Thesis Proposal

Distributed Algorithms for Probabilistic Inference and Learning

Stanislav Funiak

January 12, 2009

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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

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

Abstract

Probabilistic inference and learning problems arise naturally in distributed systems such as sensor networks, teams of mobile robots, and recommendation systems. In these systems, the data resides at multiple distributed locations, and the network nodes need to collaborate, in order to perform the inference or learning task.

This thesis has three thrusts. First, we propose distributed implementations of several state-of-the-art centralized inference algorithms. Our solutions address challenges, such as effective MAP estimation, scheduling of messages in loopy belief propagation, and assumed density filtering.

Many algorithms for probabilistic inference are described by graphical models, such region graphs or junction trees. These graphical models, together with the update schedule, entirely determine the behavior of the inference algorithm in a centralized settings. Yet, in distributed settings, the graphical model crucially interacts with the physical network and determines properties, such as robustness or communication complexity. In this thesis, we propose a unified view where the graphical model and its placement is optimized jointly to match both the network and the probabilistic model. In this manner, our distributed algorithms will not only attain accurate solutions, but will also have a low message complexity.

Recent advances in peer-to-peer networks offer interesting opportunities for learning latent variable models for collaborative filtering. Peer-to-peer networks simplify many aspects of distributed learning, but open an interesting challenge of supporting recommendation queries with stale local models. We propose a pull-based approach that updates the model parameters, in order to minimize its regret with respect to the optimal set of recommendations.

We demonstrate our algorithms on real-world applications in large-scale modular robot localization, camera networks and movie recommendation systems. We demonstrate that our algorithms scale to large networks and provide improved robustness and convergence properties.

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 3
- 1.2 Overview of the results . . . 3

2 Background **4**
- 2.1 Factorized probabilistic models . . . 4
- 2.2 Probabilistic inference methods . . . 4
  - 2.2.1 First-order optimization methods . . . 5
  - 2.2.2 Junction tree inference . . . 5
  - 2.2.3 Loopy belief propagation . . . 6
- 2.3 Literature review . . . 7
  - 2.3.1 Related work in sensor networks . . . 8
  - 2.3.2 Related work in robot localization and mapping . . . 9
  - 2.3.3 Related work on distributed inference . . . 10
  - 2.3.4 Parallel inference algorithms . . . 10
- 2.4 Discussion . . . 11

3 MAP estimation with global steps **12**
- 3.1 Application: Localization in modular robots . . . 12
- 3.2 Inference with rigid alignment . . . 12
- 3.3 Distributing the rigid alignment . . . 14
- 3.4 Discussion . . . 15

4 Distributed junction tree inference **16**
- 4.1 Application: Sensor calibration . . . 16
- 4.2 Relating the local and the global models . . . 17
- 4.3 Decomposable representation . . . 19
- 4.4 Discussion . . . 20

5 Distributed filtering **21**
- 5.1 Application: Simultaneous Localization and Tracking . . . 21
- 5.2 Problem formulation . . . 22
- 5.3 Approach: Assumed density filtering . . . 23
  - 5.3.1 Filtering in face of missing information . . . 24
- 5.4 Discussion . . . 26

6 Optimization of inference overlays **27**
6.1 Generalized belief propagation . . . 27
6.1.1 Cluster placement . . . 27
6.1.2 Robustness . . . 29
6.1.3 Co-optimization . . . 29
6.2 Tree-based parameterization . . . 30
6.2.1 Centralized pose estimation . . . 30
6.2.2 Network-aware optimization . . . 31
6.3 Relation to overlay networks . . . 31
6.4 Improving the convergence . . . 32
6.5 Discussion . . . 33

7 Learning models for collaborative filtering **34**
7.1 Application: Distributed recommendation systems . . . 34
7.2 Latent variable models for collaborative filtering . . . 35
7.3 Distributed learning . . . 37
7.4 Recommendations with an out-of-date model . . . 37
7.5 Discussion . . . 38

8 Conclusions and thesis plan **39**

Bibliography **41**

Chapter 1

Introduction

In many systems, the data is gathered at multiple distributed locations. A key problem in these systems is to reason with or build a probabilistic model that captures observations across all the locations in the network. By combining the data from multiple nodes, it is possible to infer more than is visible to an individual node locally.

Wireless sensor networks One of the first tasks in deploying a sensor network is to determine the physical locations of nodes. Often, it is not practical to determine the sensor locations manually or through GPS, such as when the nodes are deployed as a part of an emergency response system. Instead, the locations need to be estimated from approximate distance measurements to other nodes in the vicinity of each sensor node. By combining distance measurements from a large number of node pairs with a few anchor nodes whose physical location is known, the physical location of each node can be recovered. (Shang et al., 2003; Biswas et al., 2006). Furthermore, it is often desirable to compute the sensor locations in a distributed manner, without communicating all information to a central node. Such a solution may decrease the communication cost and provide greater robustness against node loss. Distributed sensor network localization has been tackled in a number of recent papers (Ihler et al., 2004; Biswas and Thrun, 2005; Djugash et al., 2008).

Modular robots Self-reconfigurable modular robots have received a growing interest from the robotics community (Yim et al., 2007; Støy, 2003). In order to perform its activities, a modular robot (Figure 1.1(a)) needs the ability to establish relative pose amongst its individual modules. The modules do not have access to long distance measurements, such as global time-of-flight measurements, or external beacons, but can observe the modules immediately adjacent to them, e.g., with short-range infrared. Probabilistic methods are key to accurately estimating the poses from such noisy observations (Roufas et al., 2000).

Simultaneous localization and tracking 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 (see Figure 1.1(b)). As with the wireless sensor networks, camera data is only useful if we know from where the images are captured, i.e., the real world location of the cameras, but manually measuring the **pose** (location and orientation) of all cameras is a

(a) modular robot (b) camera network (c) music recommendation system

Figure 1.1: Some of the applications considered in this thesis. (a) Modular robot localization. (b) Simultaneous localization and tracking. (c) Music recommendation systems.

very tedious and time consuming task. Instead, the locations of cameras can be recovered by tracking a moving target, such as a person or an LED marker. A key component in those solutions is to use a probabilistic model that relates the camera pose and the observed location of the target (Rahimi et al., 2004; Funiak et al., 2006a).

Multi-robot planning In many applications, it is desirable to deploy a team of robots to explore an environment. For example, emergency response teams may need to locate a lost person in a disaster scenario (Hollinger and Singh, 2008). Or, in environmental monitoring, scientists may wish to deploy a team of robots to gather measurements along a path, taking into account the informativeness of the observed locations (Singh et al., 2007). Decentralized versions of these approaches would allow the robots to replan online and operate in environments where the communication is severely constrained.

Collaborative spam filtering A key challenge in spam filtering is that spammers often change the IP addresses of the infected machines, which weakens the blacklisting schemes based on IP addresses. Several approaches aim to address this challenge, by relying on the sender's sending pattern (Ramachandran et al., 2007) or features, such as distance in IP space to other email senders and the geographic distance between sender and receiver (Syed et al., 2008). Distributed techniques play a key role in these approaches: for example, Ramachandran et al. (2007) require that the sending pattern be aggregated to a single node, while Syed et al. (2008) could benefit by learning classifiers that incorporate data from multiple email servers.

Recommendation systems With a number of online music services, such as Last.fm or Netflix, collaborative filtering methods now play role in many mainstream commercial services. By tracking the movie viewing or music listening patterns of each user, the services are able to provide recommendations for new music or movies a user may like. Side information about the genre or performs can be incorporated to improve the accuracy of prediction (Singh and Gordon, 2008). A natural extension, considered in this thesis, is to provide recommendations by storing the preferences at each user's machine locally, without revealing them to a central server. In doing so, we may be able to scale better and provide the service at no cost.

The applications listed above reflect a common theme, considered in this thesis: the observed data may be held locally at each node, but the nodes wish to obtain the same (or approximately the same) results as the corresponding centralized inference or learning task.

1.1 Problem statement

In this thesis, we consider several problems related to distributed inference and learning in graphical models:

- **Inference with global coordination:** Many centralized probabilistic inference algorithms perform global steps to provide fast convergence. Our goal is to design analogous distributed algorithms that perform a small amount of global coordination.
- **Optimizing inference overlays:** Many inference algorithms, such as the Generalized belief propagation (Yedidia et al., 2005), are formulated in terms of a graph datastructure that determines the structure of the updates. Our goal is to optimize the execution of the algorithm on the network, to minimize the communication cost and to trade communication for accuracy.
- **Collaborative filtering in P2P networks**: In some applications, the data is partitioned across the nodes in the network, and our goal is to determine the maximum-likelihood estimate of the model parameters, given all the data in the network.

In all three cases, we are looking for algorithms that have realistic communication properties: in an ad-hoc network, the node should only communicate with its neighbors, and the communication complexity should be sufficiently low. Furthermore, the algorithms should degrade gracefully when nodes run out of processing time or when a subset of the nodes fail.

1.2 Overview of the results

This thesis will present an in-depth exploration of distributed probabilistic inference and learning. The contributions of this thesis will be threefold:

- Understanding of **structural** and **convergence properties** that permit distributed implementations of standard centralized inference and learning algorithms. We will provide a simplified interpretation of existing distributed algorithms, and develop new algorithms that approximate state-of-the-art centralized approaches.
- A unified view of the distributed inference and learning algorithms in terms of their communication patterns. As a part of this unification, we will develop algorithms for co-optimizing the probabilistic model and its placement on the network nodes, relative to the communication pattern of the underlying inference algorithm.
- A comprehensive evaluation on novel applications in robotics and collaborative filtering. As a part of the evaluation, we will demonstrate our algorithms on a large network with several hundreds of nodes.

Parts of the work, outlined in this proposal, have been published, see (Funiak et al., 2006a,b, 2008).

Chapter 2

Background

This thesis rests on common concepts in machine learning, including factorized distributions and maximum-likelihood learning. In this section, we briefly review the relevant models, inference algorithms, and learning methods. The discussion in this section concerns centralized algorithms; we will examine the distributed aspects in the subsequent sections.

2.1 Factorized probabilistic models

Throughout this thesis, we assume that the probabilistic model we work with takes on a form of a factorized distribution,

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{A} \psi_A(\mathbf{x}_A), \tag{2.1}$$

where each $\mathbf{X} = \{X_i : i \in V\}$ are the random random variables in the model and each $A \subset V$ is a subset of the indices. Z is a normalization constant to ensure that the distribution sums to 1. We call each term ψ_A a **potential**. As a special case, we sometimes consider models that only contains unary and binary potentials:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{i \in V} \psi_i(\mathbf{x}_i) \times \prod_{\{i,j\} \in E} \psi_{i,j}(x_i, x_j) \tag{2.2}$$

We can describe the models (2.1) and (2.2) graphically, with a graph in which each node represents a variable, and for each potential ψ_A there is a clique over the nodes A. Figure 2.1 shows a simple model when variables are arranged in a grid and the potentials have at most two variables (in this case, the cliques are either individual nodes, or pairs of nodes connected by an edge).

2.2 Probabilistic inference methods

One fundamental task with the factorized models is to compute the marginals of the distribution (2.1) for a subset of variables. For example, in sensor network localization, we may wish to compute the marginal distribution over the location of each node. Or, in a temperature monitoring application, we may wish to compute the marginal temperature at each node. In the following sections, we briefly summarize the

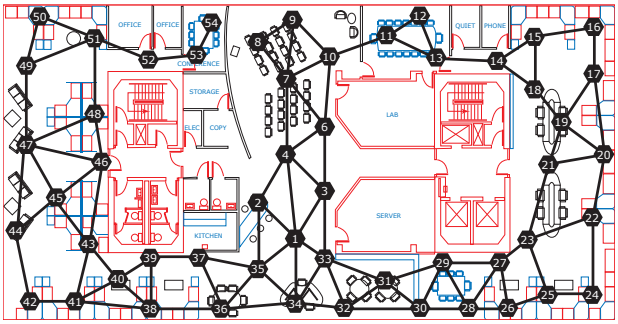


Figure 2.1: A graph that represents a factorized model with binary potentials.

inference methods that play a central role in this thesis. We refer the reader for a comprehensive review to the related literature (Cowell et al., 1999).

2.2.1 First-order optimization methods

The simplest method of probabilistic inference is to directly optimize the log-likelihood:

$$\max_{\mathbf{x}} \log p(\mathbf{x}) = \max_{\mathbf{x}} \sum_{A} \log \psi_A(\mathbf{x}_A) + \text{constant}. \tag{2.3}$$

Under mild conditions (when the variables $\mathbf{x}$ are real, and the potentials are continuously differentiable), we can find a local maximum of (2.3) with gradient ascent. The gradient of the maximum-likelihood decomposes linearly, which often leads to a very simple update. While this approach only recovers only the mode of the distribution, rather than its marginals, the mode is often sufficient in applications, especially when the distribution is known to be peaked.

2.2.2 Junction tree inference

A standard method for computing a set of marginals is junction tree inference. A junction tree is a special kind of a *clique tree*:

Definition 1. *A **clique tree** $(T, \mathbf{C})$ is an undirected tree T, in which each node $i \in V$ is associated with a set of random variables $C_i \in \mathbf{C}$, called the **clique** at i. For each undirected edge $\{i, j\} \in E$, the **separator between i and j** is*

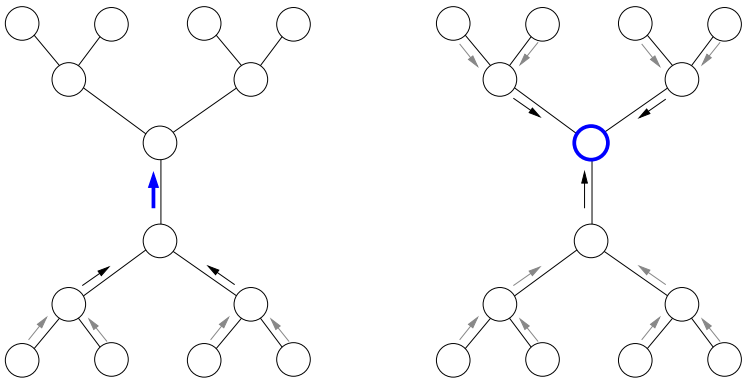
$$S_{i,j} \triangleq C_i \cap C_j, \tag{2.4}$$

the variables that are common to the cliques of nodes i and j.

A junction tree for a distribution $p(\mathbf{x}) = \prod_A \psi_A(\mathbf{x}_A)$ is a clique tree whose cliques include the arguments of p's factors and have a special structure:

Definition 2. *A clique tree is a junction tree for $\mathbf{p}(\mathbf{x})$ iff:*

- *For each factor $\psi_A(\mathbf{x}_A)$, there is a clique $C_i \in \mathbf{C}$ s.t. $A \subseteq C_i$; we say that the clique C_i **covers** the variables A.*



(a) message (b) node belief

Figure 2.2: Direct and indirect message dependencies in the sum–product algorithm. Each circle represents a collection of variables. The object of computation is shown in thick, blue lines. The messages that the computation depends upon directly are shown in black. The messages that the computation depends upon indirectly are shown in gray.

- *For every pair of nodes i and j of T, $C_i \cap C_j \subseteq C_k$ for all nodes k on the (unique) path between i and j; equivalently, for each variable X_a, the nodes whose cliques contain a form a connected subtree of T. We say that the clique tree satisfies the **running intersection property**.*

Given a junction tree $(T, \mathbf{C})$ for the distribution $p(\mathbf{x})$, we can compute the marginal $p(\mathbf{x}_C)$ over each cliques $C \in \mathbf{C}$ with the **junction-tree algorithm**: First, we assign each potential to some vertex of the tree i; denote the product of the potentials at vertex i with $\psi_i(\mathbf{x}_{C_i})$. Then we compute a **message** $\mu_{i,j}$ between every pair of neighboring vertices $(i, j) \in T$, which is defined recursively as follows:

$$\mu_{i,j}(\mathbf{x}_{S_{i,j}}) \triangleq \sum_{\mathbf{x}_{C_i - S_{i,j}}} \psi_i(\mathbf{x}_{C_i}) \prod_{k \in N(i) \setminus j} \mu_{k,i}(\mathbf{x}_{S_{k,i}}). \tag{2.5}$$

Here, $N(i)$ are the neighbors of i in T (see Figure 2.2(a). Equation 2.5 implements a dynamic programming algorithm that satisfies an invariant that the message $\mu_{i,j}$ is the marginal of all the potentials on i-th side of the tree, over the variables $S_{i,j}$. It is then easy to show that the marginal the marginal $p(\mathbf{x}_{C_i})$ can be computed as a product of incoming messages (see Figure 2.2(b)):

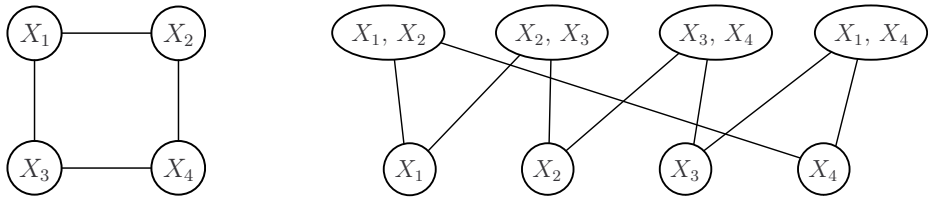
$$p(\mathbf{x}_{C_i}) \propto \psi_i(\mathbf{x}_{C_i}) \prod_{j \in N(i)} \mu_{j,i}(\mathbf{x}_{S_{i,j}}) \tag{2.6}$$

2.2.3 Loopy belief propagation

For many models, junction tree inference has a very large computational complexity.[1] A standard approach for approximate inference is generalized belief propagation (GBP). GBP relies on the fact that inference can be formulated as the result of an optimization problem

$$\min_q D(q \,\|\, p),$$

[1]The computational complexity is determined the **tree width** of the density, which the size of the largest clique of the best possible junction tree for p.



(a) a small grid graph (b) Bethe approximation

Figure 2.3: (a) A small graphical model with pairwise potentials. (b) The corresponding region graph used in the loopy belief propagation algorithm.

where p is the exact distribution in (2.1), and q an approximate distribution that belongs to some family of functions. The relative entropy can be written as

$$D(q \,\|\, p) = -\log Z - H(q) - \sum_A \mathbb{E}_{q_A}[\log \psi_A(\mathbf{X}_A)],$$

where $\log Z$ is the normalization constant that does not depend on q. GBP approximates the entropy $H(q)$ as a weighted sum of the entropies for marginals over subsets of variables. The subsets of variables and the wights determined by a **region graph**. For example, in loopy belief propagation (LBP), the region graph contains set of root clusters, one for each factor ψ_A in the graphical model, and a set of child clusters, one for each variable in the model (see Figure 2.3). The entropy $H(q)$ is approximated as

$$H(q) \approx \sum_A H(q(\mathbf{X}_A)) - \sum_i (d_i - 1) H(q(X_i)),$$

where d_i is the number of factors, whose domains include the variable X_i. By writing the dual of the relaxed optimization problem, we obtain update rules that pass messages along the edges of the region graph (Yedidia et al., 2005, Appendix E). The update rules are the same, or similar to those in Equations 2.5 and 2.6. For LBP, it is common to combine the upward and downward pass, which yields the standard update equation:

$$\mu_{s,t}(x_t) \leftarrow \sum_{x_s} \psi_{s,t}(x_s, x_t) \times \psi_s(x_s) \times \prod_{r \in N_G(s) \setminus t} \mu_{r,s}(x_s), \tag{2.7}$$

where $\mu_{s,t}$ is the message sent from node s to node t and $N_G(s)$ is the set of neighbors of node s in Markov network for p.

2.3 Literature review

Much of the work on distributed inference has been developed in the context of specific applications, such as sensor network localization and tracking, and multi-robot simultaneous localization and mapping (SLAM). In this section, we review prior work in the context of these applications, and follow with more general algorithms.

2.3.1 Related work in sensor networks

A key problem, considered in sensor networks is **tracking**, where a sensor network monitors the location of a moving object over time. When the system is modeled with a linear dynamical model, a standard centralized approach is a Kalman filter (Kalman, 1960). In their seminal work, Manyika and Whyte (1995) used an information form of the Kalman filter, which maintains the inverse of the covariance matrix. While the computational cost of the information filter is typically higher than of the Kalman filter, the conditioning step is substantially simpler and can be implemented in a distributed manner by aggregating the observation likelihood across the nodes. The material, considered in Chapters 4 and 5, can be viewed as an extension of the information filter to the structured setting where the nodes track (overlapping) sets of variables.

While an information filter provides a principled solution, several other methods aim to solve the tracking problem with a lower computational complexity. Chu et al. (2002) describe a method, called Information-driven sensor querying (IDSQ), in which a leader node maintains the belief and requests observations from sufficiently many neighboring sensors. When the target moves, the leader hands off the belief to one of its neighbors. Liu et al. (2003) propose to use the mutual information criterion to determine the recipient of the belief. Since the observations are communicated in its raw form, these methods are not tied to any particular representation of the belief.

A second problem, considered in wireless sensor networks, is **localization from signal strength**. This problem has received a significant attention and is well understood in the centralized setting. A standard approach is to treat each network node as a vertex in a graph and each measurement as a noisy observation of the relative distance between two nodes. The localization can be then formulated as a **Euclidean embedding** problem, where we wish to assign each vertex a location in such a way that the distances between neighboring vertices are approximately preserved. This optimization problem can be solved with methods, such as classical multidimensional scaling (Shang et al., 2003) and regularized semi-definite programming relaxations (Biswas et al., 2006; Wang et al., 2006). Unfortunately, most Euclidean embedding methods do not distribute well. A simple heuristic, considered by Biswas and Thrun (2005), is to greedily decompose the network into a set of overlapping clusters. The nodes within each cluster are localized centrally at the cluster's leader, and the estimates are then merged. We use a similar heuristic in our work on localization in modular robot ensembles (Chapter 3). However, our solution is hierarchical, has a lower message complexity, and we provide a fully distributed implementation.

While the Euclidean embedding methods have their merits, it is natural to seek distributed algorithms that have a probabilistic interpretation. A popular approach (Ihler et al., 2004; Djugash et al., 2008) is to formulate localization as probabilistic inference in a pairwise Markov network, where each edge corresponds to the observation of the distance between two network nodes. The marginal probabilities can be then approximated using loopy belief propagation. In order to cope with the non-linearities of the observation model, Ihler et al. (2004) approximate the messages with a collection of samples, and Djugash et al. (2008) use a combination of multiple hypotheses and over-parameterization, introduced in our work (Funiak et al., 2006a).

A closely related problem is **simultaneous localization and tracking** (Rahimi et al., 2004), where the sensors observe a distance to a moving target, rather than among themselves. This problem combines the aspects of both tracking and sensor network localization. Taylor et al. (2006) introduce a simplifying assumption that effectively turns SLAT into a sequence of static estimation problems. With this assumption, the coordination among the network nodes is entirely local. In our work, we address the complete

dynamic problem, although when presented with their formulation, our solution has a higher communication complexity.

It is worth noting that the loopy belief propagation methods carry over to this more general problem (Djugash et al., 2008). However, depending on the amount of noise in the observations, LBP can overcount significantly.

2.3.2 Related work in robot localization and mapping

Distributed aspects have surfaced in multi-robot localization algorithms. In multi-robot localization, each robot makes observations of its surroundings and of other robots. Unlike in sensor network tracking, where there is a single entity being tracked, we wish to estimate the location of each robot. Exact distributed algorithms can incur a large communication complexity. A standard technique, employed in many algorithms is to approximate the joint distribution as a product of marginals. For example, Fox et al. (2000) maintain independent particle filters over individual robot poses, one at each robot. Whenever there is an observation that relates the two, the algorithm computes a projection to the individual marginals. Assuming that two robots can communicate whenever they observe each other, this algorithm can be implemented entirely with a local communication. Rosencrantz et al. (2003) then extend this approach to the game of tag with multiple teams of robots. This extension changes the representation of the belief at each node, but not the communication pattern of the algorithm. Both these algorithms can be viewed as a special case of the Boyen-Koller algorithm (Boyen and Koller, 1998) and can be handled by the method, presented in Chapter 5.[2]

One of the first solutions for multi-robot simultaneous localization and mapping (SLAM) was presented by Nettleton et al. (2000). This method can be viewed as an extension of the information filter (Manyika and Whyte, 1995) to the setting where the state includes not only the location of each robot, but also the location of each landmark. Each robot maintains a joint distribution over itself and all the landmarks. Thrun and Liu (2003) extend this approach using the sparse extended information filter (SEIF) (Thrun et al., 2004). The use of SEIF decreases the communication complexity. (They also consider the data association problem to match the landmarks among different robots.) While the distributed SEIF is substantially simpler to implement, our solution can have a lower computational complexity (however, we do not address the problem of cluster selection).

In addition to the landmark-based approaches, discussed above, there are several approaches that make additional simplifying assumptions. For example, Thrun et al. (2000) greedily estimate the most likely map (represented as a trajectory of most likely poses), and only maintain a factored distribution over the current robot poses. The robots build a single global map, stored at the leader; updates to the map are continuously communicated from the team members to the leader. (Konolige et al., 2006) extend this approach to a very large team of robots (up to 100). Whenever two robots meet, they synchronize their maps. Since the maps represented compactly as sets of laser range-scans annotated with the most likely robot poses, the robots are not communication-limited.

In general, the networking aspects of SLAM are very different from the inference problems, considered in this thesis. Unlike the localization and tracking in sensor networks, where each node stores a limited amount of information, each node in SLAM carries a distribution over a large number of random variables.

[2]Our algorithm would have to be modified slightly to accommodate the fact that we do not have a clique over a pair of adjacent robots.

This is because we have relatively few robots, mapping a large environment. However, some of the structural properties (such as sparse approximations) exploited in this thesis are also found in SLAM.

2.3.3 Related work on distributed inference

Many centralized inference algorithms have a message passing flavor. This fact has been exploited to directly apply centralized algorithms in distributed settings. For example, as discussed earlier in the context of wireless sensor network localization, loopy belief propagation can be used to estimate the marginal over each variable x_i, provided that a network node can communicate with the nodes that carry the variables that are adjacent to x_i in the probabilistic model. Crick and Pfeffer (2003) have used this observation to argue that loopy belief propagation is well-suited for distributed inference tasks. Indeed, there have been some successful applications of loopy belief propagation, in the context of sensor network localization (Ihler et al., 2004). Also, Pfeffer and Tai (2005) use loopy belief propagation to approximate the estimation step in a continuous-time Bayesian network.

Yet, the basic loopy belief propagation does not adequately address all the problems. In order to decrease the message complexity and avoid the overhead of synchronous message passing, Schiff et al. (2007) propose to sample the messages according to the update difference. However, their algorithm stops making progress when the residuals become small. Schmidt and Aberer (2006) propose to use a distributed hash table to provide content-based addressing and optimize the cost of sending the message among the nodes in the network. In contrast, we propose scheduling that continues to make progress, and we tackle a much broader problem of optimizing the placement and selection of region graphs in the context of generalized belief propagation (Yedidia et al., 2005).

For some problems, exact inference is a reasonable alternative to loopy belief propagation. Paskin et al. (2005) describe a distributed datastructure, **network junction tree** that allows the nodes to execute the standard junction tree inference algorithm on a network. Paskin and Guestrin (2004) uses this datastructure to perform static inference, assuming that each node starts with one or more clique marginals of a triangulated model. They show that, at convergence, their distributed algorithm computes the same answer as the centralized alternative. In Chapter 4, we provide a simplified interpretation of their algorithm with new partial correctness guarantees.

So far, we have discussed estimation in the context of structured models, when the model contains multiple random variables, and each network node wishes to compute (an approximation to) the marginal over one or more variables. In some cases, the inference can be formulated a simple averaging problem. In these cases, different forms of distributed consensus (Xiao and Boyd, 2003; Mehyar et al., 2005) can be used.

2.3.4 Parallel inference algorithms

The algorithms, considered in this thesis, are closely related to parallel algorithms for clusters and multicore machines. Map-Reduced, introduced by Dean and Ghemawat (2008), is a popular framework for implementing algorithms that parallelize easily. It has been used, for example, to parallelize algorithms that fit the statistical query model (Chu et al., 2006). However, as shown by Gonzalez et al. (2009), naive application of Map-Reduce to inference algorithms, such as loopy belief propagation, can be substantially slower than the sequential algorithm. They propose an algorithm, called ResidualSplash, that performs more fine-grained parallelization.

In general, while the parallel and distributed inference algorithms share similar goals, distributed algorithms need to address a number of additional challenges, such as synchronization and communication in a network-constrained environment.

2.4 Discussion

In this section, we gave an overview relevant concept in probabilistic inference and reviewed related papers in distributed probabilistic inference. We have side-stepped a large body of literature on probabilistic inference, as well as specific application areas. For a review of probabilistic inference algorithms, see (Cowell et al., 1999). For a review of SLAM algorithms, see (Thrun, 2002).

Chapter 3

MAP estimation with global steps

In many cases, the structure of the computation naturally matches the network. This property has been advocated by Crick and Pfeffer (2003), and has been used extensively in the sensor network localization literature. Yet, often, a direct application of techniques, such as gradient is insufficient for the task at hand, and needs to be combined with domain-specific heuristics, whose computational structure does not match the underlying graphical model. In this section, we consider one example of such an approach in the context of modular robot localization, and demonstrate how it can be distributed.

3.1 Application: Localization in modular robots

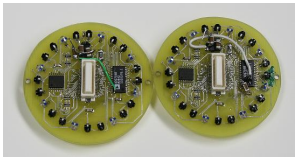
A key problem in a modular robot is to estimate the poses of individual modules within the entire ensemble. Each module has onboard sensors that allow it to detect when other modules are in its immediate neighborhood. The observations made by the sensors are noisy; hence the observations from the entire ensemble need to be combined, in order to obtain an accurate estimate of the modules' relative pose. Let $x_i \in \mathbb{R}^d \times SO(d)$ denote the pose of module i in 2D or 3D. Since the observations are local and occur only between pairs of neighboring modules, the distribution over the robot poses can be written as a pairwise Markov network:

$$p(\mathbf{x}) \propto \prod_{i,j} \psi_{i,j}(x_i, x_j; z_{i,j}, z_{j,i}), \tag{3.1}$$

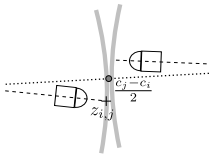
where $\psi_{i,j}(x_i, x_j; z_{i,j}, z_{j,i})$ represents the information observed by two neighboring modules i and j and $z_{i,j}$ is the observation of module j, made by module i. In the simplest case, the factor $\psi_{i,j}$ represents the disagreement between the locally observed point of contact between module i and j ($z_{i,j}$) and the mid-point of the estimated module centers (see Figure 3.1(b)). Modular robot localization can be then formulated as an optimization problem, in which we wish to recover an assignment to $\mathbf{x}$ that maximizes the likelihood in Equation 3.1.

3.2 Inference with rigid alignment

A simple approach to maximize (3.1) is to perform gradient ascent on the log-likelihood $\log p(\mathbf{x})$. As shown in Equation 2.3, the log-probability of an assignment $\mathbf{x}$ decomposes linearly across the factors of



(a) module prototypes



(b) sensor model

Figure 3.1: (a) Sensor board from module prototype. (b) Sensor model, used in the paper. Each observation is represented as the location of the sensor, projected to the perimeter of the module. The circle indicates the midpoint of the two modules' centers. The model penalizes the module locations x_i and x_j, based on the distance between the midpoint and the observations $z_{i,j}$ and $z_{j,i}$.

the distribution, and has a particularly simple sparsity structure: the gradient with respect to the variable x_i only involves the terms for module i and its neighbors. This structure leads to an very simple distributed algorithm: each node carries the current estimate of its pose x_i. At each iteration, the node collects the estimates x_j of its neighbors, and then locally performs a gradient ascent update.

Unfortunately, gradient ascent is often ineffective in modular robot localization, where rotational components lead to non-linearity, slow-convergence, and local optima. Even methods, such as preconditioned conjugate gradient ascent, do not yield adequate results. One heuristic (Funiak et al., 2008) to speed up the MAP estimation is to hierarchically decompose the optimization problem (3.1) into simpler subproblems, the solutions of which are combined to obtain a good estimate for the overall problem.

The approach is illustrated in Figure 3.2. Focusing on the operations performed at a single level, we partition the variables (or, equivalently, the modules) into two connected components A and B; for the details of the partitioning scheme, see (Funiak et al., 2008). We compute the maximum-likelihood estimate for A and B, using only observations from A and B, respectively:

$$\mathbf{x}_A^* = \arg\max_{\mathbf{x}_A} \prod_{i,j \in A} \psi_{i,j}(x_i, x_j; z_{i,j}, z_{j,i}) \tag{3.2}$$

$$\mathbf{x}_B^* = \arg\max_{\mathbf{x}_B} \prod_{i,j \in B} \psi_{i,j}(x_i, x_j; z_{i,j}, z_{j,i}) \tag{3.3}$$

The partial solutions $\mathbf{x}_A^*$ and $\mathbf{x}_B^*$ are then combined to form an initial solution that incorporates not only observations within each component, but also observations between them. Specifically, treating the observations $\mathbf{z}_{A,B} = \{z_{i,j} : i \in A, j \in B\}$ and $\mathbf{z}_{B,A} = \{z_{j,i} : i \in A, j \in B\}$ as two point clouds, we can compute an optimal rigid transform Q that minimizes the square error between the two components:

$$\min_{Q \in SO(d) \times \mathbb{R}^d} \sum_{\{i,j\} \in E : i \in A, j \in B} \|x_i^* \circ z_{i,j} - Q \circ x_j^* \circ z_{j,i}\|. \tag{3.4}$$

Here, E are the edges of the connectivity graph, and $\circ$ denotes the composition operator, i.e., $Q \circ v$ is the result of applying the transform Q to the vector v. The optimal transform (3.4) can be computed in closed form using singular value decomposition (Umeyama, 1991).

Figure 3.3(a) illustrates the performance of the approach on an ensemble with 1000 modules. We see that the algorithm outperforms alternative approaches, based on Euclidean embedding and stochastic gradient ascent. Also, the number of conjugate gradient ascent iterations to reach a fixed accuracy is small (Figure 3.3(b)).

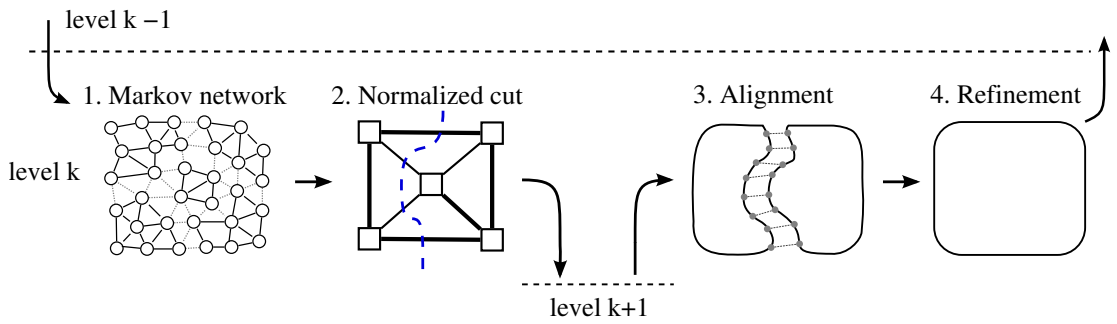
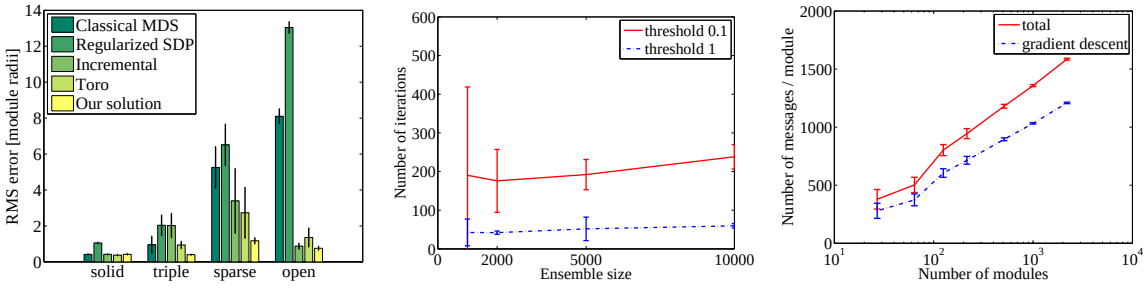


Figure 3.2: Control flow for one level of the hierarchical localization.



(a) comparison (b) convergence time (c) communication complexity

Figure 3.3: Experimental results on 2D ensembles. (a) Comparison with prior approaches. Classical MDS and Regularized SDP are two approaches based on Euclidean embedding. Toro and Incremental are two local approaches. (b) The number of iterations per module required to attain the same accuracy. (c) The number of messages per module.

3.3 Distributing the rigid alignment

A key step in the above approach is to obtain the optimal rigid body transform between two sides of the partition. This step needs to be performed efficiently and with limited communication: since the alignment is performed for very large components, the observations cannot be simply collected to a central location. However, a closer look at the method in (Umeyama, 1991) reveals that the method only depends on the first-and second-order statistics between the two point clouds $\mathbf{z}_{A,B}$ and $\mathbf{z}_{B,A}$ (transformed according to the poses $\mathbf{x}_A^*$ and $\mathbf{x}_B^*$. These statistics can be aggregated from the boundary between the nodes A and B towards a leader. The leader then computes the optimal transform and disseminates the result. Since the aggregated information is small (12 parameters for 3D localization), the communication cost of aggregating and disseminating the optimal transform is small.

Figure 3.3(c) shows the number of neighbor-to-neighbor messages per module required by our distributed algorithm (this plot shows the results for the complete distributed implementation, which includes the alignment). We see that the communication complexity increases only logarithmically in the total number of modules in the ensemble.

3.4 Discussion

In this section, we described a distributed implementation of an algorithm for localization in modular robots. A key step in the algorithm was to hierarchically partition the optimization problem into independent subproblems, and combine the partial solutions to the subproblems using rigid alignment. The rigid alignment had a particularly simple form of the solution that permitted an implementation using a combination of data aggregation and dissemination techniques. This lead to a distributed algorithm with a small communication complexity.

The discussion presented in this section focused on a single part of a larger system. A complete distributed implementation of the algorithm includes several additional details that we omit for brevity. Furthermore, a key component of the implementation was a declarative specification using Meld (Ashley-Rollman et al., 2007). We will revisit the localization problem in Section 6.2 in the context of overlay networks.

Chapter 4

Distributed junction tree inference

In the previous section, we considered the problem of determining a MAP estimate for a distribution, described as a Markov network. While MAP estimates are useful, sometimes it is desirable to recover not only a point estimate, but also its uncertainty. A standard approach for computing the exact marginals of a distribution in the centralized setting is the sum—product algorithm (Section 2.2.2). Paskin and Guestrin (2004) proposed an algorithm, called Robust message passing, that performs exact inference in distributed systems and attains additional robustness properties not exhibited by the sum—product algorithm. In this chapter, we briefly review their algorithm, providing a simplified interpretation of their algorithm and propose a modification with stronger partial correctness guarantees.

4.1 Application: Sensor calibration

Consider the following temperature monitoring application. After a sensor network is deployed, the sensors can be adversely affected by the environment, leading to biased measurements. The distributed sensor calibration task involves automatic detection and removal of these biases (Bychkovskiy et al., 2003; Ihler et al., 2004). This is possible because the quantities measured by nearby nodes are correlated but the biases of different nodes are independent.

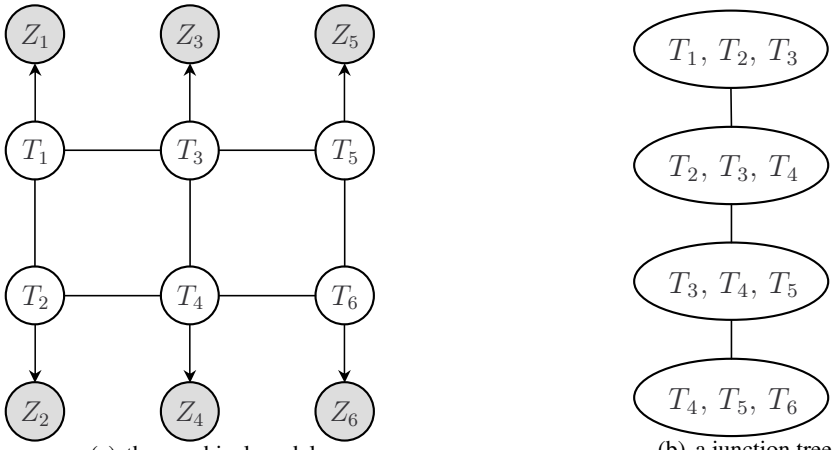
In the sensor calibration problem, each observed variable Z_i is a temperature measurement taken by one of the sensor nodes, and the hidden variables are the true temperatures T_i and sensor biases B_i of the network nodes. The temperature prior is a multivariate Gaussian distribution characterized by the Markov network in Figure 2.1; the biases of different nodes are independent. The complete joint probability density is given by:

$$p(\mathbf{t}, \mathbf{b}, \mathbf{z}) = \underbrace{\frac{1}{Z} \prod_{i,j} \psi_{i,j}(t_i, t_j)}_{\text{temperature prior}} \times \prod_i \underbrace{p(b_i)}_{\text{bias prior}} \underbrace{p(z_i \mid t_i, b_i)}_{\text{measurement model}} , \tag{4.1}$$

where each $\psi_{i,j}(t_i, t_j)$ is a factor of the Gaussian prior over temperature.

Given observations for the measurement variables $\mathbf{Z} = \underline{\mathbf{z}}$, the goal of probabilistic inference is to compute the marginal distribution $p(\mathbf{x}_Q \mid \underline{\mathbf{z}})$ for a subset of **query** variables $Q \subseteq V$.[1] For example, in the sensor

[1] We use the notation $\underline{\mathbf{z}}$ to emphasize that the value of $\mathbf{Z}$ is fixed throughout the inference process.



(a) the graphical model (b) a junction tree

Figure 4.1: An example probability model (a) and an associated junction tree (b).

calibration problem, we may be interested in computing the true temperature as well as the measurement bias at a location, conditioned on all measurements made in the network: $p(t_i, b_i \mid \underline{\mathbf{z}})$.

Figure 4.1 shows a small inference problem that we will use as a running example. In this example, we have six network nodes, each with a true temperature T_i and a temperature measurement Z_i (for simplicity, the bias variables are omitted from this example model). Each temperature measurement depends upon the node's true temperature. The posterior distribution $p(\mathbf{t} \mid \underline{\mathbf{z}})$ thus factorizes as $p(\mathbf{t} \mid \underline{\mathbf{z}}) = \frac{1}{Z} \prod_{\{i,j\}} \psi_{i,j}(t_i, t_j) \times \prod_i p(\underline{z}_i \mid t_i)$. A junction tree for $p(\mathbf{t} \mid \underline{\mathbf{z}})$ is given in Figure 4.1(b). Note that each edge of the Markov network is covered by some clique in the tree; similarly, the argument T_i of each observation likelihood $p(\underline{z}_i \mid t_i)$ is included in some clique. The clique tree satisfies the running intersection property; for example, the nodes whose cliques contain T_4 form a connected subtree.

4.2 Relating the local and the global models

Suppose that we wish to implement the junction tree inference algorithm (2.5) in a distributed setting. The simplest approach is to assign each clique to some node, and forward the messages between the corresponding nodes. While conceptually simple, this approach has two substantial drawbacks. First, unlike the previous chapter where the structure of the Markov network is typically locally observed, the mapping of cliques to nodes can be arbitrary. Thus, the nodes would need to figure out how to locate the adjacent cliques and the nodes that carry these cliques. More importantly, when a node fails or the communication network is fragmented, the corresponding factors of the distribution will be lost. With some factors missing, the algorithm may converge to arbitrarily bad results (Paskin, 2004).[2]

In order to address these issues, Paskin and Guestrin proposed an algorithm, called Robust Message Passing. The main idea of the algorithm is delay the marginalization operations, performed by the sum-product algorithm. Each message sent between two nodes in the network consists of a collection of factors; these are the factors that have not yet been summed out. Before a network message is sent, the

[2]This is because factorized probability models often lack locality: removing a factor from a probability distribution may not preserve the marginal over the remaining variables.

algorithm detects which variables can be marginalized out, and performs the updates in Equation (2.5). In this manner, the distributed algorithm emulates the centralized sum–product algorithm, without explicit knowledge of where each clique resides (we will revisit the robustness issues shortly). The algorithm is illustrated in Figure 4.2(a).

A key question is, how a node can relate a collection of cliques, held locally, to the global junction tree. Specifically, how can we determine the edges of T by looking at a subset of the cliques $\mathbf{C}'$ alone? A standard result (Cowell et al., 1999) is that a junction tree T for a set of cliques $\mathbf{C}$ can be obtained as a maximum spanning tree of a complete graph, in which the weight of each edge C_i—C_j is the size of the separator $|C_i \cap C_j|$. Suppose that we build a maximum spanning tree T' for the cliques $\mathbf{C}'$. Figure 4.2(b) shows a maximum spanning tree T' for the cliques that form the message from node 4 to node 2. We see that T' and the global junction tree T in Figure 4.1(b) have many edges in common. Intuitively, if two cliques $C_i, C_j \in \mathbf{C}'$ are neighbors in T, then the intersection $|C_i \cap C_j|$ is large, and we would expect them to also be neighbors in T'. This property holds in general:

Lemma 1. *Let $\mathbf{C}$ be the cliques of a triangulated graph and let $\mathbf{C}_M \subseteq \mathbf{C}$ be a subset of cliques with indices M. If $(T_M, \mathbf{C}_M)$ is a maximum–intersection clique tree, then there exists a junction tree $(T, \mathbf{C})$ s.t. for any $i, j \in M$, $\{i, j\} \in E \implies \{i, j\} \in E_{T_M}$.*

Thus, whenever two cliques among $\mathbf{C}'$ are neighbors in T, they are also neighbors in T'. A technicality arises due to the fact that there may be several equivalent junction trees for a set of cliques (the trees are equivalent in the sense that they have the same weight and the same set of separators). Lemma 1 states that there exists a junction tree whose edges among $\mathbf{C}_M$ match those of T'.

Naturally, for distributed inference, we need the converse result: starting from a maximum spanning tree T' for $\mathbf{C}'$, we would like to determine which edges of T' are also present in an external junction tree T for $\mathbf{C}$. Clearly, T' may contain edges that are not present in T. For example, the cliques $\{T_1, T_2, T_3\}$ and $\{T_3, T_4, T_5\}$ are neighbors in the maximum spanning tree T' in Figure 4.2(b) but not in the global junction tree in Figure 4.1(b). A simple condition can be characterized as follows. In some cases, the leaves of the maximum spanning tree correspond to the leaves of the external junction tree:

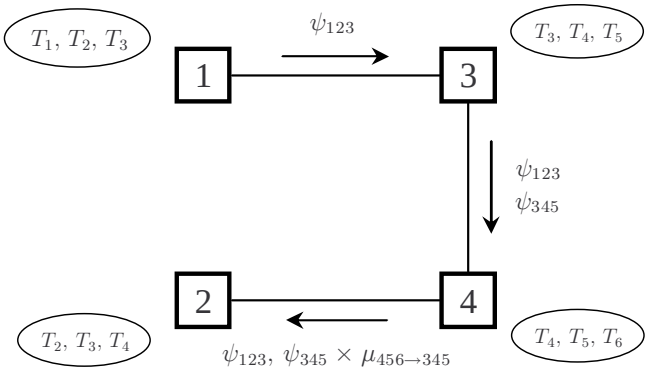
Corollary 1 (Paskin and Guestrin, 2004)**.** *Let $\mathbf{C}$ be the cliques of a triangulated graph, let $\mathbf{C}_U \subseteq \mathbf{C}$ contain all cliques that meet a subset U of variables, and let T_U be a maximum intersection clique tree for $\mathbf{C}_U$. If C_i is a leaf clique of T_U with neighbor C_j and*

$$C_i - U \subseteq C_j, \tag{4.2}$$

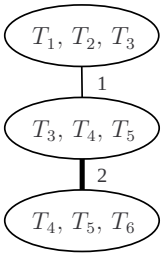
then there exists a junction tree for $\mathbf{C}$ in which i is a leaf and j is its neighbor.

We will call C_i a **dangling leaf**.

Paskin and Guestrin use Corollary 1 to suggest the following distributed algorithm. The nodes build a routing tree over the network. Whenever a node sends a message to its neighbor in the routing tree, it form a maximum spanning tree for the message cliques $\mathbf{C}'$ and repeatedly prunes any leaf clique C_i s.t. $C_i - C_j \subseteq U$. In order to determine which variables can be eliminated, Paskin and Guestrin employ the architecture (Paskin et al., 2005). This architecture computes the set of variables $S_{m,n}$ that are present on both node m's and node n's side of the routing tree, which suffices to evaluate the condition (4.2).



(a) example messages



(b) cliques at node 4

Figure 4.2: (a) The messages sent towards node 2 in the robust message passing algorithm for the model in Figure 4.1(b). The network consists of four nodes, with clique factor assigned to each node as shown above. In computing the message to its neighbor node 2, the node 4 has marginalized out the variable T_6 (which amounts to computing the term $\mu_{456,345}$ in Equation 2.5). (b) A spanning tree used in determining the variables that can be marginalized out. Note that the cliques $\{T_3, T_4, T_5\}$ and $\{T_4, T_5, T_6\}$ are adjacent both here and in the original model in Figure 4.1(b). Therefore, it is safe to use this portion of the local model for variable elimination.

4.3 Decomposable representation

As presented so far, the distributed message passing algorithm addresses one challenge: it works without explicitly encoding the edges of the junction tree in the nodes in the network. Also, assuming that no nodes fail and that the network is connected, the algorithm eventually computes the exact marginal at each node. Yet, as indicated earlier, it is desirable to also provide some partial correctness guarantees. Figure 4.3(a) illustrates the setting in the case where a node wishes to compute a result before the messages have converged. In this case, the result computed at a node only incorporates a part of the probabilistic model that was successfully communicated towards the node. Therefore, we seek a representation of the model that permits approximations in the face of missing components.

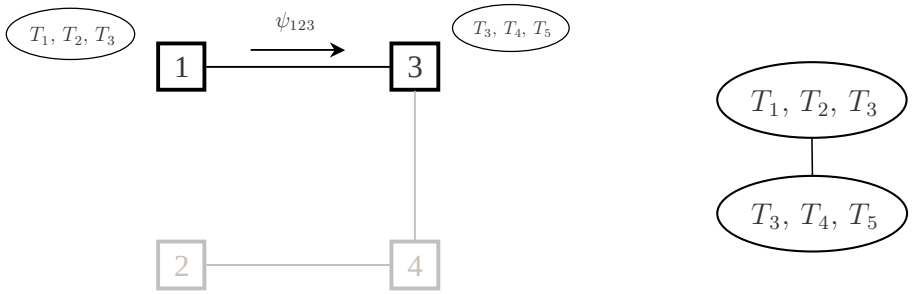
Paskin and Guestrin (2004) address his problem by parameterizing the distribution in terms of its marginals. Suppose that the distribution factorizes as prior $p(\mathbf{x})$ and the likelihood $p(\underline{\mathbf{z}} \mid \mathbf{x})$ for some fixed set of observations $\underline{\mathbf{z}}$. Let $(T, \mathbf{C})$ be a junction tree for the density p. Then the joint distribution $p(\mathbf{x}, \underline{\mathbf{z}})$ can be written as

$$p(\mathbf{x}, \underline{\mathbf{z}}) = \frac{\prod_{i \in N} p(\mathbf{x}_{C_i}) p(\underline{\mathbf{z}}_i \mid \mathbf{x}_{C_i})}{\prod_{\{i,j\} \in E} p(\mathbf{x}_{S_{i,j}})}. \tag{4.3}$$

(see e.g., (Cowell et al., 1999)). Thus, instead of an original factor ψ_i, we can use a **P/L factor** $\langle \pi_i, \lambda_i \rangle$, which is a pair that consists of a clique prior and the corresponding likelihood:

$$\langle \pi_i, \lambda_i \rangle \triangleq \langle p(\mathbf{x}_{C_i}), p(\underline{\mathbf{z}}_i \mid \mathbf{x}_{C_i}) \rangle.$$

The P/L factors $\{\langle \pi_i, \lambda_i \rangle\}$ play the same role in junction tree inference algorithm (2.5) as the original factors $\{\psi_i\}$. Furthermore, Paskin (2004) shows that when the P/L factors are used inside the distributed message passing algorithm, the result is a sequence of projection and inference operations. This property



(a) messages before convergence (b) partial belief

Figure 4.3: An illustration what it means to have a partial belief.

can be strengthened by performing a maximum-entropy computation on the prior marginals, and altering the pruning rule in a way that detects and isolates independent maximum-entropy computations.

4.4 Discussion

In this section, we presented an overview of the robust message passing algorithm (Paskin and Guestrin, 2004). We provided a cleaner interpretation of the algorithm, and adjusted the algorithm to provide stronger guarantees.

Chapter 5

Distributed filtering

A key problem in probabilistic inference is filtering. In filtering, we wish to estimate the state of a dynamic system, based on past observations. Many problems in robotics can be formulated as filtering, including Simultaneous Localization and Mapping (SLAM) and Simultaneous Localization and Tracking (SLAT). In this chapter, we show how the ideas from the preceding chapters carry over to filtering. We demonstrate our approach on a problem of camera localization (Funiak et al., 2006a).

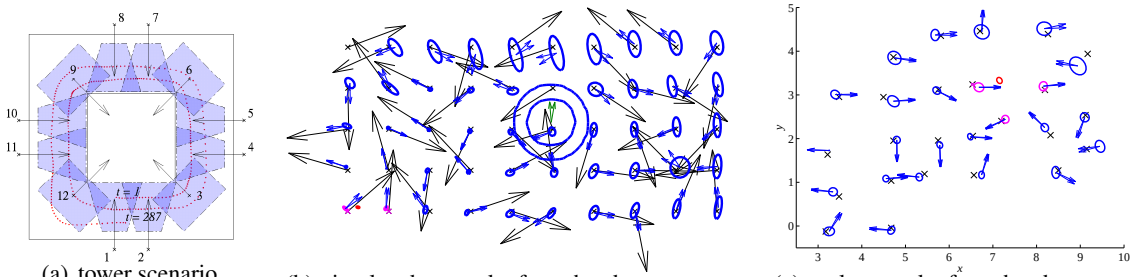
5.1 Application: Simultaneous Localization and Tracking

Suppose that a moving object is seen in the field of view of a camera and, a few moments later, the same object is observed by another camera; if we knew the trajectory of this object, we could infer information about the relative position of the two cameras. Similarly, if we knew the poses of the cameras, we could infer the trajectory of the object. We can address the camera network calibration task by solving a **simultaneous localization and tracking** (SLAT) problem, where we estimate both the trajectory of the object and the poses of the cameras. An effective solution of the SLAT problem leads to a very simple camera network deployment procedure: cameras are placed throughout the environment at unknown locations, then, as an object (e.g., a person) moves throughout the environment following an unknown trajectory, the network automatically calibrates itself.

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, SLAT can be formulated as a *probabilistic inference* task, where we maintain a joint distribution over possible object locations and poses of all cameras, given the images collected by the network. An advantage of this approach is that it provides an explicit representation of the uncertainty in the estimate of camera poses. By representing uncertainty, we have a direct measure of the quality of the solution, indicating when the calibration procedure can be stopped, and what parts of the network need more information to improve their calibration, potentially enabling an active control of the path of the object that optimizes the quality of the solution.

We model the SLAT problem using a linear dynamical system (Rahimi et al., 2004). The variables of this system are the location L_t of the object at each time step[1], and for each camera i, the pose of the camera

[1] In our experiments, we also include the velocity of the object; we omit the velocity here to simplify the discussion.



(a) tower scenario (b) simulated network of overhead cameras (c) real network of overhead cameras

Figure 5.1: Camera localization. (a) Example scenario with 12 cameras. (b) Results on a simulated overhead camera network with 50 nodes. where the cameras are overhead, the estimates are more uncertain because the object is observed less frequently. (c) The results of our algorithm when run on a real camera network of twenty-five cameras.

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

$$L_t = L_{t-1} + \mathcal{N}\left(0, \sigma^2 I\right) \tag{5.1}$$

where $\mathcal{N}\left(0, I\sigma^2\right)$ is a Gaussian random variable with mean 0 and a diagonal covariance matrix with entries σ^2. When the object appears in the image of camera i, an observation is generated which is represented by a point, $z_{=}(z^x, z^y)$, in the image coordinates of that camera. This observation depends upon the object's state L_t and the camera's pose R_i via

$$\begin{bmatrix} z^x \\ z^y \end{bmatrix} = g(l_t, r_i) + \begin{bmatrix} \delta^x_{t,i} \\ \delta^y_{t,i} \end{bmatrix}, \tag{5.2}$$

where g is the (non-linear) projective transformation for camera i and δ are white noise variables with a small standard deviation (e.g., 3 pixels).

To complete our definition of the probability model, we must specify the prior distribution over the object location at the first time step, $p(l_1)$, and the poses of the cameras $p(r_i)$. 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).

At each time step, we get some set of object observations $\underline{\mathbf{z}}^{(t)}$. The goal is to compute the **posterior distribution** at time t, which is the conditional distribution over the object location and camera poses, given the observations made at each time step:

$$p(l_t, \mathbf{r} \mid \underline{\mathbf{z}}^{(1)}, \underline{\mathbf{z}}^{(2)}, \ldots, \underline{\mathbf{z}}^{(t)}). \tag{5.3}$$

5.2 Problem formulation

The SLAT application, described in the previous section, can be generalized as follows. We will model the system as a dynamic Bayesian network (DBN). A DBN consists of a set of **state processes**, $\mathbf{X} =$

$\{X_1, \ldots, X_L\}$; these random processes characterize the state of the sensor network's environment, and a set of observed **measurement processes** $\mathbf{Z} = \{Z_1, \ldots, Z_K\}$; each measurement process $\mathbf{Z}_k$ corresponds to one of the sensors on one of the nodes. State processes are not associated with unique nodes. A DBN defines a joint probability model over steps $1 \ldots T$ as

$$p(\mathbf{x}^{(1:T)}, \mathbf{z}^{(1:T)}) = \underbrace{p(\mathbf{x}^{(1)})}_{\text{initial prior}} \times \prod_{t=2}^{T} \underbrace{p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)})}_{\text{transition model}} \times \prod_{t=1}^{T} \underbrace{p(\mathbf{z}^{(t)} \mid \mathbf{x}^{(t)})}_{\text{measurement model}} .$$

The initial prior is given by a factorized probability model $p(\mathbf{x}^{(1)}) \propto \prod_A \psi(x_A^{(1)})$ where each $A \subseteq V$ is a subset of the state processes. The transition model factors as

$$p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) = \prod_{i=1}^{L} p(x_i^{(t)} \mid \text{Pa}[x_i^{(t)}]),$$

where $\text{Pa}[x_i^{(t)}]$ are the values of parents of $x_i^{(t)}$ in the previous time step. The measurement model factors as

$$p(\mathbf{z}^{(t)} \mid \mathbf{x}^{(t)}) = \prod_{k=1}^{K} p(z_k^{(t)} \mid \text{Pa}[z_k^{(t)}]),$$

where $\text{Pa}[z_k^{(t)}] \subseteq \mathbf{X}^{(t)}$ are the parents of $z_k^{(t)}$ in the current time step.

The goal of **centralized filtering** is to compute the **posterior distribution** $p(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t)})$ for $t = 1, 2, \ldots$ as the observations $\underline{\mathbf{z}}^{(1)}, \underline{\mathbf{z}}^{(2)}, \ldots$ arrive. In **distributed filtering**, each node n needs to compute (an approximation to) the posterior distribution over some set of query variables Q_n, given all measurements made in the network up to the current time step t: $p(\mathbf{x}_{Q_n}^{(t)} \mid \mathbf{z}^{(1:t)})$. 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.

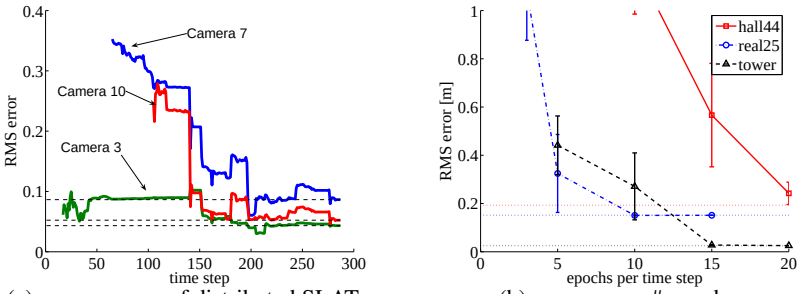
5.3 Approach: Assumed density filtering

The basic approach is to filtering is to recursively compute $p(\mathbf{x}^{(t+1)} \mid \underline{\mathbf{z}}^{(1:t)})$ from $p(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t-1)})$. in three steps:

1. **Estimation**: $p(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t)}) \propto p(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t-1)}) \times p(\underline{\mathbf{z}}^{(t)} \mid \mathbf{x}^{(t)})$;
2. **Prediction**: $p(\mathbf{x}^{(t)}, \mathbf{x}^{(t+1)} \mid \underline{\mathbf{z}}^{(1:t)}) = p(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t)}) \times p(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)})$;
3. **Roll-up**: $p(\mathbf{x}^{(t+1)} \mid \underline{\mathbf{z}}^{(1:t)}) = \int p(\mathbf{x}^{(t)}, \mathbf{x}^{(t+1)} \mid \underline{\mathbf{z}}^{(1:t)}) \mathrm{d}\mathbf{x}^{(t)}$.

Thus, in the estimation step we multiply the current belief by the observation likelihood $p(\underline{\mathbf{z}}^{(t)} \mid \mathbf{x}^{(t)})$, in the prediction step we multiply in the transition model $p(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)})$, and finally, in the roll-up step, we marginalize out the state variables $\mathbf{X}^{(t)}$.

Exact filtering in DBNs is usually expensive or intractable because the belief state rapidly loses all conditional independence structure. An effective approach, proposed by Boyen and Koller (Boyen and Koller, 1998), hereby denoted "B&K98", is to periodically project the belief to a distribution that satisfies independence assertions encoded in a junction tree (Cowell et al., 1999). The intuition behind this approximation is that, to avoid the cost of maintaining dependency information between all variables, we can



(a) convergence of distributed SLAT (b) accuracy v. # epochs

Figure 5.2: (a) RMS of estimated poses versus number of time steps, in the tower scenario, using our distributed algorithm; (b) RMS of the resulting solution as the number of epochs per time step increases. Horizontal lines indicate the quality of the corresponding centralized solution.

instead maintain dependencies between small overlapping subsets of variables. Given a junction tree T, with cliques $\{C_i\}$ and separators $\{S_{i,j}\}$, the approximation amounts to computing the clique marginals. In other words, the filtered distribution is approximated as

$$p(\mathbf{x}^{(t)} \,|\, \underline{\mathbf{z}}^{(1:t-1)}) \approx \tilde{p}(\mathbf{x}^{(t)} \,|\, \underline{\mathbf{z}}^{(1:t-1)}) = \frac{\prod_{i \in N_T} \tilde{p}(\mathbf{x}_{C_i}^{(t)} \,|\, \underline{\mathbf{z}}^{(1:t-1)})}{\prod_{\{i,j\} \in E_T} \tilde{p}(\mathbf{x}_{S_{i,j}}^{(t)} \,|\, \underline{\mathbf{z}}^{(1:t-1)})}, \tag{5.4}$$

where N_T and E_T are the nodes and edges of T, respectively.

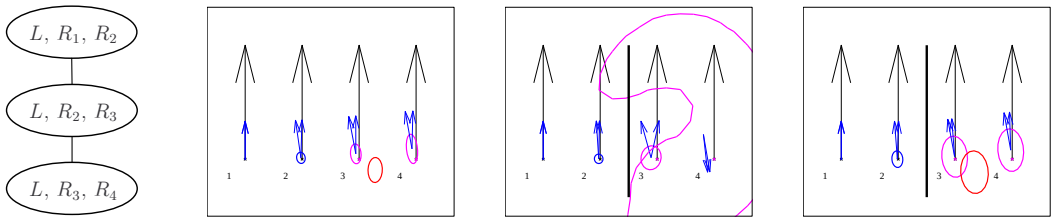
Notice that the filtered distribution (5.4) takes on exactly the same form as the prior distribution in Equation 4.3. This observation suggests a natural distributed implementation of the B&K98 algorithm: Each node maintains the approximate marginal distribution for one or more cliques $\tilde{p}(\mathbf{x}_{C_i}^{(t)} \,|\, \underline{\mathbf{z}}^{(1:t-1)})$. Then, at each time step, the nodes implement the estimation step using the robust message passing algorithm, reviewed in Section 4.3. At convergence, the algorithm computes at each node n the posterior distribution $p(\mathbf{x}_{Q_n}^{(t)} \,|\, \underline{\mathbf{z}}^{(1:t)})$ over a set of query variables Q_n. We will make sure that the set Q_n is large enough, so that the node can locally compute the prediction $\tilde{p}(\mathbf{x}_{C_i}^{(t+1)} \,|\, \underline{\mathbf{z}}^{(1:t)})$.[2] In short, at each time step, the filtering algorithm implements the estimation in a distributed manner, while the prediction and the roll-up step is performed locally.

Consider the tower scenario in Figure 5.1(a). Figure 5.2(a) shows that our distributed algorithm converges to the same solution as the centralized one. 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.

5.3.1 Filtering in face of missing information

As described so far, our distributed filtering algorithm is conceptually simple: at each time step, we run the robust message passing algorithm for certain amount of time, and then we perform the prediction step

[2]This condition can be ensured by including the parents $\mathrm{Pa}[\mathbf{X}_{C_i}]$ in Q_n for each clique C_i, maintained by the node.



(a) density (b) BK solution (c) alignment rooted at 1 (d) alignment rooted at 4

Figure 5.3: (a) Assumed density for a four-camera network. (b-d) Alignment results after partition (shown by vertical line). The circles represent 95% confidence intervals in the estimate of the camera location. (b) The exact solution, computed by the BK algorithm in the absence of partitions. (c) Solution obtained when aligning from node 1. (d) Solution obtained when aligning from node 4.

locally and move on to the next time step. Yet, in realistic deployments, the network may be sufficiently large that the estimation step is never run to convergence. Figure 5.2(b), evaluates the effects of terminating the estimation step early on the quality of the solution at the end of the experiment. We see that, if we allow enough time at each time step, the algorithm converges to the centralized solution. Yet, if the amount of communication is very small (5–10 updates, depending on the size of the network), the algorithm performs very poorly. 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 not have consistent beliefs. In this section, we briefly describe one approach that resolves these inconsistencies.

Consider the example, illustrated in Figure 5.3, in which a network of cameras localizes itself by observing a moving object. Each camera i carries a clique marginal over the location of the object L_t, its own camera pose variable R_i, and the pose of one of its neighboring cameras: $\pi_1(l_t,r_1,r_2)$, $\pi_2(l_t,r_2,r_3)$, and $\pi_3(l_t,r_3,r_4)$. Suppose communication were interrupted due to a network partition: observations would not propagate, and the marginals carried by the nodes would no longer form a consistent distribution, in the sense that π_1,π_2,π_3 might not agree on their marginals, e.g., $\pi_1(l_t, r_2) \neq \pi_2(l_t, r_2)$. The goal of **alignment** is to obtain a consistent distribution $\tilde{p}(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t-1)})$ from marginals π_1, π_2, π_3 that is close to the true posterior $p(\mathbf{x}^{(t)} \mid \underline{\mathbf{z}}^{(1:t-1)})$ (as measured, for example, by the root-mean-square error of the estimates). For simplicity of notation, we omit time indices t and conditioning on the past evidence $\underline{\mathbf{z}}^{(1:t-1)}$ throughout this section.

One way to define a consistent distribution $\tilde{p}$ is to start from a vertex of the junction tree, and allow each clique marginal to decide the conditional density of C_i given its parent, e.g.,

$$\tilde{p}_1(l, \mathbf{r}) = \pi_1(l, r_1, r_2) \times \pi_2(r_3 \mid l, r_2) \times \pi_3(r_4 \mid l, r_3).$$

This density $\tilde{p}_1$ forms a coherent distribution over $L, \mathbf{R}$, and we say that $\tilde{p}_1$ is **rooted** at node 1. Thus, π_1 fully defines the marginal density over L, R_1, R_2, π_2 defines the conditional density of R_3 given L, R_2, and so on. If the clique $\{L, R_3, R_4\}$ were the root, then node 1 would only contribute $\pi_1(r_1 \mid l, r_2)$, and we would obtain a different approximate distribution.

The distributions for two choices of the root are illustrated in Figure 5.3(c) and 5.3(d). Notice that the latter distribution yields sufficiently accurate estimates and can serve as a meaningful approximation of the centralized solution in Figure 5.3(b). The main result of our work is that these different approximations can be evaluated in terms of their *informativeness*, as measured by the entropy of the approximate

distributions $\tilde{p}_i(l, \mathbf{r})$. The best distribution can be then selected, using a dynamic programming algorithm with a structure identical to the robust message passing algorithm. Therefore, we can modify the original robust message passing algorithm to implicitly perform alignment and recover from partitions:
Theorem 1. *Given sufficient communication 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. In the presence of partitions, each partition will reach a consistent belief that minimizes the entropy within this partition.*

For the details of our algorithm and further experimental results, see (Funiak et al., 2006b).

5.4 Discussion

In this chapter, we showed how the robust message passing algorithm can be used to implement assumed density filtering in a distributed manner. We demonstrated the approach on a sensor network application, showing convergence to the centralized solution. A key component of the algorithm was an alignment procedure to resolve the inconsistencies that resulted from updates.

One drawback of assumed density filtering is that once we advance to the next time step, we effectively prune all the information about the observations and beliefs at the previous time steps. This drawback can play a key role in distributed filtering, where the estimation step may never be run to completion. Also, for large networks, the junction tree approach may not scale, since information will need to be propagated a large distance around the network. These issues may make other approximate inference approaches appealing, such as those based on generalized belief propagation (GBP) (Yedidia et al., 2005). We will discuss these approaches in the next chapter.

Chapter 6

Optimization of inference overlays

As we saw in the previous chapters, many inference algorithms use graph structures, such as junction trees or region graphs, in their computation. The performance of a centralized algorithm is determined entirely by i) how the graph is constructed for a given probabilistic model, c.f. (Yedidia et al., 2005)), and ii) by the order in which updates are propagated along the graph, c.f. (Elidan et al., 2006). For example, in the context of generalized belief propagation, a region graph determines the fixed points and the quality of the approximate solution, while the order of updates determines the convergence rate. Yet, in a distributed algorithm, a third aspect needs to be considered: depending on how the graph is mapped to the physical nodes in the network, the algorithm may have a vastly different communication complexity. The graph and its mapping to nodes also affect on the robustness of the distributed algorithm in the presence of node and link failures. In this chapter, we will examine this problem and propose an approach that optimizes the model placement in the context of generalized belief propagation (Yedidia et al., 2005) and tree-based approximations (Grisetti et al., 2007a). We will also describe a distributed algorithm for message scheduling, based on normalized random sampling, that approximates the residual message passing algorithm (Elidan et al., 2006).

6.1 Generalized belief propagation

Recall (Section 2.2.3) that a GBP algorithm is defined in terms of a region graph that must satisfy certain validity conditions. In the simplest, two-way GBP algorithm (Yedidia et al., 2005, Appendix E), the updates are propagated along the edges of this graph. One way to implement the algorithm in a distributed setting is to map each region to a single network node. An update is then implemented by passing a message between the underlying network nodes. A key question is to understand how this mapping affects the communication complexity of the algorithm, and other networking aspects, such as robustness to node failures and sudden changes in the communication topology.

6.1.1 Cluster placement

Let $G = (V, E)$ denote the region graph. Consider a pair of neighboring clusters i and j that are placed to network nodes m and n, respectively. The cost of sending a message between this pair of clusters is determined by the per-unit communication cost between m and n and the size of the message $\mu_{i,j}$.

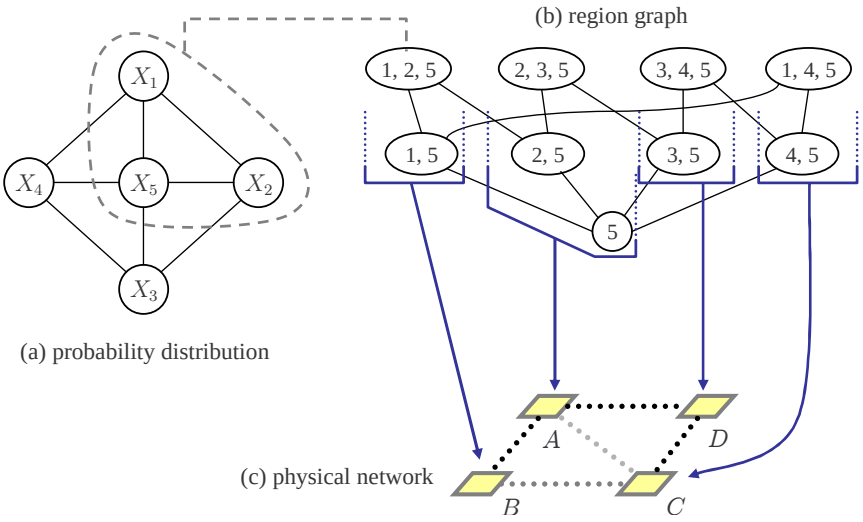


Figure 6.1: (a) A factorized probability distribution. (b) A region graph for the distribution in (a). Each vertex of the graph is associated with a set of variables, i.e., a cluster. (c) The clusters are mapped to physical nodes in the network. The nodes can communicate with varying costs.

For each pair of neighboring clusters i and j, the size of $\mu_{i,j}$ is constant. The overall communication cost is then determined by summing per-edge cost over all edges of the region graph G.[1] The goal is to obtain an optimal **cluster placement**—a mapping of clusters to network nodes that minimizes the overall communication cost.

Formally, cluster placement can be formulated as a graph labeling problem. Let y_i denote the node assigned to cluster i and let $d(y, y') \geq 0$ denote the per-unit communication cost between nodes y and y'. Also, for each edge $\{i, j\} \in E_G$, let $w_{i,j}$ denote the size of the message sent between clusters i and j. Cluster placement can be then formulated as an optimization problem

$$\min_{\mathbf{y}} \sum_{\{i,j\} \in E} w_{i,j}\, d(y_i, y_j). \tag{6.1}$$

We may wish to incorporate one or more additional constraints to this problem. For example, we may fix one or more clusters to a specific node, which amounts to fixing the values of one or more variables y_i. We may require that the clusters at each node cover (i.e., include the arguments of) the factors held at that node. Or, we may impose a budget on the amount of computation performed at each node, by placing a bound on the total number of clusters or edges held at each node. These constraints lead to different variants of the optimization problem (6.1), and we will explore these variants more closely in this thesis.

The choice of d has an important effect on the complexity of the graph labeling problem. Boykov et al. (1999) suggest two versions of the problem:

1. **Semi-metric labeling:** In addition to $d(y, y') \geq 0$, we assume that $d(y, y') = 0$ iff $y = y'$. This is a weak assumption that naturally models direct (single-step) communication, where the communication costs between nodes are always positive.

[1]Here, we assume a uniform message schedule. The approach generalizes to the case when the messages are updated at different rates.

2. **Metric labeling:** Here, we assume that d is a metric, i.e. for any three labels a, b, c, we have $d(a, c) \leq d(a, b) + d(b, c)$. This requirement naturally models multi-hop communication, where a message may be forwarded over multiple nodes to improve the communication cost. The metric d then captures the shortest-path costs.

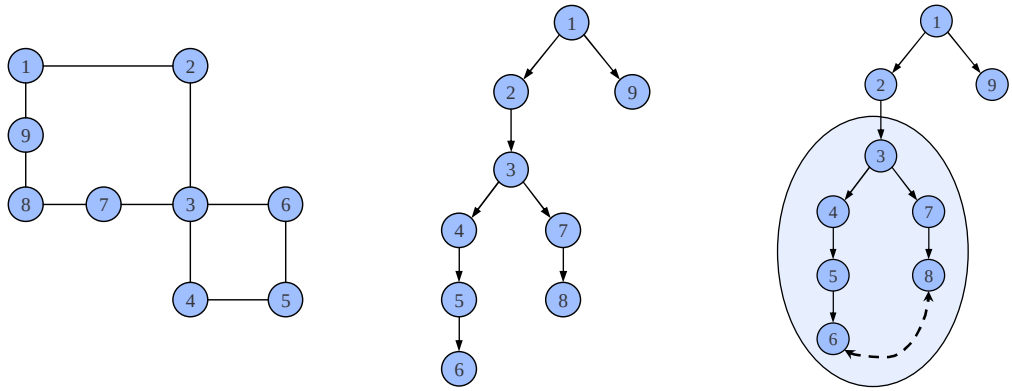
Both semi-metric and metric labeling has been considered in the prior literature. Boykov et al. (1999) propose two methods, called $\alpha\beta$-expansion and α-expansion that locally optimize a version of the problem (6.1) for the semi-metric and the metric case, respectively. Kleinberg and Éva Tardos (2002) propose an LP-relaxation approach that is based on a tree embedding of the metric d. We will examine generalizations of these methods to incorporate the constraints that occur in cluster placement and to understand the decentralized aspects of the problem.

6.1.2 Robustness

While the approach, described in the previous section will decrease the communication complexity of generalized belief propagation, it does not address a key issue: robustness to node failures. If a node fails, the corresponding clusters are removed from the GBP update equations. This change can have a large effect on the approximation quality. For example, consider the region graph in Figure 6.1(b). Suppose that the node that carries the cluster $\{X_5\}$ fails, which corresponds to removing the cluster $\{X_5\}$ from the region graph. The resulting region graph violates the region graph condition (because the counting numbers for X_5 do not sum to 1). One approach is to adjust the placement of clusters dynamically to account for failed nodes. Yet, adjusting the placement (and performing the corresponding content-based addressing) can be slow. An alternative approach is to *replicate* the cluster on several nodes. For example, in Figure 6.1(b), the cluster $\{X_5\}$ could be replicated over the nodes that carry the middle-layer clusters ($\{X_1, X_5\}$, $\{X_2, X_5\}$, and so forth). Replicating the cluster is an effective strategy that is common in the contexts where a variable does not belong to any single node (e.g., the object variable in simultaneous localization and tracking). In some instances, replicating a cluster can actually decrease the communication complexity of an algorithm. For example, if the cluster $\{X_5\}$ is placed on a separate network node, each update will incur 8 messages (four downward, and four upward). Yet, if the cluster is replicated over the nodes that carry the middle-layer clusters, and if the algorithm is executed with a broadcast communication, each update will cost only 4 messages (the upward message can be computed locally at each node). In the context of the graph labeling formulation, broadcasting can be incorporated by defining a special "broadcast" label. When the broadcast label is assigned to a cluster, the cost of the assignment will depend on the labels of all adjacent clusters.

6.1.3 Co-optimization

So far, we have discussed how the graph placement can be adjusted given a fixed region graph. We have largely side-stepped the question of obtaining the region graph in the first place. In practice, it is often easy to select a region graph using domain-specific knowledge (for example, by selecting sufficiently large root regions, and applying the Kikuchi construction (Kikuchi, 1951; Pakzad and Anantharam, 2002)). Similar heuristics can be applied in distributed settings, by selecting the regions of variables within communication range. Yet, many region graphs yield comparable approximation quality, and we may wish to examine methods that trade the quality for the communication complexity. One insight comes from the work by Welling (2004) who proposed a centralized algorithm, called **region graph pursuit**. Noting that several local moves leave the energy invariant, his algorithm identifies irreducible regions to be added to the



(a) representation (b) update

Figure 6.2: The representation, used by (Grisetti et al., 2007a). (a) A Markov network and the representation of the state variables with a tree structure. (b) The update affects all the variables (poses) on the unique path between the constrained variables.

approximation. In a distributed setting, we can first select those irreducible regions that match the network topology, until a fixed budget is reached.

6.2 Tree-based parameterization

Many inference algorithms work with a representation of estimates or beliefs that take on a form of a tree. For example, the expectation propagation algorithm can be used to approximate the belief with as a tree (Minka and Qi, 2003); tree-based parameterizations have been proven effective in the context of rigid body alignment (Grisetti et al., 2007a,b). The vertices of the tree correspond to a variable in the problem (for example, the pose of a module in modular robot localization), while the topology is determined heuristically by forming a spanning tree over the nodes in the network. A key observation is that many spanning trees lead to a comparable quality of approximation. In this case, we can choose to optimize the spanning tree to match the network. In this section, we describe these optimizations in the context of the algorithm of Grisetti et al. (2007a).

6.2.1 Centralized pose estimation

The algorithm of Grisetti et al. (2007a) solves the maximum-likelihood estimation for the localization problem, discussed in Sections 3.1 and 5.1. The input to the algorithm is a pairwise Markov network

$$p(\mathbf{x}) \propto \prod_{(i,j)\in E} \psi_{i,j}(x_i, x_j), \tag{6.2}$$

Each variable x_i is the pose (rotation and translation) of one object (robotic module, camera), and $\psi_{i,j}$ is the spatial constraint that encodes the relation of pose i to pose j. As discussed in Chapter 3, a challenge with maximizing the likelihood of (6.2) is that first-order methods, such as gradient descent, are very slow to converge. A key observation, made by Grisetti et al. (2007a), is that the estimates can be adjusted globally if the poses are arranged in a tree. Specifically, the poses $\mathbf{x}$ are represented by a tree T (Figure 6.2(a)),

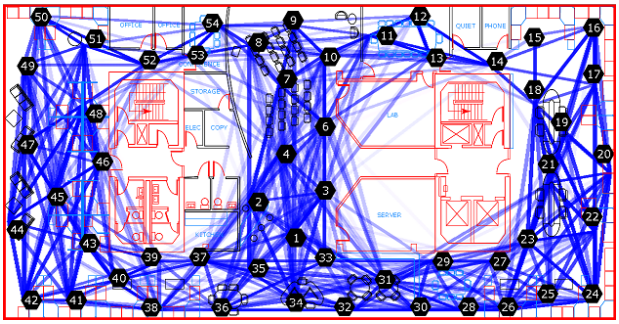


Figure 6.3: The strengths of the communication links in the Intel Research Berkeley deployment.

where each vertex i corresponds to one state variable (pose) x_i and x_i is represented *relative* to its parent j: $x_i \triangleq r_i \circ x_j$. Here, x_j is the pose of the i's parent j, and $\circ$ represents the composition of two rigid body transforms. Whenever an observation is made between x_i and x_j, the error (as determined by the gradient of $\log \psi_{i,j}(x_i, x_j)$) is redistributed across all the nodes on the (unique) path between i and j (see Figure 6.2(b)). In this manner, the algorithm makes a global change that adjusts multiple poses around the loop. Furthermore, since the poses are parameterized relatively to their parents, all the descendants of these nodes are automatically adjusted, too.

6.2.2 Network-aware optimization

When distributing a tree-parameterized algorithm, such as the one described in the previous section, two important questions arise. First, it is not clear how updates from different nodes interact. In a centralized algorithm, each update is executed atomically, and the updates are executed sequentially. In a distributed setting, however, the updates will be executed in parallel, and the nodes will be using stale information in determining the local gradients $\nabla \log \psi_{i,j}(x_i, x_j)$. Also, the spanning tree will need to be maintained, to adjust for changes in the network topology. We anticipate that the analysis techniques from approaches, such as distributed consensus (Mehyar et al., 2005), will help us in design and formal evaluation of the proposed distributed algorithm.

The second question, considered earlier in this section, is to understand how changing the structure of the parameterization tree T affects the message complexity and the convergence of the algorithm. Looking at the Intel Research Berkeley sensor network dataset (Figure 6.3), we notice two things. First, not all pairs of nodes that are in visible range have strong links between them. Therefore, it is vital that the parameterization tree be optimized to avoid these links. Second, there are several links that span a long range. Thus, it may be possible to build a spanning tree that is not too deep and allows the updates to propagate faster.

6.3 Relation to overlay networks

In networking literature, an overlay network is a distributed datastructure that is built on top of another network. Each edge in the overlay network can be implemented with multiple links in the underlying physical network. For example, most peer-to-peer networks are overlay networks because they are built on top of Internet. Also, distributed hash tables, such as Chord (Stoica et al., 2001) are implemented with overlay

networks with structure that permits efficient look-ups. The graphs, considered in this chapter (region graphs, tree parameterizations etc.) can be viewed as overlay networks where each vertex corresponds to an object (cluster, variable, etc.) in the network, rather than a specific physical node. This connection is attractive, because there has been an extensive research in recent years in developing programming languages that simplify the description of network overlays (Loo et al., 2006; Ashley-Rollman et al., 2007). Using languages, such as P2 or Meld may significantly simplify the implementation of the algorithms in this chapter. More importantly, such a connection may offer opportunity for a general procedure, through which the computational graphs can be optimized.

6.4 Improving the convergence

A concern with synchronous iterative algorithms, such as loopy belief propagation, is that they update messages indiscriminately, rather than focusing on the important messages that need to be updated to obtain fast convergence. In the distributed setting, where bandwidth and power consumption are often the limiting factors, updating the messages indiscriminately can be especially costly. An effective centralized approach was proposed by Elidan et al. (2006). Suppose that the update equation (2.7) form a contraction operator:

$$\left\|\boldsymbol{\mu}' - \boldsymbol{\mu}^*\right\| \leq \alpha \left\|\boldsymbol{\mu} - \boldsymbol{\mu}^*\right\|,$$

where $\boldsymbol{\mu}' = \left\{\mu'_{q,r}\right\}$ are the messages computed after an update along a single edge $\{s,t\}$, $\boldsymbol{\mu}^*$ is a fixed point and α is a suitably chosen constant. Then the **residual** $r_{s,t} \triangleq \|\boldsymbol{\mu} - \boldsymbol{\mu}'\|$ (the difference between the previous and newly computed message) is a lower bound on the distance between $\boldsymbol{\mu}$ and the fixed point $\boldsymbol{\mu}^*$. Elidan et al. (2006) use this observation to form a greedy algorithm, called **residual belief propagation** (RBP), that applies the update (2.7) greedily in the decreasing order of residuals $r_{s,t}$.

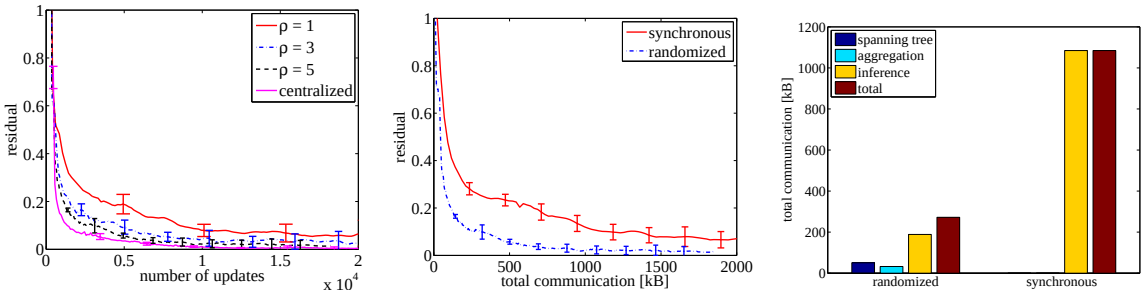
Unfortunately, it is difficult to implement residual belief propagation distributedly, since the algorithm requires maintaining a global priority queue and blocks the computation of the nodes while the node with the leading residual performs the update. A simple strategy (Schiff et al., 2007) is to delay messages with smaller residuals. This strategy can be implemented by performing a sequence of independent Bernoulli trials. At each iteration, the message $\mu'_{s,t}$ is transmitted with probability, determined by the residual $r_{s,t}$:

$$p_{s,t} = \left\|\mu'_{s,t} - \mu_{s,t}\right\|^{\rho} \tag{6.3}$$

for a suitably chosen norm $\|\cdot\|$ and constant $\rho \geq 0$. Effectively, the messages with larger residuals $r_{s,t} \triangleq \|\mu'_{s,t} - \mu_{s,t}\|$ are transmitted more often (here, we assume that the residual $r_{s,t} \leq 1$).

Still, the strategy of Schiff et al. (2007) has a significant drawback: as the messages get closer to the fixed point, the transmission probabilities $\{p_{s,t}\}$ go to 0. Therefore, the algorithm will eventually stop making progress and will *never* converge. Nevertheless, we can use an aggregation overlay to compute the sum of the norms (6.3), and normalize the probabilities $\{p_{s,t}\}$ to some pre-determined update rate u. The normalization constant $(\sum_{s,t} r^{\rho}_{s,t})/u$ is computed periodically, which ensures that the algorithm continues to make progress. As illustrated in Figure 6.4, the resulting algorithm offers substantial improvement in bandwidth usage over the naive message schedule.

We can show that each step of the residual belief propagation algorithm is approximated by the normalized sampling. Let $X_i \sim Geom(p_i)$ denote the number of trials until message i is successfully sent. Let $Y = \min X_i$ be the step when the earliest message is sent (multiple messages may be sent at the same time). Let R denote the total residual of messages sent at step Y. We would like to show that only a few



(a) comparison to centralized (b) convergence (c) bandwidth

Figure 6.4: Results on 10×10 random Ising grid models. Following (Elidan et al., 2006), some of the potentials are attractive, while others are repulsive, with potential strength $\exp(\beta)$ for β uniformly sampled in the range $[-5; 5]$. (a) The effect of ρ parameter in the randomized belief propagation algorithm. As ρ increases, the algorithm converges to the centralized residual BP algorithm. (b) The convergence of two distributed loopy BP algorithms as a function of the bandwidth used. (c) The bandwidth requirements of different components of our algorithms.

messages are sent at step Y, and that the total residual of these messages R is close to the residual $\max_i r_i$ selected by RBP that starts with the same set of residuals.[2]

Proposition 1. *Let $m = 2|E|$ denote the number of messages, let $u = \sum p_i$ denote the update rate, let M denote the number of messages transmitted at step Y, and let R denote the total residual of messages transmitted at step Y. Then $\mathbb{E}[M] \leq \frac{u}{1-e^{-u}}$. Furthermore,*

$$\frac{\mathbb{E}[R]}{\max_i r_i} \geq \frac{1 + (m-1)r^{\rho+1}}{1 + (m-1)r^{\rho}},$$

where r is the unique positive root of the polynomial $(m-1)r^{\rho+1} + (\rho+1)r - \rho = 0$. In particular, when $\rho = 1$, $\frac{\mathbb{E}[R]}{\max_i r_i} = \Omega(m^{-1/2})$.

Note that the normalized sampling approach is not limited to the standard belief propagation. A similar approach could be used for distributed Generalized Belief Propagation.

6.5 Discussion

In this chapter, we discussed how graph datastructures, employed in inference algorithms such as GBP or stochastic gradient descent, interact with a physical network in a distributed setting. We observed that, in order to obtain a robust algorithm with a low message complexity, the mapping of the graph datastructure to the physical network needs to be taken into account. This consideration leads to natural formulations that optimize the graph placement and the graph topology. Finally, we described an effective message scheduler, based on normalized random sampling.

[2] We make no statement about the relative performance of the two algorithms executed over multiple steps.

Chapter 7

Learning models for collaborative filtering

In the previous chapters, we considered a distributed inference problem, in which the nodes coordinate, in order to recover the estimates that incorporate the observations from all the network. Yet, in many interesting applications, the key difficulty does not lie with the inference; rather, the challenging part is to construct a global model that is based on data scattered across the network. In this chapter, we consider one such problem in the context of collaborative filtering for recommendation systems.

7.1 Application: Distributed recommendation systems

In recent years, recommendation systems have become increasingly popular. Services, such as Last.fm, Netflix, or iTunes Genius provide users with automated suggestions that are computed using the data gathered from a large pool of users. By comparing the user's own preferences against the preferences of other users, the services are able to provide recommendations for new items and substantially improve the user experience. Side information, such as the music genre or actors' names, can be used to refine the recommendations.

A natural task, considered in this thesis, is to design a recommendation system that can operate in a peer-to-peer setting. Such a system can be very compelling among the users who wish to receive recommendations for content, but do not subscribe to a commercial service, such as Netflix. The system can be particularly interesting to users of home media centers, such as Xbox or XBMC.[1] Open-source media center software, such as XBMC, has a large community of active users and developers, which may prove useful in the deployment and potential adoption of a distributed recommendation system.

A distributed recommendation system needs to address several challenges. It needs to scale to hundreds of thousands of users, it needs to use only a moderate amount of bandwidth (most users only have a DSL connection), and it must tolerate a high fluctuation of nodes that enter and leave the network. In order to better understand the application requirements, we consider a number of common usage scenarios. Table 7.1 summarizes the parameters of a hypothesized deployment.

[1] `http://xbmc.org/`

Requirement	Movie recommendations	Music recommendations
Items (movies, songs)	10,000	1,000,000
Latent variables	100	100
Number of ratings	5	2
Size of a complete model	40MB	800 MB
Queries	10 / week	1000 / week
New items	1000 / year	100,000 / year

Table 7.1: Requirements of a distributed recommendation system. The values were estimated based on publicly available information about existing centralized services, such as Last.fm and iTunes Store.

- **Recommendation queries:** This is the most frequent operation, provided by the service. In order to provide visual cues and obtain feedback from the user, the system needs to provide several (10 to 20) high-score recommendations at a time, with a minimal delay (at most 1 second). This performance parameter enables instant movie recommendations and dynamic playlist updates.
- **Rating:** New ratings need to be processed less frequently than queries. On average, the user will rate a few movies per week and tens of songs per day. Unlike movie ratings, which take on a numerical value, music ratings often take on only a binary value ("loved" or "banned" songs in Last.fm), or are measured indirectly, based on how long the user listened to the song.
- **New item:** We may assume that each item (movie or music) is associated with a unique id, which can be a CDDB track id, an IMDB movie id, or a canonical title string. Side information about a movie or a music can be retrieved from services, such as Last.fm or IMDB. Alternatively, side information can be entered by the users, using the service.
- **New user:** When a user joins, he or she needs to be provided with meaningful ratings from the very beginning. These ratings can be seeded with user's initial selection (as performed by Last.fm or Pandora), as well as by the songs that are present on the user's drive.

In summary, the system will need to support rapid queries, moderate updates, and we may safely assume that the name matching problem is solved. The system will need to continuously update the ratings, and each node may or may not be able to store the model for all the songs.

7.2 Latent variable models for collaborative filtering

Following the prior work (Salakhutdinov et al., 2007; Singh and Gordon, 2008), our approach learns a latent variable model that characterizes the relationship between the rating of an item and the type of a user. Here, we focus on the Restricted Boltzmann machine (RBM) model of Salakhutdinov et al. (2007); the discussion in the following sections generalizes to other latent variable models, such as that of Singh and Gordon (2008).

In the RBM model of Salakhutdinov et al. (2007), each user is associated with two sets of random variables. The vector of binary **latent variables** $\mathbf{Y}$ represents the (unknown) type of a user. This vector can be thought of as capturing features, such as the gender or age group of the user and their combination, but the actual interpretation is not recovered in the learning procedure. The (partially observed) set of

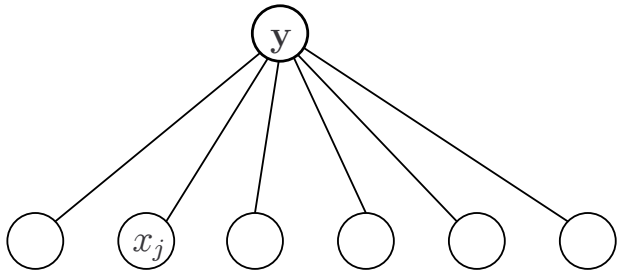


Figure 7.1: The graphical model for the RBM model for collaborative filtering.

rating variables $\mathbf{X} = \{X_1, \ldots, X_N\}$ represent how the user would rate each movie in the database.[2] For each user i, the latent variables $\mathbf{Y}^{(i)}$ and the ratings $\mathbf{X}^{(i)}$ are distributed according to a generalized linear model

$$p(\mathbf{y}, \mathbf{x}; \mathbf{w}) \propto \exp \sum_j (w_j^T \mathbf{y}) x_j, \tag{7.1}$$

where w_j are the parameters that specify the interactions between the latent variables and the rating of movie j. The bias terms can be added, which is equivalent to fixing one latent variable and one rating variable to 1. Figure 7.1 shows the Markov network for the model (7.1).

A key property, exploited in this thesis, is that once the model has been trained, making predictions on a previously unseen movie is simple. Let A denote the set of movies rated by user i. Looking at Figure 7.1, we see that the the prediction on movie j can be made by first computing the conditional distribution over the latent variables given the observed movies: $p(\mathbf{y} \,|\, \mathbf{x}_A = \mathbf{x}_A^{(i)})$.[3] Then, we multiply in the conditional distribution of the rating x_j for the new movie i and marginalize out the latent variables:

$$p(x_j \,|\, \mathbf{x}_A = \mathbf{x}_A^{(i)}) = \sum_{\mathbf{y}} p(x_j \,|\, \mathbf{y}) \times p(\mathbf{y} \,|\, \mathbf{x}_A = \mathbf{x}_A^{(i)}).$$

Provided that a network node carries the entire model (or parts of the model for movies $A \cup \{j\}$, if we employ an approximation implicitly assumed by Salakhutdinov et al. (2007)), the prediction step is entirely local. A similar property is exhibited by the matrix factorization model of Singh and Gordon (2008).

The model (7.1) is shared by all the users in the system. It is typically trained by maximizing the marginal log-likelihood of the observed ratings, summed over all the users:

$$\log(\mathcal{D}; \mathbf{w}) = \sum_i \log p(x_{A_i} = x_{A_i}^{(i)}; \mathbf{w}). \tag{7.2}$$

Here, A_i is the set of movies rated by user i. The log-likelihood (7.2) is maximized using gradient descent. While computing the exact gradient is intractable, a standard approximation that uses contrastive divergence (Hinton, 2002) performs well. Importantly, in computing the approximate gradient, each user contributes only to the parameters for the movies the user has rated. This sparsity structure is present in other methods, such as (Singh and Gordon, 2008), and is central for the proposed work.

[2]For simplicity of discussion, we will assume that the variables X_j are binary, see (Salakhutdinov et al., 2007) for a complete model.

[3]More precisely, we compute the conditional distribution in the part of the model that includes all the movies other than j.

7.3 Distributed learning

In a distributed setting, the ratings for each user reside locally at the user's machine. Since a model learnt from a single user's ratings is uninformative, the nodes need to collaborate, in order to learn a model that incorporates the data from all the users. We propose two approaches to perform such a learning:

- **Fully decentralized learning:** When the nodes in the network are homogeneous and frequently enter and leave the network, it may be desirable to employ a fully decentralized algorithm. Each node participates in updating the parameters for a subset of the movies. The node computes its local contribution to the gradient and periodically communicates the estimated gradient and the parameters to its neighbors. A node may perform multiple local gradient steps before between the communication rounds. One approach is to perform distributed consensus (Mehyar et al., 2005). Since Internet is a highly connected network (virtually any pair of nodes can communicate), the communication patterns of the consensus algorithm can be adjusted to guarantee fast convergence.
- **Learning with supernodes:** It has been observed that, while nodes fluctuate in P2P, connecting and disconnecting rapidly, a small set of stable, high-bandwidth nodes is often present. (Yang and Garcia-Molina, 2003) These nodes often form supernodes in today's P2P protocols, and can be used to perform learning. A simple approach is to partition the movies into groups (for example, with uniform hashing), and then let each supernode participate in updating the parameters for a single group. A regular node can either submit its ratings to a supernode, or participate in the updates for the movies it has rated. Since each user only rates a small number of items, the amount of communication by the regular node will be small.

A fully decentralized learning offers interesting opportunities and may be more robust to malicious attacks by a small group of users. A solution that employs supernodes is conceptually simpler and more tangible in the short term.

7.4 Recommendations with an out-of-date model

As discussed earlier, predicting the user's rating for a specific movie is an easy task: the prediction can be performed locally, using only the distribution over the latent variables $p(\mathbf{y} \mid \mathbf{x}_{A_i}^{(i)})$ and the parameters for movie j. Yet, in realistic situations, we are not interested in determining the rating of a single movie; rather, we wish to suggest several top-rated movies. Therefore, in addition to designing a distributed learning algorithm, we also need to design a scheme to recommend top-rated movies.

One way to perform the recommendations is to store an out-of-date copy of the complete model at each user's machine and periodically update the parameters whose values are expected to have a large impact on the prediction quality. This approach has a benefit in that it does not require the user to have a permanent connection to the network. Let S denote the set of movies recommended to a user, using the local copy of the model. The prediction quality can be measured by the regret between the scores of the movies and the scores of the best set A:

$$\max_{A:|A|=|S|} \sum_{j \in A} s_j - \sum_{j \in S} s_j. \tag{7.3}$$

Here, s_j represents the score of movie j for the user using the true model parameters (e.g., the predicted rating of the movie or the log-odds).

Naturally, without any further information, it is very difficult to determine, the parameters for which movie need to be updated. Sudden spikes in ratings frequency triggered by new movie releases can substantially change the optimal prediction. Even to detect slow trends, one may need to perform multiple samplings over the entire dataset. Nevertheless, we can assume that a small amount of information is provided to each node by the supernodes at regular intervals. For example, each node may receive the number of new ratings that were performed for a movie, or the smoothness of the parameter surface. This assumption can lead to two formulations:

- **Bounded score:** Under certain assumptions, the number of new ratings can be used to derive a loose bound on the change in the score. Given such a bound, it is easy to upper-bound the regret (7.3) and then derive a greedy strategy that optimistically selects the movie with a maximum decrease in the regret. It may be also possible to use a k-armed bandit-style analysis to bound the regret with respect to the optimal update ordering.
- **Distribution on the parameters:** A recent paper (Crammer et al., 2009) described a learning setting where each parameter is associated with uncertainty. A distribution on the parameters can be directly converted to a distribution on the score. It may be then possible to apply the value-of-information approaches, to select a near-optimal sequence of updates.

7.5 Discussion

In this chapter, we proposed an approach for distributed collaborative filtering. A key observation is that P2P networks typically contain stable nodes that can perform learning. In order to make predictions, the nodes will periodically update their parameters with a pull-based method.

Chapter 8

Conclusions and thesis plan

We presented several algorithms for probabilistic inference in distributed systems. To summarize the main contributions to-date, we have covered the following topics:

- We have developed a distributed algorithm for maximum likelihood estimation for modular robot localization. By interleaving global alignment and local refinement steps, the algorithm substantially improved upon state-of-the-art approaches.
- We provide several results that improve upon existing inference algorithms:
    1. We developed a simplified interpretation of the robust message passing algorithm (Paskin and Guestrin, 2004). We show that the algorithm identifies the edges in the global junction tree using information that is locally available to each node. This interpretation can be used to design an algorithm with stronger approximation guarantees.
    2. We propose an approach, based on normalized sampling, to schedule messages in loopy belief propagation, and relate the algorithm the greedy schedule in the residual message passing algorithm (Elidan et al., 2006).
- We developed a distributed algorithm for dynamic inference using assumed density filtering with junction trees that generalizes upon approaches used in distributed SLAM. We identified an important problem, inconsistency, that arises when communication is interrupted, and proposed an algorithm that selects the most informative distribution among a set of candidates.
- We demonstrated our algorithms on realistic applications in sensor networks. In the process, we developed a novel parameterization of camera poses that allows the posterior be effectively approximated with a Gaussian distribution.

In addition to the contributions listed above, the proposed work will include the new techniques and applications, proposed in Chapters 6 and 7. The additional expected contributions will be as follows:

- We will describe the application of generalized belief propagation to distributed filtering. We will demonstrate the feasibility of centralized GBP on the camera localization application, and evaluate the effects of network partitions and missing messages on the beliefs.
- We will develop an algorithm for mapping the region graph in GBP to the physical network, in order to minimize the communication cost of inference. We will consider extensions that increase

robustness, such as replication of clusters.

- We will develop a method that adjusts the topology of tree parameterizations to match the network.
- We will develop a distributed algorithm for learning latent variable models. The algorithm will provide performance that is close to the centralized learner that has access to all the data.
- We will demonstrate our distributed learning algorithm on a novel application in collaborative filtering that provides recommendations for movies or music on home media centers.
- Optionally, we will develop tools for analyzing and proving the convergence of distributed inference algorithms that make asynchronous, non-local steps. These tools will allow us to understand iterative methods for distributed optimization and inference, and may generalize to a larger set of problems.

The following is a proposed timeline leading up to a thesis defense in Spring 2010:

- Spring 2009
  - Investigate the pull-based update scheme for collaborative filtering.
  - Experiment with the GBP algorithm for the SLAT application.
  - Develop a basic algorithm for optimizing the placement of a region graph on the network.
  - Explore an approach that simultaneously optimizes the region graph and its placement on the network.
- Fall 2009
  - Develop a distributed algorithm for optimizing tree parameterization in the context of SLAT or modular robot localization.
  - Understand formal methods for analyzing convergence of distributed inference algorithms. Develop a distributed version of the learner based on consensus. (optional)
  - Experiment with other datasets for collaborative filtering, such as Last.fm.
  - Begin writing the thesis.
- Spring 2010
  - Wrap up the implementation and experiments
  - Finish writing thesis.
  - Thesis defense.

Bibliography

Michael Ashley-Rollman, Seth Copen Goldstein, Peter Lee, Todd Mowry, and Padmanabhan Pillai. Meld: A declarative approach to programming ensembles. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2007. 3.4, 6.3

Pratik Biswas, Tzu-Chen Lian, Ta-Chung Wang, and Yinyu Ye. Semidefinite programming based algorithms for sensor network localization. *ACM Transactions on Sensor Networks (TOSN)*, 2(2):188–220, May 2006. 1, 2.3.1

Rahul Biswas and Sebastian Thrun. A distributed approach to passive localization for sensor networks. In *AAAI*, pages 1248–1253, 2005. 1, 2.3.1

X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Annual Conference on Uncertainty in AI*, pages 33–42, 1998. 2.3.2, 5.3

Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Computer Vision, IEEE International Conference on*, 1:377, 1999. 6.1.1, 6.1.1

Vladimir Bychkovskiy, Seaphan Megerian, Deborah Estrin, and Miodrag Potkonjak. 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, 2003. 4.1

C.T. Chu, S.K. Kim, Y.A. Lin, Y. Yu, G.R. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288. MIT Press, 2006. 2.3.4

M. Chu, H. Haussecker, and F. Zhao. Scalable information-driven sensor querying and routing for ad hoc heterogenous sensor networks. *International Journal of High-Performance Computing Applications*, 16(3), 2002. 2.3.1

R. Cowell, P. Dawid, S. Lauritzen, and D. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, New York, NY, 1999. 2.2, 2.4, 4.2, 4.3, 5.1, 5.3

Koby Crammer, Mark Dredze, and Fernando Pereira. Exact convex confidence-weighted learning. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*. 2009. 7.4

Christopher Crick and Avi Pfeffer. Loopy belief propagation as a basis for communication in sensor networks. In Chris Meek and Uffe Kjærulff, editors, *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI-2003)*, San Francisco, 2003. Morgan Kaufmann Publishers, Inc. 2.3.3, 3

Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications ACM*, 51(1): 107–113, 2008. 2.3.4

J. Djugash, S. Singh, and B. Grocholsky. Decentralized mapping of robot-aided sensor networks. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 583–589, 2008. 1, 2.3.1

G. Elidan, I. Mcgraw, and D. Koller. Residual belief propagation: Informed scheduling for asynchronous message passing. In *Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI)*, Boston, Massachussetts, 2006. 6, 6.4, 6.4, 2

Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots*, 8(3):325–344, June 2000. 2.3.2

S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankar. Distributed localization of networked cameras. pages 34–42, 2006a. 1, 1.2, 2.3.1, 5

Stanislav Funiak, Carlos Guestrin, Mark Paskin, and Rahul Sukthankar. Distributed inference in dynamical systems. In *Advances in Neural Information Processing Systems 19*. MIT Press, 2006b. 1.2, 5.3.1

Stanislav Funiak, Padmanabhan Pillai, Michael Ashley-Rollman, Jason Campbell, and Seth Goldstein. Distributed localization

of modular robot ensembles. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008. 1.2, 3.2

Joseph E. Gonzalez, Yucheng Low, and Carlos Guestrin. Residual splash for optimally parallelizing belief propagation. In *submitted to AISTATS '09*, 2009. 2.3.4

G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007a. 6, 6.2, 6.2, 6.2.1, 6.2.1

Giorgio Grisetti, Slawomir Grzonka, Cyrill Stachniss, Patrick Pfaff, and Wolfram Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *Proceedings of IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2007b. 6.2

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 14(8):1771–1800, 2002. ISSN 0899-7667. doi: http://dx.doi.org/10.1162/089976602760128018. 7.2

Geoffrey Hollinger and Sanjiv Singh. Proofs and experiments in scalable, near-optimal search by multiple robots. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008. 1

Alexander T. Ihler, III John W. Fisher, Randolph L. Moses, and Alan S. Willsky. 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, NY, USA, 2004. ACM Press. 1, 2.3.1, 2.3.3, 4.1

R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45, 1960. 2.3.1

Ryoichi Kikuchi. A theory of cooperative phenomena. *Physical Review*, 81(6):988+, March 1951. 6.1.3

Jon Kleinberg and Éva Tardos. Approximation algorithms for classification problems with pairwise relationships: metric labeling and markov random fields. *J. ACM*, 49(5):616–639, 2002. ISSN 0004-5411. 6.1.1

Kurt Konolige, Dieter Fox, Charlie Ortiz, Andrew Agno, Michael Eriksen, Benson Limketkai, Jonathan Ko, Benoit Morisset, Dirk Schulz, Benjamin Stewart, and Regis Vincent. Centibots: Very large scale distributed robotic teams. pages 131–140. 2006. 2.3.2

Juan Liu, James Reich, and Feng. Collaborative in-network processing for target tracking. *EURASIP Journal on Applied Signal Processing*, 4:378–391, 2003. 2.3.1

Boon T. Loo, Tyson Condie, Minos Garofalakis, David A. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. Declarative networking: Language, execution and optimization. 2006. 6.3

James Manyika and Hugh D. Whyte. *Data Fusion and Sensor Management: A Decentralized Information-Theoretic Approach*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1995. ISBN 0133031322. 2.3.1, 2.3.2

M. Mehyar, D. Spanos, J. Pongsajapan, S. H. Low, and R. M. Murray. Distributed averaging on asynchronous communication networks. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC '05. 44th IEEE Conference on*, pages 7446–7451, 2005. 2.3.3, 6.2.2, 7.3

Thomas Minka and Yuan Qi. Tree-structured approximations by expectation propagation. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2003. 6.2

E. W. Nettleton, P. W. Gibbens, and H. F. Durrant-Whyte. Closed form solutions to the multiple platform simultaneous localization and map building (slam) problem. In *in Sensor Fusion: Architectures, Algorithms, and Applications IV*, pages 428–437, 2000. 2.3.2

Payam Pakzad and Venkat Anantharam. Minimal graphical representation of kikuchi regions. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, pages 1585–1594, 2002. 6.1.3

M. Paskin, C. Guestrin, and J. Mcfadden. A robust architecture for distributed inference in sensor networks. pages 55–62, 2005. 2.3.3, 4.2

Mark A. Paskin. Thin junction tree filters for simultaneous localization and mapping. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1157–1164, San Francisco, CA, 2003. Morgan Kaufmann Publishers. 5.3

Mark A. Paskin. *Exploiting Locality in Probabilistic Inference*. PhD thesis, University of California, Berkeley, August 2004. 4.2, 4.3

Mark A. Paskin and Carlos E. Guestrin. Robust probabilistic inference in distributed systems. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 436–445. AUAI Press, 2004. ISBN 0974903906. 2.3.3, 4, 4.3,

4.4, 1

Avi Pfeffer and Terry Tai. Asynchronous dynamic Bayesian networks. In *Proceedings UAI 2005*, 2005. 2.3.3

Ali Rahimi, Brian Dunagan, and Trevor Darrell. Simultaneous calibration and tracking with a network of non-overlapping sensors. In *CVPR*, 2004. 1, 2.3.1, 5.1

Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. 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, 2007. ACM. ISBN 9781595937032. 1

Matthew Rosencrantz, Geoff Gordon, and Sebastian Thrun. Locating moving entities in dynamic indoor environments with teams of mobile robots. In *Proceedings of Autonomous Agents and Multi-Agent Systems*, Melbourne, Australia, 2003. 2.3.2

K. D. Roufas, Y. Zhang, D. G. Duff, and M. H. Yim. Six degree of freedom sensing for docking using IR LED emitters and receivers. In *Proceedings of International Symposium on Experimental Robotics VII*, 2000. 1

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the International Conference on Machine Learning*, volume 24, pages 791–798, 2007. 7.2, 7.2, 2

J. Schiff, D. Antonelli, A. G. Dimakis, D. Chu, and M. J. Wainwright. 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, 2007. 2.3.3, 6.4, 6.4

Roman Schmidt and Karl Aberer. Efficient peer-to-peer belief propagation. In *OTM Conferences (1)*, pages 516–532, 2006. 2.3.3

Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz. 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, 2003. 1, 2.3.1

Ajit P. Singh and Geoff J. Gordon. A unified view of matrix factorization models. In *Machine Learning and Knowledge Discovery in Databases, European Conference (ECML/PKDD)*, 2008. ECML/PKDD-2008. 1, 7.2, 7.2, 7.2

Amarjeet Singh, Andreas Krause, Carlos Guestrin, William Kaiser, and Maxim Batalin. Efficient planning of informative paths for multiple robots. In *In IJCAI*, 2007. 1

Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001. 6.3

Kasper Støy. *Emergent Control of Self-Reconfigurable Robots*. PhD thesis, University of Southern Denmark, 2003. 1

Nadeem Ahmed Syed, Nick Feamster, Alexander G.Gray, and Sven Krasser. Snare: Spatio-temporal network-level automatic reputation engine. Technical Report GT-CSE-08-02, Georgia Tech, 2008. 1

Christopher Taylor, Ali Rahimi, Jonathan Bachrach, Howard E. Shrobe, and Anthony Grue. Simultaneous localization, calibration, and tracking in an ad hoc sensor network. In *Proceedings of IPSN*, 2006. 2.3.1

S. Thrun. Robotic mapping: A survey, 2002. 2.4

S. Thrun and Y. Liu. Multi-robot SLAM with sparse extended information filers. In *Proceedings of the 11th International Symposium of Robotics Research (ISRR'03)*, Sienna, Italy, 2003. Springer. 2.3.2

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping, 2000. 2.3.2

Sebastian Thrun, Yufeng Liu, Daphne Koller, , Andrew Ng, Zoubin Ghahmarani, and Hugh Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research*, 2004. 2.3.2

Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(4), April 1991. 3.2, 3.3

Zizhuo Wang, Song Zheng, Stephen Boyd, and Yinyu Yez. Further relaxations of the SDP approach to sensor network localization. Technical report, Stanford University, 2006. 2.3.1

M. Welling. On the choice of regions for generalized belief propagation. *Proceedings of the 20th conference on Uncertainty in ldots*, Jan 2004. 6.1.3

Lin Xiao and S. Boyd. Fast linear iterations for distributed averaging. In *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, volume 5, pages 4997–5002 Vol.5, 2003. 2.3.3

B. Beverly Yang and H. Garcia-Molina. Designing a super-peer network. *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60, March 2003. 7.3

J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *Information Theory, IEEE Transactions on*, 51(7):2282–2312, 2005. 1.1, 2.2.3, 2.3.3, 5.4, 6, 6.1

Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems: Challenges and opportunities for the future. *IEEE Robotics & Automation Magazine*, 14:43–52, March 2007. 1