scenario consists of eight side-facing and four overhead cameras placed around a hallway. The object makes two loops while being observed by the cameras. As the object moves, we wish to recover the 2D pose (location and orientation) of each camera, $p(l_a \mid \overline{\mathbf{z}}^{(0:t)})$.

A good heuristic for choosing the decomposable approximation in SLAT is to select cliques to cover sets of cameras that are near one another. This heuristics ensures that the strong dependencies among the cameras with overlapping fields of view are captured in the assumed density. Furthermore, since the object position is strongly coupled to the camera poses (via the observations), we add $M^{(t)}$ to all cliques and separators. The approximate prior thus takes on the following form:

$$\tilde{p}(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) = \frac{\prod_{i \in N_T} \tilde{p}(\mathbf{l}_i, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})}{\prod_{\{i,j\} \in E_T} \tilde{p}(\mathbf{l}_{i,j}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})},$$

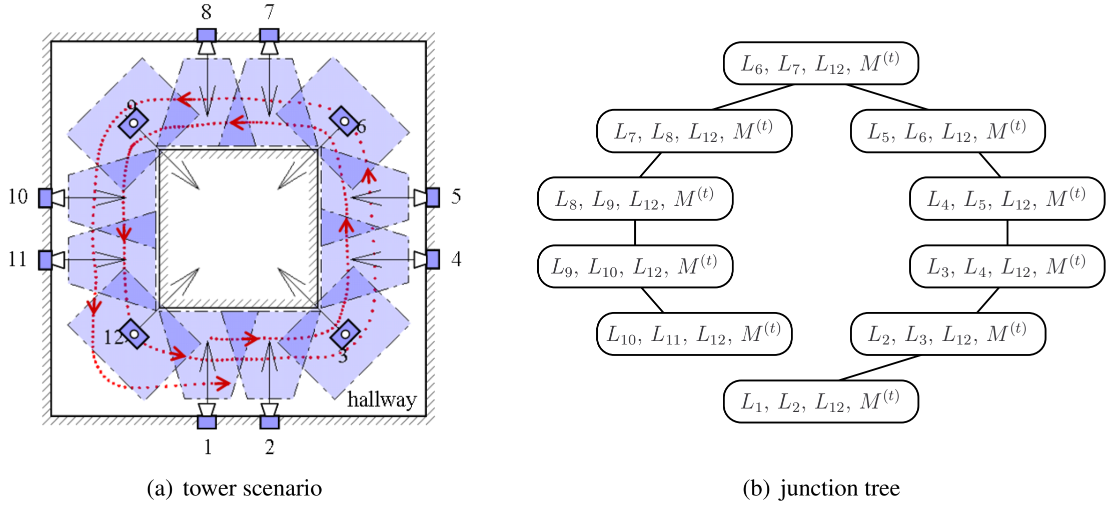
where $\mathbf{l}_i$ are subsets of the camera poses $\mathbf{l}$ assigned to clique i and $\mathbf{l}_{i,j} = \mathbf{l}_i \cap \mathbf{l}_j$ are the poses of the cameras shared by the cliques i and j. An example junction tree for the problem in Figure 3.6(a) is shown in Figure 3.6(b). Section 4.4 provides a more detailed justification for this approximation.

The B&K98 algorithm starts by constructing the decomposable model for the initial prior. Since the camera poses and the object position are independent in the initial prior, each clique marginal $p(\mathbf{l}_i, m^{(0)})$ is simply the product of marginals for individual cameras,

$$p(\mathbf{l}_i, m^{(0)}) = \prod_{l_a \in \mathbf{l}_i} \underbrace{p(l_a)}_{\text{pose prior}} \times \underbrace{p(m^{(0)})}_{\text{prior of starting position}}.$$

In each time step t, the algorithm begins by multiplying the prior by the likelihoods for all the measurements made at that time step. The parents of each observations $Z_a^{(t)}$ are the pose variable for camera a and the object position $M^{(t)}$. Since each clique includes $M^{(t)}$, we can multiply the observation likelihood to any clique that includes the camera pose L_a. After we run the Lauritzen–Spiegelhalter algorithm, we obtain a representation of the approximate posterior where all the cliques have been conditioned on the latest observations $\overline{\mathbf{z}}^{(t)}$:

$$\tilde{p}(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) = \frac{\prod_{i \in N_T} \tilde{p}(\mathbf{l}_i, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})}{\prod_{\{i,j\} \in E_T} \tilde{p}(\mathbf{l}_{i,j}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})}.$$



(a) tower scenario

(b) junction tree

Figure 3.6: SLAT application. (a) An example scenario with eight side-facing and four overhead cameras (in the corners), placed around in a hallway. The long arrows indicate the true location and orientation of the cameras; the dark-shaded regions represent the overlapping fields of view. The dotted line shows the location of the object at each time step. (b) A junction tree for the problem in (a). Each clique contains the pose variables for nearby cameras, as well as the object position $M^{(t)}$.

The prediction and the roll-up phases of filtering can now be implemented very efficiently: for each clique (and separator) marginal, we independently multiply in the object motion model and marginalize out the old object state:

$$\tilde{p}(\mathbf{l}_i, m^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)}) = \sum_{m^{(t)}} \tilde{p}(\mathbf{l}_i, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) \times p(m^{(t+1)} \mid m^{(t)}).$$

This simplification is possible because the clique $\{\mathbf{L}_i, M^{(t)}\}$ is a minimal subtree of $(T, \mathbf{C})$ that covers the parent of $M^{(t+1)}$, along with the static variables $\mathbf{L}_i$.

3.2 The distributed filtering problem

We assume the same networking model as the one considered in distributed inference problem in Section 2.2.1. Each node performs local computation, and the nodes communicate over a lossy channel. We assume message-level errors: messages are either received without error, or not received at all. Only the recipient is aware of successful transmission; neither the sender nor the recipient are aware of a failed transmission. The nodes may enter or leave the network, and the link qualities may change over time. We assume that node clocks are synchronized, so that transitions to the next time step are simultaneous. However, the communication between the nodes is asynchronous.

We assume that each node has a partial view of a global DBN model that describes the environment and

the observations at the nodes. Each node is given a fragment of the initial prior; the fragment is a partial description of $p(\mathbf{x}^{(0)})$ that is initially available to the node. We assume that the fragments are valid, that is, the set of fragments across all the nodes uniquely defines the initial prior $p(\mathbf{x}^{(0)})$. Each node also has access to the transition models for a subset of the state processes. In some applications, such as SLAT, the transition models can be "built in" to the nodes. In other cases, the transition models can be obtained through an initial dissemination or model learning stage.

At each time step, a node makes measurements of its environment. We will denote the indices of the variables observed at node n as E_n. Similarly to the static inference problem in Section 3.2, we require that each variable $Z_a^{(t)}$ is observed by exactly one node n. We assume that each node has access to the observation model for E_n, so that it can locally compute the observation likelihood $p(\overline{z}_a^{(t)} \mid \mathbf{x}_{\mathrm{Pa}[Z_a]}^{(t)})$ for all its measurements $\overline{z}_a^{(t)} : a \in E_n$.

In the **distributed filtering** problem, each node n is associated with a set of processes $Q_n \subseteq V$; these are the state processes about which node n wishes to reason about. The nodes need to collaborate, so that each node can obtain the posterior distribution over Q_n given all the measurements made in the network up to the current time step t. However, since exact filtering may not be tractable even in the centralized setting, we do not require that the nodes computes the exact posterior $p(\mathbf{x}_{Q_n}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$. Instead, we wish to compute a principled approximation to the true posterior, such as the B&K98 belief.

3.3 Approximate distributed filtering

In principle, the B&K98 algorithm could be applied to a distributed system, for example, by communicating the observations made in the network to a central location that performs all computations, and distributing the answer to every node in the network. While conceptually simple, this approach has substantial drawbacks, including the high communication bandwidth required to scale to a large number of observations, the introduction of a single point of failure to the system, and the fact that nodes do not have valid estimates when the network is partitioned. In this section, we present a distributed filtering algorithm where each node obtains an approximation to the posterior distribution over subset of the state variables. Our estimation step builds on the robust message passing algorithm of Paskin and Guestrin (2004b), while the prediction, roll-up, and projection steps are performed locally at each node.

3.3.1 Outline of the algorithm

The B&K98 algorithm represents the approximate prior distribution as a decomposable model that consists the tree and a collection of clique and separator marginals. In a distributed setting, it is undesirable to store the entire approximate prior or posterior distribution on a single node; each node has limited memory and

limited power to perform the computation and to coordinate with other nodes. Instead, each node should only store a portion of the approximate prior distribution.

Recall that by Property 2.1, a decomposable model can be represented implicitly by the clique marginals: given the clique marginals, the tree structure as well as the separator marginals can be recovered. This property was exploited in the robust message passing algorithm to represent the prior distribution of the static model in a distributed manner: each node only stored the marginals for a subset of the cliques, and the union of the clique marginals from all the nodes uniquely defined the prior distribution. In our approximate distributed filtering algorithm, each node n computes an approximate prior over the cliques $C_i : \mathcal{I}_n$ assigned to the node. Each clique is assigned to at least one node; thus, the clique marginals from all the nodes form a distributed representation of the approximate prior distribution $\tilde{p}(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$ at time step t. The approximate prior distribution changes from one time step to another, so the nodes need to recompute the approximate prior marginals $\tilde{p}(\mathbf{x}_{C_i}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$ at every time step. However, the external junction tree for the assumed density and the cliques assigned to each node remain the same throughout the execution of the algorithm.

Similarly to the B&K98 algorithm, the computation proceeds recursively in two phases. In the first, **estimation** phase, the network starts with a distributed representation of the approximate prior distribution at time step t. The nodes make the observations $\overline{\mathbf{z}}^{(t)}$ and coordinate to condition on the observations made in all of the network. Unlike the B&K98 algorithm, where the entire approximate posterior distribution was computed in a single location, each node n only obtains a marginal of the approximate posterior $\tilde{p}(\mathbf{x}_{Q'_n}^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ for some subset of the state variables $Q'_n \subseteq V$ such that $Q'_n \supseteq Q_n$. When the system is about to advance to the next time step, the nodes locally compute the clique marginals of the approximate prior distribution at the next time step, $p(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$. The clique marginals across all the nodes constitute a distributed representation of the approximate prior distribution at the next time step.

3.3.2 Estimation as robust message passing

Recall that, in the estimation phase, each node starts with the prior marginals $\tilde{p}(\mathbf{x}_{C_i}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$ and observation likelihoods $p(\overline{z}_a^{(t)})\mathbf{x}_{\mathrm{Pa}[Z_a]}^{(t)}$ and wishes to compute a marginal of the approximation posterior distribution that conditions on all the measurements in the network. Notice that, if we were to drop the conditioning on the past observations $\overline{\mathbf{z}}^{(0:t-1)}$ and the time indices, the problem would look exactly like the inference problem in Section 2.2.1. Indeed, the estimation phase is an instance of a **static inference problem**, because it only concerns with the distribution at a single time step. We can therefore directly apply the robust message passing algorithm from Section 2.3 to obtain a baseline implementation of the estimation phase in this setting.

We initialize the robust message passing algorithm by setting each prior marginal π_i for $i \in \mathcal{I}_n$ to a clique marginal of the approximate prior distribution, $\pi_i(\mathbf{x}_{C_i}^{(t)}) = \tilde{p}(\mathbf{x}_{C_i}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$. The algorithm takes

the observation likelihoods and pairs them up with the cliques that cover the parents of the observation, $\text{Pa}[Z_a] \subseteq C_i$.[2] The likelihood factor $\lambda_i(\mathbf{x}_{C_i}^{(t)})$ is equal to the product of observation likelihoods assigned to clique C_i. Together, the cliques C_i, the clique priors π_i, and the likelihoods λ_i define the robust factor ψ_n at node n.

The algorithm now proceeds, as discussed in Section 2.3.3. The nodes form a network junction tree, whose cliques cover the arguments of the local factor $\text{Scope}[\psi_n]$ and the query variables Q'_n. The nodes compute messages $\mu_{m\to n}$ using Algorithm 1. At any point, node n can compute its belief β_n (2.34), by combining the incoming messages $\mu_{m\to n}$ with its local factor ψ_n. At convergence, the belief is a prior/likelihood decomposable model for the approximate posterior distribution $\tilde{p}(\mathbf{x}_U^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ with $U \supseteq Q'_n$. Before convergence the belief represents a principled approximation, see Section 2.3.4.

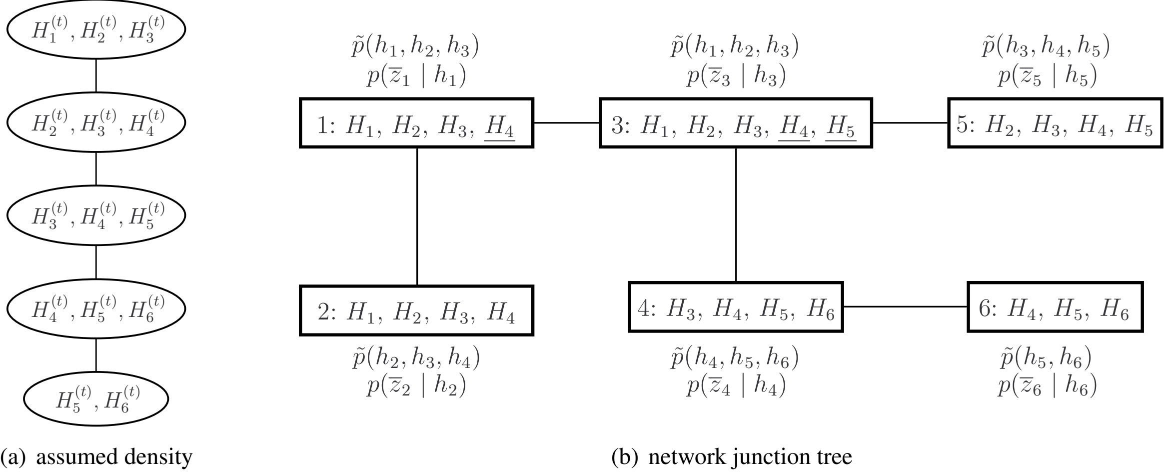
3.3.3 Prediction, roll-up, and projection

In order to advance to the next time step, each node must perform prediction, roll-up, and projection, obtaining the marginals $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$. Recall from Section 3.1.3 that, in order to compute a marginal $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$, the node needs the posterior distribution $\tilde{p}(\mathbf{x}_U^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$, where U covers the parents $\text{Pa}[\mathbf{X}_{C_i}]$ of all variables in the clique. This task can be accomplished by including these parents in the query variables at node n: $\text{Pa}[\mathbf{X}_{C_i}] \subseteq Q'_n$. The next time step marginal $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$ can be then computed by multiplying the posterior distribution with the transition model $p(x_a^{(t+1)} \mid \mathbf{x}_{\text{Pa}[X_a]}^{(t)})$ for each $a \in C_i$ and eliminating all variables but $\mathbf{X}_{C_i}^{(t+1)}$.

For example, consider a DBN for the temperature estimation example, where each $H_a^{(t+1)}$ depends on the temperature at the same location $H_a^{(t)}$ and one nearby location $H_{a-1}^{(t)}$, as illustrated in Figure 3.1(b). Suppose that we use the assumed density structure, shown in Figure 3.7(a); this external junction tree coincides with the running example in Section 2.3. Figure 3.7(b) shows the assignment of approximate prior marginals to nodes and the likelihoods for measurements made at each node (the time step t indices as well as the conditioning on the past observations were omitted for brevity). For example, node 4 maintains the marginal over the clique $\{H_4^{(t)}, H_5^{(t)}, H_6^{(t)}\}$. To advance to the next time step, node 4 needs to compute prior marginal at time step $t+1$ over $\{H_4^{(t+1)}, H_5^{(t+1)}, H_6^{(t+1)}\}$. To achieve this task, the node includes the parents of these three variables in its query: $\{H_3^{(t)}, H_4^{(t)}, H_5^{(t)}, H_6^{(t)}\} \subseteq Q'_n$. The robust message passing algorithm then instructs the distributed inference architecture to include the query variables in the clique of the network junction tree.

The robust message passing algorithm permits early stopping, which is useful, for example, when the estimation step cannot be run to convergence within the allotted time. In this case, the variables $\text{Scope}[\beta_n]$ covered by the partial belief β_n at node n may not include the entire parent set $\text{Pa}[\mathbf{X}_{C_i}]$. For example,

[2] Similarly to the robust message passing algorithm, we assume that the parents of each observation Z_a, $a \in E_n$, are covered by some clique assigned to the node.



(a) assumed density

(b) network junction tree

Figure 3.7: (a) Assumed density structure for monitoring temperature at six sensor locations, analogous to the one in Figure 2.7(a). (b) The prior clique marginals, likelihoods, and the network junction tree for a temperature estimation example with assumed density in (a). The time step t indices as well as the conditioning on the past observations $\overline{\mathbf{z}}^{(0:t-1)}$ were omitted for brevity. The clique of the network junction tree at each node includes the clique C_i assigned to the node, the parents of the clique at time step $t+1$, as well as any variables required to satisfy the running intersection property (underlined).

if node 4 receives the message from node 6 but not from node 3, the belief β_4 will consist of marginals and likelihoods over the cliques $\{H_4^{(t)}, H_5^{(t)}, H_6^{(t)}\}$ and $\{H_5^{(t)}, H_6^{(t)}\}$. Node 4 does not yet have any information about $H_3^{(t)}$, which is required to compute the approximate prior over $\{H_4^{(t+1)}, H_5^{(t+1)}, H_6^{(t+1)}\}$ at time step $t+1$. In this case, multiplying in the standard transition model is equivalent to assuming a uniform prior for the missing variables, which can lead to very poor solutions in practice.

In order to address this problem, we take a look at how the transition models are obtained in the first place. When the transition model is learned from data, $p(x_a^{(t+1)} \mid \mathbf{x}_{\text{Pa}[X_a]}^{(t)})$ is usually computed from the empirical distribution $\hat{p}(x_a^{(t+1)}, \mathbf{x}_{\text{Pa}[X_a]}^{(t)})$ that captures sufficient statistics, such as the occurrences of joint assignments to the variable $X_a^{(t+1)}$ and its parents. When p is discrete or Gaussian, the maximum-likelihood estimate of the transition model is obtained by dividing out the joint empirical distribution by the marginal over the parent variables:

$$p_{MLE}(x_a^{(t+1)} \mid \mathbf{x}_{\text{Pa}[X_a]}^{(t)}) = \hat{p}(x_a^{(t+1)}, \mathbf{x}_{\text{Pa}[X_a]}^{(t)}) \,/\, \hat{p}(\mathbf{x}_{\text{Pa}[X_a]}^{(t)}). \tag{3.6}$$

For example, $p_{MLE}(h'_4 \mid h_3, h_4) = \hat{p}(h'_4, h_3, h_4) \,/\, \hat{p}(h_3, h_4)$. Building on these empirical distributions, we can obtain an improved solution for the prediction and roll-up phase when a node does not have a distribution over the entire parent set $\text{Pa}[\mathbf{X}_{C_i}]$. We compute a valid approximate transition model $\tilde{p}(x_a^{(t+1)} \mid \mathbf{x}_W^{(t)})$, where $W = \text{Scope}[\beta_n] \cap \text{Pa}[X_a]$ is a subset of the parents covered by the belief. The approximate transition model is obtained online by marginalizing the empirical distribution $\hat{p}$ down to $\hat{p}(x_a^{(t+1)}, \mathbf{x}_W^{(t)})$ and computing the maximum-likelihood estimate for the parent variables W using an equa-

tion analogous to (3.6). This procedure is equivalent to introducing an additional independence assumption to the model: at time step $t+1$, $X_a^{(t+1)}$ is independent of $\mathbf{X}^{(t)}_{\text{Pa}[X_a]-W}$, given $\mathbf{X}_W^{(t)}$. In the example above, since node 4 has does not have any information about $H_3^{(t)}$ in its belief, it would compute an approximate transition model $\tilde{p}(h'_4 \mid h_4) = \hat{p}(h'_4, h_4) \;/\; \hat{p}(h_4)$. This approximate transition model introduces the conditional independence assumption $H_4^{(t+1)} \perp H_3^{(t)} \mid H_4^{(t)}$. If the belief β_n covers all the parents of $X_a^{(t+1)}$, the computed transition model is exact, and no approximations are made in the model.

If the transition model is not learned from data, it may be necessary to manually specify the approximate transition models for all subsets of the parents $\mathbf{X}^{(t)}_{\text{Pa}[X_a]}$. In some cases, these approximate transition models can be specified implicitly. For example, if we assume a simple diffusion transition model for the temperature monitoring network, where the temperature $H_i^{(t+1)}$ is a weighted average of time-t temperature at the same location $H_i^{(t)}$ and the nearby locations, a good approximation is to simply omit the locations that are not covered by the belief β_n from the averaging. Alternatively, in some applications, it may be possible to guarantee that the belief β_n always covers the parent set Pa$[\mathbf{X}_{C_i}]$. For example, as indicated at the end of Section 3.1.3, in simultaneous localization and tracking (SLAT), the parent set of each clique $\{\mathbf{L}_i, M^{(t+1)}\}$ is simply the clique itself at the previous time step, $\{\mathbf{L}_i, M^{(t)}\}$. Since this clique is always present in the belief, the belief will always cover the parents of $\{\mathbf{L}_i, M^{(t+1)}\}$, and we can always use the exact transition model $p(m^{(t+1)} \mid m^{(t)})$.

3.3.4 Summary of the algorithm

Our distributed approximate filtering algorithm can be summarized as follows:

- Using the distributed inference architecture (Paskin et al., 2005), construct a network junction tree such that the clique C_n at each node n includes the cliques C_i assigned to the node, the parents of each clique C_i, and the query variables Q_n:

$$C_n \supseteq \left(\bigcup_{i\in\mathcal{I}_n} C_i\right) \cup \left(\bigcup_{i\in\mathcal{I}_n} \text{Pa}[\mathbf{X}_{C_i}]\right) \cup Q_n.$$

- For $t = 1, 2, \ldots$, at each node n,

  1. run the robust message passing algorithm (Paskin and Guestrin, 2004b) until the end of time step t, obtaining a (possibly approximate) node belief β_n;
  2. for each $a \in C_i$, $i \in \mathcal{I}_n$, compute a (possible approximate) transition model $\tilde{p}(x_a^{(t+1)} \mid \mathbf{x}_{W_a}^{(t)})$, where $W_a = \text{Scope}[\beta_n] \cap \text{Pa}[X_a]$;
  3. for each clique, C_i, $i \in \mathcal{I}_n$, compute the clique marginal $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$ from β_n^* and from each $\tilde{p}(x_a^{(t+1)} \mid \mathbf{x}_{W_a}^{(t)})$, locally, using variable elimination.

Note that the cliques C_i and the parent sets Pa$[\mathbf{X}_{C_i}]$, $i \in \mathcal{I}_n$ at each node n do not change from one time step to another. Therefore, the network junction tree can be reused throughout the execution of the algorithm. This property speeds up the convergence of the robust message passing algorithm at each step, because the algorithm does not initially introduce additional approximations due to an invalid junction tree. Naturally, the network junction tree continues to be adapted to changing network conditions and to nodes entering or leaving the network.

The global convergence property of the robust message passing algorithm (Theorem 2.5) can be used to prove that, under suitable conditions, our approximate distributed filtering algorithm converges to the centralized B&K98 algorithm. Specifically, suppose that the robust message passing algorithm starts with a distributed representation of the approximate B&K98 prior $\tilde{p}(\mathbf{x}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$. If the network connectivity graph forms a single connected component, and if there is sufficient communication, the robust message passing algorithm converges to the marginal of the approximate posterior distribution $\tilde{p}(\mathbf{x}_U^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$, where U includes the parent variables of each clique $\mathbf{X}_{C_i}^{(t+1)}$ for $i \in \mathcal{I}_n$. Since U includes all the parent variables of each clique maintained by node n, the computed transition model $p(x_a^{(t+1)} \mid \mathbf{x}_{\mathrm{Pa}[X_a]}^{(t)})$ is exact for each $a \in C_i$. Multiplying in the exact transition model to the approximate posterior distribution and marginalizing out all the variables but $\mathbf{X}_{C_i}^{(t+1)}$ yields an approximate prior distribution at time t, $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$. This marginal is equal to the distribution obtained by running the B&K98 algorithm with the assumed density given by $(T, \mathbf{C})$. By induction on the time steps, we obtain the following theorem:

Theorem 3.1. *For a set of nodes running our approximate filtering algorithm, if at each time step there is sufficient communication for the robust message passing algorithm to convergence, and the network is not partitioned, then for each node n, for each clique $i \in \mathcal{I}_n$, the distribution $\tilde{p}(\mathbf{x}_{C_i}^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$ obtained by node n is equal to the distribution obtained by running the* B&K98 *algorithm on the same sequence of observations, with assumed density given by $(T, \mathbf{C})$.*

3.4 Robust distributed filtering

As described so far, our distributed filtering algorithm is conceptually simple: at each time step, we run the robust message passing algorithm for a fixed duration of time; then we perform the prediction step locally and move on to the next time step. From the global convergence of robust message passing, we were able to prove that given sufficient communication, the distributed filter converges to the centralized B&K98 solution. Yet, in realistic deployments, the network may be sufficiently large that the estimation step is never run to convergence. Furthermore, when interference causes a network partition, the nodes on the two sides of the partition may not share information for many time steps. In this case, the nodes at different sides of the partition will incorporate different sets of observations in their prior clique marginals; the nodes may also make different approximations in their beliefs. Consequently, the distributions computed by the nodes on the different sides of the partition may not agree on the shared variables as shown in

Figure 3.8, and the prior clique marginals in the network may no longer represent a valid distribution. In this section, we present two algorithms that address this problem.

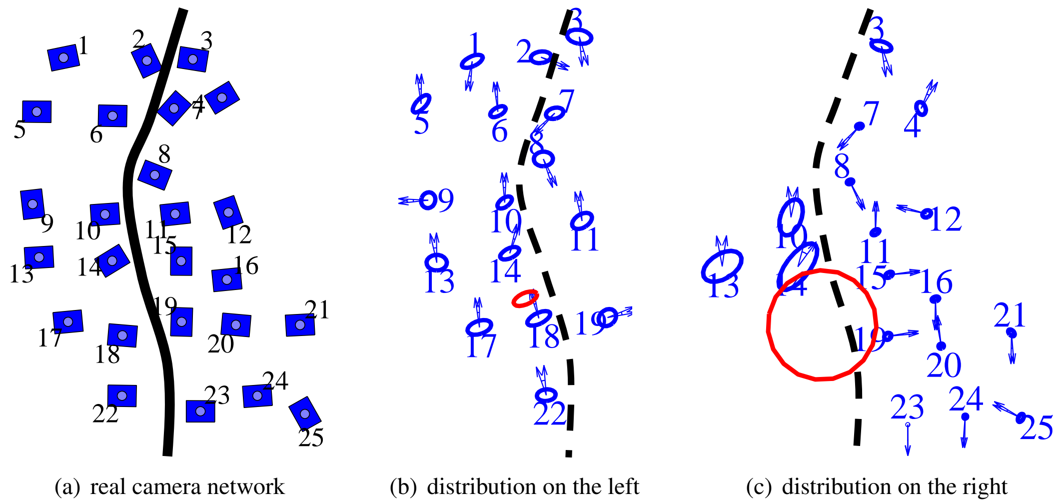
3.4.1 The alignment problem

The robust message passing algorithm (Paskin and Guestrin, 2004b) has an important property. If the network is partitioned into two or more disconnected components, the algorithm propagates the observations within each component. The algorithm makes principled approximations: the belief at each node is a sequence of projection and marginalization operations. When the communication is restored, the algorithm adapts the network junction tree and eventually converges to the correct posterior distribution.

Unfortunately, the approximate distributed filter, described in Section 3.3 does not have such a property. If the network partition spans several time steps, any approximations made by the robust message passing algorithm are incorporated into the clique priors, and our ability to undo these approximations is irreversibly lost. Consider the example, illustrated in Figure 3.9, in which a network of four cameras localizes itself by observing a moving object. Each camera i carries a clique marginal over the location of the object $M^{(t)}$, its own camera pose variable L_i, and the pose of one of its neighboring cameras: $\pi_1(l_1,l_2,m^{(t)})$, $\pi_2(l_2,l_3,m^{(t)})$, and $\pi_3(l_3,l_4,m^{(t)})$. Suppose that a network partition occurs: starting from some point in time, no messages are propagated between the two sides of the partition. The beliefs at nodes on the different sides of the partition are conditioned on different sets of observations. As the algorithm advances in time, these approximations are incorporated into the clique priors, and the prior marginals carried by the nodes no longer form a consistent distribution, in the sense that π_1,π_2,π_3 may not agree on their marginals, e.g., $\pi_2(l_3, m^{(t)}) \neq \pi_3(l_3, m^{(t)})$.

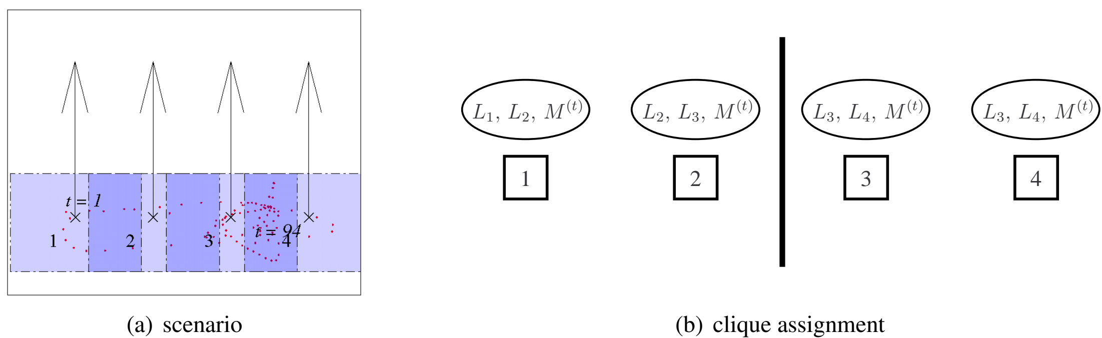
Inconsistent prior marginals can introduce inaccuracy in the estimates. For example, in Figure 3.8, the nodes on the right side of partition have an uncertain estimate of the object location and of a few cameras from the other side. This uncertainty is reflected in one or more clique marginals $\pi_i(\mathbf{l}_i, m^{(t)})$ with a large variance and weak correlations among $\mathbf{L}_i$. If a node on the left maintains a marginal over the same clique, once the communication is restored, the node could accidentally use the uncertain marginal π_i instead of its certain one, making its own estimate less informative. Furthermore, uncertain marginals can slow down the information flow captured by the likelihood updates: maintaining strong correlations among the cameras is required for quick convergence when the object closes the loop (Paskin, 2004, Section 4.2), that is, revisits the same area after moving about the environment for a while. Inconsistencies have also an unpleasant theoretical consequence: they prevent us from interpreting the state maintained by the network as a single valid distribution. Therefore, it is desirable to define some procedure through which the network can recover an informative consistent prior distribution $\tilde{p}(\mathbf{x}^{(t)} \mid \bar{\mathbf{z}}^{(0:t-1)})$ from the marginals $\{\pi_i\}$; we will say that the nodes **align** their inconsistent marginals.

In general, it is difficult to characterize the approximations made by our approximate distributed filter in



(a) real camera network (b) distribution on the left (c) distribution on the right

Figure 3.8: (a) A real camera network with 25 nodes. The blue rectangles indicate the locations and orientations of the cameras. The thick black line indicates a network partition. (b) The distribution computed by the nodes on the left; (c) the distribution computed by the nodes on the right. The blue ellipses with arrows indicate the 95% confidence regions for the estimated camera locations and orientations. The red ellipses without arrows indicates the 95% confidence regions for the estimated object position. Note that the two distributions (b) and (c) do not agree: the distributions are less certain about the locations of the cameras on the other side of the partition, because they have not been conditioned on observations from nodes on that side. Also, the distribution on the right is very uncertain about the location of the object, because none of the cameras on the right side of the partition have observed the object recently. These inconsistencies can lead to inaccurate results and prevent us from interpreting the state maintained by the network as a single valid distribution.



(a) scenario (b) clique assignment

Figure 3.9: (a) A simulated camera network with four nodes. At time step 14, the communication network becomes partitioned into two components with nodes $\{1, 2\}$ and $\{3, 4\}$. (b) Clique assigned to each node. There is an overlap between the cliques on the two sides of the partition; both sides maintain a distribution over the pose of camera L_3. The thick vertical line indicates the network partition.

the presence of partitions, because the robust message passing algorithm provides no guarantees when the likelihoods are computed with respect to inconsistent priors. Therefore, it is difficult to guarantee that an aligned distribution will be close to the B&K98 prior $\tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)})$. Nevertheless, we can select a distribution q from some family of distributions $\mathcal{F}$ that minimizes a suitably chosen measure over $q \in \mathcal{F}$ and demonstrate empirically that this distribution is in some sense close to the B&K98 prior (as measured, for example, by the root mean square error of the estimates). Naturally, if the marginals $\{\pi_i\}$ are **consistent** to begin with, that is if $\pi_i(\mathbf{x}_{S_{i,j}}) = \pi_j(\mathbf{x}_{S_{i,j}})$ for all edges $\{i, j\} \in E_T$, then an alignment procedure should not alter these marginals. We will say that alignment **preserves** a set of consistent marginals if it obtains an approximate distribution $\tilde{p}(\mathbf{x}^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t-1)})$ such that

$$\tilde{p}(\mathbf{x}^{(t)}_{C_i} \,|\, \overline{\mathbf{z}}^{(0:t-1)}) = \pi_i(\mathbf{x}^{(t)}_{C_i}).$$

This property ensures that, in the absence of partitions and if the estimation phase is run to completion, the distributed filter will still obtain the same solution as the centralized B&K98 algorithm (Theorem 3.1).

In the rest of this section, we outline two alignment procedures with different families $\mathcal{F}$. Both procedures preserve a set of consistent marginals. For simplicity of notation, we omit time indices t and conditioning on the past evidence $\overline{\mathbf{z}}^{(0:t-1)}$ throughout this section.

3.4.2 Optimized conditional alignment

One way to define a consistent distribution $\tilde{p}$ is to start from a vertex of the external junction tree, and allow each clique marginal to decide the conditional density of $\mathbf{X}_{C_i}$ given its parent. For example, for the junction tree in Figure 3.10(a), we can define

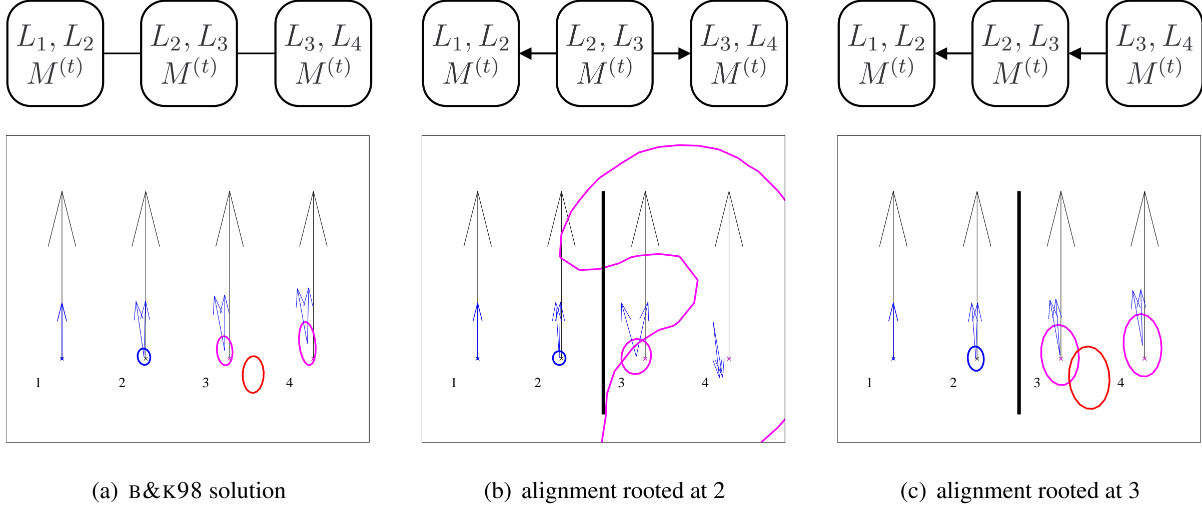
$$\tilde{p}_1(l_{1:4}, m) = \pi_1(l_1, l_2, m) \times \pi_2(l_3 \,|\, l_2, m) \times \pi_3(l_4 \,|\, l_3, m). \tag{3.7}$$

This density $\tilde{p}_1$ forms a coherent distribution over $\mathbf{L}, M$, and we say that $\tilde{p}_1$ is **rooted** at vertex 1. Thus, the prior π_1 fully defines the marginal distribution over L_1, L_2, M, the prior π_2 defines the conditional density of L_3 given L_2, M, and so on. If the clique $\{L_3, L_4, M\}$ were the root, then vertex 1 would only contribute $\pi_1(l_1 \,|\, l_2, m)$, and we would obtain a different approximate distribution.

In general, given a collection of marginals $\pi_i(\mathbf{x}_{C_i})$ over the cliques of a junction tree T, and a root vertex $r \in N_T$, the distribution obtained by **conditional alignment** from r can be written as

$$\tilde{p}_r(\mathbf{x}) = \pi_r(\mathbf{x}_{C_r}) \times \prod_{i \in N_T \setminus r} \pi_i(\mathbf{x}_{C_i - S_{\text{up}(i),i}} \,|\, \mathbf{x}_{S_{\text{up}(i),i}}) \tag{3.8}$$

where $\text{up}(i)$ denotes the upstream neighbor of i on the (unique) path between r and i. The root π_r defines the marginal over the root clique, $\tilde{p}_r(\mathbf{x}_{C_r}) = \pi_r(\mathbf{x}_{C_r})$, while the the remaining π_i define the conditional



(a) B&K98 solution (b) alignment rooted at 2 (c) alignment rooted at 3

Figure 3.10: Solutions, computed by different algorithms and the corresponding junction trees. (a) The target solution, computed by the B&K98 algorithm in the absence of partitions. The ellipses with arrows indicate the 95% confidence regions of the estimated camera locations and orientations. (b) Solution obtained when aligning from vertex 2. Note that the estimate of camera 4 is highly uncertain, as indicated by the large confidence region (the confidence region is non-elliptical, because we used relative over-parameterization to represent the camera pose, as discussed in Section 4.3.2). (c) Solution obtained when aligning from vertex 3.

densities $\tilde{p}_r(\mathbf{x}_{C_i - S_{\mathrm{up}(i),i}} \,|\, \mathbf{x}_{S_{\mathrm{up}(i),i}}) = \pi_i(\mathbf{x}_{C_i - S_{\mathrm{up}(i),i}} \,|\, \mathbf{x}_{S_{\mathrm{up}(i),i}})$. Each factor in (3.8) corresponds to one clique in the external junction tree; the arguments of each factor are exactly $\mathbf{X}_{C_i}$. We can easily reparameterize the distribution $\tilde{p}_r$ as a decomposable model, by traversing the tree from the root and computing the clique marginals $\tilde{p}_r(\mathbf{x}_{C_i})$. For example, in the conditional alignment in (3.7), we can marginalize π_1 down to the separator $\{L_2, M\}$ to obtain $\tilde{p}_1(l_2, m)$. Multiplying in the conditional $\tilde{p}_1(l_3 \,|\, l_2, m) = \pi_2(l_3 \,|\, l_2, m)$ yields the clique marginal $\tilde{p}_1(l_2, l_3, m)$.

As discussed in the previous section, an alignment procedure should preserve a set of consistent marginals. Conditional alignment satisfies this property:

Theorem 3.2. *Let $\{\pi_i\}$ be a set of consistent marginals. Then the conditional alignment procedure recovers the original marginals:*

$$\tilde{p}_r(\mathbf{x}_{C_i}) = \pi_i(\mathbf{x}_{C_i})$$

for all $i \in N_T$, independently of the choice of the root r.

Theorem 3.2 is proved in Appendix 3.A, by induction on the clique, starting from the root.

The choice of the root r often crucially determines how well the aligned distribution $\tilde{p}_r$ approximates the true prior. Suppose that in the example in Figure 3.9, the nodes on the left side of the partition do not observe the object while the communication is interrupted, and the prior marginals π_1, π_2 are uncertain about M. If we were to align the distribution from π_2, multiplying $\pi_3(l_4 \,|\, l_3, m)$ into the marginal

$\pi_2(l_2,l_3,m)$ would result in a distribution that is uncertain in both M and L_4 (Figure 3.10(b)). The clique $\{L_3, L_4, M^{(t)}\}$ is a much better choice of the root, because it reduces the uncertainty of the estimates, as shown in Figure 3.10(c).

The standard metric of uncertainty of a distribution is the entropy:

Definition 3.1. *The* ***entropy*** *of a distribution $p(\mathbf{x})$, denoted as $H_p(\mathbf{X})$, is the expectation*

$$H_p(\mathbf{X}) = \mathbb{E}_p[-\log p(\mathbf{X})] = -\sum_{\mathbf{x}} p(\mathbf{x}) \log p(\mathbf{x}).$$

Given two subsets of variables $U, W \subseteq V$ with $U \cap W = \emptyset$ and $U \cup W = V$, the ***conditional entropy*** *of $\mathbf{X}_U$ given $\mathbf{X}_W$, denoted as $H_p(\mathbf{X}_U \mid \mathbf{X}_W)$, is the expectation*

$$H_p(\mathbf{X}_U \mid \mathbf{X}_W) = \mathbb{E}_p[-\log p(\mathbf{X}_U \mid \mathbf{X}_W)] = -\sum_{\mathbf{x}_V} p(\mathbf{x}_V) \log p(\mathbf{x}_U \mid \mathbf{x}_W).$$

For discrete variables $\mathbf{X}$, the logarithm is typically taken with base 2, and the entropy represents the number of bits required to transmit one instance of $\mathbf{X}$ with the optimal encoding, assuming that the instances are generated from p. For p Gaussian, the logarithm is typically taken with the natural base e, and the entropy has a closed formula.

The distribution in Figure 3.10(c) is more certain than the distribution in Figure 3.10(b); the former distribution has a lower entropy and is thus preferable. In general, maximizing the certainty of $\tilde{p}_r$ over different choices of r amounts to selecting the root that minimizes the entropy of $\tilde{p}_r$. Selecting a root r that leads to an approximate distribution of the smallest entropy is a reasonable choice in Gaussian distributions: in Gaussians, conditioning on observations can only decrease the entropy of the distribution, so we would not expect $\tilde{p}_r$ to be overconfident.

When a distribution is written as a product of conditional probability distributions, the entropy decomposes into a sum of corresponding conditional entropies. We can use this fact to decompose the entropy for the aligned distribution $\tilde{p}_r(\mathbf{x})$ into a sum of conditional entropies over the cliques:

$$H_{\tilde{p}_r}(\mathbf{X}) = H_{\tilde{p}_r}(\mathbf{X}_{C_r}) + \sum_{i \in N_T \setminus r} H_{\tilde{p}_r}(\mathbf{X}_{C_i - S_{\mathrm{up}(i),i}} \mid \mathbf{X}_{S_{\mathrm{up}(i),i}}) \tag{3.9}$$

For example, the entropy of $\tilde{p}_2$ in the earlier example can be written as

$$H_{\tilde{p}_2}(L_{1:4}, M) = H_{\pi_2}(L_2, L_3, M) + H_{\pi_3}(L_4 \mid L_3, M) + H_{\pi_1}(L_1 \mid L_2, M), \tag{3.10}$$

where we use the fact that, for Gaussians, the conditional entropy of L_4 given L_3,M only depends on the conditional distribution $\tilde{p}_2(l_4 \mid l_3, m) = \pi_3(l_4 \mid l_3, m)$.

A naïve algorithm for obtaining the best root would exploit the decomposition (3.9) to compute the entropy

of each $\tilde{p}_r$, and pick a root that leads to the lowest entropy; the running time of this algorithm is $O(|N_T|^2)$. We propose a dynamic programming approach that significantly reduces the running time. Comparing (3.10) with the entropy of the distribution rooted at a neighboring vertex 3, we see that they share a common term $H_{\pi_1}(L_1 \mid L_2, M)$, and

$$\begin{aligned} & H_{\tilde{p}_3}(L_{1:4}, M) - H_{\tilde{p}_2}(L_{1:4}, M) \\ & \quad = H_{\pi_3}(L_3, L_4, M) - H_{\pi_3}(L_4 \mid L_3, M) - H_{\pi_2}(L_2, L_3, M) + H_{\pi_2}(L_2 \mid L_3, M) \\ & \quad = H_{\pi_3}(L_3, M) - H_{\pi_2}(L_3, M) \triangleq \triangle_{2,3}. \end{aligned}$$

If $\triangle_{2,3}$ is positive, vertex 2 is a better root than 3; if $\triangle_{2,3}$ is negative, we have the reverse situation. Thus, with Gaussian distributions, when comparing neighboring vertices i and j as root candidates, the difference in entropy of the resulting distribution is simply the difference in entropy their local distributions assign to their separator:

$$\triangle_{i,j} \triangleq H_{\tilde{p}_j}(\mathbf{X}) - H_{\tilde{p}_i}(\mathbf{X}) = H_{\pi_j}(\mathbf{X}_{S_{i,j}}) - H_{\pi_i}(\mathbf{X}_{S_{i,j}}). \tag{3.11}$$

The property (3.11) could be used to speed up the naïve algorithm, by traversing the junction tree along the edges and incrementally computing the entropy with different roots, remembering the smallest one. However, we are ultimately interested in a distributed algorithm, and this algorithm does not distribute well: it requires coordination to select the initial vertex of the junction tree to start the traversal, and it is not anytime, in the sense that the computation cannot be stopped to acquire the best root in some neighborhood. Instead, we propose the following anytime dynamic programming algorithm that determines the root r with minimal $H_{\tilde{p}_r}(\mathbf{X})$ in $O(|N_T|)$ time for trees T with bounded degree; we call this algorithm the **optimized conditional alignment** (OCA).

The basic unit of the OCA algorithm is the **optimization message** $d_{i \to j}$, which is sent along each directed edge of the tree T. The message $d_{i \to j}$ represents the difference in entropy with root vertex j, compared to the best root on i's side of the tree. In particular, if $d_{i \to j} < 0$ then j is a better root than any node on i's side of the tree. Formally, the message is defined as:

$$d_{i \to j} = \begin{cases} \triangle_{i,j} & \text{if } d_{k \to i} < 0, \ \forall k \in N_T(i) \setminus j, \\ \triangle_{i,j} + \max_{k \in N_T(i) \setminus j} d_{k \to i} & \text{otherwise.} \end{cases} \tag{3.12}$$

The two cases in (3.12) are illustrated in Figure 3.11. In the first case, the algorithm detects that vertex i is the best root among the vertices on i's side of the tree, so the message is message is simply $\triangle_{i,j}$. In the second case, the vertex k chosen in the maximization points to the best root r on node i's side of the tree, and the message is the sum of the entropy differences $\triangle$ along the path from r to j. By the additivity of the entropy differences, we can show the following lemma:

Lemma 3.1. *Let $T_{i,j}$ denote the subtree of T rooted at vertex i, away from vertex j. Then $d_{i \to j}$ is the*

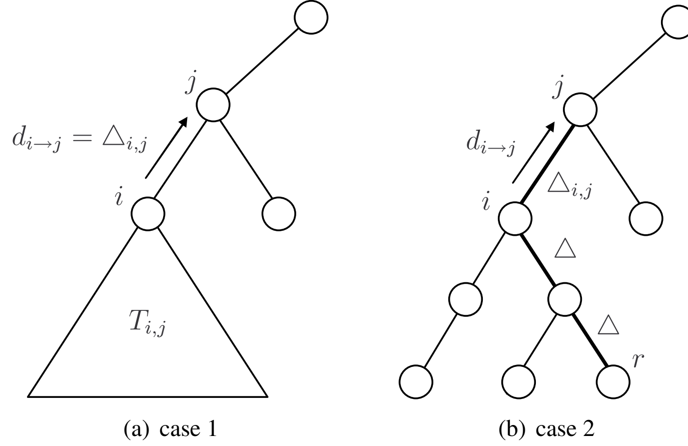


Figure 3.11: The two cases in computing the optimization message $d_{i\to j}$. $T_{i,j}$ denotes the subtree of T rooted at vertex i, away from vertex j. (a) In the first case, vertex i is the best root among $T_{i,j}$, and the message is simply the entropy difference $\triangle_{i,j}$. (b) In the second case, the message is the sum of the entropy differences $\triangle_{,}$ along the path from the best root r in $T_{i,j}$ to j (shown in bold).

difference between the entropy of $\tilde{\mathrm{p}}_j$ and the entropy attained by the best root among the vertices in $T_{i,j}$:

$$d_{i\to j} = H_{\tilde{\mathrm{p}}_j}(\mathbf{X}) - \min_{k\in T_{i,j}} H_{\tilde{\mathrm{p}}_k}(\mathbf{X}).$$

Lemma 3.1 is proved in Appendix 3.A.

Once all the messages have been computed, it is easy to decide for each vertex i whether it is the optimal root and if not, what is its upstream neighbor up(i). Suppose first that the optimal root is unique. If

$$\max_{k\in N_T(i)} d_{k\to i} < 0, \tag{3.13}$$

then the entropy with root i is smaller than the entropy with the second best contender in some subtree, rooted away from i (that is, anywhere else in the tree). Thus, i is the optimal root. Otherwise,

$$\text{up}(i) = \operatorname*{argmax}_{k\in N_T(i)} d_{k\to i} \tag{3.14}$$

points to the subtree containing the best root.

To resolve ties between root vertices with the same entropy, we augment each optimization message $d_{i\to j}$ with the ID of the best root in $T_{i,j}$. Thus, each optimization message is a pair (h, r), where h is the entropy difference as stated in Lemma 3.1 and r is the id of the best root in $T_{i,j}$ that attains this difference. We then use lexicographical ordering $\prec$ in all the comparisons (such as when computing $\max d_{k\to i}$), and the condition $d_{k\to i} < 0$ in (3.12) becomes $d_{k\to i} \prec (0, i)$ and similarly for the optimal root selection. With these changes, the algorithm determines a root r with minimal $H_{\tilde{\mathrm{p}}_r}(\mathbf{X})$ and of the roots that attain the

same entropy, the algorithm selects the one with the maximal id:

Theorem 3.3. *The* OCA *algorithm selects a single root r with minimal $H_{\tilde{p}_r}(\mathbf{X})$ and determines the upstream neighbors up(i) for each vertex i.*

The proof follows naturally from Lemma 3.1 and the uniqueness of the comparison operations.

So far, we have discussed a version of the OCA algorithm, where each optimization message $d_{i \to j}$ from vertex i to vertex j is computed only once all the messages to i (except that from j) have been computed. We will call this version **synchronous**, because the messages are synchronized, in order to satisfy the message dependencies. The synchronous OCA algorithm is appropriate in centralized settings, where the algorithm is typically executed until completion. In distributed settings, however, it is useful to consider the partial results of the algorithm obtained when the algorithm is stopped early, for example, due to communication delays. This challenge motivates us to consider an **asynchronous** version of the OCA algorithm, where the optimization messages are computed repeatedly and in an arbitrary order. In computing the message in (3.12), the maximum is now taken only over the incoming messages that have already been computed. At any point, Equations 3.13 and 3.14 specify a partial result of the algorithm: the condition (3.13) determines if a vertex is a root, and if a vertex is not a root, up(i) computed by (3.14) points to the best root in some neighborhood of i. At convergence, this partial result coincides with the exact result obtained by the synchronous OCA algorithm. Before convergence, the partial result has a meaningful interpretation, as we will now show.

Recall from Section 2.2.3 that the partial results obtained by the asynchronous sum–product algorithm have a particularly simple interpretation: the partial result at each node is the exact marginal for a subset of factors in a subtree of the junction tree. The subtree is the set of nodes, whose factors have contributed to the belief at the node, as illustrated in Figure 2.16. Similarly, the partial result obtained by the asynchronous OCA algorithm at each vertex i is the exact result for a subtree of the external junction tree containing i. Thus, vertex i is determined to be a root if and only if it is the best root in a subtree that includes all the vertices whose messages have contributed to the result at i. Otherwise, up(i) points to the best root in that subtree. Note that before convergence, multiple vertices can be selected as roots and each up(i) points towards one of them. Importantly, as we show in Lemma 3.3 in Appendix 3.A, the root pointers up(i) are non-conflicting: it is never the case that two neighboring vertices i and j point towards each other. It is then easy to show that the root pointers partition the external junction tree into a collection of subtrees:

Theorem 3.4. *At any point, the subtrees traced by the pointers up(i) of the* OCA *algorithm at each vertex i form a partition the external junction tree, where each subtree is rooted at a vertex r with minimal $H_{\tilde{p}_r}(\mathbf{X})$ among the vertices in the subtree.*

The theorem is proved in Appendix 3.A.

While the partial results of the OCA algorithm may not be sufficient to fully align the inconsistent marginals $\{\pi_i\}$, they can be used to align them *partially*, by applying conditional alignment separately to each sub-

tree in Theorem 3.4. For instance, in the earlier example in Figure 3.10(a), if cliques $\{L_1, L_2, M\}$ and $\{L_3, L_4, M\}$ are determined to be the roots, with the clique $\{L_2, L_3, M\}$ pointing to $\{L_3, L_4, M\}$, then the OCA algorithm computes two marginal distributions:

$$\begin{aligned}\tilde{p}_1(l_1, l_2, m) &= \pi_1(l_1, l_2, m) \\ \tilde{p}_3(l_2, l_3, l_4, m) &= \pi_3(l_3, l_4, m) \times \pi_2(l_2 \mid l_3, m).\end{aligned}$$

The two marginal distributions $\tilde{p}_1$ and $\tilde{p}_3$ may not agree on the shared variables L_2 and M, but they are both valid marginals of the globally aligned distributions $\tilde{p}_1(\mathbf{l}, m)$ and $\tilde{p}_3(\mathbf{l}, m)$.

3.4.3 Distributed optimized conditional alignment

Since the robust message passing algorithm was designed to work with consistent prior marginals, some of its operations may not be well-defined when the marginals are inconsistent. In particular, if the prior marginals of the belief are inconsistent, the belief cannot be interpreted as a P/L decomposable model using (2.31). This is true, for instance, in the camera example discussed in the previous section. Nevertheless, Paskin and Guestrin (2004b) describe a **flattening** operation that permits one to interpret the belief as a valid distribution. In the flattening operation, we select an arbitrary root r and compute each separator marginal from the clique that is farther away from the root:

$$\tilde{p}_r(\mathbf{x} \mid \overline{\mathbf{z}}) \propto \frac{\prod_{i \in N_T} \pi_i(\mathbf{x}_{C_i})}{\prod_{i \in N_T \setminus r} \pi_i(\mathbf{x}_{S_{\mathrm{up}(i),i}})} \times \prod_{i \in N_T} p(\overline{\mathbf{z}}_{E_i} \mid \mathbf{x}_{C_i}).$$

By pairing up each clique $i \in N_T \setminus r$ in the numerator with the separator marginal in the denominator, we see that the clique contributes $\pi_i(\mathbf{x}_{C_i - S_{\mathrm{up}(i),i}} \mid \mathbf{x}_{S_{\mathrm{up}(i),i}})$. Thus, with this operation, the robust message passing algorithm can be viewed as performing conditional alignment from r. However, the alignment is applied to the local belief at each node, rather than the global distribution, and the nodes may not agree on the choice of the root r. Thus, the network is not guaranteed to reach a globally consistent, aligned distribution. We could fix a root clique arbitrarily, but this clique would not be guaranteed to lead to a distribution of minimal entropy. Instead, in this section, we show that robust message passing can be extended to incorporate the optimized conditional alignment (OCA) algorithm from the previous section.

By Theorem 2.2, at convergence, the priors at each node form a subtree of an external junction tree for the assumed density. If we were to apply OCA to this subtree, the node would have an aligned distribution, but nodes may not be consistent with each other. Intuitively, this happens because the optimization messages $d_{i \to j}$ were not propagated between different nodes. Our distributed OCA algorithm piggy-backs on the pruning operation of the robust message passing algorithm, computing an optimization message $d_{i \to j}$ whenever clique i is pruned from clique j. The incoming optimization messages $d_{i \to j}$ are stored at clique

j; to compute the optimization message $d_{i \to j}$; cliques also carry their original, unaligned priors.[3] At convergence, each node will not only have a subtree of an external tree, but also the incoming optimization messages that result from pruning of all other cliques of the external tree. Each node n can now locally compute the remaining optimization messages between the cliques that comprise its belief β_n (using the centralized OCA algorithm) and determine if one of the cliques in its belief is the root of the conditional alignment. If the root of the conditional alignment is a clique in β_n, node n aligns the prior marginals in its belief from this root. Otherwise, if the root of the conditional alignment is not a clique in β_n, the node determines the leaf that is closest to the root (by following the pointers up(i)), and aligns the prior marginals in its belief with respect to this leaf.

To illustrate the algorithm, we will revisit the temperature example from Section 2.3.2. Recall that in this example, we repeatedly prune the dangling leaves in the messages towards node 1, as shown in Figure 2.21. For example, when node 4 computes the message to node 3, it forms a maximum intersection tree (Figure 3.12(a)) comprising its local clique $\{H_4, H_5, H_6\}$ and the clique $\{H_5, H_6\}$ from its neighbor node 6. As discussed in Section 2.3.2, the clique $\{H_5, H_6\}$ is pruned, and we compute the optimization message $d_{56 \to 456}$, which is stored at clique $\{H_4, H_5, H_6\}$. Similarly, when computing its message to node 1, node 3 forms a maximum intersection tree of its local clique $\{H_1, H_2, H_3\}$ and the cliques $\{H_3, H_4, H_5\}$,$\{H_4, H_5, H_6\}$ in the incoming messages, as shown in Figure 3.12(b). Clique $\{H_4, H_5, H_6\}$ is pruned; since the incoming optimization message $d_{56 \to 456}$ has already been computed and stored at this clique, the downstream optimization message $d_{456 \to 345}$ can now be computed using (3.12) and stored at clique $\{H_3, H_4, H_5\}$. The final belief at node 1 (Figure 3.12(c)) consists of four cliques, some of which carry the optimization messages computed by other nodes. Node 1 can now compute the remaining optimization messages among the cliques in its belief to determine the optimal root.

Recall (Theorem 2.4) that at convergence, the computation that leads to the belief at some node n follows the structure of some external junction tree for the assumed density: the pruned cliques are guaranteed to be leafs of some external junction tree, and they are guaranteed to be pruned from the correct neighbors. Unfortunately, as strong as this property may seem, it does not guarantee that the nodes in the network will make consistent decisions about the optimal root. While the pruned cliques are always leaves of some external junction tree, the external junction trees traced by the pruning operations may *differ* from one node n to another. Even within a single node, the external junction trees may not be uniquely defined. For example, in the converged belief at node 1 (Figure 3.12(c)), there are two cliques $\{H_1, H_2, H_3\}$ that carry two, potentially inconsistent priors. Depending on how these two identical cliques are placed relative to each other in the maximum–intersection tree, we may get different optimal roots. This kind of ambiguity is not specific to identical cliques, and may occur whenever a maximum intersection tree is formed in the robust message passing algorithm (such as in message computations).

In order to ensure that the nodes agree on the choice of the external junction tree, we modify the robust

[3] Alternatively, only the maximum of the incoming messages to j can be stored. This maximum is initialized to the pair $(0, j)$ to indicate that, before any cliques are pruned from j, clique j is the best known root among the vertices in the subtree at j.

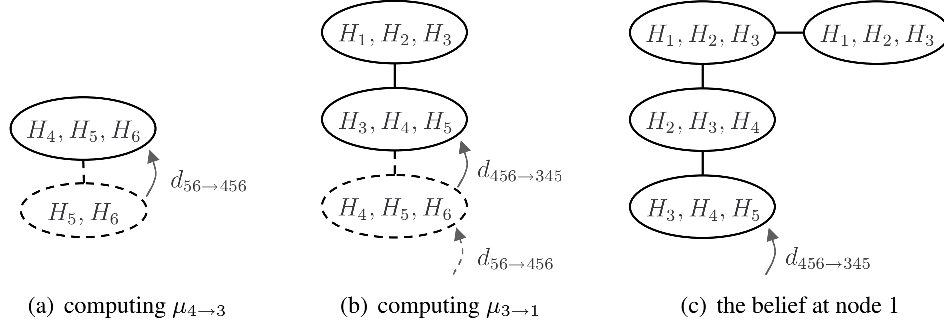


Figure 3.12: The integration of the distributed OCA algorithm into the robust message passing algorithm: (a) computing the inference message $\mu_{4\to3}$, (b) computing the inference message $\mu_{3\to1}$, (c) the converged belief at node 1. The dashed ellipses indicate the pruned cliques; the solid ellipses indicate the remaining cliques that are present in each message or belief. The arrows indicate the computed optimization messages d. The clique $\{H_1, H_2, H_3\}$ appears multiple times, because it appeared at both nodes 1 and 3 in Figure 3.7(b). The two copies are treated as two different cliques to ensure that nodes agree on the choice of the external junction tree.

message passing algorithm as follows. First, we explicitly remove any nondeterminism in the process of forming maximum intersection trees in the algorithm. This modification can be implemented simply by defining a lexicographical ordering over the pairs of neighboring clique IDs. In this manner, if two or more pairs of cliques have the same separator size, the maximum spanning tree algorithm connects the pairs with a higher lexicographical order. Second, whenever a network node m sends a message $\mu_{m\to n}$ to its neighbor n in the network junction tree, it includes the clique priors from the opposing message $\mu_{n\to m}$ in $\mu_{m\to n}$, omitting any likelihoods and OCA messages. The prior cliques of $\mu_{n\to m}$ are included in the set of cliques used to form the message $\mu_{m\to n}$ in Algorithm 1 in Section 2.3.3 and do not get pruned. With this change, we are effectively computing the messages by pruning the cliques of the belief, without double-counting the likelihoods. Including the clique priors from the opposing message does not modify the convergence properties of the original algorithm; by Theorem 2.2, given sufficient communication, each node n still obtains a subtree of some external junction tree for the assumed density $\tilde{p}$. Furthermore, at convergence, the opposing messages $\mu_{m\to n}$ and $\mu_{n\to m}$ carry the same set of cliques and are the subtrees of the beliefs β_m and β_n. Therefore, at convergence, neighbors in the network junction tree will have overlapping subtrees of the external tree. These overlapping subtrees can be "pasted" together to form a unique external junction tree. Combining these properties, we prove that distributed OCA yields a consistent global belief:

Theorem 3.5. *At convergence and in the absence of network partitions, nodes running distributed* OCA *reach a globally consistent belief based on conditional alignment, selecting the root clique that leads to the joint distribution of minimal entropy.*

While we proved that the distributed OCA algorithm reaches a globally consistent belief if it converges, proving convergence is not an easy task. In the original robust message passing algorithm, convergence

was attained as soon as all the messages were computed in an order of message dependencies. That is, if the nodes communicate with all their neighbors in each round and no messages are ever lost, the algorithm converges in the number of rounds equal to the diameter of the network junction tree. Unfortunately, since we include the prior cliques of the opposing message $\mu_{n\to m}$ in $\mu_{m\to n}$, this creates circular dependencies among the messages, which complicates the analysis. We would expect that the fact that message computations are deterministic would help the convergence, but proving convergence remains an open problem. Nevertheless, the algorithm appears to converge in practice.

The distributed OCA algorithm runs continuously until convergence, but we can conceptually distinguish two phases of its execution. In the first phase, the nodes do not yet agree on the external junction tree used in the pruning operations, and the distributed OCA algorithm is not guaranteed to align, or even partially align the inconsistent prior marginals. On the other hand, once the nodes agree on the choice of the external junction tree, running the distributed OCA algorithm is equivalent to running the centralized asynchronous OCA algorithm on this junction tree. In particular, Theorem 3.4 shows that before convergence, the distributed OCA algorithm computes a partial alignment of the inconsistent marginals from a set of cliques, each of which is the best root in some neighborhood within the external junction tree.

3.4.4 Jointly optimized alignment

While conceptually simple, there are situations where optimized conditional alignment will not provide a good aligned distribution. Suppose that we modify the example in Figure 3.9, so that cameras 2 and 3 carry marginals $\pi_2(l_2, l_3, m)$ and $\pi_{2'}(l_2, l_3, m)$, respectively. If both cameras observe the object when the network is partitioned, node 2 will have a better estimate of L_2, while node 3's estimate of L_3 will be more accurate. If either node is chosen as the root, the aligned distribution will have a worse estimate of the pose of one of the cameras, because performing rooted alignment from either direction effectively overwrites the marginal of the other node. In this example, rather than fixing a root, we want an aligned distribution that attempts to simultaneously optimize the distance to both $\pi_2(l_2, l_3, m)$ and $\pi_{2'}(l_2, l_3, m)$.

Recall that KL divergence (Definition 2.8) is one natural measure of the dissimilarity between two distributions. When the distribution $p(\mathbf{x})$ is well-defined, minimizing the KL divergence $D(p(\mathbf{x}) \,\|\, q(\mathbf{x}))$ over some family of distributions q yields the KL projection (Definition 2.9). KL projection was used in the definition of the B&K98 algorithm and in Section 2.3.4 when discussing partial correctness of the robust message passing algorithm. Unfortunately, in alignment, we do not have access to the complete distribution $p(\mathbf{x})$; we only have access to the potentially inconsistent clique marginals $\pi_i(\mathbf{x}_{C_i})$. Therefore, we cannot formulate an optimization problem that compares two distributions over $\mathbf{X}$ as a whole. Nevertheless, we can attempt to solve an optimization problem where we compare the marginals π_i to the clique marginals of the approximate distribution:

$$\tilde{p}_{\text{KL1}}(\mathbf{x}) = \operatorname*{argmin}_{q(\mathbf{x}):q\models T} \sum_{i\in N_T} D(\pi_i(\mathbf{x}_{C_i}) \,\|\, q(\mathbf{x}_{C_i})). \tag{3.15}$$

Here, $q \models T$ denotes the constraint that q can be represented as a decomposable model with junction tree $(T, \mathbf{C})$. This means that $q(\mathbf{x})$ will not be represented as a single monolithic distribution in the optimization problem (3.15); rather, it will be represented as a set of marginals $q(\mathbf{x}_{C_i})$, with a constraint that the neighboring marginals $q(\mathbf{x}_{C_i})$ and $q(\mathbf{x}_{C_j})$ for $\{i, j\} \in E_T$ agree on the separator indices $S_{i,j}$, that is,

$$\sum_{\mathbf{x}_{C_i - S_{i,j}}} q(\mathbf{x}_{C_i}) = \sum_{\mathbf{x}_{C_j - S_{i,j}}} q(\mathbf{x}_{C_j}). \tag{3.16}$$

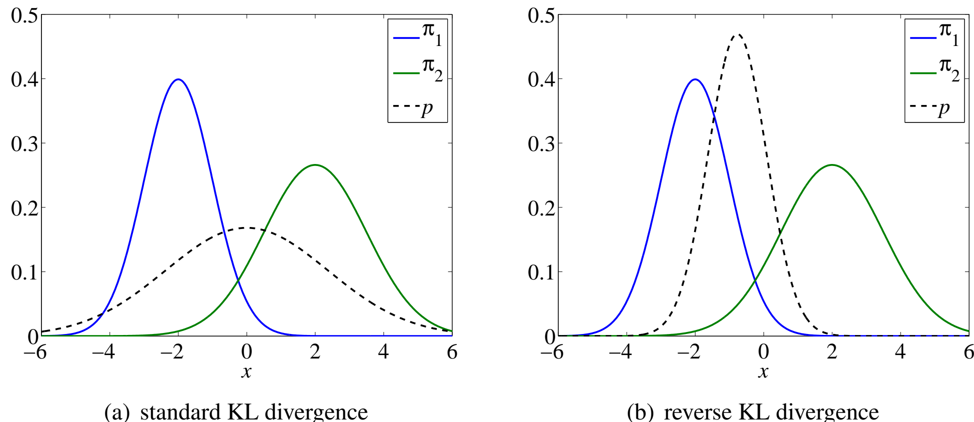
When the approximate priors π_i are Gaussian, the optimization problem (3.15) can be rewritten in the moment form. In the moment parameterization, consistency between neighboring marginals of q is very easy to enforce: we can represent the distribution q as a Gaussian $\mathcal{N}(\mathbf{x}; \mu, \Sigma)$. By the rule of marginalization in moment Gaussians, each marginal $q(\mathbf{x}_{C_i})$ is simply $\mathcal{N}(\mathbf{x}_{C_i}; \mu_{C_i}, \Sigma_{C_i})$, where we have selected the elements of μ indexed by C_i to form the mean vector of the distribution $q(\mathbf{x}_{C_i})$ and similarly for the covariance matrix. Substituting the formula for the KL divergence between two Gaussian distributions into (3.15), the optimization problem becomes

$$\begin{aligned} &\underset{\mu,\Sigma}{\text{argmin}} && \sum\nolimits_{i \in N_T} \log\det \Sigma_{C_i} + \text{tr}(\Sigma_{C_i}^{-1}\Sigma_i) + (\mu_{C_i} - \mu_i)^\top \Sigma_{C_i}^{-1}(\mu_{C_i} - \mu_i) \\ &\text{subject to} && \Sigma_{C_i} \succeq 0, \qquad \forall i \in N_T, \end{aligned} \tag{3.17}$$

where μ_i, Σ_i are the means and covariances of the marginals π_i. The constraint $\Sigma_{C_i} \succeq 0$ specifies that each Σ_{C_i} is positive semi-definite, that is, $q(\mathbf{x}_{C_i})$ is a valid Gaussian distribution. The optimization problem (3.17) only recovers the covariance parameters $(\Sigma)_{a,b}$ for the pairs of variables $\{a, b\}$ that are jointly present in some clique C_i, i.e. $a, b \in C_i$ for some $i \in N_T$; the other parameters do not appear in the objective function and can be omitted from the optimization problem entirely. This result is sufficient, since we are only interested in the clique marginals $q(\mathbf{x}_{C_i})$, rather than the monolithic representation of the distribution $q(\mathbf{x})$.

Figure 3.13(a) illustrates the results obtained by the optimization problem (3.17) on a instance with two inconsistent marginals $\pi_1(x)$ and $\pi_2(x)$; the marginal π_1 on the left is more certain (peaked) than the marginal π_2 on the right. The approximate distribution $\tilde{p}_{\text{KL1}}$ has fatter tails than either marginal π_1 or π_2; this is a standard result that stems from KL divergence penalizing q whenever $q(\mathbf{x}_{C_i}) < \pi_i(\mathbf{x}_{C_i})$. Furthermore, since all differences of the means $\mu_{C_i} - \mu_i$ have the same penalty $\Sigma_{C_i}^{-1}$, the approximate distribution is centered between μ_1 and μ_2. In alignment, both these properties are undesirable. As mentioned earlier, conditioning in Gaussians only decreases the entropy (uncertainty) of the estimates, so the approximation should be no less certain than either π_1 or π_2. Furthermore, if one of the marginals is more certain than the other, it typically incorporates more observations, and the alignment should bias $\tilde{p}$ towards this estimate. Therefore, (3.17) is not a desirable objective to optimize.[4]

[4]It also happens to be difficult to optimize, since the problem (3.17) is non-convex in either the moment or the information form.



(a) standard KL divergence (b) reverse KL divergence

Figure 3.13: Minimizing the KL divergence with two inconsistent Gaussian marginals $\pi_1(x)$ and $\pi_2(x)$. (a) Minimizing the KL divergence with the standard ordering of arguments yields a conservative estimate that covers both marginals. (b) Minimizing the reverse KL divergence yields an optimistic estimate that is biased towards the more peaked distribution.

As mentioned in Section 2.3.4, KL divergence is a non-symmetric measure: minimizing $D(p \,\|\, q)$ does not yield the same solution as minimizing $D(q \,\|\, p)$. We exploit this fact to propose an alternative optimization problem that minimizes the sum of reverse KL divergence from the aligned distribution to the clique marginals π_i:

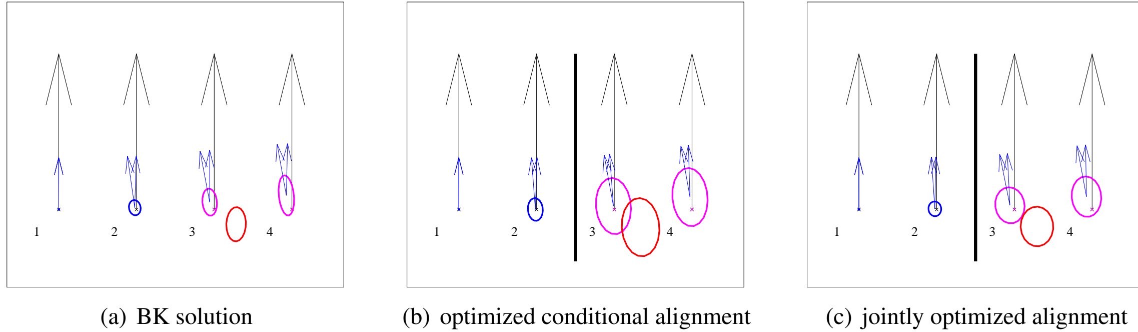
$$\tilde{p}_{\mathrm{KL2}}(\mathbf{x}) = \operatorname*{argmin}_{q(\mathbf{x}):q \models T} \sum_{i \in N_T} D(q(\mathbf{x}_{C_i}) \,\|\, \pi_i(\mathbf{x}_{C_i})), \tag{3.18}$$

where, once again, $q \models T$ denotes the constraint that $\tilde{p}_{\mathrm{KL2}}$ can be represented as a decomposable model with junction tree $(T, \mathbf{C})$. For Gaussian distributions, this optimization problem corresponds to

$$\begin{aligned} &\operatorname*{argmin}_{\mu,\Sigma} && \sum\nolimits_{i \in N_T} -\log\det\Sigma_{C_i} + \mathrm{tr}(\Sigma_i^{-1}\Sigma_{C_i}) + \sum\nolimits_{i \in N_T} (\mu_i - \mu_{C_i})^\top \Sigma_i^{-1} (\mu_i - \mu_{C_i}), \\ &\text{subject to} && \Sigma_{C_i} \succeq 0, \qquad \forall i \in N_T. \end{aligned} \tag{3.19}$$

The problem in (3.19) consists of two independent convex optimization problems over the covariances and the means of q, respectively. The former problem is an instance of log-determinant maximization (Wu et al., 1996), while the latter problem is an instance of linear regression. Both problems can be solved efficiently.

The typical intuition about the reverse KL divergence is that it produces overconfident estimates. Surprisingly, in our setting, the reverse KL divergence is better than the standard form, substantially improving the estimate quality. Figure 3.13(b) illustrates the approximation $\tilde{p}_{\mathrm{KL2}}$ on the earlier example with two inconsistent marginals $\pi_1(x)$ and $\pi_2(x)$. We see that the approximation correctly correctly captures our intuition that it should be biased towards the more peaked marginal, and its uncertainty roughly matches that of π_1 and π_2. This method, which we call **jointly optimized alignment**, provides very good aligned



(a) BK solution (b) optimized conditional alignment (c) jointly optimized alignment

Figure 3.14: Comparison of the presented alignment methods on the example with four cameras. (a) The exact solution, computed by the B&K98 algorithm in the absence of partitions, repeated from Figure 3.10(a). (b) The solution obtained by optimized conditional alignment, repeated from Figure 3.10(c). (c) Solution obtained by jointly optimized alignment.

distributions in practice, as illustrated in Figure 3.14(c).

Similarly to the optimized conditional alignment, the jointly optimized alignment preserves a consistent set of marginals:

Theorem 3.6. *Let $\{\pi_i\}$ be a set of consistent marginals. Then the jointly optimized alignment procedure recovers the original marginals:*

$$\tilde{p}_{KL2}(\mathbf{x}_{C_i}) = \pi_i(\mathbf{x}_{C_i})$$

for all $i \in N_T$.

Theorem 3.6 follows immediately from the fact that KL divergence $D(q \,\|\, p)$ is zero if and only if p and q are equal.

We do not describe a fully distributed solution for jointly optimized alignment; however, the original robust message passing algorithm can be extended to perform jointly optimized alignment. The second optimization problem in (3.19) can be solved in a distributed manner using distributed linear regression (Guestrin et al., 2004). The distributed linear regression algorithm has the same structure as the sum–product algorithm, but as suggested by Paskin (2004), the algorithm can be extended to work with a factorized form analogous to the robust factors of robust message passing. The first optimization problem in (3.19) can be solved using a distributed version of an iterative method, such as conjugate gradient descent (Bertsekas and Tsitsiklis, 1997).

3.5 Experimental results

To evaluate the methods presented in this chapter, we performed experiments on the SLAT application from Example 3.2 and the dynamic calibration of the temperature monitoring network from Example 3.3.

Since we did not have a wireless sensor network, we evaluated our distributed algorithms using the event-based distributed-systems simulator described in (Paskin and Guestrin, 2004b). The simulator incorporates message loss and faithfully captures the networking aspects of the problem.

3.5.1 Applications

In simultaneous localization and tracking (SLAT), a set of cameras simultaneously localizes itself by tracking a moving object. The inference task is for each node to estimate its camera pose, along with the position of a moving object. The error metric is the root mean square (RMS) error of the estimated camera location. The data for this task are the observations of the object by the cameras; we use the data from the real camera deployment shown in Figure 1.1(a), along with two simulated scenarios (the real camera network was manually calibrated to allow us to evaluate the localization error). An example DBN model for this task was shown in Figure 3.3; the details of the model are described in the next chapter. For the networking aspects, we simulated link qualities using an exponentially–decaying function of the squared distance between nodes, where nearby cameras (about 1 meter apart) had about 20% packet loss.

In the second application, a network of temperature sensors calibrates itself by sensing their local temperatures with some additive bias and Gaussian noise. We sample the measurement bias at each location independently according to the $\mathcal{N}(0, 1)$ distribution. The inference task is for each node to estimate its temperature and the measurement bias; the error metric is the root mean square (RMS) error of the estimated measurement bias. The model and the observations in our experiments are based on the sensor network dataset collected at Intel Research Laboratory, Berkeley.[5] The dataset contains temperature measurements for 54 sensors over the course of several weeks. We set the time step duration to 1 hour and test on three days of data, totalling 72 time steps. The initial prior for each temperature location is uncertain, and the initial prior for the biases matches the sampling distribution. In order to better capture the evolution of the temperatures, we divide the day into four time periods of approximately 6 hours each. We learn a separate transition model for each time period based on the data from two days; when performing inference, we cycle through these four transition models. The parents of each temperature variable i are the same in all four transition models; the parents are chosen by identifying the three most informative temperature locations in some physical neighborhood of i, as measured by the conditional entropy of the temperature $H_i^{(t+1)}$ given $\mathbf{H}^{(t)}_{\mathrm{Pa}[H_i]}$. Each clique in the external junction tree contains both the temperature and the bias variables for each of the locations present in the clique. Finally, in order to simulate the message loss, we use the aggregate connectivity data from the dataset, averaged over all time. The connectivity is not symmetric—node A may hear B better than B hears A.

[5] The dataset can be found at `http://www.select.cs.cmu.edu/data/labapp3/index.html`

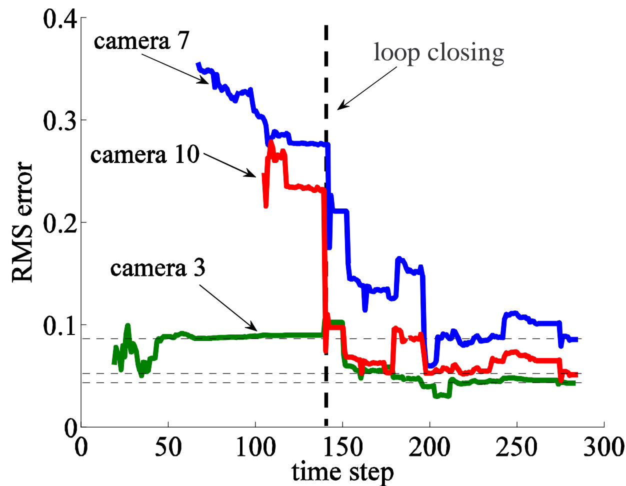
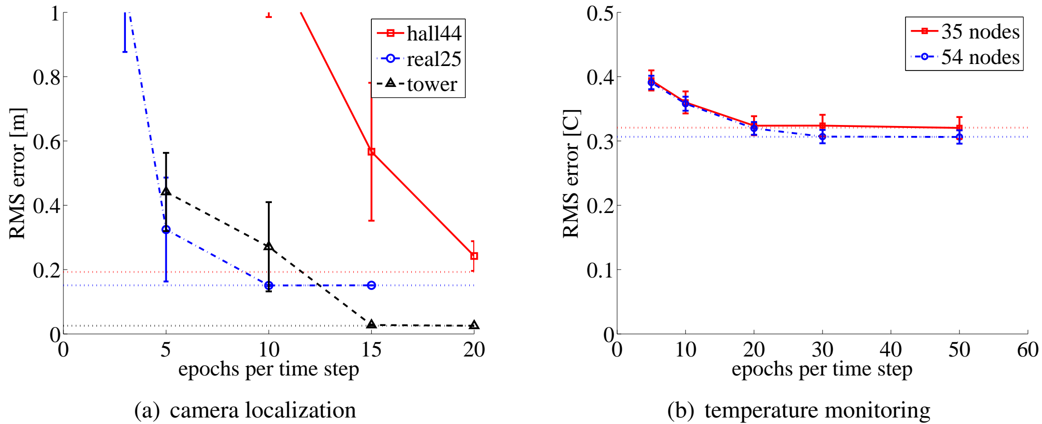


Figure 3.15: RMS of estimated poses versus number of time steps, in the tower scenario, using our distributed algorithm. Horizontal lines indicate the corresponding centralized B&K98 solution at the end of the experiment. The vertical dashed line indicates when the object closes the loop.

3.5.2 Convergence

Our first experiment applies our approximate distributed filtering algorithm to the tower scenario from Figure 3.6(a). We take the basic algorithm, described in Section 3.3, and run the estimation phase to convergence at every time step. Figure 3.15 shows that our distributed algorithm converges to the same solution as the centralized B&K98 algorithm. Note that the convergence curve is different for different cameras, since their estimate is uninformative until they first observe the object. Interestingly, in this figure, we can clearly see a "loop-closing" effect (Paskin, 2003) after about 150 time steps: the first camera to observe the object is certain about its location; when the object returns to the field of view of this camera, its position becomes more certain, and the estimates of all cameras become more accurate.

In Figure 3.16, we evaluate the sensitivity of the algorithm to incomplete communication. At every time step, we run the estimation phase for a given number of epochs. In each epoch, each node attempts to send any new messages it has queued up; about 30% of messages are lost due to lossy communication. The results are qualitatively different for SLAT and temperature network calibration. In the temperature network, missing information about the temperatures at neighboring locations can only introduce a small error, since the algorithm employs a meaningful approximate transition model and the observations are informative. In SLAT, however, failing to propagate the observations can introduce substantial errors: the algorithm may fail to close the loop. Therefore, the error curve is much steeper in SLAT. For the real camera experiment, the algorithm converges much quicker: due to the small size of the network, fewer messages are needed to propagate the information around the network. Overall, these results show that with about 15–20 epochs per time step for SLAT and 20–30 epochs per time step for the temperature network, our distributed algorithm converges to the same solution as the centralized one.



(a) camera localization

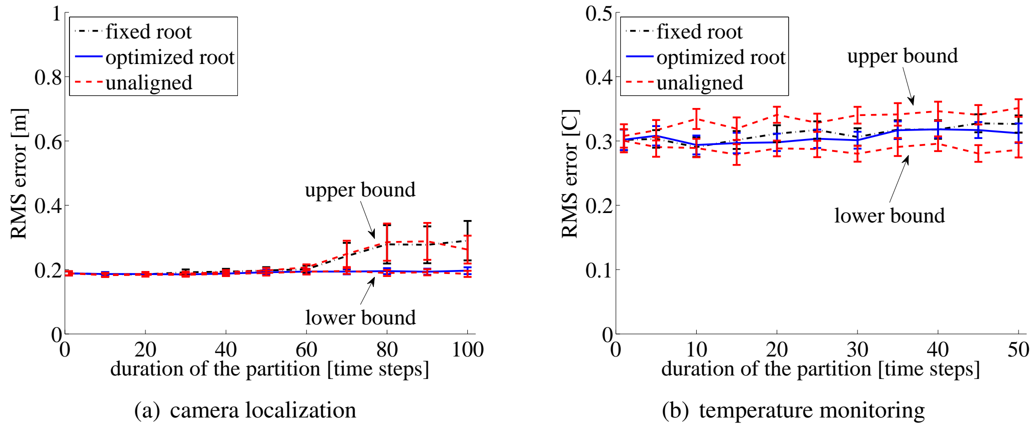
(b) temperature monitoring

Figure 3.16: Convergence versus amount of communication per time step: (a) for the SLAT application and (b) for a temperature network of 35 and 54 real sensors, respectively. Horizontal lines indicate the RMS error of the corresponding centralized B&K98 solutions at the end of the experiment.

3.5.3 Alignment

In the second set of experiments, we evaluate the alignment methods, presented in Section 3.4. We first evaluate the sensitivity of the methods to partitions of varying duration. In Figure 3.17(a), the nodes in a real camera network first get a rough estimate of their locations. At some point, the network is split into four components; in each component, the nodes communicate fully, and we evaluate the quality of the solution if the communication were to be restored after a given number of time steps. The vertical axis shows the RMS error of estimated camera locations at the end of the experiment. For the unaligned solution, the nodes may not agree on the estimated pose of a camera, so it is not clear which node's estimate should be used in the RMS computation. The plot shows an "omniscient envelope" of the RMS error, where, given the (unknown) true camera locations, we select the best and worst estimates available in the network for each camera's pose. The results show that, in the absence of optimized alignment, inconsistencies can degrade the SLAT solution: observations collected after the communication is restored are not sufficient to make up for the errors introduced by the partition. Figure 3.17(b) shows the results for a similar experiments in the temperature monitoring network with 54 nodes; the network is split into three components. The results are less pronounced, but conditional alignment fully recovers from the partition (the optimized and the fixed root obtain the same quality of solution in this case).

The fourth experiment evaluates the performance of the distributed algorithm in highly-disconnected scenarios. Here, the sensor network is hierarchically partitioned into smaller disconnected components by selecting a random cut through the largest component. The communication is restored shortly before the end of the experiment. Figure 3.18(a) shows the importance of aligning from the correct node: the difference between the optimized root and an arbitrarily chosen root is significant, particularly when the network becomes more and more fractured. In our experiments, large errors often resulted from the nodes



(a) camera localization

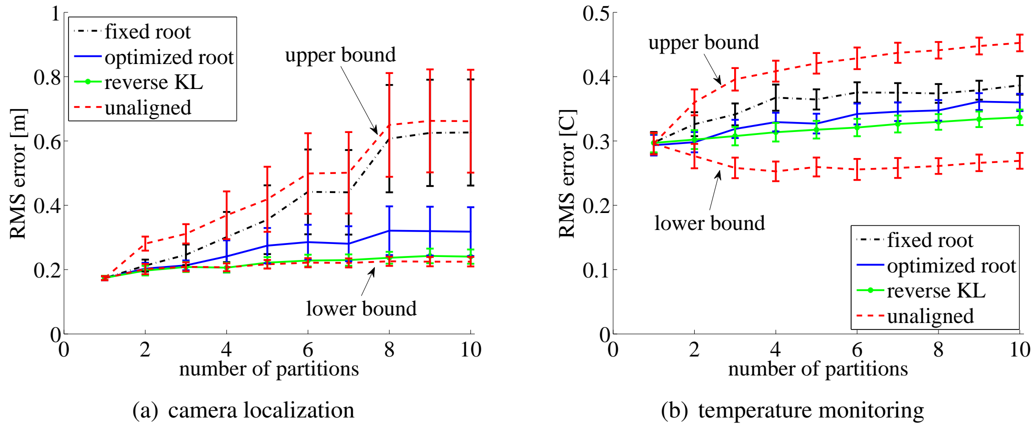
(b) temperature monitoring

Figure 3.17: RMS error vs. duration of the partition. For the unaligned solution, the plots show bounds on the RMS error: given the unknown true camera locations (sensor biases, respectively), we select the best and worst estimates available in the network. (a) Partitioning of the camera network into four components. In the absence of optimized alignment, inconsistencies can degrade the quality of the solution. (b) Partitioning of the temperature monitoring network into three components. Conditional alignment fully recovers from the partition.

having uncertain beliefs, hence justifying the objective function. We see that the jointly optimized alignment described in Section 3.4.4 (marked as "reverse KL"), tends to provide the best aligned distribution, though often close to the optimized root, which is simpler to compute. Finally, Figure 3.18(b) shows the alignment results on the temperature monitoring application. Compared to SLAT, the effects of network partitions on the results for the temperature data are less severe. One contributing factor is that every node in a partition is making local temperature observations, and the approximate transition model for temperatures in each partition component is quite accurate, hence all the nodes continue to adjust their estimates meaningfully while the partition is in progress. Still, optimized alignment is necessary, so that nodes do not accidentally use inaccurate estimates present in the network.

3.6 Related work

Distributed inference in dynamical systems has been typically examined in the context of specific applications. Tracking (Example 3.1) is a classical problem, where a sensor network monitors the location of a moving object over time. When the transition model is a conditional linear Gaussian, a standard centralized approach is a Kalman filter (Kalman, 1960). In their seminal work, Grime and Durrant-Whyte (1994); Manyika and Durrant-Whyte (1995) used an information form of the Kalman filter; the **information filter** maintains the belief as a Gaussian in the information form. Because multiplying two Gaussians in the information form is very simple (it is implemented using simple addition), the estimation phase can be implemented in a distributed manner by aggregating the observation likelihoods across the nodes. The



(a) camera localization (b) temperature monitoring

Figure 3.18: RMS error vs. number of partitions. In camera localization (a), the difference between the optimized alignment and the conditional alignment from an arbitrarily chosen fixed root is significant. For the temperature monitoring (b), the differences are less pronounced, but follow the same trend.

prediction/roll-up is entirely local. In tracking, our approximate distributed filtering algorithm, presented in Section 3.3, effectively reduces to the algorithm of Grime and Durrant-Whyte (1994): the assumed density consists of a single clique with all the state variables, so no dangling leafs are ever pruned in the estimation phase, and the belief at each node is simply the product of the prior distribution and the likelihoods across the nodes in the network.

For problems with larger state spaces, some approximations are typically made to keep the state representation and communication tractable. For example, in multi-robot localization, the state consists of the pose of each robot. The robots observe each other, which couples the estimates of the different robot poses, so exact distributed filtering is no longer tractable. Fox et al. (2000) address this problem by maintaining a fully factored representation, where the robots store independent particle filters over their individual poses, one at each robot. Whenever a measurement is made that relates two robots, the algorithm projects the coupled belief back to the individual robot marginals. Our approximate distributed filtering algorithm could also be used to solve the multi-robot localization problem, with some changes to account for the fact that the robots do not start with a clique that covers the arguments of the likelihood. Unfortunately, while our algorithm solves the multi-robot localization problem, it is suboptimal. The network junction tree transport layer facilitates large-scale coordination, which is important for reasoning about an assumed density that captures some dependencies among the random variables in the model. However, in this case, the assumed density is fully factored (that is, it is a product of independent marginals, each stored at one node). With such a simple model, reasoning can be entirely local to the robot's neighborhood: the nodes in (Fox et al., 2000) directly collect the relevant priors and likelihoods from their neighbors, which lowers the communication complexity and speeds up the convergence of the algorithm.

The simultaneous localization and tracking (SLAT) problem (Example 3.2) has also received some attention. For example, Taylor et al. (2006) describe a SLAT algorithm for problems with an uninformative

motion model, effectively assuming that the object can move arbitrarily through the world. Assuming an uninformative motion model substantially simplifies the problem, because influence cannot flow between distant nodes through the motion model, so exact filtering is tractable, and the posterior distribution can be represented as a Gaussian with a sparse information matrix. The algorithm of Taylor et al. (2006) is appealing for its simplicity and ease of distributed implementation: each node stores only a portion of the information matrix, and all the operations in the algorithm are local to the node neighborhoods. However, using an uninformative motion model is a restrictive assumption that prevents accurate localization in settings where there is only a minimal overlap in the range of the sensors. Such scenarios are very common in camera networks, described in the next chapter. By comparison, our approach allows for a variety of DBN models, including the general SLAT. Neither algorithm assumes reliable communication among the nodes, but the behavior of our algorithm is easier to characterize. In particular, in the absence of reliable communication, some of the operations in the algorithm of Taylor et al. (2006) may not be well-defined, as the nodes may have inconsistent views of the information matrix. In the worst case, these inconsistencies could lead to an estimate that is an invalid Gaussian distribution (because the information matrix is not positive definite).

In our algorithm, the approximation structure (the external junction tree) is chosen in advance. Sometimes, it may be better to select the approximation structure *adaptively*, by pruning the weak dependences in the belief given the observations collected so far. For example, the Sparse Extended Information Filter (SEIF) (Thrun et al., 2004) maintains the approximate posterior distribution as a Gaussian in the information form. The algorithm introduces conditional independence assumptions into the belief, by effectively zeroing out some entries of the information matrix that are small. The algorithm is centralized and has been designed for the Simultaneous localization and mapping (SLAM) problem, where a robot localizes itself while building a map of its environment, but the algorithm also works for other problems with the same DBN structure, including SLAT. Conceptually, the SEIF algorithm could be distributed in the same way as the algorithm of Taylor et al. (2006), by letting each node maintain a subset of the entries of the information matrix. Unfortunately, in the presence of communication failures and delays, the nodes may once again obtain inconsistent views of the information matrix, and the operations in the algorithm may not be well-defined.

An alternative approach to solving the SLAT problem was proposed by Djugash et al. (2008), who considered a version of the problem where the object is a moving robot, capable of performing sensing and computation. Similarly to our approach, each node in their algorithm maintains an approximate posterior distribution over a small sets of variables. However, rather than formally projecting the distribution to a family of decomposable models, they heuristically reason in terms of a small neighborhoods of nodes. In order to propagate the information between the different nodes, their algorithm employs a variant of loopy belief propagation (Pearl, 1988) in each step. Their algorithm has certain benefits over ours: the nodes do not need to choose an approximation structure in advance and can rely purely on local coordination to condition on each other's evidence, which may in turn speed up the convergence. However, their algorithm

obtains overconfident estimates, because it propagates the evidence in loop. Overconfident estimates can be particularly problematic when obtaining a Gaussian approximation to the observation likelihoods and in deciding, what parts of the network need more information to improve their calibration. By comparison, our solution does not suffer from overconfident estimates, because we obtain a principled approximation equivalent to the centralized B&K98 algorithm at convergence.

Although much of the work on distributed dynamic inference focuses on specific applications, there is also some prior work on addressing the general family of DBN models. In the centralized contexts, DBN inference has been addressed by a variety of techniques, including loopy belief propagation (LBP) (Murphy and Weiss, 2001). LBP is implemented by passing updates between variables adjacent in the model. Since the coordination is entirely local, LBP can be easily distributed (Crick and Pfeffer, 2003). Pfeffer and Tai (2005) generalized LBP to continuous-time Bayesian network (Nodelman et al., 2002), by allowing the nodes to operate asynchronously at different points in time. A key difference between loopy belief propagation and our work is that each node in LBP reasons in terms of an entire *history* of a random process, rather than just the value of the process at the latest time step. By maintaining a belief over the past variables (or a window), the algorithm can incorporate the evidence in any order and continue to propagate information about past observations, even after the system has advanced in time. By comparison, our algorithm reasons about each time step separately; once the system advances to the next time step, the observations from the past are no longer propagated (this is why our algorithm may obtain inconsistent estimates even in the absence of partitions). Naturally, being able to incorporate past observations is desirable; however, in LBP, this ability comes with over-confident estimates and lack of guaranteed convergence (except for some special cases). In our experiments with LBP, we have found that the algorithm of Murphy and Weiss (2001) converged for the temperature monitoring network (with overconfident estimates), but failed to converge for our camera SLAT application, described in Chapter 4. Thus, the relative merits of LBP and our method are application-dependent: if the model is very noisy and if simple, overconfident estimates are sufficient, then LBP performs better; if the model is accurate and a correct estimate over an arbitrary query is desired, then our method is preferred.

In the methods described above, the nodes communicate sufficient statistics about their observations. In some applications, communicating the measurements is appropriate. For example, Rosencrantz et al. (2003a) present an algorithm for decentralized tracking of opponents in a game of robotic laser tag. Each robot on one team runs a particle filter (Rosencrantz et al., 2003b) that captures the locations of all the robots on the opposing team. The particle filter at each robot incorporates a subset of the team's evidence; the robots on the same team periodically share informative measurements to improve each other's estimates. While our methods are not applicable to robotic laser tag, it is interesting to compare the methods in terms of the communication and space complexity. For the applications considered in this chapter, both approaches require only a few packets of communication per node per round. However, while sending the measurements may be viable from the communications perspective, the complete posterior distribution may not fit into the sensor's memory. While the amount of memory on today's devices continues to grow,

so does the scale of the sensor networks, so algorithms running on these networks need to work with a decentralized representation of the model.

Finally, one interesting problem in distributed dynamic inference is multi-robot simultaneous localization and mapping (SLAM), a generalization of multi-robot localization to the setting with unknown maps. In multi-robot SLAM, several robots construct a map of an environment, while localizing themselves in the map. Multi-robot SLAM requires a different granularity of distributed reasoning than the problems considered in this chapter, because each robot needs to carry an estimate of a map that covers a large fraction of the environment. However, some of the challenges, identified in our work, are also present in multi-robot SLAM. For example, each robot in the algorithm of Thrun and Liu (2003) carries an independent Gaussian estimate of the map, updated using the Sparse Extended Information Filter (Thrun et al., 2004). Thrun and Liu (2003) assume that the robots communicate only once, at the end of the experiment, permanently merging their maps at that time. Yet, in large-scale deployments, the robots will need to communicate intermittently, exploring different parts of the environment and updating each other's maps when in range. The process of merging two maps is akin to the alignment problem, identified in Section 3.4.1: the robots carry inconsistent estimates and wish to obtain a distribution that informatively incorporates evidence from both maps. Multi-robot SLAM has certain specifics (for example, the robots need to solve the data association problem to relate the features of one map to another). However, the joint alignment method, presented in Section 3.4.4, may carry over to this domain.

To summarize, compared to prior work, our algorithm addresses a general class of problems that can be modeled with dynamic Bayesian networks. In some cases, the custom algorithms designed for the specific applications outperform our general solution, but our approach addresses several challenges not considered before: we obtain a principled approximation equivalent to the centralized B&K98 algorithm at convergence, we provide robustness guarantees to node failures and network partitions, and we identify and address the belief inconsistency problem that arises in distributed systems.

3.7 Discussion

In this chapter, we proposed an algorithm for robust assumed density filtering in distributed systems. Graphical models, and in particular junction trees, once again played a key role in the algorithm. We showed how the estimation phase of the algorithm can be implemented by robust message passing (Paskin and Guestrin, 2004b); the prediction, roll-up, and projection phase is then entirely local. We showed that, at convergence, the algorithm obtains the same solution as the centralized B&K98 algorithm. If the estimation phase is not run to completion or in the presence of network partitions, the estimates obtained at each node may not be consistent. If left unattended, these inconsistent estimates could degrade the quality of the solution. We identified the alignment problem, and outlined a basic distributed algorithm, distributed optimized conditional alignment (OCA), that lets the nodes obtain a globally consistent, aligned

solution. We also presented a more advanced alignment approach, jointly optimized alignment, that simultaneously minimizes a measure of dissimilarity between the inconsistent marginals and the approximate distribution.

One potential disadvantage of our approach is that it supports only Gaussian distributions and, to a lesser extent, discrete distributions (we did not present an alignment method for discrete distributions). This limitation is inherited from the B&K98 algorithm, and it stems from the need to perform division operations to interpret the belief at the node. Gaussian and discrete distributions are the only types of multivariate distributions known to support such operations efficiently. While these two types of distributions are very common, many systems have nonlinearities which prevent them to be modeled exactly as a Gaussian. A standard approach is to adopt a method of linearization (employed, for example, in the Extended Kalman Filter) to approximate the exact posterior with a Gaussian. We will see an instance of such a strategy in the next chapter, when we elaborate on the details of the SLAT model for camera localization.

Appendix 3.A Proofs

Proof of Theorem 3.2. The theorem is proved by induction on the distance from the root r. The discussion at the beginning of Section 3.4.2 showed that $\tilde{p}_r(\mathbf{x}_{C_i}) = \pi_r(\mathbf{x}_{C_r})$, independently of whether the marginals $\{\pi_i\}$ are consistent. Let $i \neq r$ be a vertex in T other than the root, with an upstream neighbor $\text{up}(i) = j$, and suppose that the theorem holds for j: $\tilde{p}_r(\mathbf{x}_{C_j}) = \pi_j(\mathbf{x}_{C_j})$. Then

$$\begin{aligned}\tilde{p}_r(\mathbf{x}_{C_i}) &= \sum_{\mathbf{x}_{V-C_i}} \tilde{p}_r(\mathbf{x}) \\ &= \tilde{p}_r(\mathbf{x}_{S_{i,j}}) \times \pi_i(\mathbf{x}_{C_i - S_{i,j}} \,|\, \mathbf{x}_{S_{i,j}}).\end{aligned}$$

Here, we use the fact that marginalizing out variables downstream of C_i in the decomposition (3.8) amounts to simply removing the corresponding conditionals $\pi_k(\mathbf{x}_{C_k - S_{\text{up}(k),k}} \,|\, \mathbf{x}_{S_{\text{up}(k),k}})$ for k downstream of i. Since, by the inductive hypothesis, $\tilde{p}_r$ and π_j agree on the clique C_j, they also agree on the marginal over the separator $S_{i,j} \subseteq C_j$, that is, $\tilde{p}_r(\mathbf{x}_{S_{i,j}}) = \pi_j(\mathbf{x}_{S_{i,j}})$. We now obtain

$$\begin{aligned}\tilde{p}_r(\mathbf{x}_{C_i}) &= \pi_j(\mathbf{x}_{S_{i,j}}) \times \pi_i(\mathbf{x}_{C_i - S_{i,j}} \,|\, \mathbf{x}_{S_{i,j}}) \\ &= \pi_i(\mathbf{x}_{S_{i,j}}) \times \pi_i(\mathbf{x}_{C_i - S_{i,j}} \,|\, \mathbf{x}_{S_{i,j}}) \\ &= \pi_i(\mathbf{x}_{C_i}),\end{aligned}$$

where the first step follows from the consistency of π_i and π_j, and the second step follows from the standard decomposition $p(\mathbf{x}_{A\cup B}) = p(\mathbf{x}_A) \times p(\mathbf{x}_B \,|\, \mathbf{x}_A)$ for $A \cap B = \emptyset$. This completes the proof of the theorem. □

Proof of Lemma 3.1. The lemma is proved by induction on the depth of $T_{i,j}$. When i is a leaf, with

neighbor j, then i does not have any neighbors other than j, and the condition $d_{k\to i} < 0,\ \forall k \neq j$ is trivially satisfied. Thus, the message $d_{i\to j}$ is simply $\triangle_{i,j}$, which proves the Lemma when $T_{i,j}$ consists of a single vertex i.

Suppose that i is a non-leaf vertex, and that the lemma holds for all messages $d_{k\to i}$ with $k \in N_T(i) \setminus j$. If $d_{k\to i} < 0, \forall k \neq j$, then the entropy with root i is smaller than the entropy with any vertex upstream of i, that is, i is the best root in $T_{i,j}$, and the entropy difference between j and i is simply $\triangle_{i,j}$. This is exactly the quantity computed by (3.12). Otherwise, there is a better root in $T_{i,j}$, denote it by r. Then $\max_{k\neq j} d_{k\to i}$ is the entropy difference between root i and r, and

$$H_{\tilde{p}_j}(\mathbf{X}) - H_{\tilde{p}_r}(\mathbf{X}) = \big(H_{\tilde{p}_j}(\mathbf{X}) - H_{\tilde{p}_i}(\mathbf{X})\big) + \big(H_{\tilde{p}_i}(\mathbf{X}) - H_{\tilde{p}_r}(\mathbf{X})\big) = \triangle_{i,j} + \max_{k\neq j} d_{k\to i}.$$

Again this is exactly the quantity computed by (3.12), which proves the lemma. □

In order to prove Theorem 3.4, let N_i denote the set of vertices, whose messages have contributed to the result at vertex i; this set includes i, as well as the source vertex of any message that was used (directly or indirectly) in computing the result at vertex i. We will call N_i the **contributing neighborhood** of vertex i; at convergence, the contributing neighborhood of each vertex is simply the set of all the vertices N_T. In general, the contributing neighborhoods of two adjacent vertices i and j may overlap. Furthermore, the vertices that are present in N_i but *not* in N_j always lie on i's side of the tree:

Lemma 3.2. *Let i, j be two adjacent vertices in the external junction tree T, that is, $\{i, j\} \in E_T$. Then $N_i - N_j \subseteq N_{T_{i,j}}$.*

Proof. Suppose that $k \in N_i$ and $k \notin N_{T_{i,j}}$, that is, k is in the contributing neighborhood of i and lies on j's side of the tree. Then $k = j$ or k is the source vertex of some message that was used (directly or indirectly) to compute the optimization message $d_{j\to i}$. In either case, k must have been in the contributing neighborhood at vertex j sometime in the past, right before the message $d_{j\to i}$ was computed. Since the contributing neighborhoods cannot shrink over time, $k \in N_j$ at this point in the execution of the algorithm. This proves $N_i - N_{T_{i,j}} \subseteq N_j$, which is equivalent to the statement in the lemma. □

In Section 3.4.2, we have briefly mentioned that root pointers determined by the OCA algorithm at two adjacent vertices do not conflict. We formalize the statement below:

Lemma 3.3. *Let i, j be two adjacent vertices in the external junction tree T, that is, $\{i, j\} \in E_T$. Then it is never the case that up$(i) = j$ and up$(j) = i$.*

Proof. Recall that i is determined to be the root if it minimizes $H_{\tilde{p}_r}(\mathbf{X})$ among all the roots $r \in N_i$; otherwise, up(i) points towards the best root among N_i. Consider the candidate roots in $N_i \cup N_j$. The best root r among $N_i \cup N_j$ can fall into one of three sets:

1. $r \in N_i \cap N_j$: In this case, up(i) and up(j) both point towards the same root.

2. $r \in N_i - N_j$: By Lemma 3.2, the best root among $N_i \cup N_j$ is in $N_{T_{i,j}}$, hence $\text{up}(i) \neq j$.
3. $r \in N_j - N_i$: By Lemma 3.2, the best root among $N_i \cup N_j$ in in $N_{T_{j,i}}$, hence $\text{up}(j) \neq i$.

□

We are now ready to prove Theorem 3.4:

Proof of Theorem 3.4. Using Lemma 3.3, it is easy to establish that the subtrees traced by the pointers $\text{up}(i)$ at each vertex i form a partition of the external junction tree: since two adjacent vertices i and j never point towards each other, each edge $\{i, j\} \in E_T$ of the external junction tree is associated with exactly one direction if either $\text{up}(i) = j$ or $\text{up}(j) = i$ and with no direction otherwise. Suppose that we omit the edges from T that are not associated with any direction. This leaves us with a directed forest, where each vertex has a unique parent or is a root.

To establish the second part of the theorem, let T' denote one subtree of the directed forest constructed above. It is sufficient to show that for each $i \in N_{T'}$ other than the root, $\text{up}(i)$ points in the correct direction; in other words, the best possible root among $N_{T'}$ does not lie at i or downstream of i. This fact can be proved by induction, starting from the leaves of the subtree towards the root (excluding the root itself). In the base case when i is a leaf, $\text{up}(i)$ points in the right direction, because $\text{up}(i)$ points to the best root in N_i. In the inductive case, when i is not a leaf and not a root, $\text{up}(i)$ points to a better root than i. Furthermore, by inductive hypothesis, the best possible root does not lie at k or downstream of k for $k \in N_{T'}(i) \setminus \text{up}(i)$. Thus, the best possible root does not lie at i or downstream of i. □

Chapter 4

Simultaneous localization and tracking for camera networks

Camera networks are perhaps the most common type of sensor network. These networks are ubiquitous in a variety of real-world applications including surveillance, intelligent environments and scientific remote monitoring. In most applications, camera network data is only useful if we know from where the images were captured, that is, the real world locations and orientations of the cameras. In the previous chapter, we described an approach, simultaneous localization and tracking (SLAT), that lets the cameras solve this calibration task automatically, by tracking a moving object. An effective solution of the SLAT problem leads to a very simple sensor network deployment procedure: sensors are placed throughout the environment at unknown locations, then, as an object (such as a person) moves throughout the environment following an unknown trajectory, the network automatically calibrates itself, up to a global translation and rotation. In the previous chapter, we demonstrated that a camera network can be localized by solving the SLAT problem in a distributed manner, but we did not elaborate on the modeling aspects of the problem. Indeed, if the SLAT model is Gaussian, filtering can be performed by simple matrix operations and by efficient methods, such as the Boyen–Koller algorithm. Unfortunately, the camera calibration problem has non-linearities (for example, due to periodicity in the angles) that cannot be directly represented by Gaussian distributions. This chapter presents two techniques that enable us to represent the complex distributions in the SLAT problem effectively using a single Gaussian. These techniques lead to very precise SLAT solutions in practice and require only a minimal overlap between the cameras.

4.1 Simultaneous localization and tracking

Simultaneous localization and tracking is a recent problem in wireless sensor networks (Rahimi et al., 2004; Funiak et al., 2006b; Taylor et al., 2006). In SLAT, the goal is to recover the locations of the sensors,

by observing a moving object. In this chapter, we focus on camera networks, where each sensor is a camera with known intrinsic parameters. The camera network simultaneously performs two tasks:

1. **Localization**: The camera network needs to estimate the **pose** (location and orientation) of each camera. The cameras can use an estimate of the object position, together with the observations (the recorded images), to relate one camera pose to another.

2. **Tracking**: The camera network needs to estimate the position of the object as it moves through space. To track the object, the cameras can use the current estimates of their poses, together with the observations.

In general, the pose of a camera can be represented by six parameters: three position parameters x, y, z, and three angles (for example, roll, pitch, yaw). This chapter focuses on recovering three of these parameters: the (x, y) location of the camera and an angle θ, that is, the rotation around the z-axis; the remaining parameters (such as pitch and height) are assumed to be known. We call (x, y, θ) the **absolute parameterization** of a camera's pose. This parameterization can represent a wide range of camera poses, including downward-facing cameras attached to a ceiling and wall-mounted cameras at known heights.

We assume that the tracked object maintains an (approximately) known height throughout its motion, so that its location can be characterized by (x, y) coordinates. We assume very little about the motion of this object except smoothness—the object can stop, change direction, or change speed. For example, the moving object could be a visually distinct marker carried by a person. The images observed by the cameras are governed by the perspective projection, and are therefore highly nonlinear both in the camera's pose parameters and the object's location.

4.2 Dynamic probabilistic SLAT

Cameras provide noisy observations about possible locations of the moving object, and there may be times when the object is not visible by any camera. Thus, we describe the motion and the object and the measurements with a probabilistic model. We then formulate SLAT as a probabilistic inference task, where where we maintain a joint distribution over possible object locations and poses of all cameras, given the images.

4.2.1 Model

We model the SLAT problem using a linear dynamical system, similarly to (Rahimi et al., 2004). The variables of this system are the location and the velocity of the object at each time step, $M^{(t)}$, and for each camera a, the pose of the camera L_a.

The motion of the object is modeled using a Brownian motion:

$$M^{(t)} = \begin{bmatrix} M_x^{(t)} \\ M_y^{(t)} \\ M_{\dot{x}}^{(t)} \\ M_{\dot{y}}^{(t)} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} M_x^{(t-1)} \\ M_y^{(t-1)} \\ M_{\dot{x}}^{(t-1)} \\ M_{\dot{y}}^{(t-1)} \end{bmatrix} + \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_{\dot{x}} \\ \epsilon_{\dot{y}} \end{bmatrix}, \tag{4.1}$$

where $(M_x^{(t)}, M_y^{(t)})$ is the object's position, $(M_{\dot{x}}^{(t)}, M_{\dot{y}}^{(t)})$ is its velocity, and ϵ is a white (zero-mean) Gaussian noise variable giving additive noise in the position and the velocity. This model assumes little about the motion of the object except smoothness. It is a linear-Gaussian model, because $M^{(t)}$ is a linear function of $M^{(t-1)}$ and some additive Gaussian noise. We will denote the matrix in (4.1) as F; the object motion can be thus written as $M^{(t)} = FM^{(t-1)} + \epsilon$. This means that we can represent the **motion model** $p(m^{(t)} \mid m^{(t-1)})$ using a compact parametric form.

When the object appears in the image of camera a, an observation is generated which is represented by a point, $\overline{z} = (\overline{z}_x, \overline{z}_y)$, in the image coordinates of that camera. This observation depends upon the camera's pose L_a and the object's state $M^{(t)}$ via

$$\begin{bmatrix} Z_x \\ Z_y \end{bmatrix} = g(L_a, M^{(t)}) + \begin{bmatrix} \delta_{x,a}^{(t)} \\ \delta_{y,a}^{(t)} \end{bmatrix}, \tag{4.2}$$

where g is the (nonlinear) projective transformation for camera a and δ are white Gaussian noise variables with a small standard deviation (for example, 3 pixels). For instance, when an overhead (downward-facing) camera with known focal length, f, located at (L_x, L_y) rotated at an angle of θ observes an object located at $(M_x^{(t)}, M_y^{(t)})$ and a known height offset h, the observation is given by

$$\begin{bmatrix} Z_x \\ Z_y \end{bmatrix} = \frac{f}{h} R_{-\theta} \begin{bmatrix} M_x^{(t)} - L_x \\ M_y^{(t)} - L_y \end{bmatrix} + \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix}, \tag{4.3}$$

where $R_{-\theta}$ represents a clockwise rotation by θ. This measurement equation is not linear-Gaussian; therefore, we cannot represent the **observation model** $p(z_a^{(t)} \mid l_a, m^{(t)})$ exactly using linear-Gaussian parameters. Our approach, described below, is to use linearization to find a good linear-Gaussian approximation to the observation model.

To complete our definition of the probability model, we must specify the prior distribution over the object location at the first time step, $p(m^{(0)})$, and the poses of the cameras $p(l_a)$. Our observations give us only relative information, so any translation or rotation of the coordinate frame is equally reasonable. To resolve the coordinate system, we initialize the prior of the first camera that observes the object to a point mass at the origin and set its orientation to zero. The remaining priors (over the object location and the other cameras' parameters), are "uniform," represented by a Gaussian with a large variance (Cowell et al., 1999). The camera poses and the object locations are independent in the first time step; the prior

distribution over the system state in the first time step is the product of the marginals:

$$p(\mathbf{l}, m^{(0)}) = \prod_a p(l_a) \times p(m^{(0)}).$$

4.2.2 Exact filtering for SLAT

Given the model, described in the previous section, our goal is to estimate the camera locations and object state from the object observations $\overline{\mathbf{z}}^{(t)}$, made at each time step t. In Section 3.1.2, we described a general technique, **filtering**, that computes the **posterior distribution** at each time t. Recall that the posterior distribution is the conditional distribution over the current state of the system (that is, the object location and camera poses), given all observations made so far:

$$p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0)}, \overline{\mathbf{z}}^{(1)}, \ldots, \overline{\mathbf{z}}^{(t)}). \tag{4.4}$$

As described in Section 3.1.2, filtering can be performed recursively, by computing the **prior distribution** at time $t+1$, $p(\mathbf{l}, m^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)})$, from the prior distribution at time t. The filtering updates were described in the context of the general dynamic Bayesian networks; for the specific SLAT problem, the procedure simplifies substantially. We will denote the state of the filter at each stage of the computation as the **belief state**. Thus, for example, at the beginning of the procedure before any updates have been performed, the belief state of the filter is the prior distribution at time step 0, that is, the initial prior.

Recall that the filtering update can be viewed in terms of a three step process:

1. **Estimation**: In this step, we condition on the observations of the current time step by computing

$$\begin{aligned} p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) &\propto p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) \times p(\overline{\mathbf{z}}^{(t)} \mid \mathbf{l}, m^{(t)}), & (4.5)\\ &= p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) \times \prod_a p(\overline{z}_a^{(t)} \mid l_a, m^{(t)}). & (4.6) \end{aligned}$$

   The first term on the right hand side of (4.5) is the prior distribution at time t, and the second term is the likelihood of the current observations. In (4.6), we have used the assumption that observations are conditionally independent given the position of the object and the pose of the camera that made the observation. This fact allows us to decompose the likelihood into a product of likelihoods, one per object observation: $\overline{z}_a^{(t)}$ is the ath observation, and l_a is the pose of the camera that received the observation. Note that each observation depends upon the location of the object and the pose of the observing camera—not upon the joint state vector. To summarize, estimation is accomplished by multiplying the prior distribution by a likelihood for each observation (and then renormalizing).

2. **Prediction**: In this step we augment the belief state with the new object state variable by computing

$$p(\mathbf{l}, m^{(t)}, m^{(t+1)} \mid \overline{\mathbf{z}}^{(0:t)}) = p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)}) \times p(m^{(t+1)} \mid m^{(t)}) \tag{4.7}$$

The first term on the right hand side is the result of estimation, and the second term is the object's motion model from (4.1).

3. **Roll-up**: In this step we marginalize out the object's state variable from the current time step by computing

$$p(\mathbf{1}, m^{(t+1)} \,|\, \overline{\mathbf{z}}^{(0:t)}) = \int p(\mathbf{1}, m^{(t)}, m^{(t+1)} \,|\, \overline{\mathbf{z}}^{(0:t)}) \; \mathrm{d}m^{(t)}. \tag{4.8}$$

   The first term on the right hand side is the result of prediction, and the left hand side is the prior distribution at the next time step.

Thus, the exact filtering for SLAT is a sequence of estimation steps, interleaved with the prediction and roll-up steps. In estimation, the joint estimate of the camera poses and the object state is improved, by conditioning on the object observations. In the prediction/roll-up step, the distribution over the object state is diffused, incorporating the uncertainty in the motion model.

4.3 Addressing nonlinearities

The object location and the poses of the cameras are continuous variables in our model. Unfortunately, representing general distributions over continuous variables is a very challenging task, and most representations lead to intractable inference. If the model is Gaussian, however, the filtering operations can be implemented by simple matrix operations in the Kalman Filter (Kalman, 1960). In our setting, the model contains a nonlinear component (the projection function g in the observation model (4.2)); therefore, we cannot implement filtering exactly. Instead, we approximate the posterior distribution with a multivariate Gaussian distribution $\tilde{p}(\mathbf{1}, m^{(t)} \,|\, \overline{\mathbf{z}}^{(0:t)})$. In this section, we will describe two techniques to ensure that $\tilde{p}$ is an accurate approximation of the exact distribution p.

4.3.1 Gaussian representations in absolute parameters

We begin by considering the simple problem of approximating the posterior distribution over one camera, given that the camera has made a single observation: $p(l_a \,|\, \overline{z}_a)$; solving this problem will lay the foundation for the multi-camera case. In order to study the Gaussian approximations, we first need to specify how the camera pose is represented in our filter (thus far, we have referred to the pose of camera a abstractly as L_a). One possible representation for this pose uses the absolute parameters (x, y, θ), that is, the camera pose is represented as the camera center (x, y) and the camera orientation θ. This parameterization is illustrated in Figure 4.1.

While conceptually simple, the absolute parameterization does not permit an accurate Gaussian approximation. To demonstrate this fact, we need one result about approximating general distributions with Gaussians:

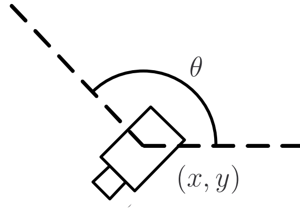


Figure 4.1: Standard parameterization in terms of camera center (x, y) and orientation θ.

Theorem 4.1. *Let $p(\mathbf{x})$ be an arbitrary distribution over a set of real-valued variables $\mathbf{X}$. Then the distribution $\tilde{p}$ that minimizes the KL divergence $D(p \,\|\, q)$ over the family of multivariate Gaussian distributions q is the one where*

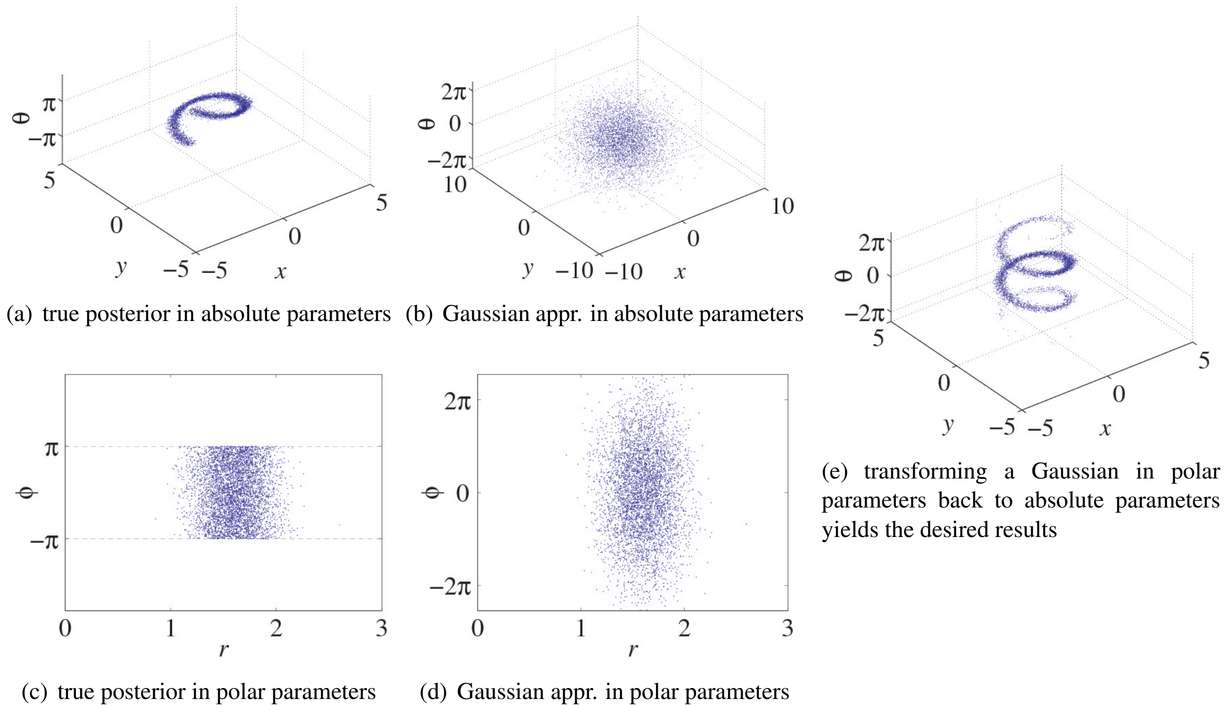
$$\begin{aligned}\mathbb{E}_{\tilde{p}}[\mathbf{X}] &= \mathbb{E}_p[\mathbf{X}] \\ \mathbb{E}_{\tilde{p}}[\mathbf{X}\mathbf{X}^\top] &= \mathbb{E}_p[\mathbf{X}\mathbf{X}^\top].\end{aligned}$$

Thus, in order to find the best Gaussian approximation in the sense of minimizing the KL divergence (Definition 2.8), we simply need to compute the first two moments (the mean and the covariance) of the true distribution p and use these moments in our approximation. Because of this property, the process of finding the best Gaussian approximation is also called **moment matching**. For a proof of Theorem 4.1, see (Lerner, 2002, Proof of Theorem 3.8).

To see how Theorem 4.1 applies to absolute parameterization of the camera pose, suppose that a camera with an unknown pose observes the object at a known position. Given the heights of the object and camera are known, we can estimate the distance from the camera to the object using a simple inverse projection. Unfortunately, we cannot recover the orientation of the camera θ, as the camera could be anywhere in a ring around the object's location. Figure 4.2(a) illustrates this phenomenon by visualizing the true posterior distribution $p(l_a \,|\, \overline{z}_a)$ over the possible camera poses in absolute coordinates given an observation of an object with known location. This ring-like distribution is highly non-Gaussian, and if we try to approximate it with a Gaussian, the problem structure is lost, as shown in Figure 4.2(b).[1] In particular, while the approximation correctly captures the means and the variances of the parameters, it has the highest density in the center of the ring, where the camera is very unlikely to be located according to the true posterior distribution. Because the approximation is so poor, applying the Kalman filter to solving the SLAT problem fails when the camera poses are represented in absolute parameters, as shown in Figure 4.5(b).

[1]Placing Gaussian distributions over angular variables requires some care because of periodicity. In our convention, a Gaussian-distributed angle Θ with mean μ and variance σ^2 is distributed so that for all $-\pi \leq \theta_0 \leq \theta_1 < \pi$,

$$\Pr \theta_0 \leq \Theta \leq \theta_1 = \sum_{k=-\infty}^{\infty} \int_{\theta_0+2k\pi}^{\theta_1+2k\pi} \mathcal{N}\left(\theta; \mu, \sigma^2\right) \, \mathrm{d}\theta.$$



(a) true posterior in absolute parameters (b) Gaussian appr. in absolute parameters

(c) true posterior in polar parameters (d) Gaussian appr. in polar parameters

(e) transforming a Gaussian in polar parameters back to absolute parameters yields the desired results

Figure 4.2: An example demonstrating the folly of Gaussian representations in absolute parameters, and the improvement obtained with relative parameters. (a) The posterior distribution of a camera's pose in absolute parameter space, given that it has observed the object at the origin. The distribution forms a spiral in the (x, y, θ) space and a ring in the (x, y) space. (b) The best Gaussian approximation to this posterior; note the bad approximation. In contrast, when expressed in polar coordinates (c), the posterior can be effectively approximated with a single Gaussian (d). By transforming this distribution back to the absolute parameters (e), we can verify that we have obtained an accurate approximation of the true posterior in the original posterior distribution in (a).

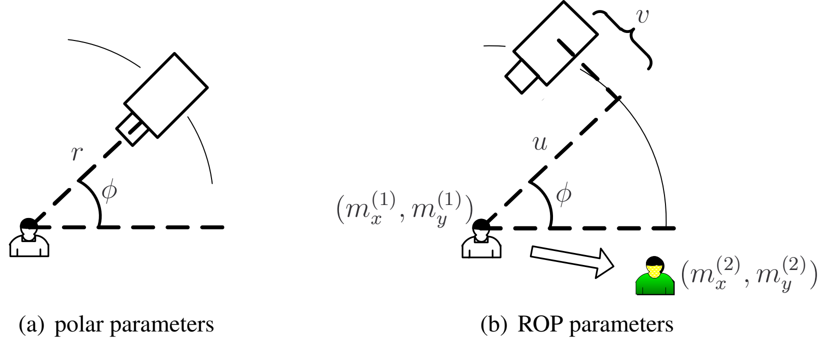
4.3.2 Relative over-parameterization

One approach for representing such ring-like distributions is to use a mixture of Gaussians (Ihler et al., 2004). In mixture of Gaussians, we represent the posterior distribution as a weighted sum of Gaussian distributions:

$$p(\mathbf{l}, m^{(t)} \mid \bar{\mathbf{z}}^{(0:t)}) \approx \sum_i w_i f_i(\mathbf{l}, m^{(t)}),$$

where each f_i is a multivariate Gaussian distribution with some mean μ_i and covariance Σ_i. Unfortunately, computations with mixtures of Gaussians are significantly more costly, losing the simplicity of the Kalman filter approach; typically an exponential number of mixture components are required to represent the pose of multiple cameras simultaneously. We now present a novel, simple reparameterization of the problem that allows us to represent these complex distributions with a single Gaussian.

Our reparameterization is based on a simple intuition. Figure 4.2(b) shows that a Gaussian in absolute



(a) polar parameters (b) ROP parameters

Figure 4.3: Two relative parameterizations that represent the pose of a camera. (a) Polar parameterization in terms of the angle ϕ and distance r from some fixed point. (b) ROP parameterization that expresses the camera pose as a composition of a translation and a rotation about a hypothetical location (m_x, m_y) of the object when first observed. At $t = 1$, the camera makes its first observation of the tracked object (person), and represents its distribution in terms of $m_x^{(1)}, m_y^{(1)}, u, v$, and ϕ. $(m_x^{(1)}, m_y^{(1)})$ is the unknown location of the person at $t = 1$, ϕ is the camera orientation, u represents the distance of the camera's image plane from the person, and v represents the lateral offset (if v were equal to 0, the camera would observe the person head on). If we vary ϕ from $-\pi$ to π, the camera traces a circle around $(m_x^{(1)}, m_y^{(1)})$. The object location at $t = 1$, $(m_x^{(1)}, m_y^{(1)})$ remains a part of the camera's belief state even after the object has moved to a different location $(m_x^{(2)}, m_y^{(2)})$ at the next time step.

parameters cannot represent the ring structure of the position variables in Figure 4.2(a). Nevertheless, this structure can be represented well with a Gaussian in *polar coordinates* (r, ϕ), Figure 4.3(a), where the origin corresponds to the observed object's true location, r is the distance to the camera's position, and the angle ϕ describes both the orientation of the camera and its orientation with respect to the object, since the camera must be looking inward toward the object. The ring structure in the polar coordinates is shown in Figure 4.2(c). A Gaussian with a small variance for r and a high variance in ϕ (to represent a uniform distribution) would provide a good approximation, see Figure 4.2(d). Comparing Figure 4.2(e) to the exact posterior in Figure 4.2(a), we see that we have obtained a sensible approximation to the true distribution of the camera pose.

This intuition has two problems that we must correct. First, the object location—the origin of our polar coordinate system—is not known with certainty when the object is observed; to remedy this, we can add its position (m_x, m_y) position to the pose variables, so that they too can be estimated from observations. Second, the camera does not necessarily observe the object head–on, but at an angle according to the orientation of the camera. To correct this, we can substitute for the radius r a pair of parameters, u and v, which describe the distance from the object to its projection on the camera's image plane (u), and the distance from this projection to the camera's center (v). Thus, our new parameterization of a camera's pose is given by (m_x, m_y, u, v, ϕ), as illustrated in Figure 4.3(b). Note that this representation has more parameters than is strictly necessary if we were to disregard the need for accurate Gaussian approximations; hence we will call the representation in Figure 4.3(b) as the **relative over-parameterization** (ROP)

of the camera pose.

Using the ROP representation in filtering requires two small changes to the model. First, the observation model (4.2) must be expressed in terms of (m_x, m_y, u, v, ϕ), which requires the projection operation $g(l, m)$ to first perform a transformation of the ROP camera pose l into absolute parameters, (x, y, θ), and then apply the standard projection. From the definition of ROP, the transformation into the absolute parameters is simply

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} m_x \\ m_y \end{bmatrix} + R_\phi \begin{bmatrix} u \\ v \end{bmatrix}, \qquad \theta = \phi + \frac{\pi}{2}, \tag{4.9}$$

where R_ϕ represents the rotation by ϕ. Thus, combining (4.9) with the observation model for the overhead camera (4.3), we obtain the model

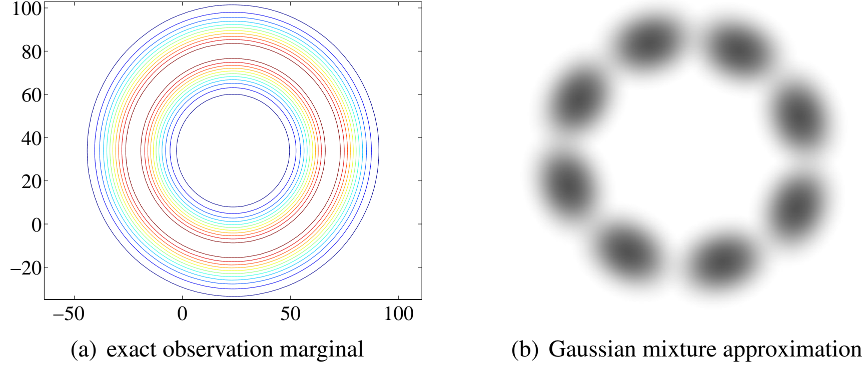
$$\begin{bmatrix} Z_x \\ Z_y \end{bmatrix} = \frac{f}{h} \left(R_{-\phi-\frac{\pi}{2}} \begin{bmatrix} M_x^{(t)} - M_x \\ M_y^{(t)} - M_y \end{bmatrix} - R_{-\frac{\pi}{2}} \begin{bmatrix} U \\ V \end{bmatrix} \right) + \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix}. \tag{4.10}$$

Second, the prior over camera poses must also be converted to the ROP representation. This prior is similar to the one in absolute coordinates: if the camera first observes the object at time t, the prior over the coordinates (U, V, ϕ) is uniform, and the prior over (M_x, M_y) is defined such that these variables are exactly equal to the (unknown) object position $(M_x^{(t)}, M_y^{(t)})$. This conversion preserves the strong correlations among the cameras that arise when the person is seen by multiple cameras at once: the cameras will be correlated through their values of (M_x, M_y).

4.3.3 Hybrid conditional linearization

Recall that our observation model in (4.2) includes a projection operation that is highly nonlinear. This makes it impossible to directly apply the Kalman filter to SLAT, because its linear-Gaussian assumptions are violated. We have seen that the ROP parameterization correctly captures the structure of the posterior distribution with a simple Gaussian. However, even with our ROP representation, the observation model is not linear in its inputs: For example, the observation model in (4.10) contains the state-dependent rotation matrix $R_{\phi-\frac{\pi}{2}}$ and is thus not a linear function of $L, M^{(t)}$. Therefore, in the estimation step in (4.6), when we multiply the prior with the likelihood of the observation, the resulting distribution is not Gaussian.

A standard approach to applying the Kalman filter to nonlinear systems is to adopt a strategy for **linearization**, which chooses a linear approximation to the observation model in the region that has highest probability according the prior distribution. Suppose that we have a Gaussian approximation of the belief state $p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$. We wish to condition this belief state on a nonlinear observation $\overline{z}_a$ made by camera a. It is sufficient for us to consider the simpler problem of computing a Gaussian approximation to $p(l_a, m^{(t)} \mid \overline{z}_a)$ from $p(l_a, m^{(t)})$; this allows us to focus on the state of the object and the camera making



(a) exact observation marginal (b) Gaussian mixture approximation

Figure 4.4: The exact observation marginal and its approximation with a Gaussian mixture. (a) The marginal of the exact distribution $p(l_a, m^{(t)}, z_a)$; this marginal forms a ring in the Z_a space. (b) The mixture Gaussian approximation.

the observation (we will address the complete problem in the next section). In linearization, we compute this Gaussian approximation by first approximating the joint over the hidden state and the observation $p(l_a, m^{(t)}, z_a)$, and then instantiating the observation $Z_a = \overline{z}_a$ using exact Gaussian conditioning.

The most sophisticated techniques for linearization are based on numerical integration, including Gaussian Quadrature and Exact Monomials (Lerner, 2002, Section 6), which include as a special case the Unscented Kalman filter (UKF) (Julier and Uhlmann, 1997; Wan and Van Der Merwe, 2002). Unlike the extended Kalman filter, which relies on a first-order Taylor expansion of the observation function g, numerical integration methods invoke the observation function directly and are far more accurate. The joint distribution $p(l_a, m^{(t)}, z_a)$ is approximated as follows. First, we select some small number of **integration points** in the $(L_a, M^{(t)})$ space to characterize the prior $p(l_a, m^{(t)})$; these points typically include the mean and points along a confidence ellipse to characterize the uncertainty in the prior. Then we evaluate the nonlinear observation function $g(l_a, m^{(t)})$ for each integration point to compute their images. The desired Gaussian approximation to the joint $p(l_a, m^{(t)}, z_a)$ is then computed by estimating its mean and covariance from the integration points and their images. In general, numerical integration techniques provide formal guarantees when the function g is a polynomial of bounded degree.

Unfortunately, as shown in Figure 4.5(c), the standard numerical integration methods do not provide effective linearization in the SLAT problem (here, we used the Exact Monomials method of degree 5). The main cause of this problem is the periodicity of the angle ϕ, which represents the orientation of the camera. This periodicity in the projection function cannot be approximated well by a polynomial of bounded degree.

We address this problem with an approach we call **hybrid linearization**. By fixing the value of the angle ϕ in the projection operator, the periodicity problem is eliminated and our integration problem can be approximated well with numerical integration methods. Building on this idea, we redefine our integration

problem by selecting a number of integration points ϕ_i for the angle, and then we use numerical integration to compute a Gaussian approximation of $p(l_a, m^{(t)}, z_a \mid \phi_i)$ for each ϕ_i (note that ϕ_i is a component of the camera pose l_a). The integration points for ϕ_i are evenly spaced in the interval $[\bar{\phi} - \alpha, \bar{\phi} + \alpha]$, where $\bar{\phi}$ is the prior mean over the angle, and $\alpha = \min(3\sigma_\phi, \pi)$, where σ_ϕ is the prior standard deviation over the angle. The approximation of our joint distribution is then given by:

$$p(l_a, m^{(t)}, z_a) \approx \frac{\sum_i p(\phi_i)\tilde{p}(l_a, m^{(t)}, z_a \mid \phi_i)}{\sum_i p(\phi_i)}. \tag{4.11}$$

In this equation, each $\tilde{p}(l_a, m^{(t)}, z_a \mid \phi_i)$ is a Gaussian, making $p(l_a, m^{(t)}, z_a)$ a mixture of Gaussians, as shown in Figure 4.4. In order to approximate this mixture as a single Gaussian, we use Theorem 4.1 to find the optimal Gaussian approximation by moment matching (Lerner, 2002, Theorem 3.8). Unlike the mixture-Gaussian representations of the complete distribution $p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$ (Ihler et al., 2004), which may require an exponential number of mixture components, our Gaussian mixture is applied to the state of a single camera and the object, and is constructed only during linearization. Therefore, the simplicity of the Kalman Filter is preserved.

The linearization method described thus far improves the quality of the SLAT results, but these results are still not satisfactory. The reason for this is the mixture of Gaussians in (4.11), illustrated in Figure 4.4(b), is often a complex, multi-modal distribution that cannot be approximated well by a single Gaussian. On the other hand, the distribution *after* the observation is instantiated is more focused and better approximated by a single Gaussian. This leads us to our **hybrid conditional linearization** technique, where the observation is instantiated in each mixture component:

$$p(l_a, m^{(t)}, \overline{z}_a) \approx \frac{\sum_i p(\phi_i)\tilde{p}(l_a, m^{(t)}, \overline{z}_a \mid \phi_i)}{\sum_i p(\phi_i)}. \tag{4.12}$$

Note that the i-th mixture component can be written as

$$\tilde{p}(l_a, m^{(t)}, \overline{z}_a \mid \phi_i) = \tilde{p}(l_a, m^{(t)} \mid \overline{z}_a, \phi_i)\tilde{p}(\overline{z}_a \mid \phi_i).$$

Thus, each mixture component i is weighted by the prior density at the integration point ϕ_i, as well as the likelihood of the observation $\overline{z}_a$ conditioned on the integration point. The mixture (4.12) is typically much closer to a Gaussian than the one in (4.11); thus, when we approximate it by a Gaussian with moment matching, the approximation is precise.

While hybrid conditional linearization is very effective in addressing the nonlinearities that stem from the uncertainty in the camera orientation, there is one source of nonlinearity that we have not considered yet. In side-facing cameras, the projection is singular: as the object approaches the plane that contains the focal point and is parallel to the image plane, the observation goes to infinity. Numerical integration typically operates far from the line of singularity, because the prior is sufficiently focused. However, when a camera makes its first observation, its pose prior is uniform (that is, a wide Gaussian), forcing

the numerical integration to operate in a region that cannot be characterized by polynomials of bounded degree. To address this problem, we use use a different procedure to condition on the first observation. Specifically, we express each mixture component in (4.12) as a product of conditionals using the chain rule:

$$p(l_a, m^{(t)}, \overline{z}_a, \delta_a \mid \phi_i) \propto p(l_a \mid m^{(t)}, \overline{z}_a, \delta_a, \phi_i) \times p(m^{(t)} \mid \overline{z}_a, \delta_a, \phi_i) \times p(\delta_a \mid \overline{z}_a, \phi_i). \tag{4.13}$$

Here, $p(l_a \mid m^{(t)}, \overline{z}_a, \delta_a, \phi_i)$ is deterministic and corresponds to the inverse projection function g^{-1} that computes the camera parameters (u, v) from $(m^{(t)}, \overline{z}_a, \delta_a, \phi_i)$. Furthermore, $p(m^{(t)} \mid \phi_i, \overline{z}_a, \delta_a) = p(m^{(t)})$, because the prior distribution of L_a is uninformative, and we approximate $p(\delta_a \mid \phi_i, \overline{z}_a)$ as a normal distribution $\mathcal{N}\left(0, \sigma_\delta^2 I\right)$. The Gaussian approximation to (4.13) has a redundant component δ_a; this component is removed using exact Gaussian marginalization before the estimate is incorporated in the mixture (4.12).

4.3.4 Approximate Kalman filter

So far, we have focused on reasoning about the distribution over the object and a single camera pose. Once we know how to approximate the posterior distribution over a single camera pose accurately, the remaining filtering operations become very simple. Our filtering algorithm conditions the belief state on the observations one-by-one, each time approximating the posterior with a Gaussian. Periodically, the state is advanced to the next time step using exact prediction and roll-up.

Conditioning the belief state on an observation is straightforward. Similarly to the exact estimation step in (4.6), we condition on the observation $\overline{z}_a$ by multiplying the approximate prior distribution with the exact observation likelihood for camera a:

$$p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}, \overline{z}_a) \propto p(\mathbf{l}, m^{(t)}, \overline{z}_a \mid \overline{\mathbf{z}}^{(0:t-1)}) \approx \tilde{p}(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) \times p(\overline{z}_a \mid l_a, m^{(t)}).$$

The approximate prior distribution can be decomposed into a Gaussian marginal over L_a and $M^{(t)}$, and a conditional linear Gaussian over the poses of the remaining cameras, given L_a and $M^{(t)}$:

$$p(\mathbf{l}, m^{(t)}, \overline{z}_a \mid \overline{\mathbf{z}}^{(0:t-1)}) \approx \tilde{p}(\mathbf{l}_{\setminus a} \mid l_a, m^{(t)}, \overline{\mathbf{z}}^{(0:t-1)}) \times \tilde{p}(l_a, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}) \times p(\overline{z}_a \mid l_a, m^{(t)}), \tag{4.14}$$

where $\mathbf{l}_{\setminus a}$ is the vector of poses for all cameras except for a. This decomposition is useful, because it let us focus the Gaussian approximation on the camera making the observation. If we group the last two terms in (4.14) together, their product is approximated using the hybrid conditional linearization method, described in the previous section. The result is a Gaussian estimate over the camera pose L_a and $M^{(t)}$ that has been conditioned on all the observations so far. This estimate is now multiplied with the first term on the right hand side of (4.14) (using exact Gaussian multiplication) to obtain a Gaussian approximation to the complete posterior $p(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}, \overline{z}_a)$. Separating out the camera pose L_a is equivalent to per-

forming hybrid conditional linearization with the complete prior $\tilde{p}(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})$, but is substantially more efficient, because the complexity of numerical integration increases with the dimensionality of the prior. Overall, the running time of our approximate estimation step is comparable to the estimation step in the Extended Kalman Filter.

Given our linear Gaussian assumptions on the motion model, the prediction and the roll-up steps can be implemented exactly, in closed form. Specifically, given a Gaussian approximation $\mathcal{N}\left(\mathbf{l}, m^{(t)}; \mu, \Sigma\right)$ of the posterior distribution at time t, the prediction step multiplies this Gaussian distribution with a conditional linear Gaussian $p(m^{(t+1)} \mid m^{(t)})$, specified by the object motion model (4.1). The result is a Gaussian distribution over the augmented state $\mathbf{L}, M^{(t)}, M^{(t+1)}$. Marginalizing out $M^{(t)}$ yields the approximate prior distribution at the next time step $\mathcal{N}\left(\mathbf{l}, m^{(t+1)}; \mu', \Sigma'\right)$, with the mean vector and covariance matrix given by

$$\mu' = \begin{bmatrix} \mu_{\mathbf{L}} \\ F\mu_M \end{bmatrix} \qquad \Sigma' = \begin{bmatrix} \Sigma_{\mathbf{L},\mathbf{L}} & \Sigma_{\mathbf{L},M}F^\top \\ F\Sigma_{M,\mathbf{L}} & F\Sigma_{M,M}F^\top \end{bmatrix}. \tag{4.15}$$

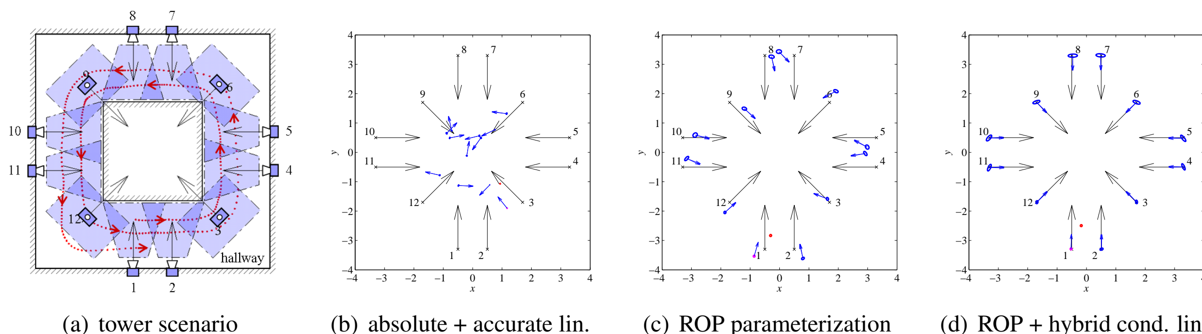
Here, μ_M selects the components of the mean vector μ that correspond to the object state $M^{(t)}$, while $\Sigma_{X,Y}$ is a submatrix of Σ that contains the covariance terms between X and Y. F is the matrix that describes the linear component of the motion model (4.1). The update (4.15) coincides with the prediction step in the standard Kalman Filter, and can be computed in in $O(N)$ time, where N is the number of cameras in the model.

To evaluate our approximate filter, we revisit the scenario from Figure 3.6(a). This scenario consists of both side-facing and overhead cameras and tests all the aspects of linearization, discussed in this section. The approximate filter obtains very precise pose estimates, as shown in Figure 4.5(d), using a single Gaussian.

4.4 The BK approximation

While effective, our approximate Kalman Filter does not scale well: the computational complexity of estimation step is quadratic in the number of cameras, for each observation being conditioned on. The reason for this inefficiency is that the algorithm maintains a quadratic number number of terms in the covariance matrix Σ, all of which are updated in the estimation step. We cannot simply make the covariance matrix sparse: maintaining correlations among the cameras is important for accurate localization and for events, such as loop closing (Paskin, 2004). In Section 3.1.3, we described a principled approximation, the B&K98 algorithm (Boyen and Koller, 1998), that periodically *projects* the exact belief into a sparser approximation.[2] In this section, we discuss the details of applying the B&K98 algorithm to our SLAT application.

[2]This projection maps one distribution to another, and is not related to the camera projection in our observation model (4.2).



(a) tower scenario (b) absolute + accurate lin. (c) ROP parameterization (d) ROP + hybrid cond. lin.

Figure 4.5: The performance of the Kalman Filter on a simulated network of cameras. The long arrows indicate the true location and orientation of the cameras, the ellipses (often small) are 95% confidence regions in the position estimates, and the two short arrows (often overlapping) are the 95% confidence intervals in the estimate of orientation. (a) Example from Figure 3.6(a): Eight side-facing and four overhead cameras (in the corners) are placed around a hallway. The dark-shaded regions represent the overlapping fields of the view. The dotted line shows the actual location of the object at each time step. (b) Nonlinearities give poor results when camera poses are represented as (x, y, θ), even with accurate linearization of the observations. (c) The ROP representation of camera poses improves results. (d) Combining the ROP representation with the hybrid conditional linearization techniques gives excellent results.

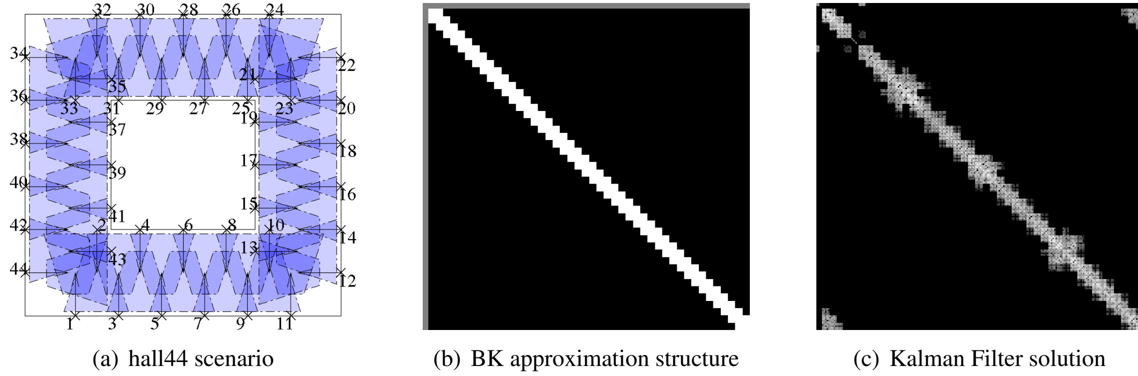
4.4.1 Selecting the BK structure

Recall from Section 3.1.3 that the B&K98 algorithm algorithm maintains its belief in the form of a decomposable model (Definition 2.3). The decomposable model approximates the prior distribution as a ratio of clique and separator marginals over some junction tree $(T, \mathbf{C})$:

$$p(\mathbf{l}, m^{(t)} \mid \bar{\mathbf{z}}^{(0:t-1)}) \approx \frac{\prod_{i \in N_T} \tilde{p}(\mathbf{l}_i, m^{(t)} \mid \bar{\mathbf{z}}^{(0:t-1)})}{\prod_{\{i,j\} \in E_T} \tilde{p}(\mathbf{l}_{i,j}, m^{(t)} \mid \bar{\mathbf{z}}^{(0:t-1)})}. \tag{4.16}$$

Here, $\mathbf{L}_i$ are the poses of cameras assigned to clique i, and $\mathbf{L}_{i,j} = \mathbf{L}_i \cap \mathbf{L}_j$ are the poses of cameras assigned to the separator. The representation (4.16) is sparse, because it only stores Gaussian distributions over small, overlapping sets of random variables. The clique marginals $\tilde{p}(\mathbf{l}_i, m^{(t)} \mid \bar{\mathbf{z}}^{(0:t-1)})$ incorporate two kinds of approximations—the results of the projection operations in the B&K98 algorithm, as well as the Gaussian approximations, described in the previous section. Here, we will focus on the former approximation.

The junction tree $(T, \mathbf{C})$ is a parameter of the B&K98 algorithm that is determined at the beginning of the experiment. At the end of Section 3.1.3, we suggested a heuristic for selecting the cliques in the SLAT application: we select the cliques to cover sets of camera that are near one another. (The cliques are selected using gross, imprecise topological information about the environment, not the precise locations of the cameras, which are determined by our approach.) Intuitively, this heuristic ensures that the strong dependencies among the cameras with overlapping fields of view are captured in our approximation,



(a) hall44 scenario (b) BK approximation structure (c) Kalman Filter solution

Figure 4.6: (a) A simulated scenario with 44 side-facing cameras, arranged along both walls in a square corridor. (b) The dependences, captured by a junction tree, where each clique is assigned the poses of adjacent cameras L_a and L_{a+1}. White indicates a camera-camera dependence; gray indicates a camera-object dependence. (c) The information matrix of a Kalman Filter solution, plotted with intensities on the log scale. Note that most of the entries in the matrix are approximately zero (shown in black). Comparing the Kalman Filter solution with the B&K98 approximation structure in (b), we see that the B&K98 approximation captures most of the non-zero entries of the Kalman Filter solution.

as shown in Figure 4.6(b). To confirm this intuition, we take a look at the information matrix of the Kalman Filter solution to SLAT. Recall from Section 2.1.2 that in Gaussian distributions, the sparsity of the information matrix corresponds to the sparsity of the Gaussian Markov network: if there is no edge between a pair of variables X_a and X_b in the graph, then the corresponding entry of the information matrix is zero. In SLAT, none of the entries are exactly zero, because of the dependences introduced by the motion model, and there are no conditional independences among the variables in the exact posterior distribution. However, most of the entries are *approximately* zero, as illustrated in Figure 4.6(c), so many conditional independences hold approximately. Indeed, we see that the approximation structure in Figure 4.6(b) captures most of the “important” edges of the exact distribution. Therefore, by projecting the distribution to the family of decomposable models with the sparsity structure in Figure 4.6(b), we do not introduce much error in the belief.

4.4.2 Incorporating nonlinear observations

Conditioning on evidence in the B&K98 algorithm is slightly different from conditioning in the Kalman Filter. Recall from Section 3.1.3 that the estimation step in the B&K98 algorithm is implemented by multiplying each observation likelihood $p(\overline{z}_a^{(t)} \mid l_a, m^{(t)})$ into some clique i in (4.16) that covers the parents of the observation, that is, $L_a \in \mathbf{L}_i$. In our case, the observation likelihood is nonlinear, so we cannot directly multiply it into the clique prior. Rather, we compute a Gaussian approximation to the likelihood

$$\tilde{p}(\overline{z}_a^{(t)} \mid l_a, m^{(t)}) = \tilde{p}(\overline{z}_a^{(t)} \mid l_a, m^{(t)}, \overline{\mathbf{z}}^{(0:t-1)}) = \frac{\tilde{p}(\overline{z}_a^{(t)}, l_a, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})}{\tilde{p}(l_a, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)})}. \tag{4.17}$$

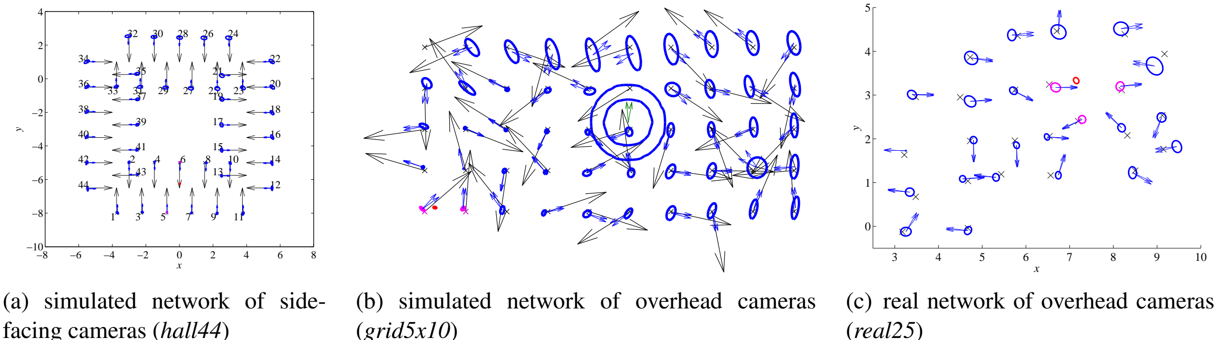
Here, in the first equality, we use the fact that $Z_a^{(t)}$ is independent of the previous observations $\mathbf{Z}^{(0:t-1)}$, given the camera pose L_a and the object state $M^{(t)}$. The ratio captures the change in the belief state after we have conditioned on the evidence $Z_a^{(t)} = \overline{z}_a^{(t)}$. We compute such approximate likelihood for each observation at the current time step, multiply the approximate likelihoods into the corresponding clique priors, and the rerun the Lauritzen–Spiegelhalter algorithm (Lauritzen and Spiegelhalter, 1988) as usual to redistribute the evidence among the cliques.

Unfortunately, as described, the procedure may not work. The reason is that our linearization procedure may increase the uncertainty in the system state $L_a, M^{(t)}$. Normally, conditioning on observations in Gaussian models only decreases the uncertainty (or keeps it the same), and the information matrices are always positive semi-definite. However, in our case, the linearization procedure creates a mixture of Gaussians; when the mixture is collapsed, some of the dimensions of the distribution may be less certain, and the information matrix of the approximate likelihood $\tilde{p}(\overline{z}_a^{(t)} \mid l_a, m^{(t)})$ may not be positive semi-definite. This fact did not cause problems in the Kalman Filter, because the observations were multiplied in one-by-one, and we ensured that the resulting belief $\tilde{p}(\mathbf{l}, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t)})$ is always a valid Gaussian distribution. However, in the B&K98 algorithm, we condition on all the observations in one time step at the same time; the deformities in the invalid likelihoods may accumulate and create an invalid belief overall.

To address this problem, we perform a correction of the approximate observation likelihoods; this correction is different from the moments correction, employed in standard linearization (Lerner, 2002, Section 6.3). Suppose that the approximate observation likelihood is $\mathcal{N}^{-1}(\eta_\ell, \Lambda_\ell, .)$ First, we take the eigenvalue decomposition of the information matrix $\Lambda_\ell = QDQ^\top$, where Q is the matrix of orthonormal eigenvectors, and D is a diagonal matrix of eigenvalues. We zero out any negative entries of D and define the new information matrix as $\Lambda'_\ell = Q \max\{D, \mathbf{0}\} Q^\top$. Since the information matrix has been adjusted, the information vector needs to be updated, too; otherwise, the likelihood information will be skewed. We update the information vector, so that the new likelihood preserves the mean of the approximate posterior distribution $\tilde{p}(l_a, m^{(t)} \mid \overline{\mathbf{z}}^{(0:t-1)}, \overline{z}_a^{(t)})$. Specifically, if the approximate prior distribution is $\mathcal{N}^{-1}(\eta_p, \Lambda_p, ,)$ and if the approximate posterior distribution has mean μ, we set the new information vector of the likelihood to

$$\eta'_\ell = (\Lambda_p + \Lambda'_\ell)\mu - \eta_p.$$

The resulting correction performs very well in practice, as demonstrated by the experimental results in Section 4.5.3.



(a) simulated network of side-facing cameras (*hall44*)

(b) simulated network of overhead cameras (*grid5x10*)

(c) real network of overhead cameras (*real25*)

Figure 4.7: Additional results from the approximate filter. As in Figure 4.5, the long arrows indicate the true locations and orientations of the camera, the ellipses are the 95% confidence regions for the estimated locations, and the small arrows indicate the 95% confidence intervals for the camera orientations. Figures (a) and (b) demonstrate very good results on simulated networks with many cameras. In (b), where the cameras are overhead, the estimates are more uncertain because the object is observed less frequently. (c) shows the results of our algorithm when run on a real camera network of twenty-five cameras.

4.5 Experimental results

4.5.1 Convergence

In addition to the smaller tower scenario in Figure 4.5, we evaluated our ROP representation and hybrid conditional linearization on several other larger simulated scenarios. We include two sample results here, Figure 4.7(a) and Figure 4.7(b). We omit the results for the absolute parameterization, because it performed very poorly in these larger scenarios.[3]

The scenario in Figure 4.7(a) contains 44 side-facing cameras, tilted down about $35°$, arranged along both walls in a square corridor; this scenario was illustrated in Figure 4.6. The object circles the loop twice. Note that all of the camera position and orientation estimates are within the estimated 95% confidence regions. Thus, we are effectively representing both the estimate and the uncertainty.

The scenario in Figure 4.7(b) consists of 50 downward-facing cameras, and the object circles the space. Again, we see that almost all of our estimates are within the 95% confidence regions. Interestingly, camera 16 in the center only sees the object once; its posterior distribution should be ring-like. Since our ROP parameterization can capture such a structure, we see that the 95% confidence region for this camera is in fact a ring.

We have also evaluated our approach on a real network of twenty-five overhead cameras. Here, a toy remote-controlled car was driven around a room carrying a colored marker, and a standard image processing algorithm was used to extract the center of the marker. Figure 4.7(c) illustrates the solution we obtain.

[3]These results are best visualized with videos, we refer the reader to `http://www.cs.cmu.edu/~sfuniak/slat`.

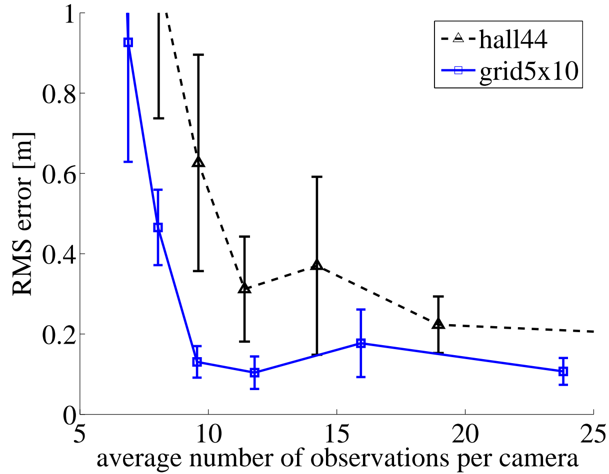


Figure 4.8: Approximation error vs. number of observations for the B&K98 algorithm with the *hall44* and *grid5x10* scenarios. The horizontal axis shows the total number of observations a camera has made. We see that with 10–12 observations per camera, we can extract the camera poses accurately.

We see that our approach generates excellent pose estimates with real data.

4.5.2 Number of observations

One of the benefits of the SLAT approach to camera calibration is its ability to recover accurate estimates with only a few observations. SLAT relies not only on observations in the regions where camera views overlap, but also on the motion model to relate one camera pose to another. In order to better understand the ability of the approach to generalize from data, we evaluate the its sensitivity to the number of observations made by the cameras. Figure 4.8 shows the accuracy of the B&K98 algorithm as we vary the number of observations made by each camera for the two large simulated scenarios. With roughly 10–12 observations per camera, the algorithm obtains accurate estimates. Only a few of these observations are made in the regions where the camera views overlap. Thus, the algorithm is very effective at generalizing from the data.

4.5.3 BK approximation and comparison with off-line optimization

As suggested by Figure 4.6, the B&K98 algorithm is well-suited for the SLAT problem. We have found that the BK approximation is excellent, and yields almost no approximation error. In Figure 4.9, we show the result of solving several SLAT scenarios using a BK approximation, for different clique sizes (for example, size 2 corresponds to maintaining a maximum of two cameras in each clique, in addition to the object state). With as few as three cameras in each clique, the algorithm attains the same level of accuracy as the Kalman Filter.

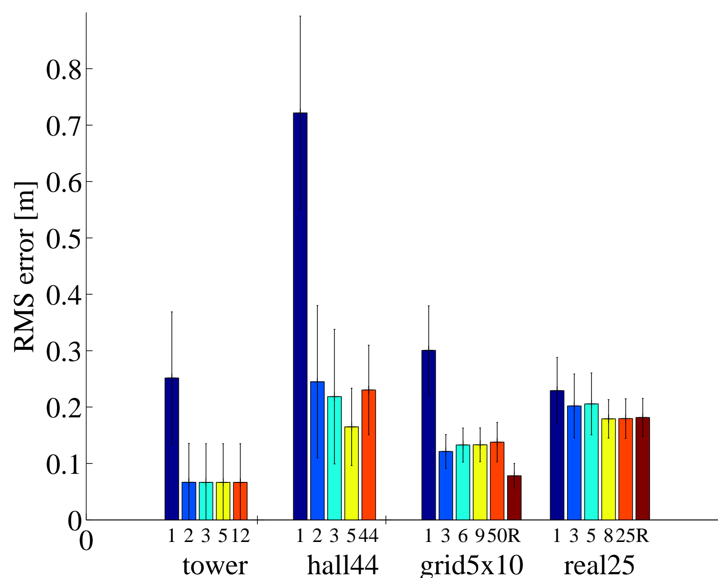


Figure 4.9: Approximation error vs. clique size with the Boyen & Koller (BK) algorithm. The horizontal axis shows the four scenarios discussed in the previous sections. The bars correspond to solutions with increasing numbers of variables per clique. The fifth bar in each group corresponds to the solution of a Kalman Filter. We see that often, a very simple approximation suffices to obtain accurate results. The last bars in the *grid5x10* and *real25* examples (labeled 'R') show the performance of the algorithm (Rahimi et al., 2004), running on our data set with the same choice of model parameters. We see that our online approach performs comparably to the offline approach (Rahimi et al., 2004).

We also compared the performance of our approach to the calibration algorithm in (Rahimi et al., 2004) on our scenarios with overhead cameras (at the time of writing, the implementation of their algorithm did not support side-facing cameras). We see that in these scenarios, our online approach and the offline optimization approach of Rahimi et al. (2004) provide solutions of comparable quality. Thus, even though our algorithm introduces approximations at every time step, it is still able to recover a solution that is close to the mode of the *exact* posterior.

4.6 Related work

SLAT is related to a problem in mobile robotics called **simultaneous localization and mapping** (SLAM) (Smith and Cheeseman, 1986; Thrun et al., 2004; Paskin, 2003). In a popular formulation of SLAM, a mobile robot observes landmarks (such as trees or corners of buildings) and from these observations, its odometry, and its control signals, the robot must jointly estimate its location and the positions of the landmarks. We can view SLAT as a SLAM problem where the cameras play the role of landmarks, and the moving object plays the role of the robot. The key difference is that the sensing is opposite: in SLAT, the cameras observe the object, while in SLAM, the robot observes the landmarks. In some ways, SLAT is easier than SLAM: in SLAT there is no data association problem, since there is only a single

object; in SLAM, there are many landmarks and the robot may need to reason about which is associated with each observation. In other ways, SLAT is more difficult than SLAM; in SLAM there is significant information about the motion of the robot (from its odometry and controls), whereas in SLAT we know little about the object's dynamics. Another feature which makes SLAT more challenging is that there are many variables that represent angles—and thus interact nonlinearly—whereas in SLAM there is typically only one angular variable, the orientation of the robot.

Of the SLAM literature, the SLAT problem is most related to **range-only SLAM** (Kantor and Singh, 2002; Newman and Leonard, 2003; Olson et al., 2004). In range-only SLAM, each landmark is equipped with an acoustic transponder or a radio frequency tag that allows the robot to estimate the range to the landmark. When the first observation of a landmark is made, the exact distribution over the landmark location forms a ring, similarly to SLAT, and cannot be directly represented as a Gaussian. This problem is typically addressed by triangulating the location of the landmark from several measurements, so that the location estimate is sufficiently focused before it is inserted into the state vector (Olson et al., 2004). In principle, triangulation could be also applied to SLAT, using the observations collected in the overlapping fields of view of the cameras. Unfortunately, triangulation often requires many measurements to initialize the locations accurately, and in camera networks, the overlap of cameras' fields of view may be very small. Using the ROP parameterization and hybrid conditional linearization, the camera pose can be included in the state vector from the beginning, and the filter recovers accurate estimates even with minimal overlap of the fields of view.

Aspects of SLAT are also related to work in computer vision in **multiple camera tracking and calibration** (Khan and Shah, 2003; Stauffer and Tieu, 2003), which is largely focused on overlapping camera configurations. Similarly to our approach, they assume that an object moves on a plane. Unlike in our work, however, where a single object is tracked, the work in multiple camera tracking and calibration considers multiple objects simultaneously and addresses a difficult problem of determining correspondences among multiple object tracks. Therefore, they are able to recover the full 3D pose of the camera. Our work, while only considering 2D poses, assumes a minimal overlap of the cameras, and recovers the uncertainty of the estimates.

Another popular problem in computer vision is **structure from motion** (SFM) (Tomasi and Kanade, 1992; Soatto and Perona, 1998; Nistér, 2001; Pollefeys et al., 2004). Given a sequence of images of a static scene captured by a moving camera, the goal of SFM is to recover the 3D geometry of the scene and the trajectory of the camera motion, typically by using correspondences between feature points. There are two key differences between SLAT and SFM. In SLAT, the positions from which images are obtained are not related by smooth motion of the camera, but by the layout of the camera network; this means that the overlap between different images is typically much smaller. Additionally, in SFM geometric information is extracted from large sets of feature point correspondences; in SLAT, only a single point is tracked—the moving object.

In the sensor networks community, there has been a large body of work on localizing nodes from pairwise distance estimates, c.f. (Ihler et al., 2004) for one such approach and (Whitehouse et al., 2005) for an interesting analysis of this problem. The assumptions of SLAT are weaker, since we localize nodes by simply tracking an external, uncontrolled object. A related approach that solves the camera localization problem distributedly but relies on feature point correspondences is presented in (Mantzel and Baraniuk, 2004). Perhaps the closest work in the sensor network community is that on **passive localization**, where sensors attempt to localize themselves using sound events of unknown origin (Thrun, 2006).

Rahimi et al. (2004) proposed to address the SLAT problem using an offline optimization algorithm. This approach is based upon a probabilistic model that is similar to ours, but rather than computing a complete posterior distribution over camera poses, they compute the most likely trajectory and the most likely pose for each camera with the Newton-Rhapson method. In Section 4.5.3, we show that our approach and that of Rahimi et al. (2004) provide solutions of comparable quality. However, the approach of Rahimi et al. (2004) is offline, centralized, and does not provide an explicit representation of the uncertainty in the solution. On the other hand, our algorithm is online, can be distributed (using the methods from the previous chapter), and provides uncertainty estimates that can be used for active control.

Taylor et al. (2006) considered a version of the SLAT problem for a set of wireless sensors, measuring their distances to a moving beacon. Similarly to our work, their approach addresses the problem of estimating the sensor locations using a nonlinear observation model and employs a Gaussian approximation of the belief state. However, rather than representing the complex distributions using a reparameterization of the sensor location, they use the Laplace's method that finds the mode of the posterior distribution in the absolute parameterization using the Newton-Rhapson method and then approximates the inverse covariance as the curvature of the negative log posterior at the mode. In our camera application, the Laplace method is likely to obtain overconfident estimates, as the initial mode of the posterior distribution lies somewhere on the ring around the object, and the local curvature does not accurately reflect the uncertainty in the camera location. Yet, more experimental data is needed to evaluate the merits of the Laplace's method; in combination with our ROP representation, the Laplace's method may perform well.

Since the initial publication of this work, others have built upon the ideas presented in this chapter. Djugash et al. (2008) use the ROP representation in the context of wireless sensor network localization. In order to handle the multi-modalities that arise in their problem, they maintain a separate hypothesis for each mode. In their application, the standard Extended Kalman Filter in combination with the ROP representation was sufficient for accurate results. Djugash et al. (2009) adapt the ROP representation to modeling the motion of a mobile robot. Angular uncertainty is a major source of errors in mobile robot localization, and in order to represent it accurately, a particle filter typically needs to be used. With the ROP representation for the robot pose, the simplicity of the Kalman Filter is once again preserved.

4.7 Discussion

This chapter demonstrated that large camera networks can be automatically calibrated by tracking a moving object. We presented two techniques, relative over-parameterization and hybrid conditional linearization, that enable efficient Kalman Filter solution to the SLAT problem, in spite of its complexity and nonlinearity. Our approach obtains the estimate of a camera's pose, as well as the uncertainty in the estimate. We demonstrated that the B&K98 algorithm gives an excellent approximation to the Kalman filter solution and performs comparably to an offline optimization approach (Rahimi et al., 2004). These results validate that our approximate distributed filter from the previous chapter not only performs well relative to the centralized B&K98 algorithm, but it indeed solves the SLAT problem well.

Chapter 5

Localization in large-scale modular robot ensembles

In the previous chapter, we have seen that inference in nonlinear models can be difficult, because the distribution forms a complicated shape. In some systems, reasoning is further complicated by the fact that the model forms a dense mesh over the hidden variables, which precludes one from applying methods based on junction trees. In this chapter, we consider one kind of such systems—large-scale modular robots—and the problem of estimating locations of the robot's internal components. We will show that this problem can be solved by identifying suitable hierarchical partitioning of a graphical model. By hierarchically building up an accurate solution rather than refining a poor initial estimate, our algorithm avoids difficult local optima that plague other solvers. A key component of the algorithm are once again overlay networks for aggregating summary statistics of the graphical model and the observations to the leader nodes. To construct these overlay networks and to integrate them with the rest of the algorithm, our distributed implementation leverages recent advances in declarative programming languages (Ashley-Rollman et al., 2009).

5.1 Internal localization in modular robots

As discussed earlier, a modular robot is comprised of many discrete, physically connected modules which can rearrange themselves to adapt the robot's shape to the task at hand. The shape adaptation is accomplished by a motion planner that rearranges the modules to reach a target shape. Some planners (Dewey et al., 2008) assume that each module is aware of its pose (location and orientation) in the ensemble. Furthermore, for many applications, the specific tasks performed by an individual module are dictated primarily by its position within the robot. For these reasons, a modular robot needs the ability to establish relative pose amongst its individual modules. Unlike the localization task in the previous chapter, where

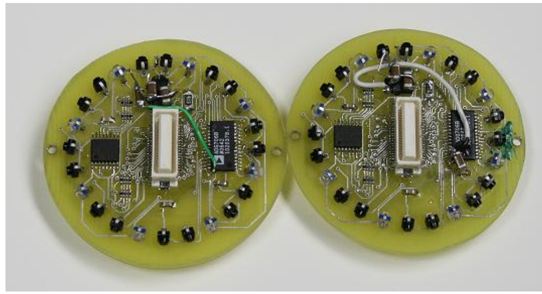


Figure 5.1: Sensor board from a module prototype. The board contains infrared transmitters and receivers, spaced evenly around the perimeter. The sensors allow the module detect when another module is in range.

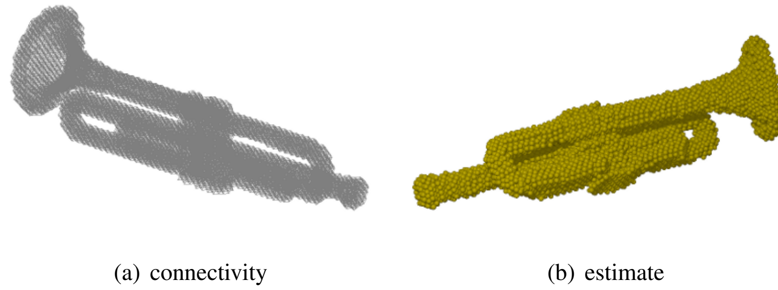
the cameras observed an external entity, the relative poses of each robotic module are estimated solely from the internal readings between the modules. Therefore, we call this task **internal localization**.

A common characteristic of the internal sensors is that they are noisy (Roufas et al., 2001), imprecise, and limited to sensing adjacent modules (that is, the modules do not have access to long distance measurements, such as global time-of-flight measurements or external beacons, to aid the localization). Figure 5.1 shows one prototype of a sensing subsystem of a module designed by the Claytronics project, a collaborative research project on programmable matter at CMU and Intel Research.[1] Each module shown has 8 infrared transmitters and 16 infrared receivers, oriented radially and spaced evenly around the circular perimeter, which allow a pair of adjacent modules to communicate and thereby detect when they are in close proximity. The sensors provide a measurement of the relative poses of the neighboring modules. Such observations are inherently uncertain: two modules may be in sensing range, but not in physical contact, or a measurement can be made when sensors are not perfectly aligned. Therefore, information from multiple sensors needs to be combined, in order to obtain an accurate solution.

In this work, a module is said to observe a nearby module if and only if it is able to receive a beacon message from the other module. The identity of the sensed module can be included in the beacon message, which greatly simplifies the localization task, since we do not need to address the data association problem. This assumption is standard in other fields, such as wireless sensor network localization. In addition, some module designs also assume strong mechanical latches (Jorgensen et al., 2004), which allows them to rely on mechanical constraints to resolve the uncertainty in alignment and orientation; we do not make this assumption here.

Internal localization poses several interesting challenges. First, similarly to the problem in the previous chapter, the relationship between measurements and the system state is nonlinear, because it includes a rotational component of the each pose. We have seen that nonlinear relationships can result in a distribution that is distinctly different from Gaussian: the shape may contain ring-like structures, and the distribution may be multi-modal. Reasoning about such distributions requires new methods that are often custom-tailored to the task at hand.

[1] http://www.cs.cmu.edu/~claytronics/



(a) connectivity (b) estimate

Figure 5.2: Connectivity graph of ensemble with 9322 nodes, and resulting estimate of module positions; the results are accurate, subject to a rotation and translation of the coordinate space.

Second, as self-reconfigurable robots scale to larger ensembles of smaller, finer grain modules, the number of modules whose pose needs to be estimated increases dramatically. For example, a simple rendering of a trumpet, shown in Figure 5.2, has over 9000 modules; experiments with motion planners often scale up to a million of modules (Ashley-Rollman et al., 2009). Not only does the number of modules increase, so does the complexity of reasoning. The complexity of the methods based on junction trees, considered in the previous chapters, increases cubicly (for Gaussians) with the size of the largest clique in the junction tree. Unfortunately, the distributions in modular robot localization form a dense mesh as shown in Figure 5.2(a), and the cliques in modular robots are very large: each clique includes an entire cross-section of the ensemble, with hundreds to thousands of modules. Therefore, junction tree methods are not applicable in this setting.

Finally, due to the limited communication capabilities of the modules and due to the scale of the ensemble, it is not feasible to collect the observations onto a single node. Instead, the algorithm needs to be decentralized to run on-board the modules. The modules themselves have very limited capabilities; for example, the current prototype of the planar Claytronics module has an ATmega1281 processor with 8kB of RAM. Thus, simple distributed methods are preferred.

5.2 Maximum-likelihood estimation for internal localization

We begin by describing the internal localization problem more formally. We formulate internal localization as a maximum-likelihood estimation in a probabilistic model. Our probabilistic model is based on a standard formulation in graph-based SLAM literature (Grisetti et al., 2007b), with refinements that are specific to modular robots.

5.2.1 Probabilistic model for internal localization

We model the modular robot as a static system. The state variables in this system are simply the poses of all the modules in the ensemble $\mathbf{L} = (L_1, \ldots, L_N)$. The pose of each module i is represented by a vector, $L_i \triangleq (L_{c,i}, L_{r,i})$, where $L_{c,i}$ is the center of the module and $L_{r,i}$ is its orientation (represented in 2D as an angle, and in 3D as a quaternion). Similarly to the SLAT problem in the previous chapter, the pose variables are fixed, and their value does not change over time. However, unlike the SLAT problem, which contained a dynamic component (the moving object), our treatment of internal localization is entirely static; our methods will only estimate a snapshot of the poses in time.

When module i is in the immediate neighborhood of module j, an observation $Z_{i,j}$ is generated at module i. The observation captures the relative pose of module j with respect to module i, but not the intensity of the readings. Specifically, the observation $Z_{i,j}$ represents the best guess of the center of module j in the frame of reference of module i, given a single measurement made at node i, as shown in Figure 5.3. The observation model penalizes the observation $Z_{i,j}$, based on how well it predicts the center of module j:

$$p(z_{i,j} \,|\, l_i, l_j) \propto \exp\left\{-\frac{1}{2}\big(f(l_i, l_j) - z_{i,j}\big)^\top \Sigma^{-1}\big(f(l_i, l_j) - z_{i,j}\big)\right\},$$

where f is a nonlinear function that transforms the center $l_{c,j}$ to the reference frame of module i. Thus, the observation model is a conditional Gaussian with mean $f(l_i, l_j)$ and covariance Σ. For simplicity, we often assume that the observations are equally uncertain in all the directions, and we take Σ to be the identity matrix. In this case, the observation model becomes

$$p(z_{i,j} \,|\, l_i, l_j) \propto \exp\left\{-\frac{1}{2}\left\|l_i \circ z_{i,j} - l_{c,j}\right\|_2^2\right\}, \tag{5.1}$$

where, $l_i \circ z_{i,j}$ denotes the observation $z_{i,j}$, transformed to the global reference frame according to the pose l_i. This model is a crude approximation to the true sensor characteristic, but it correctly captures the intuition that the readings are highest when the infrared transmitter and receiver are aligned, and the module centers are a unit distance apart. Alternatively, we could use a more accurate model that captures properties of IR transmitters and receivers, such as quadratic decay and multi-modal response, but such a refinement is not key to the methods presented in this chapter.

The model (5.1) represents the relationship between a pair of modules and the observation $Z_{i,j}$. We assume that the observations are independent, given the system state; thus, the joint observation model $p(\mathbf{z} \,|\, \mathbf{l})$ is a product of the models for individual observations. This model can be represented as a directed graph $G = (N_G, \vec{E}_G)$, where each vertex $i \in N_G$ corresponds to the pose of one module, and each edge $(i, j) \in \vec{E}_G$ corresponds to the observation $Z_{i,j}$. Instantiating the measurements $Z_{i,j} = \overline{z}_{i,j}$[2] in the

[2] Recall that we use bar to indicate that $\overline{z}_{i,j}$ is fixed assignment to the observed variable $Z_{i,j}$.

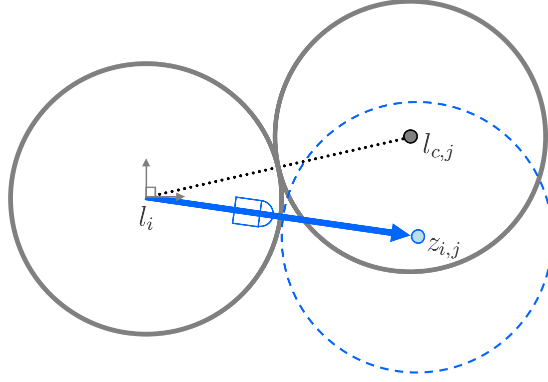


Figure 5.3: Sensor model, used in this chapter. Each observation $z_{i,j}$ represents the predicted center of module j in the frame of reference of module i. The observation model penalizes the distance between the observation and the actual center $l_{c,j}$.

observation model gives the likelihood of the joint state $\mathbf{l}$:

$$p(\overline{\mathbf{z}} \,|\, \mathbf{l}) = \prod_{(i,j)\in\vec{E}_G} p(\overline{z}_{i,j} \,|\, l_i, l_j). \tag{5.2}$$

We can write the negative log of the likelihood (5.2) equivalently as

$$-\log p(\overline{\mathbf{z}} \,|\, \mathbf{l}) = \sum_{(i,j)\in\vec{E}_G} F_{i,j}(l_i, l_j), \tag{5.3}$$

where $F_{i,j}(l_i, l_j) = \frac{1}{2}\big(f(l_i, l_j) - \overline{z}_{i,j}\big)^\top \Sigma^{-1}\big(f(l_i, l_j) - \overline{z}_{i,j}\big)$. If several observations are generated between a pair of modules (i, j), the likelihood (5.2) contains multiple terms $p(\underline{z}^k_{i,j} \,|\, l_i, l_j)$, one for each measurement $\underline{z}^k_{i,j}$. The observations do not need to be symmetric: module i may observe module j even if module j does not observe i.

Similarly to the simultaneous localization and tracking problem in the previous chapter, the poses of the modules can be only recovered up to a global translation and rotation, since all the configurations that relate by a global rigid transform are equally likely. Unlike the problem in the previous chapter, the model does not explicitly resolve this ambiguity by fixing the pose of one module through the prior. In practice, the ensemble would use external beacons to orient themselves in the global coordinate frame or orient itself with respect to a known target (such as a person), once it has performed internal localization using the methods described in this chapter. More importantly, the model does not explicitly represent the constraint that modules must not overlap; such a constraint would make inference very difficult. Instead, the model makes overlapping configurations less likely, as they would require the observations to deviate further from the predictions.

5.2.2 Maximum-likelihood estimation

In this chapter, we focus on recovering point estimates of the module poses. With our probabilistic model, solving the internal localization problem amounts to computing the maximum-likelihood estimate (MLE) of the location of all the modules, given all observations $\mathbf{Z} = \overline{\mathbf{z}}$:

$$\mathbf{l}^* = \arg\max_{\mathbf{l}} p(\overline{\mathbf{z}} \mid \mathbf{l}) = \arg\min_{\mathbf{l}} \sum_{(i,j)\in\vec{E}_G} F_{i,j}(l_i, l_j). \tag{5.4}$$

The first optimization problem above identifies a point where the likelihood $p(\overline{\mathbf{z}} \mid \mathbf{l})$ is maximized, while the second problem finds a point where the negative log-likelihood is minimized; the two problems are equivalent.

A standard approach is to solve the optimization problem (5.4) with preconditioned gradient descent.[3] Preconditioned gradient descent starts from some initial estimate $\mathbf{l}^{(0)}$, and iteratively updates the estimates as

$$\mathbf{l}^{(k+1)} = \mathbf{l}^{(k)} - \lambda \tilde{H}^{-1} \sum_{(i,j)\in\vec{E}_G} \nabla_{\mathbf{l}} F_{i,j}(l_i, l_j), \tag{5.5}$$

where $\tilde{H}$ is the **preconditioner** that speeds up the convergence by accounting for the different variances in different dimensions of the likelihood. We take $\tilde{H} = \operatorname{diag}(H)$, where H is the Hessian of the negative log-likelihood, which corresponds to the standard Jacobi preconditioner. The update (5.5) then decomposes linearly over the log-likelihoods of individual observations $\overline{z}_{i,j}$ and has a particularly simple structure: to evaluate the gradient with respect to the pose l_i, we only need to know the poses of module i and its neighbors in G:

$$\nabla_{l_i}\big(-\log p(\mathbf{z} \mid \mathbf{l})\big) = \sum_{j\in\overrightarrow{N}(i)} \nabla_{l_i} F_{i,j}(l_i, l_j) + \sum_{j\in\overleftarrow{N}(i)} \nabla_{l_i} F_{j,i}(l_i, l_j). \tag{5.6}$$

Here, $\overrightarrow{N}(i)$ is the set of outgoing neighbors of node i in G, and $\overleftarrow{N}(i)$ is the set of incoming neighbors.

Unfortunately, preconditioned gradient descent performs very poorly in our problem. The reason is that the system is highly non-linear, and gradient descent makes only local adjustments, which cannot redistribute the error effectively. One approach is to use a better preconditioner (such as the exact Hessian); however, this change increases the computational complexity substantially. A better approach, proposed by Grisetti et al. (2007b), is to cleverly reparameterize the problem. Rather than using the absolute pose parameters, the approach uses a relative parameterization of poses, arranged in a tree. The pose of module i is specified relative to its parent up(i) as $l_i \triangleq l_{\text{up}(i)} \circ x_i$, where x_i is the "difference" between the pose i and its parent.

[3] In our experiments, we use preconditioned conjugate gradient descent, because it is known to converge faster, but the results are comparable.

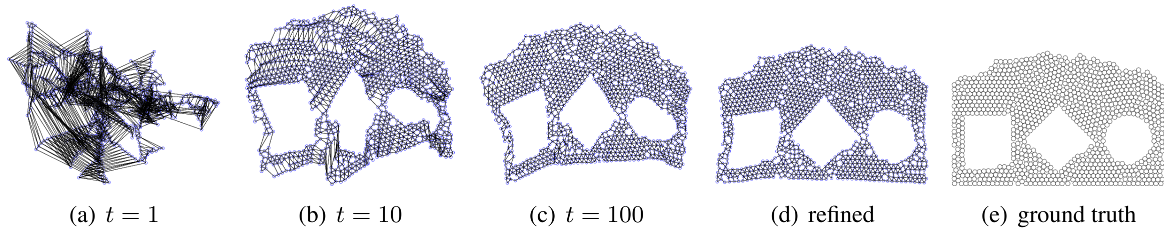


Figure 5.4: The results of running the algorithm (Grisetti et al., 2007b) on an ensemble with 1000 modules. (a-c) Solution obtained by the algorithm at various stages of the optimization. The algorithm recovers from a very bad initialization. (d) The solution in (c), refined by running a small number of global conjugate gradient descent steps. The computed answer is very close to the ground truth (e), with root mean square error less than 1 module radius.

The functions $F_{i,j}(l_i, l_j)$ are optimized in sequence; any error in the prediction of $\overline{z}_{i,j}$ is redistributed among multiple poses in the tree, so that the poses in an *entire subtree* are shifted with a single update. This approach often obtains superior estimates, as shown in Figure 5.4.

5.3 Localization with hierarchical graph partitioning

While iterative approaches have become faster at converging to a solution, they are still prone to local optima. One such case is shown in Figure 5.5. Notice that the estimated poses in this example are all locally consistent, but the solution does not have the desired global shape. The solution is thus close to the optimal objective value, but it does not correctly capture the prior information in the form of physical constraints that the module poses should be non-overlapping. This local optimum occurs because of topological twists and other global errors (inadvertently) introduced when constructing an initial solution. Because the local observations are still consistent, this local optimum is difficult to detect (especially in 3D), and is more likely to occur in modular robots than in SLAM, where the successive robot poses have accurate odometry information.

5.3.1 Overview of the algorithm

A key to avoiding the local optima like the one shown in Figure 5.5(b) is to bias the search, so that it avoids physically implausible configurations. One way to introduce this bias is to construct the solution bottom-up over progressively larger groups of modules. At the bottommost level, the partial solutions are all non-overlapping (because they consist of individual modules). We may then hope that all the partial solutions are physically plausible and lead to an accurate final solution overall.

This observation suggests the following hierarchical algorithm. At each level of the hierarchy, the algorithm starts by partitioning the ensemble into two groups of modules A and B. By applying the localiza-

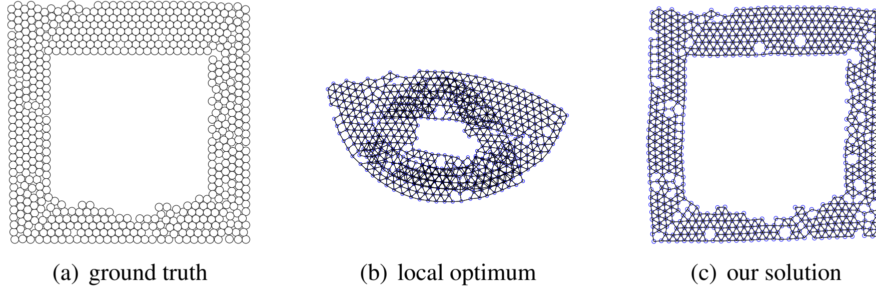


Figure 5.5: (a) An ensemble with 1000 modules and realistic configuration. (b) A locally optimal solution, obtained by running an iterative algorithm (Grisetti et al., 2007a), followed by 100 steps of conjugate gradient descent. This solution is within 2% of the optimal objective value, but has a very high positional error. (c) The solution obtained by running our algorithm.

tion procedure recursively, the algorithm computes a partial solution for the modules A, conditioned on all observations within A and similarly for the modules B. Let k be the current level of recursion, and let $\mathbf{l}_A^{(k+1)}$ and $\mathbf{l}_B^{(k+1)}$ denote the partial solutions, obtained by applying the algorithm recursively. These estimates are (approximately) the local maxima of the likelihood in (5.2), restricted to the edges whose endpoints are both in A and B, respectively:

$$
\begin{aligned}
\mathbf{l}_A^{(k+1)} &= \arg\max_{\mathbf{l}_A} \prod_{(i,j)\in \vec{E}_G : i,j \in A} p(\overline{z}_{i,j} \mid l_i, l_j) \\
\mathbf{l}_B^{(k+1)} &= \arg\max_{\mathbf{l}_B} \prod_{(i,j)\in \vec{E}_G : i,j \in B} p(\overline{z}_{i,j} \mid l_i, l_j)
\end{aligned}
$$

The partial solutions are then merged to obtain an estimate for the entire ensemble $\mathbf{l}^{(k)}$. The merging procedure involves a combination of a global alignment step, solved in closed form, and a local refinement that maximizes the joint optimization problem (5.4) using gradient descent.

5.3.2 Determining an effective partition

The choice of partitioning crucially determines the performance of the above algorithm. If the relative poses of the modules in the partial solutions $\mathbf{l}_A^{(k+1)}$ and $\mathbf{l}_B^{(k+1)}$ are far from the joint solution $\mathbf{l}^{(k)}$, the local refinement will require many iterations to solve the optimization problem. One way to measure this error is the entropy of the conditional distributions $p(\mathbf{l}_A \mid \overline{\mathbf{z}}_A)$ and $p(\mathbf{l}_B \mid \overline{\mathbf{z}}_B)$ (with an appropriately chosen prior to resolve the ambiguity of the global coordinate frame). The distributions with a larger entropy will be less certain about the poses and hence, their modes may not be as accurate.

Unfortunately, estimating the entropy of the distribution in non-linear systems like ours is a difficult task; even if we were able to estimate the entropy, we would still need to perform a search to find a partition that minimizes it. This problem may in fact be more difficult than the localization task we set out to

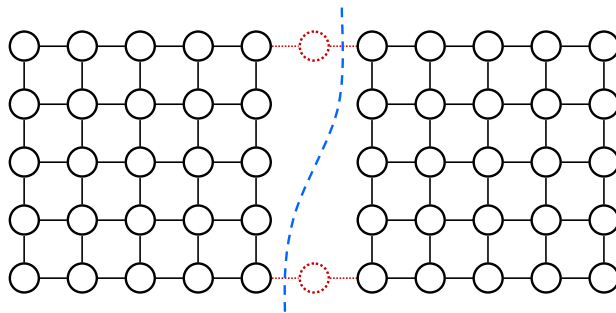


Figure 5.6: The connectivity graph for an ensemble, consisting of two tightly connected clusters. The red dotted lines indicate these weak regions; the modules in these weak regions only make a few observations. The blue dashed line indicates the optimal normalized cut, which separates the two connected clusters along the weak regions.

solve in the first place. Nevertheless, we can find a good partitioning heuristically, by identifying **weak regions** in the ensemble. A weak region is a sparsely connected group of modules, which make only a few observations among themselves (see Figure 5.6). Attempting to localize two small subgraphs connected by a weak region will lead to a large error, as there may be too few observations to effectively constrain the system. At a larger scale, however, observations from multiple weak regions can be combined to resolve the uncertainty. This intuition suggests that we should delay incorporating observations in the weak regions and seek a partition of the graph such that (i) each component is well-connected, and (ii) there are few edges that cross the boundaries between the individual components. This criterion is well captured by the objective of normalized cut (Shi and Malik, 2000):

$$\mathit{Ncut}(A, B) = \frac{\mathit{cut}(A, B)}{\mathit{assoc}(A, V)} + \frac{\mathit{cut}(A, B)}{\mathit{assoc}(B, V)}, \tag{5.7}$$

where $\mathit{Ncut}(A, B)$ is the normalized cut value, $\mathit{cut}(A, B)$ is the number of observations between module sets A and B, and $\mathit{assoc}(A, V)$ is the total number of observations between the modules in A and all modules in the graph. Minimizing (5.7) yields a partition of G into two components A, B. Note that the normalized cut procedure uses an undirected graph $G = (N_G, E_G)$, where each observation $Z_{i,j}$ contributes 1 to the weight of the (undirected) edge $\{i, j\}$.

Intuitively, normalized cut prefers partitions such that the number of observations between A and B is small, compared to all observations made by A and B. For example, in Figure 5.6, the vertical cut that separates the two well-connected components has value $Ncut = O(\frac{1}{N})$, where N is the number of modules, whereas the value of the horizontal cut is $Ncut = O(\frac{1}{\sqrt{N}})$. We see that normalized cut strongly discriminates these two cases and yields the desired ordering. The objective (5.7) also ensures that the algorithm continues to make progress, since it tends to select components of comparable size. For example, in Figure 5.5(a), the normalized cut partitions the graph horizontally through the empty space. In contrast, the minimum cut does not have this property and is free to choose an arbitrarily imbalanced partitioning of the loop. Thus, unlike normalized cut, minimum cut can effectively lead to a solution that

estimate the poses in the ensemble incrementally.

5.3.3 Merging partial solutions

Given a binary partition of G and the partial solutions $\mathbf{l}_A^{(k+1)}$ and $\mathbf{l}_B^{(k+1)}$, we need a procedure that recovers a local optimum $\mathbf{l}^{(k)}$ for the entire graph G. We compute the solution $\mathbf{l}^{(k)}$ in two steps. First, we optimize the global likelihood (5.2), subject to the constraint that the relative poses of modules within A and B remain unchanged. In other words, we view the clusters A and B as two rigid bodies, and we determine an optimal rigid transform (translation and rotation) that aligns these two rigid bodies according to the observations made between A and B. Let Q denote an arbitrary rigid transform that we apply to cluster B, in an effort to align it with cluster A. The negative log-likelihood can be then written as

$$\sum_{\substack{(i,j)\in\vec{E}_G\\ i\in A, j\in B}} \left\| l_i^{(k+1)} \circ \overline{z}_{i,j} - Q \circ l_{c,j}^{(k+1)} \right\|_2^2 + \sum_{\substack{(j,i)\in\vec{E}_G\\ i\in A, j\in B}} \left\| Q \circ l_j^{(k+1)} \circ \overline{z}_{j,i} - l_{c,i}^{(k+1)} \right\|_2^2, \tag{5.8}$$

plus additive terms that are independent of Q. As before, the log-likelihood penalizes the observations based on the distance to the predicted centers, but the poses of modules in B are parameterized as a composition of Q and $\mathbf{l}_B^{(k+1)}$, rather than simply $\mathbf{l}_B$. Minimizing (5.8) with respect to Q amounts to solving the optimization problem

$$Q^* = \underset{Q\in SO(d)\times\mathbb{R}^d}{\operatorname{argmin}} \sum_{(i,j)\in\vec{E}_{AB}} \left\| p_{i,j} - (Q\circ q_{i,j}) \right\|_2^2, \tag{5.9}$$

where d is the dimensionality of the problem, $\vec{E}_{AB}$ is the set of directed edges in G between the clusters A and B (in either direction), and $\{p_{i,j}\}$, $\{q_{i,j}\}$ are two point clouds that represent the matching observations and centers:

$$\left.\begin{aligned} p_{i,j} &= l_i^{(k+1)} \circ \overline{z}_{i,j} \\ q_{i,j} &= l_{c,j}^{(k+1)} \end{aligned}\right\} \quad \text{for } (i,j)\in\vec{E}_G,\ i\in A,\ j\in B, \tag{5.10}$$

$$\left.\begin{aligned} p_{j,i} &= l_{c,i}^{(k+1)} \\ q_{j,i} &= l_j^{(k+1)} \circ \overline{z}_{j,i} \end{aligned}\right\} \quad \text{for } (j,i)\in\vec{E}_G,\ i\in A,\ j\in B. \tag{5.11}$$

The optimal rigid transform (5.9) can be computed with a closed-form solution in time linear in the number of observations between A and B (Umeyama, 1991).

The optimal transform Q^* yields an initial estimate of the pose for the entire graph G:

$$\tilde{l}_i^{(k)} = l_i^{(k+1)} \qquad \text{for } i\in A \tag{5.12}$$

$$\tilde{l}_j^{(k)} = Q^*\circ l_j^{(k+1)} \qquad \text{for } j\in B. \tag{5.13}$$

Algorithm 2 *NormCutLocalize*(G, V, k)

1: **if** V is sufficiently small **then**
2: compute $\arg\max_{\mathbf{l}_V} p(\overline{\mathbf{z}}_V \mid \mathbf{l}_V)$ using local heuristics
3: **else**
4: Compute the normalized cut $(A, B) = \textit{NormCut}(G)$
5: $\mathbf{l}_A^{(k+1)} \Leftarrow \textit{NormCutLocalize}(G_A, A, k+1)$
6: $\mathbf{l}_B^{(k+1)} \Leftarrow \textit{NormCutLocalize}(G_B, B, k+1)$
7: Compute the aligned points $\{p_{ij}\}$ and $\{q_{ij}\}$ using Equations 5.10-5.11.
8: Compute the optimal rigid transform:

$$Q^* = \operatorname*{argmin}_{Q \in SO(d) \times \mathbb{R}^d} \sum_{\{i,j\} \in E : i \in A, j \in B} \|p_{ij} - (Q \circ q_{ij})\|_2^2 .$$

9: Let $\tilde{\mathbf{l}}_V^{(k)} \Leftarrow (\mathbf{l}_A^{(k+1)}, Q^* \circ \mathbf{l}_B^{(k+1)})$.
10: $\mathbf{l}_V^{(k)} \Leftarrow \arg\max p(\overline{\mathbf{z}}_V \mid \mathbf{l}_V)$, starting from $\tilde{\mathbf{l}}_V^{(k)}$ (computed approximately using an iterative method)

This initial estimate is used as a starting point for preconditioned gradient descent, to obtain an approximate maximum likelihood estimate for the observations among the modules in G. We have found that, for the scenarios considered in this chapter, very few iterations are sufficient (on the order of 10).

The resulting procedure is shown in Algorithm 2. The base case, when the graph G is small, is initialized using a local heuristic: we form a spanning tree of G, and initialize the poses, starting from an arbitrarily chosen root using an alignment procedure similar to (5.9). The algorithm then proceeds as described, computing partial solutions recursively, and merging them to obtain solutions at the higher level. When the function returns from the top level of the execution, it obtains the maximum-likelihood estimate $\mathbf{l}^*$ of the entire ensemble.

5.3.4 Scaling up the solution

While normalized cut is an effective heuristic for partitioning the problem, computing the exact normalized cut is costly and dominates other operations. Specifically, the total cost of all the rigid alignment steps is linear in the total number of observations, that is, it is linear in the number of nodes $O(|V|)$ (due to the locality of observations). On the other hand, the complexity of computing a *single* normalized cut is $O(|V|^{3/2})$ (Shi and Malik, 2000), dominating computation costs. A standard method to decrease the computational complexity is to compute an abstraction of the graph, using a simpler clustering algorithm, such as k-means (Shi and Malik, 2000). Each vertex a in the abstract graph G' corresponds to a cluster of nodes N_a in G; in the context of image segmentation, a is called a **super-pixel**, and the abstract graph G' is an over-segmentation of the image. Each edge $\{a, b\} \in E_{G'}$ corresponds to a set of edges and its weight is the sum of the weights of all the edges between N_a and N_b in G. The normalized cut is then computed on a smaller graph G'.

Compared to other clustering tasks, the clustering task in internal localization is simpler in two ways. First, unlike in image segmentation, where shifting the cut can adversely affect the visual quality of the segmentation, the clustering in our application is only used as a heuristic, and offsetting the cut does not substantially decrease the quality of location estimates. Furthermore, since the edges in the connectivity graph G can only exist between geometrically adjacent nodes and have unit edge weights, the increase in the cut value can be bounded as a linear (for 2D) or quadratic (for 3D) function of the hop count distance from the optimal one. Therefore, a naive strategy of partitioning the graph greedily is sufficient in this problem.

5.4 Distributed localization

While centralized localization in a self-reconfigurable modular robot is useful, distributed localization is preferred, as it can significantly reduce the communication cost, enable on-line control, and avoid a centralized point of failure. The observations are naturally distributed throughout the ensemble, so a distributed algorithm eliminates the need to collect all of the data centrally. Our distributed implementation closely follows the centralized approach. We rely on standard techniques of local communication, data aggregation, and data dissemination to implement the steps of Algorithm 2. Each layer of our distributed solution on its own is not complicated, but the integration of these components creates substantial challenges.

5.4.1 Network assumptions

The assumptions on the network in this chapter differ from those in the previous chapters. The work in previous chapters was primarily motivated by wireless sensor networks, where nodes communicate over lossy, unreliable links. By comparison, the communication in modular robots is short-range, with the transmitters / receivers operating in a close proximity. Therefore, the communication links do not suffer from interference and are more reliable. Nevertheless, communication may get interrupted if a module moves. In addition, we assume that whenever two modules i and j observe each other, they can also communicate directly. While this is a strong assumption, it is justified by our system requirements: the sensors that measure the relative pose are also used for communication. Therefore, in modular robots, the probabilistic model and the communication network are aligned.

5.4.2 Individual layers

Figure 5.7 summarizes the control flow for one level of the algorithm execution. We first describe each layer in isolation; we will discuss how the layers are integrated in the next section.

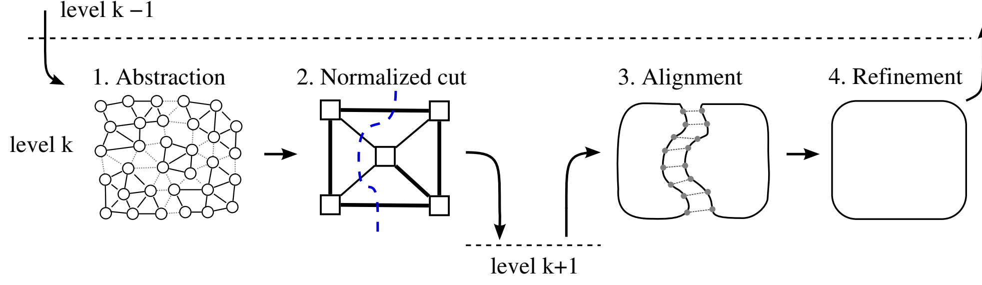


Figure 5.7: Control flow for one level of the distributed implementation.

In order to compute the graph abstraction, we use a random sampling strategy that partitions the graph into a set of connected subgraphs, centered around randomly self-selected leaders. Each module elects itself as a leader with a small probability, and becomes the root of a tree that defines a vertex of the abstract graph G'. The trees grow out from the leaders as the other nodes greedily join the tree of the nearest leader (as measured by the hop-count). Each node stores the ID of its leader and shares the leader ID with its immediate neighbors. When this process completes, each node can locally identify, which of its neighbors (if any) belong to a different subgraph. If node in subgraph a detects that one of its neighbors belongs to a different subgraph b, it records the edge (a, b). We denote the multiset of edges detected by node i as E_i; the union of all the edges in the group $\cup_i E_i$ represents all the edges in the abstract graph G'.

It may be possible to compute the normalized cut in a decentralized manner, using decentralized power iteration (Yang et al., 2008). However, we only need to obtain normalized cuts for small abstract graphs, so computing the normalized cuts in a centralized manner on one of the leader nodes works well. The leaders first select a single node among themselves, which we call the **group leader**; the group leader is responsible for computing the normalized cut at the current level of hierarchy. The nodes in the group form a spanning tree $\mathcal{T}$, rooted at the group leader. The edges of the abstract subgraph is then aggregated to the group leader along this tree, by computing messages as

$$\mu_{i \to \text{up}(i)} = E_i \cup \bigcup_{j \in N_{\mathcal{T}}(i) \setminus \text{up}(i)} \mu_{j \to i}. \tag{5.14}$$

The message $\mu_{i \to \text{up}(i)}$ represents the multiset of edges of the abstract graph downstream of i, and it is stored efficiently, by associating each edge (a, b) with its count. The edges of the abstract graph at the group leader are then simply

$$E_i \cup \bigcup_{j \in N_{\mathcal{T}}(i)} \mu_{j \to i}. \tag{5.15}$$

The group leader then computes the normalized cut with a centralized algorithm and broadcasts it out to all of the modules in the group, beginning the recursion process.

After the recursion completes, the modules on the two sides of the partition have computed the estimates $\mathbf{l}_A^{(k+1)}$ and $\mathbf{l}_B^{(k+1)}$ that are conditioned on the observations within each partition component. Recall that

a key step in Algorithm 2 is computing the optimal rigid transform that aligns the modules on the two sides of the partition, keeping the relative poses within each component fixed. To compute this transform in a decentralized fashion, we leverage the fact that the centralized algorithm (Umeyama, 1991) only depends on the first two moments of the two point clouds $\{p_{i,j}\}$,$\{q_{i,j}\}$, defined in (5.10-5.11): $\frac{1}{N}\sum_{i,j} p_{i,j}$, $\frac{1}{N}\sum_{i,j} q_{i,j}$, and $\frac{1}{N}\sum_{i,j} p_{i,j}q_{i,j}^\top$. We reuse the spanning tree $\mathcal{T}$, formed in the previous step, to aggregate these statistics from the boundary to the group leader. The leader then computes the optimal transform and disseminates the result. Since the size of the aggregated statistics depends only on the dimensionality of the aligned points (2 or 3), rather than on the number of contributing observations, the communication cost of aggregating and disseminating the optimal transform is very small. Likewise, the computational cost of the alignment is constant at the leader.

As each module receives and applies the optimal transform, it begins the preconditioned gradient descent step. Recall from (5.6) that the gradient with respect to the pose of one module l_i has a very simple structure: in order to evaluate the gradient, we only need to know the poses of module i and its neighbors. Thus, the gradient descent can be implemented very easily: each module stores its own estimate of the pose l_i, and periodically transmits its estimated pose to its neighbors. Once the node receives the estimate from all its neighbors $j \in \overleftarrow{N}(i) \cup \overrightarrow{N}(i)$, it computes the gradient (5.6) locally and applies it to its pose estimate to take one gradient step. Thus, this aspect of the algorithm is entirely local and can be distributed very easily. For simplicity, gradient descent proceeds for a fixed number of iterations. An alternative stopping condition, such as the magnitude of the gradient, could be used without affecting the other layers of the algorithm.

An important feature of our algorithm is that at each level of hierarchy, each node communicates only a constant amount of information. The graph abstraction has a constant communication complexity per node, because each node joins one vertex of the abstraction. The normalized cut computation also has a constant communication complexity per node, because the abstract graphs are bounded in size, and each node has a bounded number of neighbors from whom it may collect the information in aggregating the abstract graph to the leader. Once the leader computes the normalized cut, the cut is disseminated to all the nodes in the group, with a constant communication complexity per node. Similarly, the rigid alignment step requires a constant amount of communication per node. The communication complexity of gradient descent is bounded by the number of iterations, which is held constant in our implementation.

Since the algorithm requires only a constant amount of communication per level per node, the overall per-node communication complexity is bounded by the number of levels of hierarchy. Typically, the partitions computed by normalized cut are balanced, meaning that typically neither of the two components has more than α fraction of the nodes, for some fixed $\alpha < 1$. In this case, the number of hierarchy levels is bounded by $\log_{1/\alpha} N$, where N is the number of nodes. We thus obtain the following statement:

Theorem 5.1. *Suppose that each time the normalized cut is executed, it forms a partition where neither component has more than αN_a nodes, where N_a is the number of nodes in the group and $\alpha < 1$ is a fixed value. Then the communication complexity of the localization algorithm is $O(\log_{1/\alpha} N)$ per node.*

5.4.3 Implementation using Meld

While conceptually simple, our algorithm poses several implementation challenges. First, due to the recursive nature of the algorithm, the implementation needs to maintain parallel data structures for all of the concurrently active levels. Thus, each node needs to participate in multiple aggregation trees $\mathcal{T}$, one at each level. Furthermore, the program consists of several layers, interacting both within one level of hierarchy and across the levels. Thus, some amount of synchronization among the layers is necessary. For example, after finishing the gradient descent on one side of the partition, the nodes may need to wait for the other side to finish, before the statistics for the rigid alignment can be aggregated. However, even though we described the algorithm as logically separate layers and recursion levels, the implementation should not use explicit synchronization between the stages, as large-scale synchronization slows down the execution of the algorithm. Rather, the algorithm should operate asynchronously: as soon as an individual module has sufficient information to proceed to the next step, it should do so immediately. Finally, the implementation should be robust to changing topology. While we focus on the static problem, some small changes to the topology are possible (for example, by the sensors failing intermittently). The algorithm should compute a meaningful solution in face of these changes.

To overcome these challenges, we implemented our distributed algorithm in the Meld programming language (Ashley-Rollman et al., 2009). Meld is a logical, high-level, declarative language designed for programming large ensembles. It was motivated by P2 (Loo et al., 2006) with syntax similar to Datalog (Ceri et al., 1989). A Meld program consists of rules, running simultaneously on all of the modules in the system, that specify sufficient preconditions to derive new facts from existing ones. A fact is a tuple of one or more elements, each of which is a module identifier or a constant. By convention, the first element of each fact is a module identifier which refers to the module on which the fact will be stored. Each rule has a head that specifies a fact to be generated and a body of prerequisites to be satisfied, separated by the symbol "`:-`". The rules can refer to facts that are located on other nodes, provided that those facts are accessible through some chain of local and remote facts. This design provides a key benefit of Meld: it allows programmers to focus on the logical, information processing aspects of an algorithm, while Meld automatically takes care of the mechanics of distributed programming.

For example, a spanning tree can be represented using one fact per node, `parent(A,B)`, stating that module B is a parent of module A. If $A = B$, then A is the root of the spanning tree. By the convention noted above, each fact `parent(A,B)` is stored at the child A. A simple distributed spanning tree algorithm can be then specified in two rules: a rule that determines the root of the tree, and a rule that lets a node join a tree that extends to one of its neighbors, as shown in Figure 5.8. This specification has two benefits: First it does not require the programmer to explicitly reason about the communication, required to form the spanning tree. Instead, the Meld compiler automatically writes the low-level code, needed to establish the parent relationship. Furthermore, once the spanning tree has been formed, the `parent` facts can be used to route the messages $\mu_{i \to j}$ that carry the edges of the graph abstraction to the leader,

```
/* Declare a fact type parent(A,B), representing that the parent of A is B. */
type parent(module, first module).

/* Start the tree at the leader of a group. */
parent(ModuleA, ModuleA) :-
       leader(ModuleA, ModuleA).      /* A is a leader. */

/* Extend the tree to other modules in the same group */
parent(ModuleA, ModuleB) :-
       neighbor(ModuleA, ModuleB),
       parent(ModuleB, _),            /* B has already joined some tree. */
       leader(ModuleA, Leader),       /* Limit the tree to modules in the same group. */
       leader(ModuleB, Leader).
```

Figure 5.8: A program to generate a spanning tree across each group. The program consists of a type declaration and two rules that specify the root of the tree and a procedure for extending the tree to all other members in the group. The **`first`** keyword prevents a node from acquiring multiple parents. The "_" symbol in the second rule states that any value can be matched in the corresponding field, that is, B has already some parent.

as shown in Figure 5.9. Similarly, once the normalized cut is computed at the group leader, the spanning tree specified by the `parent` facts is used to propagate the cut down to all the nodes in the group. An analogous set of rules specify the procedure for computing and disseminating the optimal rigid transform. Thus, the steps of our algorithm at a single level of recursion can be seamlessly integrated.

In order to implement the recursive calls to the *NormCutLocalize* procedure, we augment each fact with the recursion level.[4] This simple modification lets us construct and maintain spanning trees at each level of the hierarchy. Once again, Meld semantics allows us to easily chain the results between different recursion levels, by including the correct level in the preconditions. Thus, explicit synchronization between the recursion levels can be avoided—some nodes may return to the higher recursion level, while others continue to make progress at the same level. Since Meld rules specify the dependencies among the facts, the Meld runtime can automatically gate processing on the availability of relevant data. This design maximizes parallelism, as execution is only limited by the dataflow. The entire description of our algorithm is concise—it consists of 60 Meld rules (470 lines) and a few C++ subroutines. A Meld program is compiled to C++, so it can be executed very efficiently.

At the time of initial publication of this work, Meld did not support certain key features that would make our implementation more concise and more efficient. First, Meld did not support X-Y stratification (Zaniolo et al., 1993), which is a technique to efficiently implement the aggregates, such as the unions in graph abstraction (5.14-5.15), by delaying the execution of the aggregate until all the prerequisite predicates

[4] To simplify the implementation, we perform the abstraction only once, at the topmost level of the hierarchy. However, it is not too difficult to extend our implementation to perform the abstraction at each level.

```
/* Declare a fact type edges(A,E), representing the set of edges of G' known to node A. */
type edges(module, union edge multiset).

/* Detect an edge in the graph abstraction. */
edges(ModuleA, Edges) :-
     neighbor(ModuleA, ModuleB),
     leader(ModuleA, Leader),           /* A and B are in the same group. */
     leader(ModuleB, Leader),
     vertex(ModuleA, VertexA),          /* A and B belong to different vertices
     vertex(ModuleB, VertexB),             of the abstraction. */
     VertexA != VertexB,
     Edges = {edge(VertexA, VertexB)}.  /* Create an edge between the two vertices. */

/* The edges are propagated up the tree to the root. */
edges(Parent, Edges) :-
     parent(Module, Parent),
     edges(Module, Edges).
```

Figure 5.9: A program to compute the graph abstraction at the root of the spanning tree. The program consists of a type declaration and two rules that detect the edges in the graph abstraction and aggregate them up the spanning tree. Note that both rules may produce one or more sets of edges at each node. These sets are automatically merged together, as specified by the **union** keyword in the fact type declaration.

have been gathered by the node. Without X-Y stratification, if a new predicate arrives that changes the value of the aggregate, the old value is deleted and replaced with a new one, which can trigger a significant re-execution of the program. The execution of aggregates can be delayed manually with additional rules, but Meld now supports X-Y stratification (Ashley-Rollman et al., 2009), which lowers the number of rules to approximately 44. The rules shown in Figure 5.9 do take advantage of X-Y stratification. Furthermore, Meld did not support dynamic state, which is a technique to give the programmer a tighter control when a fact is derived and underived. Without dynamic state, the gradient descent algorithm had to be implemented by deriving a new fact for each iteration, and the implementation had to remember the estimates from all the levels of hierarchy. Due to this limitation, a typical execution of the algorithm would take tens to hundreds of kilobytes of memory per node. Moreover, if a module moved or a communication link became temporarily unavailable, Meld would question the validity of any information that was derived from that module's position or the link. Thus, a minor change would trigger a significant re-execution of the algorithm. Meld now supports dynamic state (Ashley-Rollman, 2010) based on linear logic (Girard, 1987). With this feature, gradient descent can keep only the latest iterate. Furthermore, it is now possible to maintain a snapshot of the observations among the modules, which avoids the need to rederive some facts and would make the implementation more robust to minor topological changes.

5.4.4 On the role of group leaders

In our distributed localization algorithm, certain key steps of the computation (normalized cut, optimal rigid alignment) are performed at a single node within the group—the group leader—based on the statistics aggregated from all the nodes within the group. This algorithm is conceptually different from the robust message passing algorithm, reviewed in Chapter 2. In both algorithms, the nodes communicate over a spanning tree; however, in the robust message passing algorithm, all the nodes play the same role: each node computes a summary (marginal) of the data (factors) present at all the nodes in the network. Thus, in the robust message passing algorithm, each node is effectively a leader, which is necessary, because the nodes need to compute different marginals of the probability distribution. However, in modular robot localization, all the nodes in the group wish to compute the same quantity, and whether or not all the nodes play an equal role is a design decision with different trade-offs, as we will now discuss.

Our localization algorithm can be easily adapted to make each node a leader, by changing the way data is aggregated in the normalized cut and rigid alignment phases of the algorithm. Rather than aggregating the data unidirectionally towards a single leader, the data is aggregated towards each node in the group. The aggregation is performed by passing messages along the edges of a single spanning tree $\mathcal{T}$ over the nodes in the group. The messages have a similar interpretation as before. For example, in computing the graph abstraction at every node in the group, the message $\mu_{i\to j}$ sent along a directed edge $(i,j) \in \vec{E}_{\mathcal{T}}$ represents the multiset of all the edges of the abstract graph on node i's side of the tree. Formally, the message is defined as a union of the locally observed edges E_i as well as the messages from all neighbors of i other than j:

$$\mu_{i\to j} = E_i \cup \bigcup_{k\in N_{\mathcal{T}}(i)\setminus j} \mu_{k\to i}.$$

Once all the messages incoming to i have been received at node i, the abstract graph can be recovered locally using (5.15). Normalized cut of this abstract graph can be then computed locally using a standard centralized implementation. Since the algorithm sends only two messages for each undirected edge of the spanning tree $\mathcal{T}$, the communication complexity of the new algorithm is within a factor 2 of the communication complexity of the original algorithm, discussed in Section 5.4.2. Furthermore, in the absence of failures, each node in the group computes the same normalized cut and the same rigid alignment, so the two localization algorithms compute the same answer.

While the two algorithms have a similar communication complexity and compute the same answer in the absence of failures, their behavior is radically different in the presence of node and communication failures. In the presence of node failures, a single leader node constitutes a single point of failure within the group: if a leader node fails, a new spanning tree must be formed from scratch, which triggers a re-execution of the aggregation at the current level of hierarchy. On the other hand, the leader serves an important role: it ensures that the nodes agree on the normalized cut and the optimal rigid alignment. Suppose that each node is a leader and as soon as the node determines the optimal cut, it recurses to the

next level, and similarly, it begins gradient descent (and ultimately returns to the previous level of hierarchy) as soon as it determines the optimal rigid alignment. Furthermore, suppose that a communication link becomes unavailable and the spanning tree changes. This change may temporarily create an incorrect result at a node, double-counting or under-counting some of the aggregated values. The nodes may then disagree on the normalized cut or the optimal rigid alignment. The localization algorithm can likely tolerate some disagreement about the optimal rigid transform, because any differences will be smoothed out in iterative refinement. However, if the nodes disagree on the normalized cut, the components obtained by the normalized cut may no longer be connected, which prevents the localization algorithm from successfully completing its execution. Therefore, unless we compute the normalized cut in a decentralized manner, the single-leader version of the algorithm is preferred.

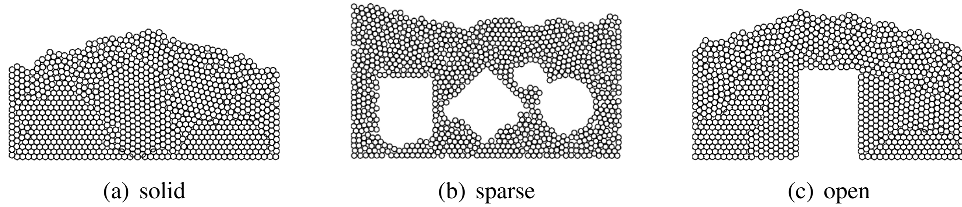
So far, we have discussed two versions of the localization algorithm—one with a single leader in each group and one in which each node is a leader. It is also possible to have a subset of the nodes act as leaders. This design works well when the normalized cut is computed in a decentralized manner using power iteration (Yang et al., 2008). In this case, each vertex of the abstract graph G' corresponds to a single leader that executes the steps of the power iteration, and the leaders communicate using multi-hop routing. The design does not help with rigid alignment, where it introduces additional complexity in the spanning tree formation, does not lower the communication complexity, and still suffers from inconsistent estimates of the optimal rigid transform.

5.5 Experimental results

In this section, we present experimental results that illustrate both the centralized and the distributed aspects of our solution. We generated input scenarios with a C++ simulator[5] that models IR sensing and physical interactions between the modules. Each module in the simulation had 12 IR transceivers (co-located emitter/detector pairs), whose IR response was modeled according to an inverse-square law, similar to the model in (Roufas et al., 2001). The threshold for detecting observations between a pair of neighboring nodes was set to 20 percent of the peak intensity. At this setting, a sensor can report a connection even if the modules are not in a physical contact and if the transmitter and the receiver are not perfectly aligned.

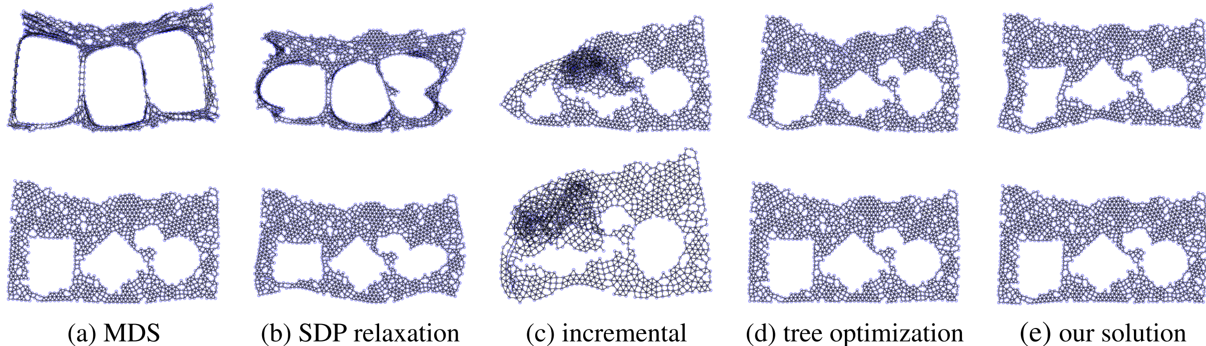
We first evaluate aspects of our algorithm pertinent to both the centralized and distributed implementations, such as the quality of the solution, sensitivity to abstraction, and scalability. These experiments were run using the centralized implementation. We then evaluate messaging costs relevant to the distributed implementation. We perform our experiments on a variety of scenarios, outlined below.

[5]DPRSim: The Dynamic Physical Rendering Simulator. http://www.pittsburgh.intel-research.net/dprweb/



(a) solid (b) sparse (c) open

Figure 5.10: 2D scenarios used in our experiments. The scenarios were generated by settling randomly inserted modules in a gravitational field.



(a) MDS (b) SDP relaxation (c) incremental (d) tree optimization (e) our solution

Figure 5.11: Example results using three algorithms on the *sparse* scenario in Figure 5.10(b). Lower images reflect results after additional iterative refinement steps.

5.5.1 Scenarios

We constructed both 2D and 3D test ensembles. The 2D ensembles were generated by randomly settling simulated spheres under a simulated gravity field into a fixed container of the desired overall shape. The 2D configurations, shown in Figure 5.10, mimic planar slices of a 3D shape capture scenario (Pillai et al., 2006). The configurations have realistic, irregular, largely amorphous structures one would expect in the shape capture scenario. Each shape in Figure 5.10 was instantiated ten times, with different initial velocities and locations of the modules, allowing us to average results across repeated runs using configurations very similar in overall shape but where module connectivity and spacing varies. We also experimented with different 3D configurations, which are detailed in Section 5.5.5.

5.5.2 Convergence

We first compare the proposed algorithm to related algorithms in wireless sensor network localization and SLAM. We evaluate the classical multi-dimensional scaling (Shang et al., 2003) (labeled as *classical MDS* in Figure 5.13) and the inequality formulation of regularized semi-definite programming (Biswas et al., 2006) (*regularized SDP*); these two methods compute a Euclidean embedding of the connectivity graph that preserves the distances between pairs of vertices in the graph. In addition, we evaluate a simple in-

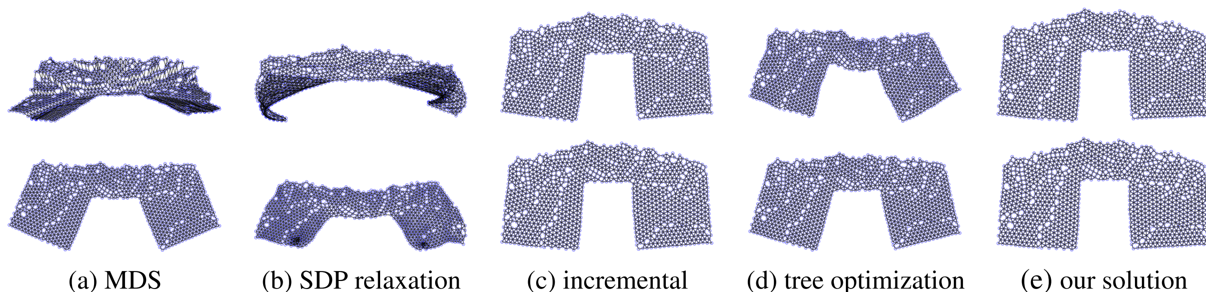


Figure 5.12: Example results using three algorithms on the *open* scenario in Figure 5.10(c). Lower images reflect results after additional iterative refinement steps.

cremental approach (*incremental*) that incrementally adds the modules to the ensemble and continuously performs gradient descent. Finally, we evaluate the tree-parameterized algorithm of Grisetti et al. (2007b) (*tree optimization*) and the proposed method, computing normalized cuts on the original graph, as opposed to the graph abstraction from Section 5.3.4. We perform experiments on the scenarios in Figure 5.10 with 1000 modules. The initial solution, obtained by each method is refined with 300 iterations of preconditioned conjugate gradient descent. For the incremental and the proposed algorithm we perform 10 steps of preconditioned gradient descent at each iteration. At the time of writing, the implementation of (Grisetti et al., 2007b) did not support multiple observations between a pair of nodes and we provide instead the average of the observations for each pair of nodes in the input. However, the preconditioned conjugate gradient descent used all the observations.

Figures 5.11 and 5.12 illustrate the performance of the evaluated methods qualitatively. We see that the embedding-based approaches suffer from overestimation of distances, and artifacts due to the projection from a manifold in a high dimensional space to a 2D space. The incremental approach can get stuck in local minima. The tree optimization and our solution perform well consistently.

This intuition is confirmed Figure 5.13, where we show the average root mean square (RMS) error for each scenario. In order to account for the overlap introduced by the objective (5.1), we uniformly scale the locations of the modules, so that the average spacing equals the module diameter. Since the algorithms do not align to any global coordinate frame, we then use the ground truth locations of the modules to compute an optimal rotation and translation, and report the error for the aligned solution. We see that approaches based on Euclidean embedding (classical MDS, regularized SDP) perform poorly in this setting, especially for the *sparse* scenario and the *open* scenario. For classical multi-dimensional scaling, the error results from approximating true distances with hop-count; for regularized SDP, the errors may come either from the SDP relaxation or the underlying solver. The incremental and the tree optimization approaches perform better, but are outperformed by our normalized cut formulation on the *sparse* scenario.

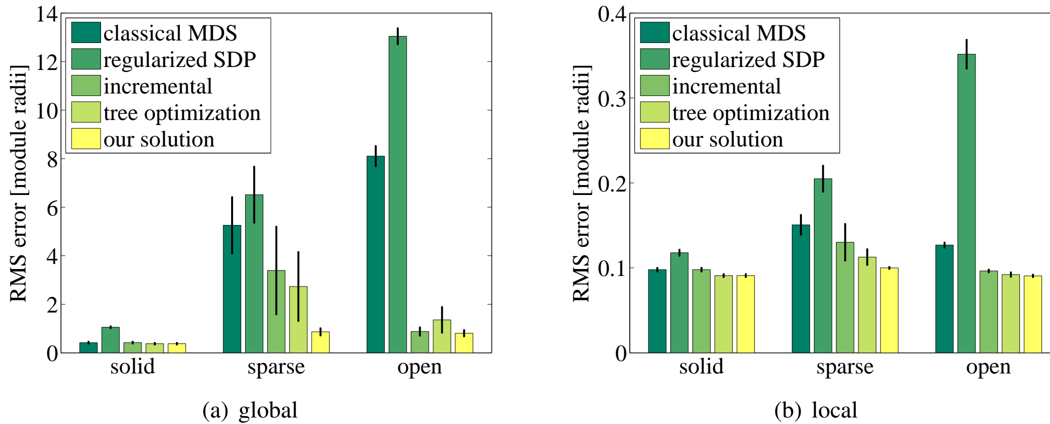


Figure 5.13: RMS error of the location estimates. (a) Global RMS error, averaged over all modules. (b) RMS error of modules relative to their neighbors. Here, we greedily partition the ensemble into connected regions with diameter of 6 modules or less and compute the RMS error using the optimal rigid alignment for each region.

5.5.3 Scalability

In the next experiment, we evaluate the sensitivity of the proposed localization method to errors introduced by performing normalized cut on the abstracted, rather than the original connectivity graph. Abstraction is performed at each level of the hierarchy. We took the *sparse* and *open* scenarios in Figure 5.10(b) with 2000 and 1000 modules, respectively. Figure 5.14 shows the root mean square (RMS) error as we vary the number of nodes in the abstraction of the connectivity graph (since we controlled the maximum diameter of clusters, rather than their count, the displayed node count is approximate). We see that the performance of the proposed localization method is insensitive to abstraction errors: with as few as 20 nodes in the abstract graph, the approach yields a sufficiently small RMS error. These results suggest that meaningful solutions can be obtained from small graph abstractions, which can be analyzed centrally at each leader node. For the rest of the experiments, we use abstractions with approximately 40 vertices.

Next, we evaluate the performance of the proposed method and the tree optimization of (Grisetti et al., 2007b) as the number of modules in an ensemble increases. We selected the *sparse* scenario in Figure 5.10(b) and formed a collection of progressively larger ensembles. At each scale, the ensemble retains the same overall shape and proportions, but the number of modules that form the shape increases. In Figure 5.15(a), we evaluate the RMS error as a function of the number of modules for fixed number of iterations at each level of hierarchy. The tree optimization was run for 1000 iterations (at which point the objective value remains fairly constant). We see that for the large ensembles, the solutions obtained by the tree optimizer become inaccurate. These inaccuracies are often caused by the local optima like the one shown in Figure 5.5(b). The error of our hierarchical algorithm, on the other hand, remains stable.

We also performed an experiment where we observed the time needed to reach a fixed quality of the

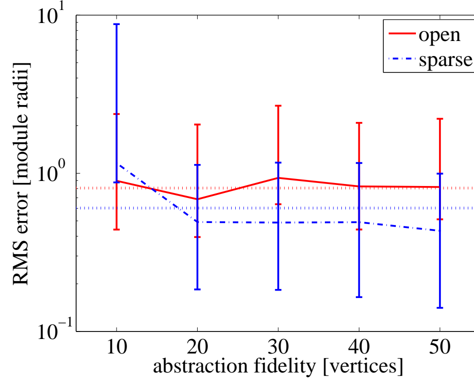


Figure 5.14: The location RMS error for the *sparse* scenario with 2000 modules and the *open* scenario with 1000 modules, when using the normalized cut approximation in Section 5.3.4. The horizontal axis is the number of vertices of the abstract graph (that is, the number of "super-pixels"); the value of 2000 on this axis would indicate no abstraction. In this figure alone, the error bars indicate the range of the measured RMS error values, rather than 95% confidence intervals. The horizontal dashed lines indicate the fidelity of the solutions, obtained with exact normalized cut.

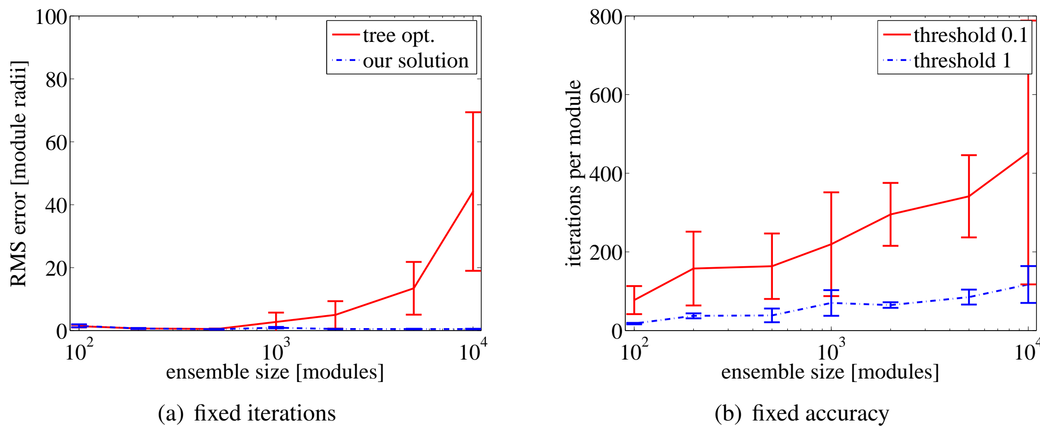


Figure 5.15: (a) The global root mean square error per module, as a function of the number of modules in the ensemble. (b) The average number of iterations per module for the *sparse* scenario as the number of modules grows. At different scales, the ensemble retains its shape and the proportions. With a threshold of 1.0, the average location RMS error was 1.26; with a threshold of 0.1, the average location RMS error was 0.80.

solution. We ran Algorithm 2 such that, at each level of the hierarchy, the estimate $\mathbf{l}^{(k)}$ reaches a fixed level of accuracy, as measured by the norm of the gradient of the likelihood function at $\mathbf{l}^{(k)}$. This procedure ensures that each estimate $\mathbf{l}^{(k)}$ is sufficiently accurate, before it is used at the higher level. Figure 5.15(b) shows the average number of iterations of preconditioned conjugate gradient descent needed as a function of the number of modules. We see that the number of iterations needed to attain the desired level of accuracy increases very slowly.

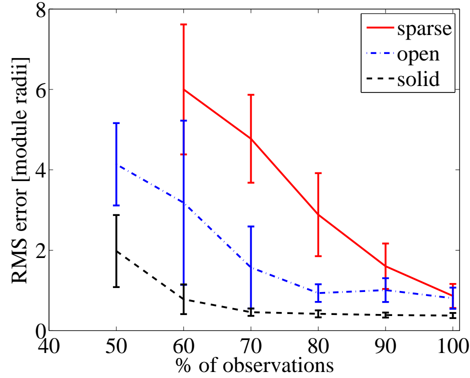


Figure 5.16: The accuracy of our hierarchical algorithm on the three scenarios in Figure 5.10 as a function of the proportion of the retained observations. The *sparse* curve starts at 60% of the observations, because at lower settings, the ensemble becomes too fragmented. We see that the accuracy of the solution improves gradually as the proportion of observations increases, and depending on the scenario, the algorithm attains accurate solutions with as few as 70–90% of the retained observations.

5.5.4 Number of observations

In the next experiment, we evaluate the sensitivity of our approach to the number of observations, simulating sensor failures. We took the *solid*, *sparse*, and the *open* scenarios in Figure 5.10, and subsampled the observations, such that for each pair of adjacent modules, either all or none of the observations are included. Since some of the observations are missing, the resulting connectivity graph may no longer be connected. For our experiments, we take only the largest connected component; this component always contains at least 90 percent of all the modules. We then execute our hierarchical algorithm without graph abstraction and refine the initial solution with 300 iterations of the preconditioned conjugate gradient descent. Figure 5.16 shows that on the *open* and *solid* scenarios, our algorithm attains accurate solutions with as few as 70 percent of observations. On the *sparse* scenario, the algorithm performs worse, because subsampling disconnects the loop around the perimeter of the ensemble, and the rotational uncertainty accumulates. Nevertheless, even in the *sparse* scenario, the algorithm performs well with a moderate observation loss of 10 percent or less.

5.5.5 3D experiments

We extended our implementation to three dimensions, using a quaternion representation for each module's 3D orientation rather than the scalar orientation parameter used in the 2D case. As we did for the the 2D experiments, we generated test scenarios by settling randomly inserted modules in a gravitational field, see Figure 5.17(a,b). In addition, we generated one small and two large 3D ensembles in Figure 5.17(c-e) by rasterizing 3D outlines into a cubic lattice, randomizing the module orientations. The two large shapes are actually hollow shells only 2 modules thick, and are particularly challenging cases. We simulated

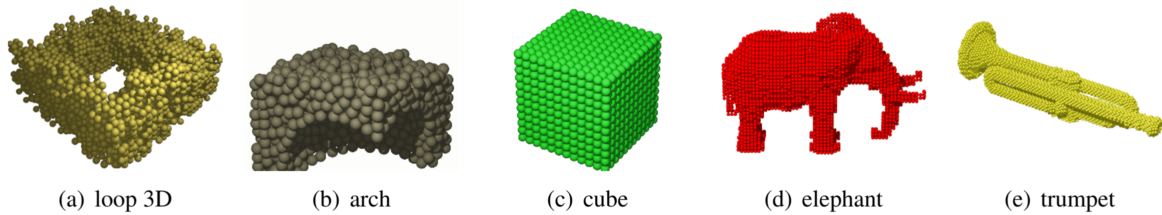


Figure 5.17: 3D scenarios used in our experiments. (a,b) Scenarios that were generated by settling randomly inserted modules in a gravitational field, with 576 and 1210 modules, respectively. (c-e) Scenarios obtained by rasterizing 3D outlines, with 1728, 8008, and 9322 modules, respectively.

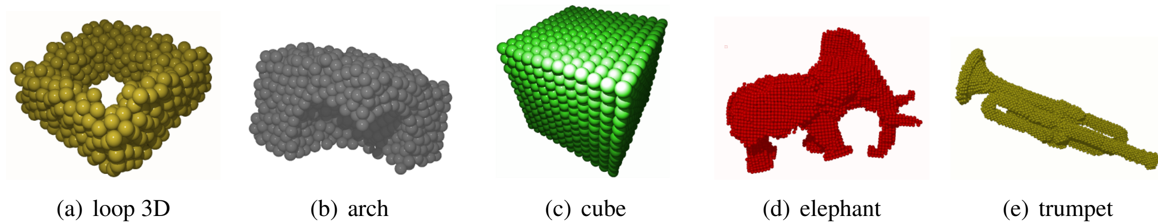


Figure 5.18: Example localization results for the 3D scenarios from Figure 5.17. (a-c) achieve very accurate results (RMS error < 0.2 module radii); (d) and (e) show some distortion, resulting in higher RMS errors of 4–6 module radii.

spherical modules that have 50 sensors scattered across their surfaces. Despite the larger number of sensors, when compared to the 2D case, the available angle constraints in the 3D test cases were generally much weaker.

Figure 5.18 shows example results for the scenarios in Figure 5.17. For the smaller scenarios (a-c), we reach a very high fidelity, with RMS error less than 0.2 module radii. The large, hollow scenarios (d,e) exhibit some distortion, due to inaccuracies at the lowest level of the hierarchy that prevent the algorithm from properly aligning the partial solutions at the larger scale. These distortions lead to a larger error of approximately 4-6 module radii. We tried to compare this result to a state-of-the-art 3D optimizer (Grisetti et al., 2007a); however, the implementation of Grisetti et al. (2007a) uses a slightly different objective function than ours and does not perform well on our scenarios. We did not compare with classical MDS and regularized SDP, since they performed poorly in the 2D experiments, and MDS does not scale to our largest examples.

5.5.6 Messaging costs

Finally, we evaluate the neighbor-to-neighbor message complexity of the distributed implementation running on varying sizes of the *cube* scenario. Figure 5.19 shows the average number of messages sent by each module, as well as the maximum sent by any module, as functions of the ensemble size. The mes-

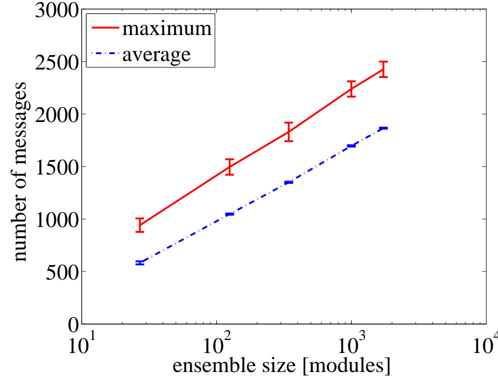


Figure 5.19: The average number of messages per module and maximum number of messages for any module as functions of the ensemble size.

Procedure / Test case	$5 \times 5 \times 5$	$10 \times 10 \times 10$
Neighbor detection	0.5%	0.3%
Graph abstraction	7.7%	7.3%
Normalized cut	6.4%	6.5%
Rigid alignment	9.7%	9.5 %
Gradient descent	75.8%	76.3 %

Figure 5.20: The relative number of messages sent by each component.

sages destined for nodes outside of the module's immediate neighborhood are counted for each module they traverse. These results show that the maximum number of messages sent by any one module is only about 40% greater than the number of messages sent by an average module. This indicates that the communication load of our localization implementation is well distributed throughout the ensemble, and no module is likely to become a bottleneck due to network traffic.

Additionally, the figure confirms that the number of neighbor-to-neighbor messages per module required by the algorithm increases only logarithmically in the total number of modules in the ensemble. Thus, the distributed implementation scales to large ensembles. Figure 5.20 shows the fraction of the messages used by different components of the algorithm for two ensemble sizes. Interestingly, the messaging costs are dominated by the gradient descent steps. In particular, the communication overhead of computing the rigid alignments are small compared to the cost of iterative refinement. Even if we take into account the larger size of the messages for determining the normalized cut, the overall message cost of the normalized cut computation is small.

5.6 Related work

Localization algorithms have received attention in the modular robot literature. When the modules form a perfect lattice, or if the modules make exact observations, their locations can be computed using standard constraint-based methods (Pillai et al., 2006; Reshko, 2004). Often, these algorithms can be distributed easily with simple message-passing schemes. Unfortunately, these algorithms are not robust to noise and are ill-suited to irregular, non-lattice structures, common in some modular robots. Local probabilistic approaches (Roufas et al., 2001) have been shown to be effective in localization of relatively small modular robots, such as PolyBot (Yim et al., 2000). Yet, these methods often make no provision to improve the speed of convergence in large ensembles. Instead, they rely on robust mechanical latching to reduce errors to a millimeter range.

Localization is a well-studied problem in wireless sensor networks. In wireless sensor networks, each node observes approximate distances to other nodes in the wireless network. By combining the distance information about other nodes in the network, the nodes can accurately triangulate their own positions. A standard formulation is to treat each observation as the weight of an edge in a graph and obtain an embedding of the graph vertices in Euclidean space that best matches the observed distances. This formal problem can be solved using a variety of methods, such as multidimensional scaling (Shang et al., 2003), or regularized semi-definite programming (SDP) relaxations (Biswas et al., 2006; Wang et al., 2006b). In contrast to sensor networks, the internal localization of modular robots assumes no long-range communications or sensing, so only adjacent neighboring nodes are detected. Although internal localization can be viewed as Euclidean embedding, only unit distances to immediate neighbors are known. As indicated by our experiments in Section 5.5, this restriction appears to impair the performance of the SDP relaxations. In addition, the sensing model for internal localization provides the approximate headings to neighboring nodes, which are typically not used in Euclidean embedding approaches to sensor network localization.

Internal localization is related to the simultaneous localization and mapping (SLAM) problem. SLAM has different goals than our work (building an environmental map rather than localizing nodes), but uses similar techniques. Related work in the SLAM literature falls into two categories. In **graph-based SLAM**, the SLAM problem is represented as a graph where each vertex is a pose of the robot at some point in time, and edges represent the observed, noisy spatial relationship between successive poses. Learning the map then involves solving a nonlinear optimization problem that incorporates all spatial relations observed so far. Lu and Milios (1997) approximate the objective with a quadratic function and solve the optimization problem directly. Their solution requires inverting a large matrix at every time step and is costly. To address this problem, most approaches (Duckett et al., 2002; Frese et al., 2005; Olson et al., 2006; Grisetti et al., 2007b,a) solve the problem iteratively, by starting from some (often highly inaccurate) initial solution and improving the solution using Gauss-Seidel relaxation. In our work, internal localization is also formulated as nonlinear optimization problem over a graph of module poses. However, rather than

refining a poor initial estimate, we propose a method to hierarchically build up a solution that is close to the global optimum. Consequently, our approach is less likely to get stuck in local optima that are prevalent in internal localization problems.

Recently, several SLAM approaches have been proposed that strive to recover a **hybrid metric-topological map** that groups semantically similar places, such as those belonging to a single room (Blanco et al., 2006; Zivkovic et al., 2005, 2007; Brunskill et al., 2007). Like our approach, these methods use hierarchical normalized-cut clustering to identify well-connected regions. To avoid the sampling bias that comes from a robot exploring certain places more densely, Zivkovic et al. (2007) propose a resampling method that subsamples the graph vertices according to their distance to the k-th nearest neighbor. Brunskill et al. (2007) propose an incremental version that performs spectral clustering to identify when a robot is starting to explore a new region. In this chapter, we also use normalized cut to perform clustering. However, unlike their work, we do not seek to recover a topological interpretation of the maps. Instead, we use normalized cut as a heuristic to speed up the convergence of iterative methods, and we describe an explicit method to merge the estimates to build global metric localizations.

Declarative specification of inference algorithms has been considered in the literature. In parallel with the work described in this chapter, Funiak et al. (2008a) and Atul (2009) presented a compact implementation of several distributed inference algorithms in P2's language Overlog. They consider two algorithms for probabilistic inference: the robust message passing algorithm (Paskin and Guestrin, 2004b), reviewed in Chapter 2, and a loopy belief propagation algorithm with a random message schedule (Schiff et al., 2007) normalized to ensure convergence. In addition, Atul (2009) presents an implementation of a distributed algorithm for collaborative spam filtering based on the SpamTracker system (Ramachandran et al., 2007), replacing the centralized clustering algorithm inside SpamTracker with a message-passing algorithm, called affinity propagation (Frey and Dueck, 2007). Since the work presented in this chapter focuses on a different kind of a system (modular robots, rather than sensor networks or peer-to-peer networks), the challenges we faced were slightly different than those faced by Funiak et al. (2008a) and Atul (2009). For example, since it is easy to detect neighbors in a modular robot, forming a spanning tree can be accomplished using two simple rules, whereas a simplified version of the spanning tree formation layer in the distributed inference architecture (Paskin et al., 2005) requires 20 rules. Nevertheless, overall, the implementations of inference algorithms in Meld and Overlog are similarly concise.

5.7 Discussion

In this chapter, we examined internal localization in large-scale self-reconfigurable modular robot ensembles using uncertain, local observations. We formulated internal localization as a maximum-likelihood estimation problem and introduced a novel approach which hinges on the selection of an effective ordering of observations using a normalized cut criterion. In combination with closed-form solutions for rigid

alignment and a simple graph abstraction scheme, this approach leads to accurate, scalable solutions. Extensive evaluation of our proposed approach on a test suite of realistic 2D and 3D configurations with up to 10,000 nodes shows that our algorithm outperforms methods in Euclidean embedding, as well as a recent iterative approach in SLAM. We presented a distributed version of our algorithm, whose per-node communication complexity increases only logarithmically with the size of the ensemble. While our distributed algorithm is somewhat complicated, because it integrates several layers over multiple levels of hierarchy, we demonstrated that it can be implemented concisely using a declarative programming language.

Similarly to the robust message passing algorithm, reviewed in Chapter 2, there is an interesting interplay between the inference algorithm and the overlay network(s) it constructs. In the robust message passing algorithm, the cliques and the edges of the network junction tree were determined by the local factors at each node, as well as the link qualities between the nodes. Similarly, in our distributed algorithm for internal localization, the connected components (and thus, the spanning trees) at the lower levels of hierarchy are determined by the normalized cut procedure executed at the higher level. Thus, both the robust message passing algorithm and our localization algorithm are instances where the inference algorithm *affects* the way the overlay network is constructed.

Chapter 6

Collaborative filtering in peer-to-peer networks

In the previous chapters, we have examined distributed probabilistic inference problems, where the nodes computed marginals or the mode of a probability distribution, conditioned on all the observations made in the network. In these problems, the global probability model was known, and the challenge was reasoning about the model efficiently and robustly. Sometimes, however, the global model is not known, and the nodes need to first estimate the parameters of this model from the observations collected across the network, before the model can be used for inference. In this chapter, we consider one instance of such a problem: distributed collaborative filtering. In distributed collaborative filtering, the nodes, such as computers or Internet-enabled cellphones, form a peer-to-peer network and wish to provide recommendations for their users, based on opinions of other like-minded users. Distributed collaborative filtering is challenging, because the standard algorithm for solving the centralized version of the algorithm is purely sequential and difficult to parallelize. Furthermore, the algorithm needs to be robust to node fluctuations, common in peer-to-peer networks, and needs to provide recommendations in an online fashion. In this chapter, we will show that these challenges can be addressed by maintaining inexact replicas of the model parameters at the nodes, continuously synchronized using distributed averaging.

6.1 Distributed collaborative filtering

Many of today's online services, such as Amazon.com, Netflix, or Last.fm, employ a **recommender system** to help the user explore the service's catalogue. A recommender system analyzes the past transactions and provides automated product recommendations that match the user's preferences. Services like Amazon.com extensively use recommender systems to personalize their web sites to each customer's interests (Linden et al., 2003).

A standard approach to recommender systems is **collaborative filtering**. With collaborative filtering, the system provides item recommendations based on the opinions of other like-minded users. The opinions can be obtained either explicitly, by asking each user to rate the items, or implicitly, by measuring the user's interest through timing or click tracking. Each user expresses their opinion of a subset of the items; the user's opinion of a specific item is represented by a **rating**, typically on a numerical scale. Based on the observed ratings from all the users, the system predicts the rating of an item the user has not rated yet. Thus, collaborative filtering is a form of a missing data estimation problem: the system observes the ratings for some user-item pairs and needs to complete the rating matrix for the unobserved pairs.

Most collaborative filtering approaches are centralized, requiring the data from all the users to be stored in a single location. While centralized approaches are often appropriate, some services, such as the music streaming service Spotify, rely on a peer-to-peer network for content distribution. These services could leverage a decentralized recommender system that also runs on the peer-to-peer network, in order to decrease their operating costs. Furthermore, some users, found among the owners of network-connected media centers (such as computers running XBMC[1]), do not have a subscription to a commercial content service. For these users, a centralized recommender system is not commercially viable, as running such a system incurs substantial maintenance costs. Instead, the users may wish to join a distributed collaborative filtering service that runs directly on the user's hardware for free.

A peer-to-peer collaborative filtering algorithm faces several challenges. First, no single node has access to all the data: each node only observes a fraction of user data, and the data cannot be disseminated across the network due to its size. Thus, the nodes need to coordinate their analysis of the data, communicating sufficient statistics about the data they have observed. Second, the algorithm needs to be scalable to thousands of nodes and be robust to network fluctuations: similarly to other situated distributed systems, a peer-to-peer network does not have a fixed node membership—the nodes may join and leave the network at any time. Finally, similarly to the centralized settings, the algorithm needs to be able to provide recommendations online. In most deployments, the algorithm is run over a long period of time, and it needs to adapt to the new ratings entered by the users and to changing user preferences. In this chapter, we will show how all these challenges can be addressed.

6.2 Latent variable models for collaborative filtering

Collaborative filtering is a rich area, with many successful approaches. In our work, we focus on model-based approaches that build a model that relates the user's (hidden) preferences to the observed ratings. We consider two approaches to collaborative filtering: matrix factorization (Sarwar et al., 2000) and the Restricted Boltzmann Machines (Salakhutdinov et al., 2007). Both approaches estimate the hidden parameters of a model, however, the models they employ are different. Matrix factorization is a simple

[1] `http://xbmc.org/`

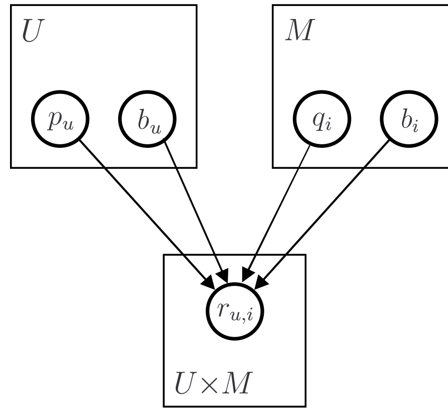


Figure 6.1: The matrix factorization model. Each rating $r_{u,i}$ combines the user and item latent factors p_u and q_i, as well as the biases b_u and b_i. The large boxes indicate the repeated parts of the model.

predictive model, whereas Restricted Boltzmann Machines are a generative probabilistic model. Matrix factorization tends to perform better than Restricted Boltzmann Machines, except for users with very few ratings (10 ratings or less for the Netflix dataset). Matrix factorization is also easier to train, the predictions require substantially less computation, and as we will show later, matrix factorization is easier to parallelize. Nevertheless, we include Restricted Boltzmann Machines here for completeness.

6.2.1 Matrix factorization

Matrix factorization is a simple model that represents a rating as an inner product of two vectors in a real vector space. Let $\mathcal{U} = \{1, \ldots, U\}$ denote the set of U users and $\mathcal{I} = \{1, \ldots, M\}$ denote the set of M items. Each user $u \in \mathcal{U}$ is associated with a vector of parameters $p_u \in \mathbb{R}^F$, and each item $i \in \mathcal{I}$ is associated with the same number of parameters $q_i \in \mathbb{R}^F$; the parameters p_u and q_i are called the **latent factors** of the user and the item, respectively. The item latent factor q_i captures the features, such as the item's genre or the target age group, while the user latent factor p_u specify the user's affinity towards these features. The dot product between p_u and q_i captures the overall affinity of the user towards the item. In addition, each user is associated with bias b_u, and each item is associated with bias b_i. These biases model the systematic tendencies for some users to give higher ratings than others and some items to be rated higher than others. Together, a rating is predicted by the equality

$$r_{u,i} = p_u^\top q_i + b_u + b_i,$$

with the dependency graph shown in Figure 6.1. Throughout this chapter, we denote the item parameters $\{(q_i, b_i) : i \in \mathcal{I}\}$ as θ; we denote the user parameters $\{(p_u, b_u) : u \in \mathcal{U}\}$ as ν.

The model parameters (ν, θ) are assumed to be hidden and need to be estimated from data. The training data consists of the observed ratings $\overline{r}_{u,i}$ for a set of user-item pairs $(u, i) \in \mathcal{K}$; the ratings are assumed

to be collected from the users prior to the experiment, but we do also consider the online setting later in Section 6.5.3. A standard formulation is to minimize the sum of squared prediction errors, penalized by the norm of the parameters:

$$\underset{\mathbf{p},\mathbf{q},\mathbf{b}}{\operatorname{argmin}} \sum_{(u,i)\in\mathcal{K}} f_{u,i}(p_u, b_u, q_i, b_i), \tag{6.1}$$

where

$$f_{u,i} = (\overline{r}_{u,i} - p_u^\top q_i - b_u - b_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

is the penalized square prediction error for a single user-item pair. The vectors **p**, **q**, **b** are the latent factors and biases for all the users and items, and λ is the regularization parameter. With $\lambda = 0$ and $\mathbf{b} = \mathbf{0}$, this optimization problem corresponds to a sparse version of singular value decomposition. Regularization decreases the amount of overfitting, by penalizing the parameter vectors with a large magnitude.

A standard approach to minimize the objective function in (6.1) is to perform stochastic gradient descent on the parameters (ν, θ). Here, we iterate over the observed ratings $\overline{r}_{u,i}$, and in each step, we update the parameters p_u, b_u, q_i, b_i for a single user-item pair $(u, i) \in \mathcal{K}$ using the gradient of $f_{u,i}$:

$$(p_u, b_u, q_i, b_i) \leftarrow (p_u, b_u, q_i, b_i) - \eta \nabla f_{u,i}, \tag{6.2}$$

leaving other parameters intact. The optimization procedure may make several passes over the entire dataset; we will refer to each pass as one iteration. The update (6.2) attempts to move the parameters for the user u and item i closer to the global optimum (6.1). Stochastic gradient descent can be viewed as a crude approximation to the exact gradient descent, which updates all the model parameters at once:

$$\nu \leftarrow \nu - \eta \sum_{(u,i)\in\mathcal{K}} \nabla_\nu f_{u,i} \qquad\qquad \theta \leftarrow \theta - \eta \sum_{(u,i)\in\mathcal{K}} \nabla_\theta f_{u,i} \tag{6.3}$$

In large-scale problems like ours, stochastic gradient descent has been shown to outperform exact gradient descent (Bottou and Bousquet, 2008). Intuitively, a matrix factorization model has many parameters, and there is not enough data to estimate these parameters accurately even with a perfect optimization procedure. Stochastic gradient descent trades the accuracy of the optimization for speed. Experiments have also shown that stochastic gradient descent is less prone to overfitting.

6.2.2 Restricted Boltzmann Machines

The Restricted Boltzmann Machine (RBM) model (Salakhutdinov et al., 2007) also describes the relationship between the user's hidden preferences and the item ratings. However, unlike the matrix factorization model, which contained a separate latent factor for each user, the RBM is a generative probabilistic model that is shared among all the users. The model consists of two sets of random variables. The vector of hidden **latent variables** $\mathbf{H} = (H_1, \ldots, H_F)$ captures the user's preferences; this vector is analogous to

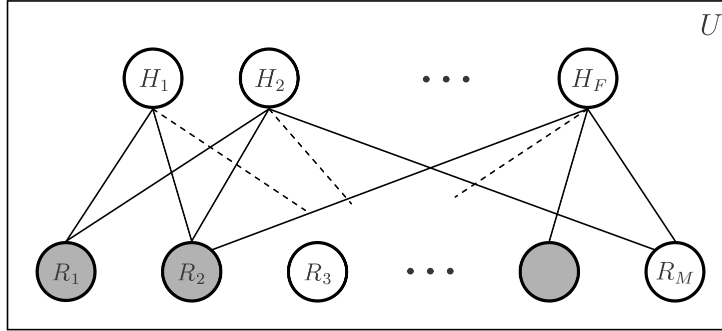


Figure 6.2: The Restricted Boltzmann Machine model. Each variable in the hidden layer $\mathbf{H}$ is connected to the rating variables $\mathbf{R}$. A subset of the ratings (shaded) is observed. The model is the same for all U users (with different sets of observed variables).

the latent factor p_u in matrix factorization, but each H_j is a binary random variable, rather than a real parameter. In addition, for each item $i \in \mathcal{I}$, the model contains a random vector of indicator variables $R_i = \{R_{i,k}\}$, such that $R_{i,k} = 1$ if the user rates item i as k. We denote the concatenation of these rating vectors as $\mathbf{R}$. RBMs model the relationship between the latent variables and the ratings as a fully connected bipartite Markov network, with the latent variables on one side and the ratings on the other, as shown in Figure 6.2. Thus, the model captures the interactions between each rating R_i and each latent variable H_j. The model takes the form of an exponential family distribution

$$p(\mathbf{h}, \mathbf{r}) \propto \exp\big(-E(\mathbf{h}, \mathbf{r})\big), \tag{6.4}$$

where the negative energy term is given by

$$-E(\mathbf{h}, \mathbf{r}) = \sum_{i=1}^{M} \sum_{j=1}^{F} \sum_{k=1}^{K} w_{i,j,k} h_j r_{i,k} + \sum_{i=1}^{M} \sum_{k=1}^{K} r_{i,k} b_{i,k} + \sum_{j=1}^{F} h_j b_j. \tag{6.5}$$

The lower values of the energy $E(\mathbf{h}, \mathbf{r})$ correspond to more likely assignments and vice versa. In (6.5), K is the number of possible ratings (typically 5), and the weights $w_{i,j,k}$ and the biases $b_{i,k}$, b_j are the model parameters. In the rest of this chapter, we denote the union of all the model parameters $\{w_{i,j,k} : i \in \mathcal{I}\}$, $\{b_{i,k} : i \in \mathcal{I}\}$, $\{b_j\}$ as θ; when the model parameters are not clear from the context, we explicitly include them in the probability model as $p(\mathbf{h}, \mathbf{r}; \theta)$.

Given a set of parameters θ, the probabilistic model $p(\mathbf{h}, \mathbf{r}; \theta)$ can be used to make predictions about the items that were not rated by the user. If we knew the values of the latent variables $\mathbf{h}$, the prediction task would be very simple: recall from Section 2.1.2 that in Markov networks, the variables $\mathbf{R}$ are conditionally independent given $\mathbf{H}$ if removing the vertices $\mathbf{H}$ from the graph leaves each R_i in a separate connected component. Since the Markov network in Figure 6.2 is bipartite, the ratings R_i is are indeed conditionally

independent given $\mathbf{H}$, with the conditional distribution given by

$$p(r_{i,k} = 1 \mid \mathbf{h}) = \frac{\exp(b_{i,k} + \sum_{j=1}^{F} w_{i,j,k} h_j)}{\sum_{l=1}^{K} \exp(b_{i,l} + \sum_{j=1}^{F} w_{i,j,l} h_j)}. \tag{6.6}$$

The numerator represents the unnormalized probability of the user rating the item i as k, while the denominator ensures that the probabilities over the different values of k sum to 1. Thus, RBMs are closely related to matrix factorization: the weights $\{w_{i,j,k} : j = 1, \ldots, F\}$ play the role of the item latent factor q_i, and the latent variables $\mathbf{h}$ play the role of the user latent factor p_u. In practice, we do not know the exact values of $\mathbf{h}$, but we can compute the conditional distribution over $\mathbf{H}$, given the observed ratings $\mathbf{R}^{(u)} = \overline{\mathbf{r}}^{(u)}$; this vector is a concatenation of the observed ratings $\{\overline{r}_i^{(u)} : i \in \mathcal{I}^{(u)}\}$ for the subset of items $\mathcal{I}^{(u)} \subseteq \mathcal{I}$ rated by user u. Computing the exact distribution $p(h_j \mid \overline{\mathbf{r}}^{(u)})$ is intractable. A standard approximation (Salakhutdinov et al., 2007; Truyen et al., 2009) is to disregard the unobserved items from the model; the conditional distribution over H_j in this approximate model $\tilde{p}$ can be computed in closed form in time linear in the number of rated items $\mathcal{I}^{(u)}$. The predicted rating is then computed by substituting the probabilities $\tilde{p}(h_j = 1 \mid \overline{\mathbf{r}}^{(u)})$ for h_j in (6.6) and taking the expectation over the random variable R_i. It can be shown that this expectation is a principled approximation to the conditional expectation $\mathbb{E}[R_i \mid \overline{\mathbf{r}}^{(u)}]$ in the exact model (6.4).

Similarly to matrix factorization, the RBM model is trained from data. In matrix factorization, two sets of parameters were estimated: the user latent factors and biases $\{(p_u, b_u)\}$ and the item latent factors and biases $\{(q_i, b_i)\}$. In RBMs, the (user) latent variables $\mathbf{H}$ are random variables in the model, so only the parameters θ need to be recovered. The training data set $\mathcal{D}$ consists of the ratings $\overline{\mathbf{r}}^{(u)}$ for each user u. The observations are assumed to be independent samples from the model (6.4). For a single user u, the likelihood $p(\overline{\mathbf{r}}^{(u)}; \theta)$ describes how well the model parameters fit the observed ratings $\overline{\mathbf{r}}^{(u)}$. In order to estimate the best set of parameters $\hat{\theta}$, a standard approach is to maximize the likelihood of the observed ratings of all the users:

$$\log p(\mathcal{D}; \theta) = \sum_u \log p(\overline{\mathbf{r}}^{(u)}; \theta). \tag{6.7}$$

The log-likelihood (6.7) is maximized approximately using stochastic gradient descent. In each step, we take a subsample (a mini-batch) of users $\mathcal{S} \subseteq \mathcal{U}$ and update the model parameters in the direction of the derivative, averaged over the users in $\mathcal{S}$:

$$\theta \leftarrow \theta + \frac{\eta}{|\mathcal{S}|} \sum_{u \in \mathcal{S}} \Delta^{(u)}, \tag{6.8}$$

where $\Delta^{(u)} = \frac{\mathrm{d} \log p(\mathbf{r}^{(u)}; \theta)}{\mathrm{d}\theta}$ is the contribution of a single user to the gradient. While computing the exact derivative $\Delta^{(u)}$ is intractable, an efficient approximation (Salakhutdinov et al., 2007) based on contrastive divergence (Hinton, 2002) exists. This approximation has an important property: the approximate derivative direction $\tilde{\Delta}^{(u)}_{w_{i,j,k}}$ is non-zero only for items $i \in \mathcal{I}^{(u)}$ rated by the user, and the computation

of $\tilde{\Delta}^{(u)}$ only depends on the biases and on the weights for items rated by user u. This sparsity structure will become important in Section 6.4.1 when evaluating architectures for distributed collaborative filtering.

6.3 Parallel collaborative filtering

The training algorithms, reviewed in the previous section, are centralized in nature. For example, the stochastic gradient descent algorithm in matrix factorization processes one rating at a time, updating the parameters for one user and one item in each step. The algorithm is designed to work on a single processor and does not parallelize naturally. RBM training can be parallelized, by computing the update for each user in a batch on a separate processor, but the parallel implementation still relies on a shared-memory architecture to sum the individual updates quickly. As the recommender systems scale to millions of users, these centralized approaches may take too long to converge. Instead, we wish to design an algorithm that estimates the model parameters in parallel, on a computer cluster. We call this task **parallel collaborative filtering**, to differentiate it from the fully distributed problem, addressed in the subsequent sections.

Parallel collaborative filtering and distributed collaborative filtering have some challenges in common. The nodes need to coordinate in the parameter estimation task, by sending messages over a physical network. Furthermore, each node in parallel collaborative filtering may have access to only a part of the dataset. However, unlike in a peer-to-peer network, the node membership in a computer cluster is stable and predictable (except for the failing nodes). Furthermore, the data management in a computer cluster is centralized: there is a single entity that determines where the data is stored. Thus, parallel collaborative filtering is substantially simpler than a peer-to-peer approach. Nevertheless, we can view a parallel collaborative filtering algorithm as a stepping stone towards a peer-to-peer solution. Since the prediction can be parallelized trivially, we focus here solely on training. We will revisit the complete collaborative filtering problem in the subsequent sections.

There is some prior work on parallelizing stochastic gradient algorithms. Langford et al. (2009) proposed a version of stochastic gradient descent that delays the updates to the parameters. The algorithm can be applied to a variety of communication paradigms. In the context of computer clusters, the algorithm is implemented by partitioning the training set across the cluster and computing the updates (gradients) sequentially on each node. A single node is designated as a state-keeper; this node collects the updates from the other nodes and applies these updates to the current state in a round-robin fashion. With some assumptions on the loss functions $f_{u,i}$, the algorithm is guaranteed to achieve a near-linear theoretical speedup. Unfortunately, to perform well in practice, the algorithm requires that the communication time is negligible, compared to the time required to compute the updates. In our case, the updates are simple to compute, so the algorithm does not scale. Furthermore, because the algorithm designates a specific node as a state-keeper, it does not easily generalize to peer-to-peer networks, where the state-keeper would

constitute a single point of failure.

Consider the following alternative approach. Suppose that each node stores the current estimate, updated using stochastic gradient descent. Node n stores its own copy of the item parameters θ_n; in addition, in matrix factorization, the node stores the user parameters for a subset of the users $\mathcal{U}_n$ assigned to that node: $\{(p_u, b_u) : u \in \mathcal{U}_n\}$. To keep the updates local, the dataset is partitioned horizontally: each user is assigned to a single node, and all the ratings for that user are stored on the node. This partitioning ensures that the update step $\Delta^{(u)}$ in RBMs can be computed locally without coordinating with other nodes. Similarly, since each gradient $\nabla f_{u,i}$ in matrix factorization updates the parameters for a single user u at a time, the user parameters (p_u, b_u) need to be stored only at one node; thus, the user parameters $\{(p_u, b_u) : u \in \mathcal{U}\}$ are partitioned across the cluster.

In each round, the algorithm iterates over the local data, computing the parameters $\hat{\theta}_n^{(t)}$ from some initial estimate $\theta^{(t)}$. Since the estimated parameter vectors $\hat{\theta}_n^{(t)}$ are computed based on different portions of the dataset, they may differ from one node to another. Periodically, the nodes synchronize their state, by computing the average of the shared parameters. The averaging initializes the next round of local updates as

$$\theta^{(t+1)} = \frac{1}{N} \sum_n \hat{\theta}_n^{(t)}, \tag{6.9}$$

where N is the number of nodes. The average (6.9) can be computed for example by aggregating the parameter values over a spanning tree in the cluster.

It is difficult to analyze the behavior of the algorithm formally, but we can provide intuition of how it performs. Suppose that we partition the data set $\mathcal{D}$ into three parts: $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$. The exact stochastic gradient descent iterates over these parts in sequence, starting from some initial iterate $\theta^{(t)}$, and reaching the next iterate $\theta^{(t+1)}$, as shown in Figure 6.3(a). On the other hand, each node in our parallel algorithm starts from the same initial iterate $\theta^{(t)}$, and the algorithm computes three separate iterates $\hat{\theta}_1^{(t)}$, $\hat{\theta}_2^{(t)}$, and $\hat{\theta}_3^{(t)}$, as shown in Figure 6.3(b). These iterates are then averaged to compute the initial iterate $\theta^{(t+1)}$ at the next time step. Since each node receives a fraction of the data $\mathcal{D}_n$, the estimate $\hat{\theta}_n^{(t)}$ at each node are biased; averaging them removes the bias and leads to a more accurate estimate overall. Since the each node processes only a fraction of the data, we may hope to estimate the parameters θ with a similar degree of accuracy as the centralized algorithm, but with substantially less computation performed at each node.

Depending on the number of nodes, our parallel algorithm can be related to different versions of the gradient descent algorithm. When the algorithm runs on a single node, no averaging is performed, and the algorithm executes the centralized stochastic gradient descent. On the other hand, when the data is so finely partitioned that each node receives the data only for one user, (6.9) corresponds to one step of exact gradient descent on the complete dataset $\mathcal{D}$, by the linearity of gradients. Between these two extremes, the algorithm combines local moves (stochastic gradient updates) with global moves (averaging). Thus, as we vary the number of nodes, our algorithm effectively interpolates between the stochastic gradient descent

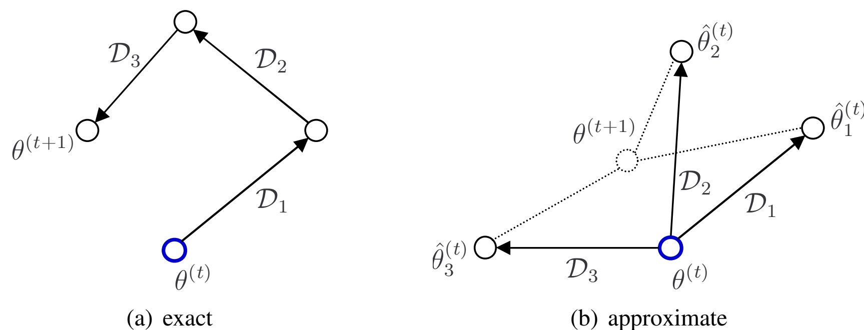
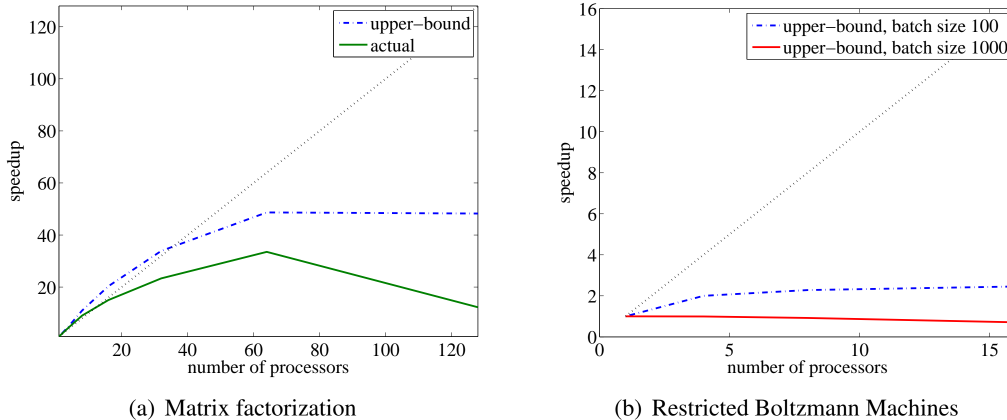


Figure 6.3: Comparing the exact stochastic gradient descent and our parallel approximation. (a) Stochastic gradient descent performs sequence of updates, starting from some initial iterate, marked as bold blue. (b) Our approximation takes parallel steps and averages the result.

and the exact gradient descent algorithms. However, because the averaging divides the estimates by the number of nodes N, the step size taken by the gradient descent in the extreme case is very small, so we would not want to run the algorithm with a very large number of nodes, as we will now show.

To validate our parallel algorithm, we performed two sets of experiments on the Netflix dataset. The dataset contains ratings for 17770 movies from more than 480,000 users, totalling over 100 million ratings, split into a training set and a test set. In the first experiment, we measured the speedup of our algorithm over the centralized stochastic gradient descent. Figure 6.4 shows the time needed to achieve a fixed accuracy relative to the centralized implementation, as we partition the Netflix dataset over a progressively larger number of nodes. In the matrix factorization plot on the left, we show two curves: one is an upper-bound on the speedup that our algorithm can possibly accomplish, disregarding the communication cost. This upper-bound was computed by comparing the number of iterations required to reach the same accuracy in both the centralized and the parallel algorithm. The other curve is the actual speed-up, as measured on a cluster of machines with an 8-core Xeon nodes, using Open MPI. We see that our parallel matrix factorization obtains a near-linear speedup with up to 32 nodes. Subsequently, the upper-bound on the speedup levels off. This behavior is not too surprising: as the number of nodes increases, the amount of data at each node decreases, and each node makes only small progress in one iteration. The actual speed-up is lower, due to the communication overhead associated with averaging over a large number of nodes. In the RBM plot on the right, we show the upper-bound for two batch sizes. We see that RBM training is not amenable to this form of parallelism, achieving only 2.5$\times$ and no speedup for batch sizes 100 and 1000, respectively.

Our approach is fairly insensitive to the frequency of averaging. In a normal execution of the algorithm, each node makes one or more passes over its local dataset before it computes the average. The averaging steps can be infrequent: In Figure 6.5, we show the root-mean-square (RMS) error after a fixed number of iterations over the entire dataset, as we vary the frequency of averaging. We see that with as many as ten passes over the local dataset per one global averaging step, we obtain results that are comparable to the



(a) Matrix factorization

(b) Restricted Boltzmann Machines

Figure 6.4: The speedup as a function of the number of nodes. Here, we compare the time to reach a solution with a fixed accuracy on the test set (0.92 and 0.935 for (a) and (b), respectively). We see that for matrix factorization, the theoretical speed-up levels off at approximately 50×, at which point the local updates become too small. Restricted Boltzmann Machines are not amenable to this form of parallelism.

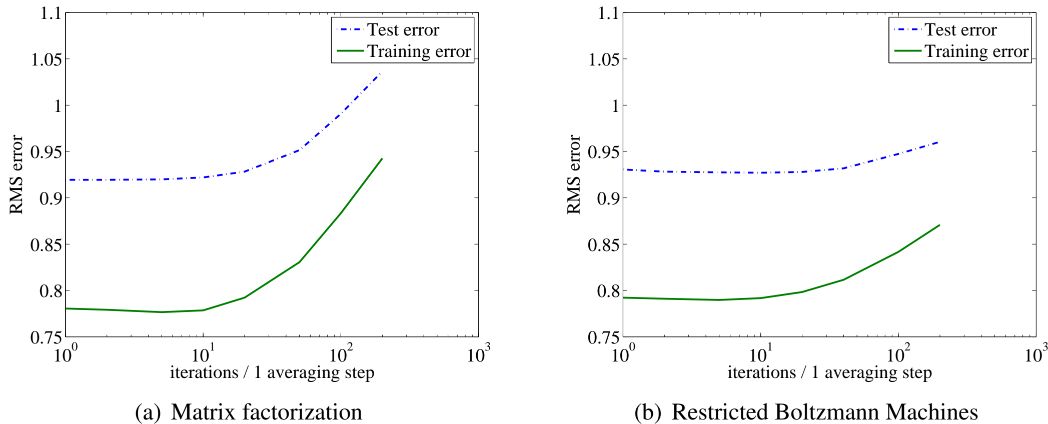
centralized algorithm.

6.4 Distributed collaborative filtering

While parallel collaborative filtering is useful, sometimes there is no centralized entity to collect and process the data. In this case, **distributed collaborative filtering** is preferred, distributing the data storage and the training of model parameters across a peer-to-peer network. When generalizing our parallel collaborative filtering approach to a peer-to-peer setting, we face two challenges. First, as indicated in the previous section, the speedup curve levels off as we partition the data more finely. This behavior can drastically increase the communication complexity in a peer-to-peer network, where each node holds the ratings for a single user. Furthermore, the node membership in a peer-to-peer network is not fixed—nodes may enter and leave the network at any time. A single global aggregation structure (such as a spanning tree) that collects the model parameters from all the nodes may not be stable enough for the network to make progress. In this section, we propose a distributed algorithm that addresses both these challenges.

6.4.1 Super-peer architecture

The parallel algorithm, presented in the previous section, could be directly applied to a peer-to-peer network. Each node would store the ratings for a single user, perform local updates using the data collected for this user, and periodically average the item parameters with other nodes. Such a design is called a **pure peer-to-peer architecture**, because all nodes play an equal role. Kad network and Freenet are examples of systems with pure peer-to-peer architecture. In collaborative filtering, the architecture is appealing



(a) Matrix factorization

(b) Restricted Boltzmann Machines

Figure 6.5: Convergence with a varying frequency of communication. The plots show the root mean square (RMS) error at the end of the experiment with 100 nodes, after each node performs fixed number of iterations over the entire dataset. We see that moderate frequency of averaging is sufficient for accurate results.

for its simplicity and is desirable from the privacy standpoint, because each node shares only the item parameters with other nodes. Unfortunately, as we saw in the previous section, partitioning the data in our algorithm finely across the network does not lead to faster convergence. Instead, the speedup levels off, requiring more passes over the local data to achieve the same accuracy of the solution. Since each pass is accompanied by an averaging step, the communication cost of the pure peer-to-peer architecture is prohibitive.

An alternative to a pure peer-to-peer architecture is a **super-peer architecture** (Yang and Garcia-Molina, 2003), illustrated in Figure 6.6. A super-peer architecture is a hybrid architecture that consists of two kinds of nodes: **super-peers** and clients. Super-peers are peers with high bandwidth, disk space, and computational capabilities. A super-peer acts both as a server to a set of clients, and as an equal peer in the network of super-peers. In our setting, each super-peer executes an algorithm analogous to the parallel algorithm, described in Section 6.3. Each clients connects to a super-peer and uploads its ratings. Typically, the user issues queries to the server, and the server responds with the recommendations. Since each super-peer serves only a fraction of the users, and since the queries are infrequent, the super-peer can easily answer the queries once it has trained the model. Nevertheless, in some applications, it may be desirable to provide predictions even when the client is not connected to the network. In this case, the client can periodically download the model from a super-peer or from another client and perform the predictions locally.

Super-peer architectures sometimes use a complicated protocol for connecting clients to the super-peers. For example, in the FastTrack protocol (used once by the Kazaa peer-to-peer file sharing application), the client disconnects from its super-peer after it has heard the reply to its query and connects to a new super-peer (Liang et al., 2005). By contrast, in our application, it is desirable that each client remains connected

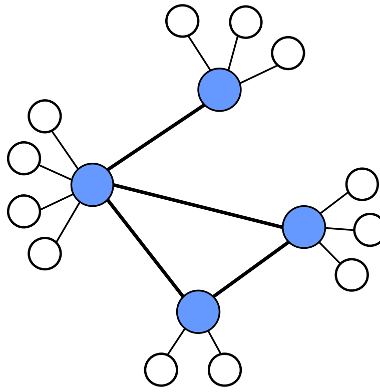
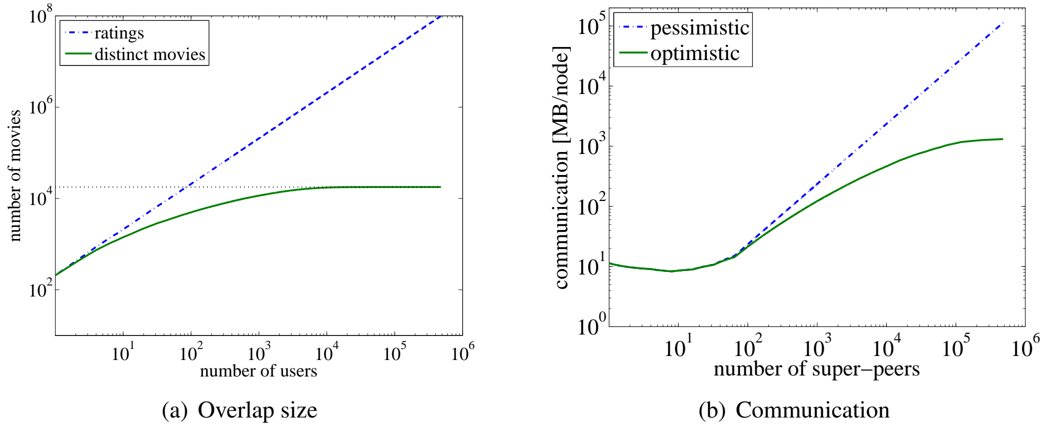


Figure 6.6: A super-peer architecture, where the stable nodes with a fast network connection (shown as larger blue circles) act as servers, and the remaining nodes act as clients.

to a single super-peer, so that the super-peer can continue receiving the new ratings from the client. In order to maintain such a stable mapping, we assign the clients to the super-peers using a distributed hash table (DHT), formed over the super-peers of the network. Conceptually, a distributed hash table provides a single operation: given a key, it maps the key to a super-peer. In our application, the keys are the user ids; each super-peer is also associated with an id. In Chord (Stoica et al., 2001), for example, the super-peers are arranged in a circle; each node is responsible for storing the keys that fall between its id and the id of the node's successor. Standard results (Stoica et al., 2001) guarantee that the assignment will be load-balanced: no server will be associated with more than $O(\frac{\log N}{N})$ fraction of the users.

One of the most important design decisions in applying a super-peer architecture to a given application is to decide how many super-peers there are in a system or, equivalently, what is the number of clients associated with each super-peer. Our approach is communication-bound: each iteration includes an averaging step, which requires each node to communicate its parameter vector to another node in the network. Therefore, the optimal super-peer count will be dictated primarily by the communication complexity of the algorithm, as a function of the number of super-peers.

We can evaluate the communication complexity by referring back to the speedup curve in Figure 6.4. Since the speedup levels off, the algorithm requires many iterations (and hence many rounds of communication) at high super-peer counts, so we would expect the optimal number of super-peers to be relatively low. To refine this intuition, we need to consider one more aspect of the problem: in the Section 6.3, we treated the item parameters as a dense vector. However, as we partition the data more finely, each super-peer may only carry a small portion of items. The averaging algorithm may be able to leverage this sparsity and only communicate a part of the parameter vector. Therefore, we may bound the communication complexity, by assuming that the averaging algorithm will communicate no fewer parameters than the set of items carried at the node and no more than a dense vector. Figure 6.7(a) shows the number of distinct movies rated by a random set of users in the Netflix dataset as we vary the number of users in the set, as well as the total number of ratings. We see that, with 1000 users or more (corresponding to 480



(a) Overlap size

(b) Communication

Figure 6.7: (a) The total number of ratings and the number of distinct movies for an increasing number of users in the Netflix dataset (taken in random order, averaged over 1000 orderings). The difference between the two curves indicates the amount of overlap among the users. The horizontal line indicates the total number of movies in the dataset. We see that, with 1000 users or more, the parameter vector is near-dense. (b) The best-case and the worst-case communication complexity required to achieve progress equivalent to one iteration of stochastic gradient descent in centralized matrix factorization. The pessimistic estimate assumes that all movie parameters need to be communicated by each super-peer, in order to compute their average. The optimistic estimate assumes that averaging can be performed by communicating the parameters for the movies rated by users held at a super-peer. Notice that even with the optimistic estimate, the pure peer-to-peer architecture (marked by the point on the right end of the curve) requires substantially more communication than the super-peer architecture with 100 super-peers or less.

super-peers or less), the parameter vector is nearly dense. Combining this plot with the speedup curve in Figure 6.4(a), we obtain the bounds on the communication required to achieve progress equivalent to one step of centralized stochastic gradient descent. The bounds on the communication costs *per super-peer* for matrix factorization are shown in Figure 6.7(b); the bounds for RBMs are analogous. We see that the optimal number of super-peers is in the range of 50–100. With this many super-peers, we leverage the parallelism in our approach, while keeping the communication costs low.

While keeping the number of super-peers low is important to keep the communication complexity low, it has an important drawback: each super-peer needs to serve many clients. For example, with 100 super-peers and a million users, each super-peer may have to serve 10,000 clients. This ratio is acceptable if the client immediately disconnects from the server after each operation. However, if the client maintains an open connection to the server, the number of open connections on a server would be prohibitive.

We can address this problem by replicating each user's ratings among multiple super-peers. This strategy effectively duplicates the users in the training set, but it does not alter the estimated model parameters, because, under the assumption that the ratings are perfectly replicated across multiple super-peers, the objective functions in (6.1) and (6.7) are simply multiplied by the multiplicity of the ratings, leaving the optimum intact. Importantly, replicating the ratings reduces the number of iterations over the local data,

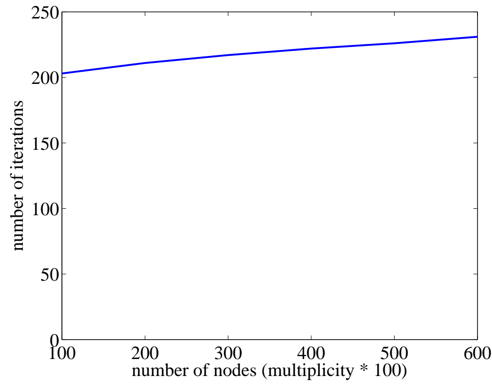


Figure 6.8: The number of iterations, required to achieve a fixed accuracy on the test set (0.92), as we simultaneously vary the number of nodes and the multiplicity of the data. We see that the number of iterations increases very slowly.

required to achieve the same quality of solution. This fact is illustrated in Figure 6.8, where we executed the parallel algorithm from Section 6.3 on a progressively larger set of nodes, keeping the amount of data per node fixed. We see that the number of iterations required to achieve a fixed accuracy increases very slowly. Since the communication complexity is determined primarily by the number of iterations, this strategy substantially reduces the communication complexity of our approach at high super-peer counts: the x axis in Figure 6.7(b) gets effectively scaled by the multiplicity of the ratings. Thus, for example, if each client's data is replicated to 10 super-peers, the optimal number of super-peers becomes 500–1000, significantly lowering the number of clients served by each super-peer.

In practice, it may be difficult to keep all the replicas perfectly synchronized, so we cannot expect to recover the same solution as the centralized stochastic gradient descent. Nevertheless, even when the ratings are not replicated perfectly, we would expect the trained model to be still accurate, because the differences (and hence the perturbations to the problems) are typically not large, and most of the inconsistencies in the replicas will arise due to missing ratings. This means that the replication can be performed with any of the standard approaches. For example, the super-peers could form groups, where only one super-peer in the group participates in the DHT, but all the super-peers in the group serve the users and inform each other of the updated ratings. Alternatively, one could leverage the replication schemes built into the DHTs: most DHTs provide a `put` operation that allows a client to store a key-value pair redundantly on multiple nodes in the network. In our application, the key would be the user id, and the value would be all the ratings associated with this user. As nodes join and leave the network, the structure of the DHT changes, and the nodes may not carry the most recent ratings for each user. This problem can be addressed by attaching a time-to-live value to all the ratings, discarding ratings that have not been refreshed sufficiently recently.

Algorithm 3 DistributedMatrixFactorization($\mathcal{I}$)

1: $(q_i, b_i) \leftarrow$ random, for $i \in \mathcal{I}$
2: **for** $t = 1, \ldots$ **do**
3: **for** new users u **do**
4: $(p_u, b_u) \leftarrow$ initialize(u)
5: **for** $(u, i) \in$ local data K_n **do**
6: $(p_u, b_u, q_i, b_i) \leftarrow (p_u, b_u, q_i, b_i) - \eta \nabla f_{u,i}$
7: Let $\theta_n = \{(q_i, b_i) : i \in \mathcal{I}\}$
8: Update θ_n with an averaging protocol

6.4.2 Outline of the algorithm

Algorithm 3 outlines our approach to distributed matrix factorization (the algorithm for estimating the parameters in the RBM model is analogous). The algorithm mirrors the parallel algorithm for matrix factorization. It starts by initializing the parameter vectors for the movies with random values. In each iteration, the super-peer first checks for any new users and initializes the parameters for each new user as described below. Then the super-peer executes stochastic gradient descent, updating the parameters for each user-movie pair in its local database using the gradient of the partial objective function $f_{u,i}$. The algorithm concludes the iteration by averaging the item parameters θ_n with a distributed averaging protocol. Since the client nodes play only a minor role in our algorithm, we will refer to them simply as "clients," and we will refer to the super-peers as nodes or servers.

6.4.3 User initialization

The first step in each iteration in Algorithm 3 is to initialize the user parameters (p_u, b_u) for the users who have newly joined the super-peer. Typically, in centralized matrix factorization, the user parameters are initialized randomly. We could similarly initialize the user parameters randomly whenever the client connects to a new super-peer. However, this approach has a significant drawback: it will cause a temporary spike in the prediction error for this user. In Section 6.5.2, we will show that these spikes can be significant.

There are two ways to address this problem. One approach is to replicate the user parameters (p_u, b_u) on the client. Periodically, the client connects to its super-peer, and downloads the most recent parameters (p_u, b_u). Then, if the client needs to connect to a new super-peer (either because the old one left the network, or because the DHT structure has changed), the new super-peer will initialize its state with the cached parameters (p_u, b_u) from the client. An alternative method is to initialize the parameters (p_u, b_u) optimally relative to the current item parameters $\{(q_i, b_i) : i \in \mathcal{I}\}$ by solving a least-squares problem similar to (6.1). This initialization is akin to one step of the alternating least squares (ALS) algorithm (Singh and Gordon, 2008), and can be solved in closed form. Perhaps surprisingly, we will demonstrate in Section 6.5.2 that the least-squares initialization does not yield as good results as the replication scheme.

Therefore, a more traditional approach of replication is preferred.

6.4.4 Distributed averaging

A key step in Algorithm 3 is a procedure for averaging the item parameters θ across all the nodes. Averaging the parameters using a global spanning tree is not robust in a peer-to-peer setting: nodes may enter or leave the network at any time, and disrupt the computation in progress. Instead, we rely on an iterative approach to average the item parameters.

We consider three averaging algorithms: the Push-Sum protocol (Kempe et al., 2003), the asynchronous gossip algorithm (Boyd et al., 2006) and a version of the synchronous gossip algorithm from Section 3 of (Boyd et al., 2006). All three algorithms compute the average iteratively: starting from some initial parameter values $\theta_n^{(0)}$, the algorithms update $\theta_n^{(t)}$, so that as $t \to \infty$, each $\theta_n^{(t)}$ converges to the average of the initial values, $\frac{1}{N}\sum_n \theta_n^{(0)}$. All three algorithms converge fast, in the sense that with high probability, the convergence time is logarithmic in the accuracy of the solution and in the size of the network N. However, the algorithms differ in how they represent $\theta_n^{(t)}$ and in the way that the updates are scheduled:

Push-Sum protocol. Push-Sum is a synchronous protocol that operates in distinct rounds. Each node maintains a positive weight $w_n^{(t)}$, initialized to 1, and the local sum $s_n^{(t)}$ (initialized to $\theta_n^{(0)}$). Locally, a node can estimate the average parameters at an time as $\theta_n^{(t)} = s_n^{(t)}/w_n^{(t)}$. In order to update its estimates, the nodes diffuse their local state to other nodes in the network. Specifically, in each round, each node sends a fraction of their weight and the sum to a randomly chosen neighbor in the network. The sum of the incoming weights and sums determine the iterate at the next step. The algorithm is asymmetric: node n may contact node m, but not vice versa.

Asynchronous gossip. Asynchronous gossip (Boyd et al., 2006) is an asynchronous protocol. Nodes communicate at random times (called ticks); assuming instantaneous communication, no two nodes communicate at the same time. In each tick, node n contacts a random neighbor m, chosen from some distribution p, and the two nodes set their values equal to the average of their current values, $\frac{1}{2}(\theta_m^{(t)} + \theta_n^{(t)})$.

Synchronous gossip. Synchronous gossip (Boyd et al., 2006) is similar to asynchronous gossip but dispenses with the assumption of instantaneous communication. The updates are performed in distinct rounds. In each round, the nodes are randomly split into active and inactive nodes: the active nodes initiate the averaging requests with random nodes chosen according to some distribution p, while the inactive nodes are passively awaiting requests. If an inactive node is contacted by exactly one active node, the two nodes set their parameter values equal to the average $\frac{1}{2}(\theta_m^{(t)} + \theta_n^{(t)})$. Otherwise, no averaging is performed. Thus, the algorithm enforces the constraint that each node communicates with at most one other node in each round.

All three algorithms assume that they can sample random nodes according to some distribution p. A simple approach to sample the random nodes is to once again form a DHT over the super-peers. In DHTs, keys are B-bit integers. In order to select a random node, we generate a key uniformly at random in the range $\{0, \ldots, 2^B - 1\}$. Then we issue a DHT lookup to find the node responsible for storing the key k. The resulting distribution p will not be uniform, since some nodes are responsible for a larger range of keys than others. However, the load-balancing properties of DHTs ensure that no p_n is greater than $O(\frac{\log N}{N})$.

Despite the upper-bound on p_n, the Push-Sum protocol may perform poorly. The reason is that, in Push-Sum, the convergence time has a multiplicative factor that is equal to the ratio of N and the *smallest* sampling probability among all the nodes (Kempe et al., 2003). This factor can be very large, since a node in a DHT can be responsible for an arbitrarily small set of keys. Intuitively, a node with a low sampling probability will almost never receive updates from other nodes in the network, and its parameter estimate is determined entirely by the local data, which may lead to overfitting. This concern can be partially alleviated by using the same DHT for sampling in the averaging procedure as the DHT for user assignment. The nodes with a lower sampling probability will be responsible only for a small set of keys. Therefore, the higher error will be exhibited only by a small fraction of the users.

In the synchronous gossip, the convergence is slowed down when several active nodes attempt to communicate with an inactive node. Nevertheless, if the distribution p does not favor a small set of nodes, the requests from active nodes will tend not to collide, and the nodes continue to make progress. In DHTs, the distribution is near-uniform, so we would expect that the algorithm would converge fast. We do not provide formal guarantees (although the convergence time can be bounded using the arguments in Theorem 5 of (Boyd et al., 2006)). Nevertheless, we demonstrate experimentally that the averaging procedure performs well in practice.

We conclude by discussing implementation aspects of the averaging algorithms. The Push-Sum and the synchronous gossip protocols assume that the nodes operate synchronously. In a fully distributed setting, where nodes join and leave the network and communicate at varying speeds, this assumption may be difficult to satisfy. Furthermore, asynchronous gossip assumes that communication is instantaneous—an assumption which is certainly violated in collaborative filtering, where the averaged parameter vectors are large. Therefore, some approximations need to be employed to make the averaging algorithms work in practice

We relax the Push-Sum and the synchronous gossip by executing them in a **weakly-synchronous** manner. The algorithms are implemented in two threads: one thread listens for incoming averaging requests, and the other one issues the requests to random nodes. The requesting thread waits for a pre-specified amount of time between the requests; this effectively limits the rate at which averaging is performed (in practice, the rate is further slowed down by communication and local computation). No effort is made to explicitly synchronize the nodes. The local state θ_n needs to be locked while the model parameters are being updated

locally, so that the local updates do not overwrite the updates received from other nodes. The Push-Sum protocol is effective at decreasing the contention at the averaged model parameters θ. Since the algorithm computes the averages by asymmetrically passing the sum-weight pairs between the nodes, the updates from other nodes can be received in parallel and continue while the model is being updated locally using stochastic gradient descent.

A naive implementation of the asynchronous gossip would lock the local state while the averaged values are communicated between a pair of nodes. Unfortunately, this implementation can lead to deadlocks if two nodes decide to contact each other before completing the transfer (more generally, the deadlock could also occur in larger cycles). Instead we implement only a weak locking mechanism: the contacting node locks its state twice—once when computing a half of the local value and once when adding the received value to the local state. In the meantime, the node is free to receive averaging requests from other nodes. While this implementation does not follow the idealized algorithm described in (Boyd et al., 2006), we demonstrate in Section 6.5.2 that it performs well.

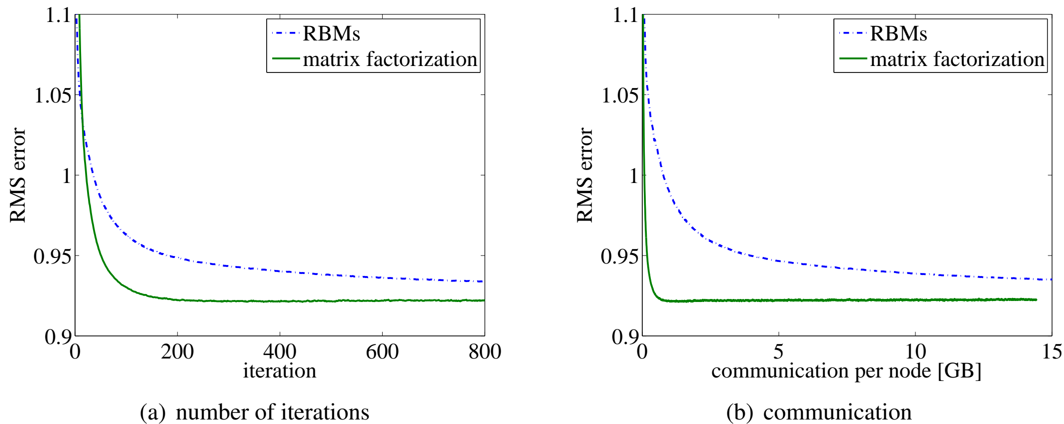
Finally, we have found it useful to limit the number of incoming connections in the Push-Sum and the asynchronous gossip protocols. If this limit is not imposed, a node with a slow connection may end up serving the request of many other nodes, thus slowing down the progress of the network overall. We have occasionally experienced such a slow-down in our experiments, and thus we limit the number of incoming connections to 3.

6.5 Experimental results

We performed a detailed experimental evaluation using a C++ implementation of the algorithms in Section 6.4 in simulation and on PlanetLab, a network of computers world-wide that serves as a testbed for research on computer networks and distributed systems. Since RBM models are substantially larger, are slower to train, and are less accurate, we focused predominantly on distributed matrix factorization. In our matrix factorization experiments, we used $F = 40$ latent variables for each user / item; we set the regularization parameter $\lambda = 0.05$ and the learning rate $\eta = 0.005$. In the RBM experiments, we used $F = 100$ latent variables; we set the step size $\eta = 0.5$ and batch size $S = 1000$.

6.5.1 Simulated experiments

We first evaluate the basic distributed approach from Section 6.4 in a shared memory setting. We uniformly partition the data across a set of working threads, each of which simulates the computation at one node. The nodes operate in perfect synchrony, waiting for all the local iteration to complete before one or more averaging steps are performed. For averaging, we used the synchronous gossip algorithm (Boyd et al.,



(a) number of iterations

(b) communication

Figure 6.9: Convergence of distributed matrix factorization and distributed Restricted Boltzmann Machines as a function of (a) the number of iterations and (b) the total communication sent/received by the averaging algorithm. We see that distributed matrix factorization converges faster and requires less communication.

2006) with uniform sampling. Since our algorithms are communication-bound, we were able to simulate the entire network on a single multicore system.

We first evaluate the convergence of distributed matrix factorization and distributed Restricted Boltzmann Machines in a batch setting. We took the complete Netflix dataset and partitioned the users uniformly across 100 nodes in the network. To improve the convergence, we performed two steps of averaging for each iteration over the local data. Figure 6.9 shows the root mean square (RMS) error as a function of number of iterations and the total communication. We see that the algorithms converge to their centralized counterparts, achieving the RMS error of 0.92 and 0.93 for the matrix factorization and RBMs, respectively. Nevertheless, as mentioned earlier, distributed matrix factorization converges much faster than distributed RBMs. The communication overhead of RBMs is especially noticeable; with RBMs, it takes 37 times more communication than matrix factorization to achieve the same accuracy of solution (0.94). This overhead is in part due to the size of the RBM model (which is roughly $10\times$ larger than the movie parameters employed by matrix factorization) and in part due to the slower convergence of RBMs. Clearly, matrix factorization models are preferred.

In the next experiment, we evaluate how well distributed matrix factorization scales as we increase the number of nodes, while keeping the number of users at each node fixed. We partitioned the complete Netflix dataset into 1000 batches of approximately 490 users each, so that with 1000 nodes, the learning was performed on the complete dataset. As before, we performed two averaging steps for each local iteration. Figure 6.10 shows the number of iterations required to achieve a fixed accuracy of the solution (0.93). We see that the number of iterations remains roughly constant; thus, the solution scales well. Note that, with 100 nodes (or equivalently, one tenth of the dataset), the algorithm does not reach the accuracy 0.93 even after 4000 iterations. This is expected, since having more data helps the algorithm

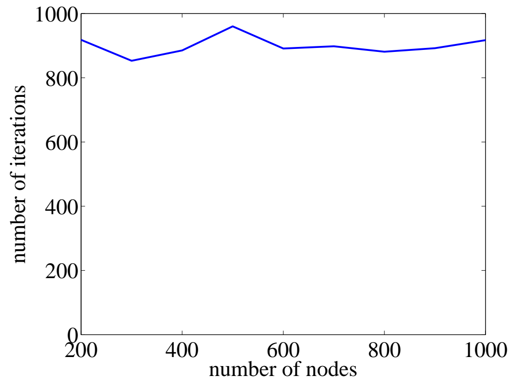


Figure 6.10: The required number of iterations to achieve a fixed accuracy of the solution (0.93), as a function of the number of nodes and the available data.

make better predictions. The number of iterations could be reduced by employing the data replication strategy, described in Section 6.4.1.

6.5.2 PlanetLab experiments

In the second set of experiments, we evaluate convergence and robustness to node loss on a real PlanetLab deployment. We implemented both the server and the client side of the algorithm, where the server makes predictions, and the client uploads its rating to a single server. We used the reference implementation of Chord for user assignment and random host sampling. To speed up the execution of the algorithm, we only use Chord for node lookups; the ratings are stored directly by our application.

We first evaluate the convergence of distributed matrix factorization on the complete Netflix dataset. We selected a subset of 100 nodes that had low average load. At 100 nodes, the processing and the communication times are comparable. Many Planetlab nodes carry a high load-average of 5 or more, and we wanted to avoid having our experiments be CPU-bound. We evaluate the convergence of the distributed matrix factorization using the three distributed averaging protocols, discussed in Section 6.4.4. Figure 6.11(a) shows the convergence of the three versions of the algorithm as a function of the average number of iterations performed locally at each node. We see that the synchronous and the asynchronous gossip perform comparably. In particular, we see that the locking approximation we have made in the asynchronous gossip protocol does not negatively impact its performance. The Push-Sum protocol appears to overfit the local data; its test error increases over time. This intuition is supported by Figure 6.11(b), which shows that the training error for Push-Sum is lowest among all three algorithms.

Figure 6.11(c) shows the convergence of the three averaging algorithms when the frequency of averaging is doubled, performing two averaging steps for each local update. Averaging more frequently can improve the convergence, by bringing the distributed algorithm closer to our parallel approach. The convergence of the asynchronous gossip improves with more frequent averaging, while the convergence of the Push-Sum

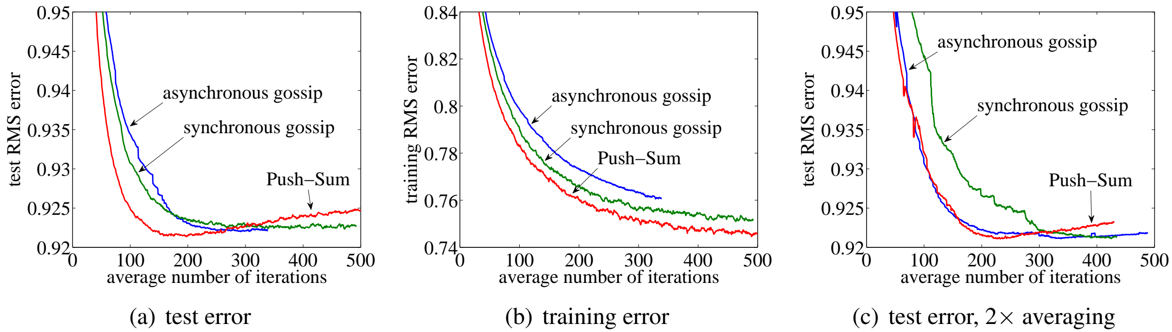


Figure 6.11: Convergence of distributed matrix factorization, using the three averaging algorithms, discussed in Section 6.4.4: (a) the test error, (b) the training error, (c) the test error when we perform two averaging steps for each local iteration.

protocol remains the same (the protocol still overfits, increasing the test error over time). Surprisingly, in our real deployment, synchronous gossip converges slower with more averaging. This behavior is repeatable, but we were unfortunately not able to identify its source. For the remaining experiments, we use the synchronous gossip protocol at 1 averaging step per local iteration, in order to keep the PlanetLab experiments aligned with the simulated experiments. However, in practice, asynchronous gossip protocol may be preferred.

In the second experiment, we evaluate the robustness of the algorithm to fluctuations in the network. As before, we start with a group of 100 nodes that learn from the entire Netflix dataset. After 85 iterations, the servers are disconnected one-by-one, Figure 6.12(a) shows the number of live servers. Each client periodically tests the liveness of the assigned server and if needed, transfers its ratings to a new location. Figure 6.12(b) shows the test RMS error for three versions of the algorithm: i) random initialization of the user parameters (p_u, b_u) without replication, ii) initialization using a replica of (p_u, b_u) stored at the client, and iii) initialization by solving the least-squares problem for (p_u, b_u) without replication. We see that replicating the latent variables smoothes out the spikes in the errors, caused by user relocation. The least-squares initialization of (p_u, b_u) also has a smooth behavior, but yields a slightly higher RMS error. To explain this behavior, we plot the training error of the three initializations in Figure 6.12(c). Note that the least-squares initialization has the lowest training error among the three initialization methods; thus, the least-squares initialization tends to overfit. This is similar to the behavior of the centralized Alternating Least Squares algorithm, which we have found to also overfit the training data, despite heavy regularization.

6.5.3 Online experiments

In most deployments, collaborative filtering runs continuously, making predictions for new movies and learning from the entered ratings. In the last set of experiments, we simulate such a behavior, using the

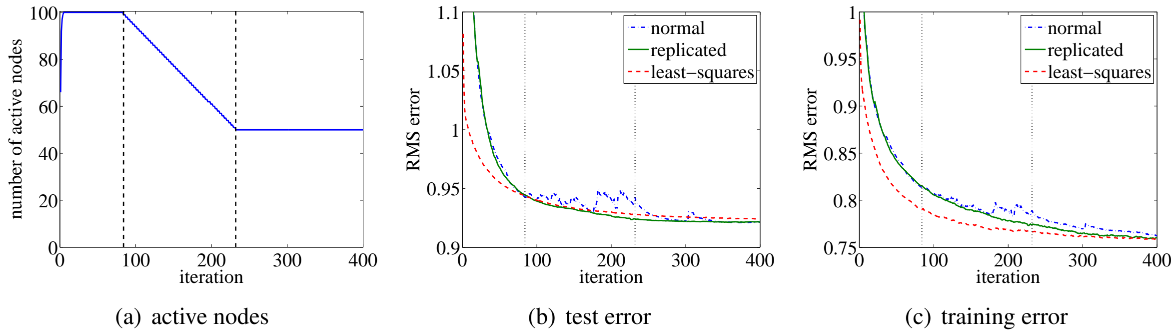
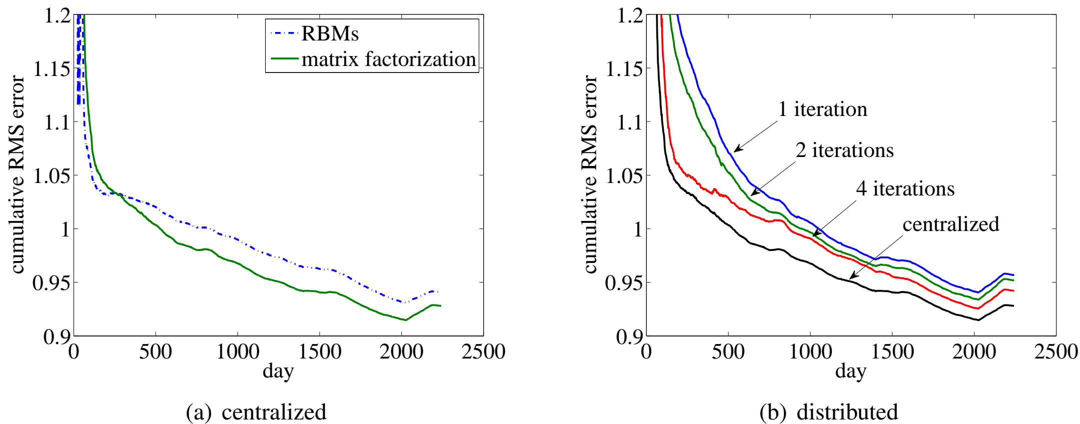


Figure 6.12: The robustness experiment: (a) the number of active nodes; (b) the test RMS error; (c) the training RMS error. We see that random initialization (labeled as normal) is sensitive to network fluctuations. Replicated initialization performs best.

time stamps present in the Netflix dataset. We evaluate both the centralized and the distributed algorithm. The centralized algorithm is executed as follows: the algorithm starts with a random model for all the movies. For each day in the dataset, we compute the prediction for movies rated in that day using the current model. Then, we add the ratings to the training set and retrain the model. To speed up the convergence, we only train on a random portion of the dataset in each day (one tenth of the current training set for matrix factorization, 10 batches of 1000 users each for Restricted Boltzmann Machines). As before, matrix factorization can greatly benefit from proper initialization of user latent variables. Since each user starts with no ratings at the beginning of the experiment, we initialize the bias of user u to the average bias over all active users at the time when the first rating of user u is observed. This initialization ensures that the users receive meaningful predictions as soon as they start using the system. Figure 6.13(a) shows the cumulative RMS error as a function of time. We see that while the RMS error is initially high, it continues to decrease as the algorithm learns a progressively more accurate model. From day 2000 to day 2200, there is a sudden spike in the prediction error. This spike does not correspond to previously documented fluctuations in the Netflix dataset, and we continue to evaluate its source. Nevertheless, the final cumulative error in this (much harder) online setting is close to the batch settings.

For the distributed experiments, the experiment is performed similarly. Due to the large communication cost of the RBM models, we only evaluated distributed matrix factorization. For each day, we upload new ratings from the clients and compute the prediction error on the new ratings using the current model. To initialize the bias for a new user, we only use the local user biases; no coordination is performed to determine the average user bias across all the nodes.[2] Then we perform a number of learning steps; each learning step consists of one iteration over the local data and one averaging step, as shown in Algorithm 3. Figure 6.13(b) shows the convergence of distributed matrix factorization for different numbers of learning steps per day. The third curve corresponds to a two-week long execution of the algorithm on 100 PlanetLab nodes. We see that performing multiple learning steps per day is key to obtaining accurate results.

[2]In a separate simulated experiment, we have found that using global biases leads to only a minor improvement early in the experiment and no improvement in the second half of the experiment.



(a) centralized (b) distributed

Figure 6.13: Convergence of incremental learning. (a) The cumulative test error over time for matrix factorization and Restricted Boltzmann Machines. The final results are comparable to the batch setting. (b) The cumulative test error for incremental distributed matrix factorization. We show the convergence for a different number of learning steps (iterations) per day. The accuracy is somewhat worse than the centralized algorithm, but still acceptably low.

The overall results are slightly worse than the centralized algorithm, but still competitive with the batch algorithm.

6.6 Related work

Centralized collaborative filtering is a rich area. The algorithms for collaborative filtering span two broad categories. The **memory-based** approaches interpolate a user-item rating by identifying similar users or similar items in the database. A standard approach (Resnick et al., 1994; Breese et al., 1998; Herlocker et al., 1999; Bell and Koren, 2007) is to predict the rating as a weighted sum of other users' ratings, by estimating the correlation between the users. The **model-based** approaches, on the other hand, construct a model that lets the system make predictions solely based on the user's own ratings. In particular, the latent-variable approaches (Sarwar et al., 2000; Salakhutdinov et al., 2007; Singh and Gordon, 2008; Koren, 2009), some of which were considered in this chapter, estimate the user's hidden preferences and train a model that relates these preferences to the rating of each movie. The two categories are complementary; memory-based and model-based approaches make different prediction errors, some of which can be eliminated by combining the approaches using an **ensemble** method. The winners of the Netflix prize used an ensemble of many memory-based and model-based algorithms (Bell et al., 2009), but in practice, a few well-selected strategies are sufficient (Bell et al., 2009).

The PipeCF algorithm (Han et al., 2004) is an early example of a distributed memory-based approach to collaborative filtering. Similarly to other memory-based algorithms, the algorithm predicts an active user's rating by taking a weighted average of ratings made by a set of neighbors. In the case of PipeCF, the

neighbors are the users who share a common rating (an item-rating pair) with the active user. Each node stores the list of users for a subset of the item-rating pairs. Similarly to our approach, this data is stored in a distributed hash table (DHT). However, the DHT serves a slightly different purpose: in our approach, DHT ensures near-uniform user assignment and data replication while in PipeCF, the DHT also provides efficient lookups.

Due to the difficulty of finding similar users in a distributed manner, most distributed collaborative filtering approaches are model-based. Miller et al. (2004) propose a simple approach called PocketLens that computes a similarity between each pair of items. Each node stores its own ratings and approximates the similarity between the items rated at the node and all other items, by communicating the ratings from a subset of other users. Miller et al. (2004) analyze different architectures for discovering other users, including a Gnutella-like flooding scheme and examining the neighborhood of a node in a DHT. One downside of their model is that a user may not be able to predict the rating for all items, since the estimated similarity matrix has some zero entries. Also, it is not clear how PocketLens compares to the state-of-the-art collaborative filtering approaches. By comparison, our approach can make predictions for any user-item pair and performs well on a modern dataset.

Another approach that examines the pairwise similarity between items was proposed by Wang et al. (2006a). Their approach uses a probabilistic relevance model based on (Ponte and Croft, 1998) that only takes into account the positive evidence for the interest of the user (for example, when the user plays a song). With some simplifying assumptions on the model, the training procedure amounts to counting the co-occurrences of items in users' download lists. These co-occurrence counts are stored in a so-called buddy table, which is located at the node that holds the multimedia file for the item. Each item is assumed to be stored at exactly one location, which is not realistic in peer-to-peer settings. By comparison, in our approach, the model storage is orthogonal to the content delivery service, with each super-peer storing the model for all items and a subset of the users.

Distributed model-based approaches have also been examined in the context of privacy-preserving collaborative filtering. The goal of privacy-preserving collaborative filtering is to provide recommendations to individual users without revealing their ratings to a central server or other nodes in the network. Only the aggregates, such as the aggregate gradient or the item latent factors q_i, are revealed. For example, Canny (2002a) propose an algorithm that performs a partial SVD decomposition of the rating matrix. The underlying optimization problem is similar to ours; however, their algorithm fills in default values for the missing ratings, rather than solving the sparse problem directly. Since the number of values that need to be filled in is large (for example, the sparsity of the Netflix dataset is almost 99%), the approach can introduce inaccuracies in the solution. Canny (2002b) addressed the problem of missing values by using a probabilistic model, trained using Expectation Maximization (Dempster et al., 1977). We do not address the privacy-preserving aspects of the distributed collaborative filtering problem and instead rely on trusted client software. However, our approach is robust to unstable node membership, and we demonstrate competetive performance on a modern dataset. Furthermore, the privacy-preserving schemes can

be communication intensive, requiring hundreds of megabytes of communication per client per iteration even on small datasets. By comparison, our approach is substantially less costly, requiring only tens of megabytes of communication per iteration for the Netflix dataset.

6.7 Discussion

In this chapter, we presented a distributed approach to collaborative filtering, based on matrix factorization and Restricted Boltzmann Machines. We proposed a simple algorithm for parallel collaborative filtering that partitions the data across the nodes and periodically averages the shared model parameters. The algorithm naturally extends to the peer-to-peer setting using a two-level, super-peer architecture, in which a subset of stable, fast peers train the model and answer queries for the clients. The resulting solution employs a combination of a distributed averaging algorithm and a distributed hash table for assigning users to super-peers and sampling random neighbors in the averaging algorithm. We performed a detailed experimental evaluation of the approach both in simulation and on PlanetLab, demonstrating convergence, scalability, robustness to node loss, and feasibility of online operation. Our distributed algorithm performs a modest amount of communication and attains results that are comparable to a centralized algorithm.

The algorithm presented in this chapter once again illustrates an important point: inconsistency arises naturally in distributed inference algorithms. Unlike in Chapter 3, where inconsistencies among the nodes' beliefs arose primarily due to the communication delays and network partitions, inconsistency in distributed collaborative filtering was introduced by the algorithm in its normal course of operation as each node updated its own estimates using the data held locally at the node. The inconsistency resolution mechanism in this chapter was simpler than the one in Chapter 3 (distributed averaging vs. the optimized conditional alignment), but it once again played a key role: it allowed the nodes exchange information about their observations, substantially increasing the prediction accuracy.

Chapter 7

Conclusions

This thesis studied inference problems, where nodes need to integrate observations from across a network to reason accurately. By building upon graphical models and overlay networks, we developed scalable solutions to one general problem and three important applications. Our solutions address the challenges of many today's networks, including the unreliable communication among the nodes and constrained computational resources.

7.1 Summary of the thesis

In this section, we will briefly summarize the approach and the key contributions presented in this thesis. We considered several types of graphical models, each of which has some benefits for distributed inference:

- *Decomposable models*, the most powerful representation considered in this thesis, have several properties that make them very useful in distributed inference. A decomposable model supports *marginalization by pruning*: the marginal distribution can be obtained simply by removing the leaf cliques and the associated marginals from the model. This pruning operation can be implemented in a distributed manner, by relating the cliques collected by a node to the global decomposable model. A decomposable model also supports a *projection* operation for approximating complex distributions with simple, sparser ones. This operation is particularly useful when exact inference is intractable.

- *Dynamic Bayesian networks* represent a system, whose state changes over time. Dynamic Bayesian networks are useful for distributed inference, because they capture the structure in the system transitions and observations. This structure permits each node to reason only about the variables that are most relevant for it, substantially reducing the memory requirements of the algorithm.

- *Markov networks* can represent any system that can be described as a factorized probability model. Markov networks can be used to efficiently implement operations, such as gradient descent. A Markov network can be also used to heuristically reason about the best ordering of observations to condition on, by identifying the *weak regions* in the graph.

- *Latent variable models*, including matrix factorization and Restricted Boltzmann Machines, are simple examples of graphical models. They are useful for distributed parameter estimation, because they support efficient local updates using stochastic gradient descent.

We also considered several types of overlay networks for large-scale coordination among the nodes:

- A *spanning tree* is the simplest kind of an overlay network that can be used to compute simple aggregates, such as sums. Typically, a spanning tree utilizes strong communication links between neighboring nodes. Sometimes, multiple spanning trees are constructed by an inference algorithm to perform multiple aggregation tasks in parallel, and the inference algorithm can *affect* how these trees are constructed.

- A *network junction tree* is a spanning tree, whose nodes and edges are annotated with sets of variables. A network junction tree can be used to implement dynamic programming algorithms with structure similar to the sum–product algorithm. A network junction tree can be *optimized* to trade the communication and computational cost of the algorithm.

- A *distributed hash table* is a distributed data structure for peer-to-peer networks that supports efficient look-ups of keys and reliable storage of key-value pairs.

Based on these graphical models and overlay networks, we developed algorithms that address several important problems. We first examined the distributed filtering problem and a powerful approach, assumed density filtering, that periodically projects the prior distribution to a family of tractable distributions, represented by a decomposable model with a fixed structure. We showed that assumed density filtering can be reduced to a sequence of static estimation steps, solved using the robust message passing algorithm. In the presence of unreliable communication or high latency, the nodes may not be able to condition their estimates on all observations in the network. Hence, the beliefs at the nodes may be conditioned on different evidence and no longer form a consistent global probability distribution over the state space. We showed that such inconsistencies can lead to poor results when nodes attempt to combine their estimates. We developed an algorithm, optimized conditional alignment (OCA) that lets the nodes recover from the inconsistency; the algorithm obtains a global distribution as a product of conditionals from local estimates and optimizes over different orderings to select a global distribution of minimal entropy. We also proposed a more global optimization approach that provides accurate solutions even when the communication network is highly fragmented. We demonstrated that our algorithm converges to the centralized B&K98 algorithm and is robust to communication failures.

We then turned to our first application, distributed localization of networked cameras. We showed that non-linearities in the camera model can lead to very poor solutions. To address this problem, we developed two novel techniques, the relative over-parameterization (ROP) and hybrid conditional linearization. We demonstrated that, with these two techniques, the complicated distributions of the camera poses can be represented with a simple Gaussian. We also demonstrated that assumed density filtering yields excellent results, hence our distributed filtering algorithm can be used for accurate camera localization.

In the next chapter, we addressed a challenging problem of internal localization in large-scale modular robots. We proposed a distributed algorithm that exploits the structure of the connectivity graph to hierarchically partition the problem into smaller subproblems, which are then solved in a bottom-up fashion. Our algorithm constructs many spanning trees in parallel; these spanning trees are used to identify the partitions and compute the partial solutions simultaneously in different parts of the ensemble. We demonstrated that the algorithm substantially improves the accuracy over the state-of-the-art algorithms in sensor networks and simultaneous localization and mapping, and its communication complexity per node increases only logarithmically with the ensemble size. We also presented a concise implementation of the algorithm in a declarative programming language.

Finally, we examined an important problem, collaborative filtering in peer-to-peer networks. We observed that a standard stochastic gradient descent algorithm can be approximated with a parallel solution that partitions the data across a moderate number of super-peers and periodically averages the shared model parameters. Using a distributed averaging protocol, we obtained a robust algorithm that achieves accurate results even when a large fraction of nodes leave the network. A key component of our algorithm was a distributed hash table, for assigning the clients to super-peers and selecting random neighbors in the averaging algorithm. We demonstrated that our approach achieves results of quality comparable to the centralized algorithm, scales, and can provide recommendations in an online fashion.

7.2 Common themes

While most of the inference problems addressed in this thesis are specific to their application domains, they share a number of common themes. These themes reflect recurring patterns of the problems or their solutions and are likely to generalize to other application domains.

Overlay networks. The algorithms presented in this thesis make an extensive use of overlay networks. Overlay networks greatly simplify the design of an inference algorithm, by providing a useful abstraction for large-scale coordination among nodes. An overlay network can play one of several roles. An overlay network can serve as an interface between the graphical model and the physical network. When the graphical model and the physical network are not directly related, as was the case in the robust message passing algorithm and our approximate distributed filter, the overlay network can provide sufficient information for the inference algorithm to perform operations on the

graphical model. An overlay network can also be used to perform data aggregation, as we have seen in modular robot localization. Aggregating data along a spanning tree is very efficient, as the message complexity of aggregation scales linearly with the number of nodes. Finally, overlay networks can provide addressing, greatly simplifying the lookup of nodes in the network. Such lookups can serve several purposes, including sampling of nodes in the network and assigning the data to the nodes.

While overlay networks provide a useful abstraction for distributed inference algorithms, they incur communication overhead, as the overlay network needs to be maintained to account for changing link qualities and node failures. Even in the absence of failures, an overlay network can incur communication cost, as the nodes may need to monitor the status of their neighbors. Nevertheless, this communication cost can be often offset by the fact that overlay networks can optimize the communication paths for the inference algorithm. For example, a network junction tree can be optimized to lower the expected cost of the inference messages (Paskin et al., 2005).

Inconsistency. When the nodes reason about overlapping sets of variables, their estimates may become inconsistent. In distributed inference algorithms, inconsistency can take on one of two forms: the nodes may disagree about a distribution, or they may disagree about the parameters of a non-probabilistic model. Inference in dynamical systems is a canonical example of how inconsistencies may occur: in the presence of network partitions and communication delays, an individual node may not condition its belief on all the observations in the network, and once the filter advances to the next time step, its ability to incorporate the remaining observations is irreversibly lost. We have examined inconsistencies in the context of our approximate distributed filter in Chapter 3, where the nodes obtain a set of marginals over a fixed external junction tree. Inconsistencies may also arise in other filtering tasks, such as multi-robot SLAM as discussed in the next section, and in model learning, where each node learns a probabilistic model, such as a mixture of Gaussians or a thin junction tree (Chechetka and Guestrin, 2007). Inconsistencies may also be introduced by the training algorithm itself when adjusting the model parameters using local data, as discussed in Chapter 6. In all these cases, inconsistencies can degrade the performance of the algorithm: in the presence of network partitions and communication delays, a distributed filter may fail to incorporate the observations from other nodes, while a learning algorithm may overfit to the local data.

In filtering, a simple approach to address inconsistency is to replay the data: the nodes maintain a history of estimates, and if they detect inconsistency, they can retract some of the estimates and restart the inference process. Data replay has been employed in the work of Rosencrantz et al. (2003a) in order to reduce the variance of importance weights in their particle filter. Unfortunately, it can be very costly to replay the data in our distributed filter: doing so requires the nodes to re-execute the filter for many time steps, incurring both computational and communication cost. Furthermore, in order to replay the data, the nodes have to store a large amount of information (the history of the estimates), which can be prohibitive on devices with little memory.

In Chapter 3, we have described two algorithms to resolve inconsistencies in distributed filtering: optimized conditional alignment (OCA) and jointly optimized alignment. Both these algorithms are instances of a broader methodology of finding a consistent distribution from a set of inconsistent marginals. In this methodology, we first select a family of candidate approximate distributions (in OCA, these are the distributions that can be written as a product of a root marginal and the conditionals for the remaining cliques, while in jointly optimized alignment, these are all the decomposable models with the given external junction tree). We then optimize an objective function over the members of this family, such as the entropy or the sum of the reverse KL divergences. The objective function should capture certain intuitive properties: it should not disregard information captured by the observations, it should match the inconsistent marginals, and if the marginals are consistent to begin with, it should recover the exact distribution. The quality of the resulting distribution is then evaluated empirically. Unlike the data replay, this approach permits an online operation, where the nodes only store the latest estimates and do not need to store past observations or exchange training data.

Data redundancy. In most applications, the solution quality degrades smoothly as we decrease the number of observations. For example, as shown in Section 4.5.2, even with a small number of observations made by each camera, simultaneous localization and tracking can recover accurate estimates of the camera locations. Similarly, in Section 5.5.4, we have shown that a modular robot can accurately localize its components even if a fraction of the observations are missing. Finally, as shown in the scaling experiments in Section 6.5.1, with as little as one fifth of the entire Netflix dataset, matrix factorization is able to make accurate recommendations. Thus, we see that even if some of the observations are left out, an inference algorithm can still recover an accurate solution.

Data redundancy is particularly important in distributed systems. In the presence of communication failures, a distributed algorithm may not be able to condition on all the observations in the network. For example, if a sensor node is located far from other nodes, its communication links to the rest of the network may be weak, and the node may not be able to communicate information about its observations to other nodes at all times. Furthermore, when some of the nodes leave the network or fail, their observations are effectively removed from the network. Data redundancy ensures that the remaining nodes can still recover accurate estimates.

In Section 1.1, we have outlined a number of challenges, common to situated distributed systems: scalability, constrained resources, unreliable communication, and unstable node membership. In each of the problems considered in this thesis, we have addressed many, but not necessarily all of these challenges. In our work on distributed filtering, we have focused on communication failures, including network partitions. We have not considered node failures, which can be thought of as a special case of network partitions, but we ensured scalability to moderately-sized networks. In our work on localization of modular robots, we focused on scalability to large ensembles. In modular robots, scalability is very important, because modular robots are envisioned to consist many fine-grained components (Goldstein and Mowry,

2004). Robustness to communication and node failures is also important, but we did not address it in this thesis, as we expect the algorithm to be executed over a short period of time when node failures are less likely. Finally, in distributed collaborative filtering, we focused on scalability to large networks and on robustness to nodes entering or leaving the network. We did not address the privacy aspects of the problem and instead rely on trusted client software.

7.3 Directions for future research

We have explored some problems, but distributed reasoning under uncertainty is a growing field with many interesting opportunities. We review some research directions that are related to the work presented in this thesis.

7.3.1 Distributed filtering with minimal communication

While our algorithm in Chapter 3 addressed the general problem of filtering in dynamic Bayesian networks and is robust to network partitions, it is still communication-intensive. The key issue is that the algorithm requires many rounds of communication (epochs) in each estimation step to perform well (around 20 in our experiments). In fast dynamic environments, where each time step only takes a fraction of a second, communicating so much may be prohibitive. Ideally, each node would communicate only a few times in each time step, transmitting only the most relevant information. After all, in most rounds, little information is introduced into the estimates.

Improving the communication complexity of our approach would require a better integration of the robust message passing algorithm within our distributed filter and improvements to the robust message passing algorithm itself. For example, currently, our algorithm discards the messages of robust message passing after the state is advanced to the next time step; thus, the robust message passing algorithm starts with a clean state in each round. This is problematic, because the messages carry important priors, which are required to transfer the observation likelihoods from one clique to another. A better approach would advance not only the cliques held locally at the node, but also the cliques in the messages. An open question is how to perform alignment in this setting. Furthermore, currently, the robust message passing algorithm sends messages indiscriminately, as long as new information is received that may change the message. A better approach would be send messages *adaptively*, propagating messages only if they introduce significant changes in the beliefs at other nodes. Such a message passing scheme was considered by Paskin (2003) in the context of the decomposable model updates. An open problem is how to implement this scheme with robust factors.

7.3.2 Adaptive DBN filtering

In our distributed filtering algorithm, presented in Chapter 3, the approximation structure (the external junction tree) was selected in advance using domain-specific knowledge. For example, in simultaneous localization and tracking for camera networks, the junction tree was selected to capture the dependences among nearby cameras. While this approach can be fully automated, the best approximation structure may depend on the observations, which are not known ahead of time (Paskin, 2004). In this case, it is desirable to employ an adaptive technique for DBN filtering that adjusts the approximation structure over time, selecting a external junction tree that minimizes the projection error.

A simple approach to adaptive DBN filtering is to borrow the ideas from the Chow–Liu algorithm (Chow and Liu, 1968) for learning tree-structured Bayesian networks. Suppose that we wish to approximate an arbitrary distribution $p(\mathbf{x})$ with a tree Bayesian network G over the same set of variables (that is, a Bayesian network where each variable except for the root has exactly one parent). Then the KL divergence from the exact distribution p to the best approximation q with structure G decomposes linearly over the edges of G:

$$D(p \,\|\, q) = -\sum_{\{a,b\} \in E_G} I(X_a; X_b) + c, \tag{7.1}$$

where $I(X_a; X_b)$ is the mutual information between the variables X_a and X_b in p, and c is a constant independent of q. The best Bayesian network can be then computed by searching for the tree G that minimizes (7.1); the search can be performed very efficiently using a maximum spanning tree algorithm. Therefore, the Chow–Liu algorithm minimizes the KL divergence simultaneously over the different choices of the graph structure G and different choices of the conditional probability distributions for each variable, given its parent in G. The Bayesian network G can then be readily converted to a junction tree, where each clique has at most two variables.

In the context of distributed filtering, the Chow–Liu algorithm can be employed in the prediction/roll-up/projection phase. Recall that in our approximate distributed filter, each node locally computes the marginal of the approximate prior distribution over its cliques, $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \,|\, \bar{\mathbf{z}}^{(1:t)})$. An adaptive algorithm can compute such marginals for several pairs of variables $\{X_a^{(t+1)}, X_b^{(t+1)}\}$. These marginals can be used to evaluate the mutual information between $X_a^{(t+1)}$ and $X_b^{(t+1)}$ in the approximate prior. The nodes need to then coordinate the search to maximize the sum of mutual informations over the edges of the Bayesian network G. The primary challenge of this approach is to ensure that the nodes obtain a valid tree structure G in the face of communication delays and node failures.

7.3.3 Distributed generalized belief propagation

Generalized belief propagation (GBP) (Yedidia et al., 2000, 2005) is an approximate inference algorithm that simultaneously generalizes both loopy belief propagation and the sum–product algorithm. Approxi-

mate algorithms for distributed inference are desirable, due to their lower computational and communication complexity. However, standard loopy belief propagation tends to perform relatively poorly for models with many tight loops and conflicting interactions (Yedidia et al., 2000). Generalized belief propagation addresses this problem by reasoning in terms of larger sets of tightly related variables, called **regions**. The regions are arranged in a directed acyclic graph called the **region graph**, where the root vertices are associated with largest regions and the descendants are their subsets. For example, in the SLAT application, each root region would contain the poses of nearby cameras, along with the object variables at two consecutive time steps, while the descendants would contain the poses of subsets of these cameras. There are several versions of generalized belief propagation algorithms; one of them (Yedidia et al., 2005, Appendix E) sends messages between the regions in a manner analogous to the sum–product algorithm and is best suited for distributed inference.

There are two key challenges in distributing a GBP algorithm: **region placement** and **region graph co-optimization**. We will first discuss region placement, since it is conceptually simpler. A natural way to distribute a GBP algorithm is to assign each region to one node and send messages between two nodes if the nodes carry regions that are adjacent in the region graph. Each such message incurs communication cost that is determined by the per-unit communication cost $d(m, n)$ between m and n (based on the link quality) and the size of the message. In region placement, we wish to optimize the assignment of the regions to nodes, so that the overall communication complexity of the GBP algorithm is minimized. This task can be formulated as an optimization problem:

$$\min_{\mathbf{y}} \sum_{\{i,j\} \in E_G} w_{i,j} d(y_i, y_j), \tag{7.2}$$

where E_G are the edges of the region graph G, $w_{i,j}$ is the size of the message between regions i and j, and y_i and y_j are the nodes assigned to regions i and j, respectively. Since a region graph is a type of an overlay network, this problem is an instance of overlay network optimization; one example of such an optimization for junction trees was demonstrated in (Paskin et al., 2005).

Region placement is an instance of a **graph labeling** problem: we label each vertex i of the region graph with a color that represents a node. Depending on the assumptions on the networking, we can further distinguish two cases. In **semi-metric labeling**, the algorithm can only communicate their inference messages $\mu_{i\to j}$ directly between the nodes that are assigned the regions i and j. With **metric labeling**, on the other hand, we assume multi-hop communication, where the inference messages can be forwarded over multiple nodes to lower the communication cost. Boykov et al. (1999) propose two methods, called $\alpha\beta$-expansion and α-expansion that locally optimize a version of the problem (7.2) for the semi-metric and the metric case, respectively. These methods can likely be implemented in a distributed manner. An open question is whether these methods perform well when we place additional constraints, such as when we fix one or more regions to a specific node or when we impose a budget on the amount of computation performed at each node.

Typically, the root regions are chosen heuristically, and the remaining regions are determined based on the root regions using the Kikuchi construction (Kikuchi, 1951; Yedidia et al., 2005). A more principled approach is to select the root regions adaptively, in a way that leads to the most accurate solution. A centralized version of this problem was considered by Welling (2004), who proposed an algorithm, called region graph pursuit. In distributed settings, however, the region pursuit must take into account not only the accuracy of the solution, but also the link quality, hence we need to **co-optimize** the region graph with respect to both the probability distribution and the network. Co-optimization is very important, because it lets us trade the accuracy of the solution for the communication and computational cost.

7.3.4 Dynamic localization in modular robots

In Chapter 5, we discussed the static localization problem for a modular robot ensemble. In this problem, the ensemble is assumed to be motionless, so that the algorithm can continue to make progress. While this assumption is reasonable during initial stages or when the ensemble is temporarily paused, in practice, the ensemble will be often in motion as it adapts its shape. Therefore, we need an algorithm that performs **dynamic localization**, localizing the ensemble over time. Dynamic localization can be viewed as a sequence of static inference tasks, where the estimates from the current time step are used as an initial solution for the next time step. Alternatively, dynamic localization can be viewed as an inference task with a temporal model, such as a DBN. The former formulation is challenging, because local approaches, such as the gradient descent, may be too slow to update the pose estimates effectively. The latter formulation may be even harder, due to the scale of the system.

It may be possible to solve the dynamic localization task by combining our approach for static localization with an effective iterative solution, such as (Grisetti et al., 2007b). In particular, while we observed that the algorithm (Grisetti et al., 2007b) was prone to local optima with a bad initialization, it could be very effective at maintaining the estimate in the dynamic setting. Recall from Section 5.2.2 that the algorithm of Grisetti et al. (2007b) parameterizes the module poses along a tree; any prediction errors between a pair of modules adjacent in the ensemble are propagated along the unique path between these modules in the tree. In their (centralized) algorithm, these updates are processed sequentially. When generalizing the algorithm to the distributed setting, the updates between multiple pairs of modules will need to be performed concurrently. Analyzing the behavior of such concurrent updates may require tools from control theory. Furthermore, in a distributed setting, the tree in the algorithm (Grisetti et al., 2007b) is a type of an overlay network: it defines the communication topology for passing the inference messages between the nodes in the network. Currently, their method uses a simple procedure that forms this tree as a spanning tree of the ensemble connectivity graph. It may be necessary to once again optimize this overlay network to better match the graph topologies found in modular robots.

7.3.5 Alignment for multi-robot SLAM

At the end of Section 3.6, we briefly mentioned the multi-robot SLAM problem. Recall that in the multi-robot SLAM problem, a group of robots simultaneously localizes itself and maps an unknown environment. In landmark-based SLAM, the map is represented as a set of landmarks—features that are easy to detect, such as corners or trees. Each robot maintains a distribution over its pose and the locations of the landmarks; this distribution is typically approximated as a Gaussian and represented in a sparse form. For example, in Sparse Extended Information Filter (SEIF) (Thrun et al., 2004), the distribution is represented as a Gaussian in the information form with a sparse information matrix. In Thin junction tree filter (TJTF) (Paskin, 2003), the distribution is represented as a decomposable model, with structure determined online.

A key challenge in multi-robot SLAM is map merging. Periodically, the robots are in communication range and need to merge their maps, so that they can benefit from each other's observations. This is a difficult problem, because, similarly to our distributed filtering algorithm, the nodes will have conditioned their beliefs (maps) on different evidence and will have made different sparsifying approximations in their beliefs by the time they merge the maps. Our joint alignment formulation could be used for map merging, too. Recall that in joint alignment, we start with a set of inconsistent marginals $\{\pi_i(\mathbf{x}_{C_i})\}$ and determine a single, global distribution $\tilde{p}(\mathbf{x})$, by minimizing the sum of reverse KL divergences,

$$\tilde{p}_{\text{KL2}}(\mathbf{x}) = \operatorname*{argmin}_{q(\mathbf{x}):q\models T} \sum_{i\in N_T} D(q(\mathbf{x}_{C_i}) \,\|\, \pi_i(\mathbf{x}_{C_i})).$$

In map merging, we could perform a similar optimization problem, but align the sparse distributions $\pi_i(\mathbf{x}_{V_i})$ that represent the individual robot's map:

$$\tilde{p}_{\text{KL2}}(\mathbf{x}) = \operatorname*{argmin}_{q(\mathbf{x}):q\models T} \sum_{i\in N_T} D(q(\mathbf{x}_{V_i}) \,\|\, \pi_i(\mathbf{x}_{V_i})).$$

Here, V_i is the set of variables (landmarks and robot poses) known to robot i. This optimization problem is tractable when q is represented as a thin junction tree T (that is, junction tree with small cliques). This optimization problem does not solve the data association problem that arises when the robots explore overlapping regions; the algorithm proposed by Thrun and Liu (2003) may generalize to this setting.

7.4 Concluding remarks

Graphical models are now widely accepted as a useful formalism to represent and reason about structure in large systems. This thesis explored the relevance of graphical models in distributed inference. Often, graphical models provide a means to scale up a distributed algorithm to large systems. However, graphical models often need to be complemented with other mechanisms to coordinate a large set of nodes. Overlay

networks provide one such mechanism. This thesis explored, in part, the relationship between graphical models and overlay networks on a set of applications and general inference problems. Sometimes, graphical models and overlay networks are largely unrelated: for example, in our distributed collaborative filtering algorithm, the overlay network was constructed indiscriminately over all super-peers in the network, and the graphical model did not affect the overlay network at all. But sometimes, the overlay network *is* a graphical model, as is the case in the robust message passing algorithm, and the communication structure and the system model are tightly integrated.

Overlay networks also helped in achieving robustness in face of communication failures or nodes entering and leaving the network. After all, overlay networks are designed to adapt themselves to changing network conditions. However, as we saw in Chapters 3 and 6, they may need to be combined with other mechanisms, such as optimized alignment or distributed averaging. These general mechanisms often carry from one problem to another.

Finally, we hope that by addressing realistic, novel applications, we convey that distributed reasoning under uncertainty is of practical interest to a broad audience of researchers in both academia and industry. As the field matures, the number of interesting applications will continue to grow.

Bibliography

Aberer, K. (2001). P-Grid: A self-organizing access structure for P2P information systems. In *Cooperative Information Systems*, pages 179–194. Springer.

Ashley-Rollman, M. (2010). personal communication.

Ashley-Rollman, M., Lee, P., Goldstein, S. C., Pillai, P., and Campbell, J. (2009). A language for large ensembles of independently executing nodes. In *International Conference on Logic Programming*, pages 265–280. Springer.

Atul, A. (2009). Compact Implementation of Distributed Inference Algorithms for Network. Master's thesis, University of California, Berkeley.

Bell, R. M., Bennett, J., Koren, Y., and Volinsky, C. (2009). Million Dollar Programming Prize.

Bell, R. M. and Koren, Y. (2007). Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *Seventh IEEE International Conference on Data Mining, ICDM 2007*, pages 43–52, Washington, DC, USA. IEEE Computer Society.

Bertsekas, D. P. and Tsitsiklis, J. N. (1997). *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific; 1st edition (January 1997).

Biswas, P., Lian, T. C., Wang, T. C., and Ye, Y. (2006). Semidefinite programming based algorithms for sensor network localization. *ACM Transactions on Sensor Networks (TOSN)*, 2(2):188–220.

Blanco, J. L., Gonzalez, J., and Fernandez-Madrigal, J. A. (2006). Consistent observation grouping for generating metric-topological maps that improves robot localization. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, ICRA 2006*, pages 818–823.

Bottou, L. and Bousquet, O. (2008). The Tradeoffs of Large Scale Learning. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 161–168, Cambridge, MA. MIT Press.

Boyd, S., Ghosh, A., Prabhakar, B., and Shah, D. (2006). Randomized gossip algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530.

Boyen, X. and Koller, D. (1998). Tractable inference for complex stochastic processes. In *Proceedings of the 14th Annual Conference on Uncertainty in AI*, pages 33–42.

Boykov, Y., Veksler, O., and Zabih, R. (1999). Fast Approximate Energy Minimization via Graph Cuts. *IEEE International Conference on Computer Vision*, 1:377.

Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence*, pages 43–52.

Brunskill, E., Kollar, T., and Roy, N. (2007). Topological mapping using spectral clustering and classification. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2007*, pages 3491–3496.

Butler, Z., Corke, P., Peterson, R., and Rus, D. (2004). Networked cows: Virtual fences for controlling cows. In *WAMES 2004*, volume i. Citeseer.

Bychkovskiy, V., Megerian, S., Estrin, D., and Potkonjak, M. (2003). A Collaborative Approach to In-Place Sensor Calibration. In *Proceedings of the Second International Workshop on Information Processing in Sensor Networks (IPSN)*, volume 2634 of *Lecture Notes in Computer Science*, pages 301–316. Springer–Verlag Berlin Heidelberg.

Canny, J. (2002a). Collaborative filtering with privacy. In *Proceedings of 2002 IEEE Symposium on Security and Privacy*, pages 45–57. IEEE Computer Society.

Canny, J. (2002b). Collaborative filtering with privacy via factor analysis. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 238–245, New York, NY, USA. ACM.

Ceri, S., Gottlob, G., and Tanca, L. (1989). What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166.

Chechetka, A. and Guestrin, C. (2007). Efficient Principled Learning of Thin Junction Trees. In *In Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada.

Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

Cowell, R., Dawid, P., Lauritzen, S., and Spiegelhalter, D. (1999). *Probabilistic Networks and Expert Systems*. Springer, New York, NY.

Crick, C. and Pfeffer, A. (2003). Loopy Belief Propagation as a Basis for Communication in Sensor Networks. In Meek, C. and Kjaerulff, U., editors, *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, San Francisco. Morgan Kaufmann Publishers, Inc.

Dean, T. and Kanazawa, K. (1990). A model for reasoning about persistence and causation. *Computational Intelligence Journal*, 5(3):142–150.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical*, 39(1):1–38.

Dewey, D. J., Ashley-Rollman, M. P., De Rosa, M., Goldstein, S. C., Mowry, T. C., Srinivasa, S. S., Pillai, P., and Campbell, J. (2008). Generalizing metamodules to simplify planning in modular robotic systems. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1338–1345.

Djugash, J., Singh, S., and Grocholsky, B. (2008). Decentralized mapping of robot-aided sensor networks. In *IEEE International Conference on Robotics and Automation, ICRA 2008.*, pages 583–589.

Djugash, J., Singh, S., and Grocholsky, B. (2009). Modeling mobile robot motion with polar representations. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2096–2101. IEEE.

Duckett, T., Marsland, S., and Shapiro, J. (2002). Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287–300.

Fox, D., Burgard, W., Kruppa, H., and Thrun, S. (2000). A Probabilistic Approach to Collaborative Multi-Robot Localization. *Autonomous Robots*, 8(3):325–344.

Frese, U., Larsson, P., and Duckett, T. (2005). A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*, 21(2):1–12.

Frey, B. J. and Dueck, D. (2007). Clustering by Passing Messages Between Data Points. *Science*, 315(5814):972–976.

Funiak, S., Atul, A., Chen, K., Hellerstein, J. M., and Guestrin, C. (2008a). Distributed inference with declarative overlay networks. Technical Report UCB/EECS-2008-135, EECS Department, University of California, Berkeley.

Funiak, S., Guestrin, C., Paskin, M., and Sukthankar, R. (2006a). Distributed Inference in Dynamical Systems. In *Advances in Neural Information Processing Systems 19*. MIT Press.

Funiak, S., Guestrin, C., Paskin, M., and Sukthankar, R. (2006b). Distributed localization of networked cameras. In *The Fifth*

International Conference on Information Processing in Sensor Networks, IPSN 2006., pages 34–42.

Funiak, S., Pillai, P., Ashley-Rollman, M. P., Campbell, J. D., and Goldstein, S. C. (2009). Distributed Localization of Modular Robot Ensembles. *The International Journal of Robotics Research*, 28(8):946–961.

Funiak, S., Pillai, P., Rollman, M. A., Campbell, J., and Goldstein, S. (2008b). Distributed Localization of Modular Robot Ensembles. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland.

Gilpin, K., Kotay, K., Rus, D., and Vasilescu, I. (2008). Miche: Modular Shape Formation by Self-Disassembly. *The International Journal of Robotics Research*, 27(3-4):345–372.

Girard, J.-Y. (1987). Linear Logic. *Theoretical Computer Science*, 50(1):1–101.

Goldstein, S. C. and Mowry, T. (2004). Claytronics: A Scalable Basis for Future Robots. In *Robosphere*.

Grime, S. and Durrant-Whyte, H. F. (1994). Data fusion in decentralized sensor networks. *Control Engineering Practice*, 2(5):849–863.

Grisetti, G., Grzonka, S., Stachniss, C., Pfaff, P., and Burgard, W. (2007a). Efficient estimation of accurate maximum likelihood maps in 3D. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3472–3478. IEEE.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007b). A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA.

Guestrin, C., Bodik, P., Thibaux, R., Paskin, M., and Madden, S. (2004). Distributed Regression: an Efficient Framework for Modeling Sensor Network Data. In *Information Processing in Sensor Networks (IPSN)*, Berkeley.

Han, P., Xie, B., Yang, F., and Shen, R. (2004). A scalable P2P recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 27(2):203–210.

Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, page 237. ACM.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Ihler, A. T., Fisher, J. W., Moses, R. L., and Willsky, A. S. (2004). Nonparametric belief propagation for self-calibration in sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, pages 225–233, New York, New York, USA. ACM Press.

Jensen, F. V. and Jensen, F. (1994). Optimal junction trees. In Mantaras, R. L. and Poole, D., editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 360–366, San Francisco. Morgan Kaufmann Publishers.

Jorgensen, M. W., Ostergaard, E. H., and Lund, H. H. (2004). Modular ATRON: modules for a self-reconfigurable robot. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2068–2073.

Julier, S. and Uhlmann, J. (1997). A New Extension of the Kalman Filter to Nonlinear Systems. In *Proceedings of AeroSense: The 11th International Symposium Aerospace/Defense Sensing, Simulation and Controls*, Orlando, FL.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45.

Kantor, G. and Singh, S. (2002). Preliminary results in range-only localization and mapping. In *Proceedings 2002 IEEE International Conference on Robotics and Automation*, number May, pages 1818–1823. Ieee.

Karagozler, M. E., Goldstein, S. C., and Reid, J. R. (2009). Stress-driven MEMS assembly + electrostatic forces = 1mm diameter robot. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2763–2769.

Kempe, D., Dobra, A., and Gehrke, J. (2003). Gossip-Based Computation of Aggregate Information. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 482+, Washington, DC, USA. IEEE

Computer Society.

Khan, S. and Shah, M. (2003). Consistent labeling of tracked objects in multiple cameras with overlapping fields of view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1355–1360.

Kikuchi, R. (1951). A Theory of Cooperative Phenomena. *Physical Review*, 81(6):988+.

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, New York, NY, USA. ACM.

Krumm, J., Harris, S., Meyers, B., Brumitt, B., Hale, M., and Shafer, S. (2000). Multi-camera multi-person tracking for EasyLiving. In *Proceedings Third IEEE International Workshop on Visual Surveillance*, pages 3–10. IEEE Computer Society.

Langford, J., Smola, A., and Zinkevich, M. (2009). Slow Learners are Fast. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C. K. I., and Culotta, A., editors, *Advances in Neural Information Processing Systems 22*, pages 2331–2339.

Lauritzen, S. L. and Spiegelhalter, D. J. (1988). Local Computations with Probabilities on Graphical Structures and their to Expert Systems. *Journal of the Royal Statistical Society*, 50(2):157–224.

Lerner, U. N. (2002). *Hybrid Bayesian Networks for Reasoning about Complex Systems*. PhD thesis, Stanford University.

Liang, J., Kumar, R., and Ross, K. (2005). The KaZaA overlay: A measurement study. *Computer Networks Journal (Elsevier)*, pages 1–25.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, (February).

Loo, B. T., Condie, T., Garofalakis, M., Gay, D. E., Hellerstein, J. M., Maniatis, P., Ramakrishnan, R., Roscoe, T., and Stoica, I. (2006). Declarative networking: language, execution and optimization. In *Proc. of the 2006 ACM SIGMOD int'l conf. on Management of data*, pages 97–108, New York, NY, USA. ACM Press.

Lu, F. and Milios, E. (1997). Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333–349.

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. (2002). Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. ACM.

Mantzel, W. and Baraniuk, R. (2004). Distributed camera network localization. In *Asilomar Conference on Signals, Systems and Computers*, pages 1381–1386. Ieee.

Manyika, J. and Durrant-Whyte, H. (1995). *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Miller, B. N., Konstan, J. A., and Riedl, J. (2004). PocketLens: Toward a personal recommender system. *ACM Transactions on Information Systems (TOIS)*, 22(3):437–476.

Murphy, K. and Weiss, Y. (2001). The Factored Frontier Algorithm for Approximate Inference in DBNs. In *Proceedings of the Seventeenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 338–378, San Francisco, CA. Morgan Kaufmann.

Newman, P. and Leonard, J. (2003). Pure Range-Only Sub-Sea SLAM. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '03)*, pages 1921–1926.

Nistér, D. (2001). *Automatic dense reconstruction from uncalibrated video sequences*. PhD thesis, Royal Institute of Technology KTH.

Nodelman, U., Shelton, C. R., and Koller, D. (2002). Continuous time Bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 378–387.

Olson, E., Leonard, J., and Teller, S. (2004). Robust range-only beacon localization. In *Proceedings of IEEE Conference on*

Autonomous Underwater Vehicles, volume 31, pages 66–75.

Olson, E., Leonard, J., and Teller, S. (2006). Fast iterative alignment of pose graphs with poor initial estimates. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2262–2269.

Paskin, M., Guestrin, C., and Mcfadden, J. (2005). A robust architecture for distributed inference in sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 55–62.

Paskin, M. A. (2003). Thin Junction Tree Filters for Simultaneous Localization and Mapping. In Gottlob, G. and Walsh, T., editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1157–1164, San Francisco, CA. Morgan Kaufmann Publishers.

Paskin, M. A. (2004). *Exploiting Locality in Probabilistic Inference*. PhD thesis, University of California, Berkeley.

Paskin, M. A. and Guestrin, C. E. (2004a). A Robust Architecture for Distributed Inference in Sensor Networks. Technical Report IRB-TR-03-039, Intel Research.

Paskin, M. A. and Guestrin, C. E. (2004b). Robust probabilistic inference in distributed systems. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 436–445. AUAI Press.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pfeffer, A. and Tai, T. (2005). Asynchronous Dynamic Bayesian Networks. In *Proceedings of the Twenty-First Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 467–476, Arlington, Virginia. AUAI Press.

Pillai, P., Campbell, J., Kedia, G., Moudgal, S., and Sheth, K. (2006). A 3D Fax Machine based on Claytronics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4728–4735.

Pollefeys, M., Van Gool, L., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J., and Koch, R. (2004). Visual Modeling with a Hand-Held Camera. *International Journal of Computer Vision*, 59(3):207–232.

Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '98*, pages 275–281, New York, New York, USA. ACM Press.

Rahimi, A., Dunagan, B., and Darrell, T. (2004). Simultaneous Calibration and Tracking with a Network of Non-Overlapping Sensors. In *Computer Vision and Pattern Recognition (CVPR) 2004*, pages 187–194.

Ramachandran, A., Feamster, N., and Vempala, S. (2007). Filtering spam with behavioral blacklisting. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 342–351, New York, NY, USA. ACM.

Rasti, A. H., Stutzbach, D., and Rejaie, R. (2006). On the Long-term Evolution of the Two-Tier Gnutella Overlay. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–6. IEEE.

Reshko, G. (2004). Localization Techniques for Synthetic Reality. Master's thesis, Carnegie Mellon University.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM.

Rosencrantz, M., Gordon, G., and Thrun, S. (2003a). Decentralized Sensor Fusion with Distributed Particle Filters. In *Proceedings of the Conference on Uncertainty in AI (UAI)*, Acapulco, Mexico.

Rosencrantz, M., Gordon, G., and Thrun, S. (2003b). Locating Moving Entities in Dynamic Indoor Environments with Teams of Mobile Robots. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia.

Roufas, K., Zhang, Y., Duff, D., and Yim, M. (2001). Six Degree of Freedom Sensing for Docking Using IR LED Emitters and Receivers. In *ISER '00: Experimental Robotics VII*, pages 91–100, London, UK. Springer-Verlag.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann machines for collaborative filtering. In *Proceedings*

of the International Conference on Machine Learning, volume 24, pages 791–798.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000). Application of dimensionality reduction in recommender systema case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop*. Citeseer.

Schiff, J., Antonelli, D., Dimakis, A. G., Chu, D., and Wainwright, M. J. (2007). Robust Message-Passing for Statistical Inference in Sensor Networks. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, pages 109–118.

Schmidt, R. and Aberer, K. (2006). Efficient Peer-to-Peer Belief Propagation. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, pages 516–532.

Shang, Y., Ruml, W., Zhang, Y., and Fromherz, M. P. J. (2003). Localization from mere connectivity. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 201–212. ACM Press New York, NY, USA.

Shi, J. and Malik, J. (2000). Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

Singh, A. P. and Gordon, G. J. (2008). A Unified View of Matrix Factorization Models. In *Machine Learning and Knowledge Discovery in Databases, European Conference (ECML/PKDD).*

Smith, R. C. and Cheeseman, P. (1986). On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68.

Soatto, S. and Perona, P. (1998). Reducing "Structure from Motion": a general framework for dynamic vision. 1. Modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(9):933–942.

Stauffer, C. and Tieu, K. (2003). Automated multi-camera planar tracking correspondence modeling. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2003*, pages 259–266. IEEE Computer Society.

Stoica, I., Morris, R., Karger, D., Kaashoek, F., and Balakrishnan, H. (2001). Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160.

Taylor, C., Rahimi, A., Bachrach, J., Shrobe, H. E., and Grue, A. (2006). Simultaneous localization, calibration, and tracking in an ad hoc sensor network. In *The Fifth International Conference on Information Processing in Sensor Networks, IPSN 2006.*

Thrun, S. (2006). Affine structure from sound. In *Advances in Neural Information Processing Systems*, volume 18, pages 1353–1360.

Thrun, S. and Liu, Y. (2003). Multi-Robot SLAM With Sparse Extended Information Filers. In *Proceedings of the 11th International Symposium of Robotics Research (ISRR'03)*, Sienna, Italy. Springer.

Thrun, S., Liu, Y., Koller, D., Ng, A. Y., Ghahramani, Z., and Durrant-Whyte, H. (2004). Simultaneous Localization and Mapping with Sparse Extended Information Filters. *The International Journal of Robotics Research*, 23(7-8):693–716.

Tomasi, C. and Kanade, T. (1992). Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154.

Truyen, T. T., Phung, D. Q., and Venkatesh, S. (2009). Ordinal Boltzmann Machines for Collaborative Filtering. In *25th Conference on Uncertainty in Artificial Intelligence (UAI).*

Umeyama, S. (1991). Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4):376–380.

Wan, E. A. and Van Der Merwe, R. (2002). The Unscented Kalman Filter for Nonlinear Estimation. In *Proceedings of Adaptive Systems for Signal Processing, Communications, and Control Symposium (AS-SPCC) 2000*, pages 153–158. IEEE.

Wang, J., Pouwelse, J., Lagendijk, R. L., and Reinders, M. J. T. (2006a). Distributed collaborative filtering for peer-to-peer file sharing systems. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1026–1030, New York, NY, USA. ACM.

Wang, Z., Zheng, S., Boyd, S., and Yez, Y. (2006b). Further Relaxations of the SDP Approach to Sensor Network Localization. Technical report, Stanford University.

Welling, M. (2004). On the choice of regions for generalized belief propagation. *Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence (UAI-04)*.

Whitehouse, K., Karlof, C., Woo, A., Jiang, F., and Culler, D. (2005). The effects of ranging noise on multihop localization: an empirical study. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 10, Piscataway, NJ, USA. IEEE Press.

Wu, S.-P., Vandenberghe, L., and Boyd, S. (1996). MAXDET Software for Determinant Maximization Problems.

Yang, B. B. and Garcia-Molina, H. (2003). Designing a super-peer network. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60.

Yang, P., Freeman, R. A., Gordon, G., Lynch, K., Srinivasa, S., and Sukthankar, R. (2008). Decentralized Estimation and Control of Graph Connectivity in Mobile Sensor Networks. In *American Control Conference*.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2000). Generalized Belief Propagation. In *Advances in Neural Information Processing Systems (NIPS)*, volume 13, pages 689–695. MIT Press.

Yedidia, J. S., Freeman, W. T., and Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312.

Yim, M., Duff, D. G., and Roufas, K. D. (2000). PolyBot: a modular reconfigurable robot. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 514–520 vol.1.

Zaniolo, C., Arni, N., and Ong, K. (1993). Negation and Aggregates in Recursive Rules: the LDL++ Approach. In *Proceedings of the Third International Conference on Deductive and Object-Oriented Databases*, pages 204–221.

Zivkovic, Z., Bakker, B., and Krose, B. (2005). Hierarchical map building using visual landmarks and geometric constraints. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2480–2485.

Zivkovic, Z., Booij, O., and Kröse, B. (2007). From images to rooms. *Robotics and Autonomous Systems*, 55(5):411–418.