June, 2005

LIDS Publication # 2644

Research supported in part by:

ODDR&E MURI: ARO Grant DAAD19-00-0466; AFOSR Grant F49620-00-0362; MIT Lincoln Laboratory Program 2209-3023.

Inference in Sensor Networks: Graphical Models and Particle Methods

Alexander T. Ihler

Inference in Sensor Networks: Graphical Models and Particle Methods

by

Alexander T. Ihler

B.S., Electrical Engineering and Mathematics, Caltech, 1998

S.M., Electrical Engineering and Computer Science, MIT, 2000

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

Doctor of Philosophy
in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

June, 2005

Signature of Author: _____
Department of Electrical Engineering and Computer Science
February 28, 2005

Certified by: _____
Alan S. Willsky
Edwin Sibley Webster Professor of Electrical Engineering
Thesis Supervisor

Certified by: _____
John W. Fisher III
Principal Research Scientist, CSAIL
Thesis Supervisor

Accepted by: _____
Arthur C. Smith
Professor of Electrical Engineering
Chair, Committee for Graduate Students

Inference in Sensor Networks: Graphical Models and Particle Methods

by Alexander T. Ihler

Submitted to the Department of Electrical Engineering
and Computer Science on March 1, 2005
in Partial Fulfillment of the Requirements for the Degree
of Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Sensor networks have quickly risen in importance over the last several years to become an active field of research, full of difficult problems and applications. At the same time, graphical models have shown themselves to be an extremely useful formalism for describing the underlying statistical structure of problems for sensor networks. In part, this is due to a number of efficient methods for solving inference problems defined on graphical models, but even more important is the fact that many of these methods (such as belief propagation) can be interpreted as a set of message passing operations, for which it is not difficult to describe a simple, distributed architecture in which each sensor performs local processing and fusion of information, and passes messages locally among neighboring sensors.

At the same time, many of the tasks which are most important in sensor networks are characterized by such features as complex uncertainty and nonlinear observation processes. Particle filtering is one common technique for dealing with inference under these conditions in certain types of sequential problems, such as tracking of mobile objects. However, many sensor network applications do not have the necessary structure to apply particle filtering, and even when they do there are subtleties which arise due to the nature of a distributed inference process performed on a system with limited resources (such as power, bandwidth, and so forth).

This thesis explores how the ideas of graphical models and sample–based representations of uncertainty such as are used in particle filtering can be applied to problems defined for sensor networks, in which we must consider the impact of resource limitations on our algorithms. In particular, we explore three related themes. We begin by describing how sample–based representations can be applied to solve inference problems defined on general graphical models. Limited communications, the primary restriction in most practical sensor networks, means that the messages which are passed in the inference process must be approximated in some way. Our second theme explores the consequences of such message approximations, and leads to results with implications both for distributed systems and the use of belief propagation more generally. This naturally raises a third theme, investigating the optimal cost of representing sample–based estimates of uncertainty so as to minimize the communications required. Our analysis shows several interesting differences between this problem and traditional source coding methods. We also use the metrics for message errors to define lossy or approximate

encoders, and provide an example encoder capable of balancing communication costs with a measure on inferential error.

Finally, we put all of these three themes to work to solve a difficult and important task in sensor networks. The self-localization problem for sensors networks involves the estimation of all sensor positions given a set of relative inter-sensor measurements in the network. We describe this problem as a graphical model, illustrate the complex uncertainties involved in the estimation process, and present a method of finding for both estimates of the sensor positions and their remaining uncertainty using a sample–based message passing algorithm. This method is capable of incorporating arbitrary noise distributions, including outlier processes, and by applying our lossy encoding algorithm can be used even when communications is relatively limited. We conclude the thesis with a summary of the work and its contributions, and a description of some of the many problems which remain open within the field.

Thesis Supervisors: Alan S. Willsky
Professor of Electrical Engineering and Computer Science

John W. Fisher III
Principal Research Scientist

Acknowledgments

Back where I come from, we have universities, seats of great learning, where men go to become great thinkers. And when they come out, they think deep thoughts and with no more brains than you have. But they have one thing you haven't got: a diploma.
The Wizard of Oz

It's true hard work never killed anybody,
but I figure, why take the chance?
R. Reagan

If ever I imagined that this process would not comprise hard work, I've certainly since been disabused of the notion. But in another sense, it is still difficult to think of these past years as being work, as they were mostly comprised of thinking about problems I'd have been glad to spend my time on anyway. Still, in order to finish I've needed to rely on the assistance and support of a great many people around me, whom I would gratefully like to acknowledge.

First and foremost among these, of course, are my advisors, Prof. Alan Willsky and Dr. John Fisher. I cannot measure, much less describe, how much they both have helped me through the years. Alan has been a constant, always quick to understand an idea and fold it into a bigger picture. Meanwhile, John was always willing to brainstorm and hear new ideas, no matter how premature or half-baked.

My thanks also go to Prof. Randy Moses of the Ohio State University without whose collaboration Chapter 6 would not exist. Many thanks to the other members of my thesis committee, Prof. Bill Freeman and Prof. Sanjeev Kulkarni, for all of their advice and assistance throughout the process. MIT has been a wonderful place to work and interact with researchers in many areas, and I've been grateful for the opportunity to interact both with members of LIDS and CSAIL. I would particularly like to thank Prof. Sanjoy Mitter, Prof. Lizhong Zheng, and Prof. Trevor Darrell for all their helpful discussions.

However, the thing I am most grateful for at MIT has been my fellow students. Thanks to the members of SSG, both past and present. First, Junmo Kim, with whom I have shared an office almost since we both arrived, and will almost until we both depart. Thanks to my grouplet, lately consisting of Lei Chen, Jason Williams, and Emily Fox, as well as Dr. Müjdat Çetin, for letting me sound ideas off of them and sharing their own. Thanks also to Pat Kreidl for our research discussions, as well as for his culinary suggestions. Andrew Kim remains the only member of SSG I've managed

to convert to karate; perhaps the resulting bruises can be considered karmic repayment for his making me a sysadmin not long beforehand. And of course Erik Sudderth, frequent collaborator and friend, whom I myself dragged into the sysadmin role but has yet to revenge himself. Meanwhile, I'm counting on Dewey Tucker, Ayres Fan, and Walter Sun to cut me in on some dizzyingly lucrative financial investment scheme. Thanks also to Martin Wainwright for sharing his directness and world–view, along with the proper way to perform an elbow–drop. And of course, Dmitry Malioutov for sharing his life–lessons involving fast cars and arm–wrestling, and Jason Johnson, our reigning arm–wrestling champ. In CSAIL, I would also like to thank Ali Rahimi, Bryan Russell, Mike Siracusa, and Kinh Tieu for innumerable discussions, both on research and otherwise.

Of course, even surrounded by all the bright fellows mentioned already, I would still have been lost without the support of everyone behind the scenes. Petr Swedock, who takes over the network administration and without whom I might never be able to depart in good conscience. Many thanks to Praneetha Mukhatira, and before her Laura Clarage and Taylore Kelly, without whose assistance the bureaucratic machinery of MIT would long since have chewed me up and spit me out.

On a more personal level, I would like to thank my family for all their understanding and support. My parents, Garret and Karin, who taught me how to do what I love, how to succeed, and most importantly how to be myself. My sister Elisabeth, who told me she would love me if I became a New Zealand sheep farmer (and presumably also if I did not). Liisa, who helps our family take time to enjoy ourselves. Erich and Dee, who have looked out for me in Boston and given me a home away from home. My cat Widget, who has helped decorate both me and my belongings with his affection and fur. And of course Michelle, who will always have my heart, for everything.

This work was supported in part by MIT Lincoln Laboratory under Lincoln Program 2209-3023, in part by ODDR&E MURI through ARO grant DAAD19-00-0466, and in part by AFOSR grant F49620-00-0362.

Contents

Chapter 1

Introduction

WIRELESS sensor networks are becoming increasingly attractive for a wide variety of applications, from tracking and surveillance to environmental monitoring. They require significantly less physical infrastructure than their wired counterparts and can be deployed at substantially lower cost. In theory, wireless networking can be used to create areas with ubiquitous sensing, for example to perform habitat or environmental monitoring [63, 66], create "smart" or interactive rooms and buildings [53], and provide surveillance of security–sensitive locations or regions of conflict [83].

However, wireless sensor networks also come fraught with a number of difficult issues, many due to the inherent energy and bandwidth limitations of a battery-powered wireless communications medium. In essence, ubiquitous sensing has the potential to provide overwhelming and undesirable volumes of raw data, making the challenge one of how to extract the relatively small amount of useful information from the network. The process of extracting useful information, without communicating an unnecessarily large volume of irrelevant data, often involves processing of the data locally at the sensors within the network.

■ 1.1 General Tools

The role of distributed processing of information for inference and estimation is one of the central themes of this thesis. We analyze this issue for a subset of problems in which we can bring to bear two basic modeling tools. The first is the popular formalism of graphical models [60, 79]; we use graphical models to describe the statistical dependency structure among the random variables of interest in our applications. Second, we use nonparametric, sample–based estimates of uncertainty [3] to capture and represent the complex distributions which can arise in these problems.

Graphical models and belief propagation (BP) have already generated some excitement for their applicability to distributed inference problems in sensor networks [10, 16, 77]. Regardless of whether they aim to perform exact or merely approximate inference, these methods begin by interpreting the structure of the problem's underlying probability distribution as a graph. This enables the direct application of methods such as belief propagation, in which the inference process can be described as a sequence of message–passing operations between parts of the graph. By assigning the responsibil-

ity for the computations involved to various sensors within the network, one readily obtains a simple, distributed algorithm for performing inference. Implementations of these ideas have already been considered for discrete–state [16] and jointly Gaussian models [77].

However, many real–world problems involve high–dimensional random variables with complex uncertainty, for which neither Gaussian nor discrete–valued approximations may be suitable. In these cases inference using sample–based estimates of uncertainty has become quite popular; particle filtering methods are widely used for state estimation in nonlinear, non-Gaussian systems [3]. Particle filters have also been applied in sensor networks to track the position of one or more objects (vehicles, people, etc.) as they move within a given region [114]. However, the application of particle filters is limited to simple, sequential estimation problems, corresponding to a relatively small class of graphical models (those which have the basic structure of a Markov chain).

■ 1.2 Problems Addressed

This thesis is framed in terms of a single primary focus problem, the usage of nonparametric, sample–based representations for inference in distributed sensor networks. In particular, we consider four specific sub-problems which comprise aspects of the larger whole.

- Using sample–based representations in general graphical models
- Understanding the implications of approximations to the messages passed in belief propagation
- Minimizing the cost of communications for sample–based representations of uncertainty, or approximations to the same
- Applying the aforementioned elements to solve a specific application problem (sensor self-localization)

Each chapter is devoted to one of these sub-problems, and is developed with an eye toward the focus problem of inference in sensor networks. However, each chapter also has implications which are much broader in scope, and after presenting the general themes and layout of the thesis we revisit each aspect and describe some of the ways in which they may be applicable to a wider class of problems and applications.

■ 1.3 Thesis Organization

This thesis considers how both graphical models and sample–based representations of uncertainty can be applied to solve difficult, distributed estimation problems. As mentioned, sample–based representations have been applied to some inference problems in sensor networks; however, many sensor network problems are best described using

more general graphical models, in which inference has been limited to Gaussian and discrete representations of uncertainty.

We extend these methods of inference by developing a nonparametric, sample–based inference method which is applicable to general graphical models, as opposed to the relatively simple graphical models to which particle filtering can be applied. In sensor networks, distributed inference is performed by passing messages between sensors; in order to consider the inherent cost in communicating these messages, we examine two important issues—the effects of approximating these messages on the inference algorithm, and the minimal size, in bits, of a representation of sample–based estimates of uncertainty. Finally, we apply our results in these areas to an example application in sensor networks. Thus, we can divide the body of the thesis into a background chapter and four distinct but closely related problems, whose focus steadily narrows on sample–based inference for sensor network applications. The chapters are organized as follows.

Background. The background sections in Chapter 2 provide a host of relevant material required by the rest of the thesis. Although our presentation of this material is by necessity brief, it provides the tools which are required to understand the algorithms and analysis presented in subsequent chapters, as well as references for the interested reader. We begin with an overview of the applications and issues inherent in the use of wireless sensor networks. We next describe some results from information theory, the study of which is central to such relevant tasks as data compression and communications in sensor networks. In this thesis, we are primarily concerned with estimation using sample–based, nonparametric representations such as kernel density estimates, which we introduce in Section 2.3. We also focus specifically on probabilistic descriptions and inference algorithms defined on graphical models, including belief propagation (BP) and particle filtering (Sections 2.5–2.7).

Nonparametric Belief Propagation. Chapter 3 presents the nonparametric belief propagation (NBP) algorithm, developed in collaboration with Erik Sudderth [46, 93]. NBP can be regarded in either of two ways—as a generalization of particle filtering which is able to be applied to a more general class of probability distributions defined on graphical models, or as a stochastic, sample-based approximation to the belief propagation algorithm. In essence, NBP works to combine several of the best qualities of both techniques. Like BP, NBP allows us to take advantage of known statistical independency structures beyond simple Markov chain structures, and like particle filtering, NBP provides a computationally efficient representation for complex non-Gaussian uncertainty about relatively high dimensional random variables.

The belief propagation algorithm has already proven to be useful for a number of sensor network applications, because it can be expressed as a potentially distributed message–passing algorithm [16]. NBP, too, is applicable to a wide variety of problems in sensor networks, several of which we consider at various points in this thesis. Among

them are such tasks as distributed tracking via particle filters (Section 5.7.2), estimation using non-Gaussian multi–scale models of spatially related phenomena (Section 5.7.3), and self–localization of sensor nodes (Chapter 6).

Message Approximations in Belief Propagation. Chapter 4 examines the idea of using approximate versions of the BP messages in more detail. There are several reasons why message approximations may be important in sensor networks. First, sensors have limited computational power; approximate messages such as those used in NBP or other forms of approximate inference [6, 13] can provide a means of reducing computational complexity. Approximation may become even more important when communications costs are considered. If the random variables of a graphical model are assigned to various nodes within a wireless sensor network, and belief propagation is performed over the graphical model, we require certain BP messages to be *communicated* from one sensor to the other, i.e., transmitted over the wireless channel. Approximations can be used to reduce the cost of these transmissions.

However, such approximations to the correct BP messages can cause errors in each subsequent stage of belief propagation, and in particular cause differences between the final results found via BP with and without message approximations. Chapter 4 considers the twin problems of how approximation error may be measured for BP messages, and how these approximations propagate to affect the estimates found via BP. As an interesting additional consequence, this analysis also helps to characterize the behavior and convergence properties of BP when no approximations are made.

Communication Cost of Particle Representations. If sample–based methods such as particle filtering and nonparametric belief propagation are to be applied to distributed inference in actual sensor networks, we also need to understand the *cost* of communicating the messages involved. In particular, given a collection of particles which represent a distribution, what is the cost of communicating that collection from one sensor to another? Chapter 5 considers the fundamental cost of communicating a sample–based estimate of a distribution, as well as several constructive methods for encoding the samples. *Lossy* approximations are of particular interest, since we may be able to obtain significant savings if we are willing to distort the form of the message slightly. Our analysis of message approximations from the previous chapter gives us the tools to understand the effects of lossy encoding of messages, and we describe an algorithm for finding and encoding representations which efficiently balance the cost of communications with any potential errors.

Sensor Network Self–Localization. Chapter 6 brings the analysis of all three previous chapters to bear on a single, canonical problem in sensor networks—that of self–localization. At the base of most sensor network applications is the fundamental assumption that each sensor has some idea of its own location in the world. However, the utility of many sensor networks depends on being able to obtain this information in an

automatic fashion, without direct intervention by a user or expensive additional equipment such as global positioning satellite (GPS) hardware. Often, there is information readily available about the *relative* locations of the sensors, for example using wireless signal strength or other measures to infer sensor distance, and this relative information can be combined in the network to provide location estimates for each sensor. We frame this problem as a statistical inference task defined on a graphical model, and apply nonparametric belief propagation to find a solution, obtaining both estimates of sensor locations and of their uncertainty. The specific inference task of localization provides a more complex problem on which we can demonstrate the utility of our analysis from the previous chapters.

■ 1.4 Contributions

Each of the problems described in the previous section, along with its chapter's analysis, has implications both for our focus problem (sample–based inference in sensor networks) and for a more general understanding of approximate inference methods. We list some of these contributions in the general order in which they appear in the thesis.

The nonparametric belief propagation method of Chapter 3 provides an algorithm which can be used to solve many problems defined on sensor networks, including the self–localization problem of Chapter 6. However, NBP is not restricted to sensor networks; it is a general–purpose method of performing approximate inference in graphical models. In addition to its application to sensor networks as covered in detail in the thesis, NBP has also been used to solve difficult problems in computer vision applications, for example estimating visual appearance models [93] and performing video-based tracking [88, 89, 94]. We describe the general structure and important concepts underlying NBP, and provide a detailed description of the algorithmic tools required to obtain an efficient implementation of NBP for inference.

Chapter 4 considers the problem of approximate belief propagation more abstractly. Perhaps the most important contribution of this chapter is to describe a novel framework in which belief propagation and many approximate versions of BP can be analyzed. In particular, this framework regards each iteration's messages as approximations to a fixed point of BP, with some quantifiable error; by analyzing the behavior of these errors, we may draw conclusions about the BP messages themselves.

We introduce one particularly convenient measure of error between BP messages, for which we are able to derive strong theoretical statements about the behavior of BP, including convergence conditions and some properties of the BP fixed points. We also obtain results which describe how BP behaves when messages or model parameters are approximated, as might arise in quantized versions of BP. Broadly speaking, this analysis is directly applicable to many uses of BP in sensor networks, in which quantization and other simplified representations are key to being able to communicate the messages in inference efficiently. Moreover, the implications of this chapter go well beyond sensor networks. BP is widely applied in such diverse fields as communications,

machine learning, computer vision, and signal processing; it is safe to say that a better understanding of the properties of BP, including its convergence and stability with respect to approximations, benefit many of these areas.

We also considered a second, less strict measure of error between messages. While we are unable to derive strong theoretical statements using this measure, we are able to use it to find useful approximations. Furthermore, it is instructive to see why the analysis becomes more difficult, and how at least some of these difficulties may be circumvented.

Our analysis of communications costs and approximations for particle–based representations (Chapter 5) has implications for many canonical sensor network applications, for example performing distributed target tracking using particle filtering [61, 114]. Again, perhaps the most important contribution of this chapter is to define the problem of communicating a sample–based density estimate. This opens the door to a number of new and interesting problems.

We characterize the optimal size of an exact representation of the density estimate under certain assumptions. As one consequence, we are able to show that this problem behaves quite differently from most common source coding problems. We describe some characteristics of "good" encoding methods and give a few constructive examples.

We also describe the problem of approximate representation of the density estimate, arguing that it is important to apply measures of loss which have some theoretical interpretation in terms of eventual inference error, for example those measures described in Chapter 4. The ability to balance the cost of communications with some measure of the resulting inference error represents a basic and extremely important element in creating efficient yet useful implementations for sensor networks. Again, the example approximation algorithm we describe by no means exhausts the possibilities, but instead serves to highlight an area of research which deserves additional attention.

Finally, Chapter 6 describes how these elements may be combined to provide a powerful set of tools for solving inference problems in sensor networks. By describing this canonical problem in terms of a graphical model, we are able to characterize a number of interesting properties of the problem, as well as gaining a sense of how "local" the problem really is. NBP provides a novel method of solving the ensuing optimization problem, and results not only in estimates of the sensor locations but also estimates of our remaining uncertainty. It can be easily distributed at a relatively low communications cost. In short, it not only illustrates how the preceding chapters' analysis can be applied to problems in sensor networks, but also appears to provide a powerful new solution to one of their fundamental tasks.

In summary, the work presented here forms a cohesive investigation of the central problem of performing inference tasks in distributed networks of sensors using nonparametric, particle–based representations of uncertainty. However, each part of the whole has its own implications for general inference and estimation problems, whether centralized or decentralized, and it is our hope that the results herein will prove useful for many more problems than those we have explicitly addressed (some of which we

outline explicitly in Chapter 7).

■ 1.5 Acknowledgements

Much of the research in this thesis has also been submitted or published in the form of conference and journal papers. Chapter 3, on the nonparametric belief propagation algorithm, consists of research done jointly with Erik Sudderth and is derived from the conference papers [46, 93]. The analysis of message errors and stability of Chapter 4 is also described in [44]. Chapter 5, on the lossless and lossy communications costs of particle–based representations of uncertainty, contains work derived from two papers [43, 45]. Finally, Chapter 6 describes our work on the sensor self–localization problem; this research was performed jointly with Prof. Randy Moses of The Ohio State University, and is also documented in the publications [40–42].

Chapter 2

Background

IN this chapter, we provide a brief overview and introduction to the prior work relevant to later parts of the thesis, and give specific references to works with more in-depth coverage. We first describe sensor networks generally, some of their current uses, and the typical issues involved. We then cover background in several basic areas: communications theory, nonparametric density estimation, efficient data structures (specifically KD-trees), and graphical models and inference algorithms. These sections provide the basic tools and notation used in the later parts of the thesis.

2.1 Sensor Networks

Sensor networks comprise a rapidly growing field of research with numerous applications for both military and civilian problems [29, 57]. Wireless ad-hoc sensor networks are appealing for a number of reasons. The idea of pervasive sensing is compelling—inexpensive sensors blanketing a region and reporting everything within. In such a scenario, there might be thousands of sensors, consisting of many different sensing modalities. In order to make such a scenario work, sensor networks must operate with relatively little infrastructure and almost no direct user intervention or calibration.

These features enable sensor networks to be deployed quickly and cheaply, and can be important in many types of environments, such as areas which are dangerous for people, for example monitoring regions of conflict [83], or are simply difficult to access, such as habitat or environmental monitoring applications [63, 66]. More mundane applications include problems in which it is simply too expensive to add or alter existing infrastructure, for example the retrofitting of old buildings [58]. Sensing technology has also progressed to the point where many useful sensors can be made extremely small, allowing information to be gathered unobtrusively. Examples of the technological progression and size of practical sensor networks are shown in Figure 2.1.

Each sensor in the network is typically equipped with certain devices and abilities. In particular, these elements usually include

Sensing: each sensor typically has some means of observing (and potentially interacting with) the environment. Some examples of relatively low-cost sensors include acoustic, seismic, or meteorological (temperature and pressure) measuring devices; higher-cost sensing units for visual or infrared imaging are also possible.

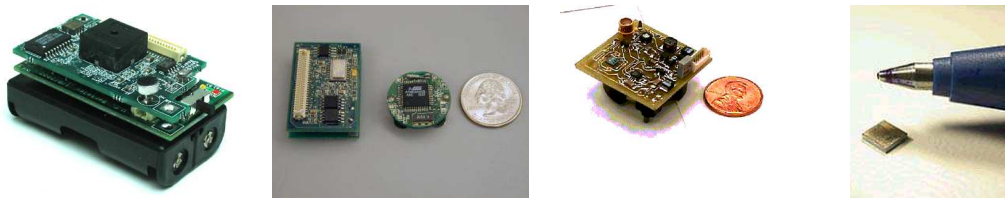


Figure 2.1. Various forms and development of the Berkeley "Mote" sensor; see e.g. [38]

Computation: each sensor has the means to perform some amount of local computation or data processing, from simple tasks such as data compression to complex distributed algorithms for calibration, event detection, and inference.

Communications: sensors are generally equipped with some form of wireless communications, enabling each sensor to exchange information with other sensors, typically those located nearby. This communications network allows data to be exported from the sensors, and can also be used to exchange relevant data locally with other sensors, enabling each sensor to benefit from the others' observations.

Power: each sensor is also equipped with a self-contained power supply of some kind. For the same reasons which make low-infrastructure sensor networks appealing, these power supplies are generally difficult to replace or recharge, and thus dictate the total lifetime of each sensor.

The primary concern of a wireless sensor network is almost always power consumption. In order to avoid a wired infrastructure, the sensors' power supplies must be self-contained. This power is slowly depleted by every action the sensor takes—observing the environment, local data processing activity, or communicating with nearby sensors. Unfortunately, battery technology has not progressed at the same rate as, for example, the technology underlying fast computation. In a typical sensor, the required battery size is many times that of the rest of the device [68], and is difficult to access for recharging or replacement. This makes power the driving factor behind sensor lifetime, and thus utility.

Limitations on available power means that problems such as inference and estimation in sensor networks must be carefully considered. In particular, communication typically takes many times the amount of energy required for computation or sensing [68]. This makes distributed algorithms and lossy forms of information transfer key issues for successful operation of a sensor network. In Chapters 4 and 5, we examine the effects of loss and approximation on inference algorithms, including for example distributed tracking, and in Chapter 6 describe a distributed algorithm for one of the most basic elements of constructing an ad-hoc sensor network, that of self-localization (automatic estimation of the position of each sensor in the network).

2.2 Information Theory

With their distributed nature and limited power supplies, sensor networks have compelling reasons to consider the communications requirements inherent in their tasks. Any study of communications, of course, must originate with information theory, which describes measures of uncertainty and information (the reduction of uncertainty); these quantities are directly related to the asymptotic performance of optimal communications. We stop short of describing constructive methods (algorithms) which approximate or achieve such performance; for a more in-depth coverage of information theory and data compression, see e.g. [15, 28].

2.2.1 Entropy

Entropy provides a quantification of randomness for a variable. Let x be a random variable taking on one of a discrete set of K values, with probability mass function $p(x)$. Shannon's measure of entropy (in bits) is given by

$$\mathrm{H}(p) = -\,\mathrm{E}[\log_2 p(x)] = -\sum_{i=1}^{K} p(i)\log_2 p(i) \tag{2.1}$$

Entropy quantifies the expected amount of information (and thus communication) required to describe the state of the random variable x, and for discrete-valued x is always strictly non-negative, and zero only when x is in fact deterministic rather than random. *Source coding*, or *data compression*, describes the process of finding an efficient representation of any particular realization of random variables. Sometimes the underlying distribution $p(x)$ is known, and can be used to design an optimal encoder; in other problems it must be estimated, or an encoder which is agnostic to the distribution applied.

In particular, (2.1) is achieved by assigning each value of x a codeword (string of bits) whose length is proportional to the negative log-probability of x. Examples of constructive methods which achieve optimal performance in this manner include the classic Huffman and arithmetic codes [28].

The *differential entropy* of a continuous random variable is a more subtle concept. Let x be a continuous-valued random variable with probability distribution function (pdf) $p(x)$ which is non-zero on some finite interval, say $[0, 2^T)$. Define x_d to be a discrete-valued version of x with probability mass function $p_d(x_d)$, where x_d has been discretized to bins of size $2^{-\beta}$. There are thus $2^{T+\beta}$ possible discrete values for x_d. The entropy of the discrete random variable x_d is given by $\mathrm{H}(p_d)$, and is a function of the discretization level β. Then, the differential entropy $\mathrm{H}(p)$ is defined by the limit of increasing resolution

$$\mathrm{H}(p) = \lim_{\beta\to\infty} \mathrm{H}(p_d) - \beta$$

and this can be shown to be equivalent to the natural generalization of (2.1),

$$= \int p(x) \log_2 p(x).$$

The differential entropy essentially measures the amount of randomness in a "very fine" discretization of the variable x, in a manner which can be decoupled from the actual discrete resolution β.

2.2.2 Mutual Information

Observing one random variable often tells us something about a related variable. The amount of randomness lost by observing one of two variables is a symmetric function, termed mutual information (MI). It can be expressed in terms of entropy as

$$\mathrm{I}(x;y) = \mathrm{H}(x) - \mathrm{H}(x|y) = \mathrm{H}(x) + \mathrm{H}(y) - \mathrm{H}(x,y) \tag{2.2}$$

Furthermore, a deterministic function of a random variable can only *lose* information; this is the *data processing inequality*:

$$\mathrm{I}(x; f(y)) \leq \mathrm{I}(x;y) \qquad \forall f(\cdot) \tag{2.3}$$

2.2.3 Relative Entropy

Relative entropy, also called the Kullback-Leibler (or simply KL) divergence, is one measure of the similarity between two distributions. It is defined as

$$\mathrm{D}(p\|q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx \tag{2.4}$$

(with the integral replaced by a summation if x is discrete). The KL-divergence has the nice property that it is zero if and only if the distributions p and q are equal. To be precise, we mean that for all events A, the probability of A is equal under both distributions: $\int_A p(x)\, dx = \int_A q(x)\, dx$. If x is a discrete-valued random variable, this means that $p(x) = q(x)$ for all values of x; if x is continuous it is possible for $p(x)$ and $q(x)$ to differ on a set of measure zero. However, for practical purposes we may ignore this subtlety.

2.3 Nonparametric Density Estimation

In many situations, we observe random processes for which we do not know, and would like to estimate, the distribution from which the observations have been drawn. If we *also* do not know the underlying form of the distribution *a priori*, nonparametric estimation methods are appealing, since they possess few underlying assumptions about the density which could potentially be incorrect. Although a nonparametric estimate

generally converges more slowly than an estimate making use of a correct parametric form, the strength of nonparametric techniques lies in the fact that they can be applied to a wide variety of problems without modification. One popular method of nonparametric density estimation, used extensively in this thesis, is the kernel density estimate.

■ 2.3.1 Kernel density estimates

Kernel density estimation, or Parzen window density estimation, is a technique of smoothing a set of observed samples into a reasonable continuous density estimate [76, 84, 90]. For interested readers, Silverman [90] provides a particularly detailed and useful introduction to the subject. Other useful references include [50, 86, 103].

In kernel density estimation, a function $K(\cdot)$, called the kernel, is used to smooth the effect of each data point onto a nearby region. For N *i.i.d.* samples $\{x_1 \dots x_N\}$, we have the density estimate

$$\hat{p}(x) = \frac{1}{N} \sum_i K_h \left(x - x_i\right) \tag{2.5}$$

where h denotes the kernel size, or bandwidth, and controls the smoothness of the resulting density estimate.

The kernel function $K_h(\cdot)$ is generally assumed positive, symmetric, and chosen to integrate to unity to yield a density estimate in (2.5). Perhaps the most common kernel function is the spherically symmetric Gaussian kernel

$$K_h(x) = \mathcal{N}(x; 0, hI) \propto \exp\left(-\|x\|^2/(2h)\right)$$

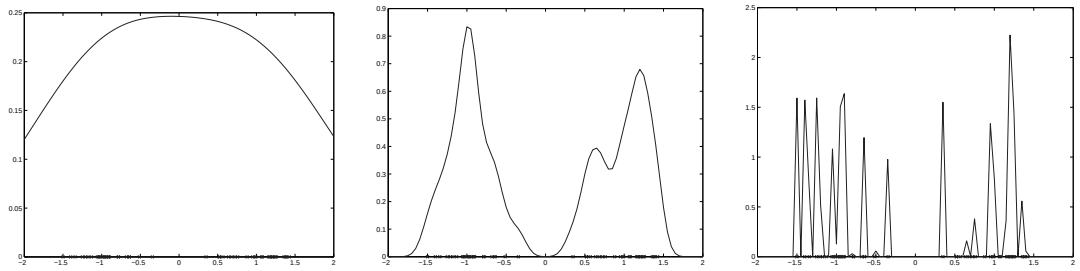
where $\mathcal{N}(x; \mu, \Sigma)$ denotes the Gaussian distribution with mean μ and covariance Σ

$$\mathcal{N}(x; \mu, \Sigma) = (2\pi)^{-d/2}|\Sigma|^{-1/2} \ \exp\left((x-\mu)^T\Sigma^{-1}(x-\mu)\right) \tag{2.6}$$

and d is the dimension of x and μ (in our notation, represented as column-vectors), and $|\Sigma|$ is the determinant of Σ. In this case, the bandwidth h controls the variance of the Gaussian kernel.

Many other kernel shapes are possible; however, empirically and theoretically, kernel shape appears to have very little effect on the quality of the resulting density estimate. Thus, in this thesis we will always use the Gaussian kernel, though as we discuss shortly, not necessarily a spherically symmetric one. The Gaussian kernel has a number of advantages in later parts of the thesis, which we mention as they arise. For a full discussion of the impact of various kernel choices, see [90].

A more crucial choice is the selection of the kernel size (or bandwidth) h. Let us begin with a simple one-dimensional problem, and consider the more general case subsequently. Figure 2.2 shows an example of the possible effects of over- and under-smoothing by poor choice of bandwidth parameter. When the bandwidth is too large, important features (such as the bimodality) are lost; however, if it is chosen to be too small, the exact values of the data begin to unduly affect the density estimate.



(a) (b) (c)

Figure 2.2. Kernel size choice affecting the density estimate: Large kernel sizes (a) produce over-smoothed densities, while small sizes (c) make densities which are too data-dependent. An appropriate middle ground is shown in (b).

In some cases the bandwidth can be chosen by hand, but it is often important to be able to select a reasonable value for the bandwidth automatically. There are a number of ways in which this can be done; we merely describe two which will be applied in later parts of the thesis. The first method, called the *rule of thumb*, is a simple, fast heuristic [86, 90]. Specifically, it assumes that the data samples are drawn from a Gaussian distribution, and computes the optimal bandwidth for the kernel density estimate as a function of the variance of the one-dimensional data by

$$h_{ROT} \approx 1.05\,\sigma^2\,N^{-2/5} \qquad \sigma^2 = \frac{1}{N}\sum(x_i - \mu)^2 \qquad \mu = \frac{1}{N}\sum x_i.$$

The conventionally accepted wisdom is that this technique has a tendency to over-smooth the distribution and thus prefers unimodal density estimates [90].

Another possibility is to choose the bandwidth in a maximum-likelihood framework. Naively, one could maximize the average log-likelihood of (2.5) at the same points $\{x_i\}$ used to construct the density estimate, giving

$$h^* = \arg\max_h \frac{1}{N}\sum_j \log\left(\frac{1}{N}\sum_i K_h\left(x_i - x_j\right)\right);$$

however, since the same points are used both for constructing the density estimate and for estimating the likelihood, the optimal value h^* will always be zero. A non-trivial solution is given by the leave-one-out maximization

$$h_{LCV} = \arg\max_h \frac{1}{N}\sum_j \log\left(\frac{1}{N-1}\sum_{i\neq j} K_h\left(x_i - x_j\right)\right). \tag{2.7}$$

We refer to this choice of bandwidth simply as *likelihood cross-validation* (LCV).

For higher dimensional distributions, kernel density estimation poses additional problems. For Gaussian kernels, the bandwidth may be defined by a general covariance

matrix, but with the larger number of parameters an optimization over likelihood can be very inefficient. Often, one restricts attention to diagonal-covariance bandwidths or even to multiples of the identity ("spherical" or isotropic Gaussian kernels). Another option is to select a bandwidth which is proportional to the standard deviation of the overall data, and optimize over the remaining (scalar) degree of freedom. The generalization of the "rule of thumb" method to higher-dimensional distributions employed in this thesis are given by [86]

$$h_{ROT} = C_d \left(\mathrm{Diag}(\Sigma)\right) N^{-2/(4+d)} \quad \Sigma = \frac{1}{N}\sum (x_i - \mu)(x_i - \mu)^T \quad \mu = \frac{1}{N}\sum x_i \tag{2.8}$$

where $\mathrm{Diag}(\Sigma)$ is the diagonal part of Σ, and thus h_{ROT} is a vector which captures only the variance in each dimension. This representation is selected for simple computational efficiency; if a more general covariance structure is desired it can be obtained by initial pre-processing (rotation) of the data [90]. The dimension–dependant constant C_d is well–approximated by $C_d \approx 1$ for all dimensions d and thus is often ignored. The generalization of the likelihood cross-validation criteria is similar; we select h_{LCV} by

$$h_{LCV} = \alpha_{LCV}\,\mathrm{Diag}(\Sigma) \quad \alpha_{LCV} = \arg\max_{\alpha} \frac{1}{N}\sum_j \frac{1}{N-1}\sum_{i\neq j} \mathcal{N}\left(x_i; x_j, \alpha\,\mathrm{Diag}(\Sigma)\right). \tag{2.9}$$

with Σ defined as in (2.8).

In subsequent chapters, we use h to indicate the vector–valued kernel bandwidth of a multi-dimensional Gaussian kernel. Specifically, this kernel is given by

$$K_h(x) = \mathcal{N}\left(x\,;\, 0, \mathrm{diag}(h)\right)$$

where $\mathrm{diag}(h)$ is the diagonal covariance matrix whose elements are specified by the vector h.

■ 2.3.2 Estimating information-theoretic quantities

Kernel density estimates provide one means of robustly estimating the quantities described in Section 2.2 for continuous random variables. Although other methods certainly exist (for an overview, see [4]) it is sufficient for our purposes to cover a few techniques of entropy estimation based on kernel methods.

One simple idea involves direct integration of $\hat{p}$, calculating the exact entropy of the estimated distribution:

$$\hat{\mathrm{H}} = -\int \hat{p}(x)\log\hat{p}(x)dx \tag{2.10}$$

However, this quickly becomes unwieldy as the number and dimension of the data grow. More feasible methods involve re-substituting the data samples back into the

kernel density estimate. This gives a stochastic approximation to the integral [1, 52]

$$\begin{aligned}\hat{\mathrm{H}} &= -\frac{1}{N}\sum_j \log \hat{p}(x_j) \\ &= -\frac{1}{N}\sum_j \log\left(\frac{1}{N}\sum_i K\left(\frac{x_j - x_i}{h}\right)\right)\end{aligned} \tag{2.11}$$

or, removing the evaluation datum from the density estimate gives a leave-one-out estimate [4]

$$\hat{\mathrm{H}} = -\frac{1}{N}\sum_j \log\left(\frac{1}{N-1}\sum_{i\neq j} K\left(\frac{x_j - x_i}{h}\right)\right) \tag{2.12}$$

Mutual information can then be estimated via (2.2):

$$\hat{\mathrm{I}}(x;y) = \hat{\mathrm{H}}(x) + \hat{\mathrm{H}}(y) - \hat{\mathrm{H}}(x,y) \tag{2.13}$$

and KL-divergence as:

$$\hat{\mathrm{D}}(p||q) = \hat{\mathrm{H}}(x) - \frac{1}{N}\sum_j \log \hat{q}(x_j) \tag{2.14}$$

where $\hat{q}(x)$ is another density estimate, for example a kernel density estimate constructed using a different set of samples.

■ 2.3.3 Implementation

The basic operations of kernel density estimation described in this section, along with many of the algorithms described subsequently in this thesis, have been made available as part of the Kernel Density Estimation (KDE) Toolbox for MATLAB [47]. The code and its documentation can be found at `http://ssg.mit.edu/~ihler/code/`.

■ 2.4 KD-Trees

K-dimensional trees (KD-trees) are data structures for representing and manipulating large sets of continuous-valued points. A KD-tree is a binary-tree structure which divides up a collection of points into a hierarchy of subsets, and caches statistics of each set which enable later computations to be performed more efficiently [5, 17, 71, 75].

Abstractly, a KD-tree stores two elements at each node of the tree. The first element is a statistic, or collection of several statistics, representing a potentially large set of k-dimensional points at each node (for example, their mean value). The second element of a node describes some method of subdividing the set of points represented by that node into two subsets which are then represented by nodes at the next level of the binary tree. Typically, this subdivision takes the form of a $k-1$ dimensional hyperplane which

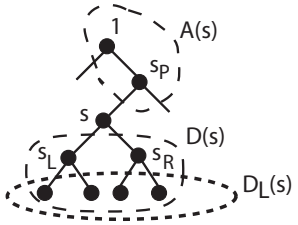


Figure 2.3. KD-tree notation: a node s has parent node s_P, left and right children s_L, s_R respectively, ancestors $A(s)$ (including the root node, 1), and leaf descendants $D_L(s)$.

splits the points into two disjoint collections of approximately equal size, one on each side of the hyperplane. Perhaps the simplest method of subdividing the data is to select between hyperplanes that are perpendicular to one of the k cardinal axes, choosing the axis in some cyclic manner. Eventually, the data are subdivided so many times that each finest-scale node stores a statistic computed from only a single point in the original collection.

■ 2.4.1 Notation

In order to describe KD-trees, the statistics which are cached, and a few of their many uses, we first require some notation to discuss the overall data structure. Figure 2.3 provides a visual depiction of our notation. We label the root node (at the top of the tree) by 1. For each node s in the tree except the root, we use s_P to indicate its parent node; assuming they exist, s_L and s_R indicate the left and right children respectively. The set $A(s)$ indicates the *ancestors* of node s—those nodes in the path between s and 1 (not including s itself). The *descendants* of s, $D(s)$, are the nodes which are separated from 1 by s, i.e., those below s in the tree. We will use the notation $D_L(s)$ to indicate the subset of the descendants $D(s)$ whose nodes are also *leaf* nodes, i.e., nodes that have no left or right children.

Each node of the KD-tree is associated with a set of points in k-dimensional space, with the complete set of points associated with the root node, and leaf nodes associated with individual points. Because each of the leaf nodes is associated with only a single point, we make no distinction between the points themselves and the leaf nodes of the KD-tree. Similarly, we can consider the set of points associated with an internal node s in the tree to be specified by its leaf descendants, $D_L(s)$.

It remains to be specified the two fundamental elements of the KD-tree. The first is to specify precisely how the tree is constructed, which is to say how we choose the $k-1$ dimensional hyperplane which subdivides the data $D_L(s)$ at each node s into $D_L(s_L)$ and $D_L(s_R)$, the collections summarized at the left- and right-hand children of s, respectively. Then, given the structure of the KD-tree, the second element is to specify exactly which statistics of the points, or equivalently leaf nodes $D_L(s)$, are to be stored at each internal node s.

■ 2.4.2 Construction Methods

There are many ways to construct KD-trees, and the method employed may impact the utility of the tree for subsequent computations; see, for example [71, 75]. However, it is not our purpose to investigate the relative merits of these methods in this thesis, nor do we assume that any particular method is used.

In our simulations and experiments for this thesis, we employ one of the simplest construction algorithms, from [75]. This procedure works via a top-down set-splitting procedure. Beginning with the root node $s = 1$, we compute the variance, along each cardinal axis, of the points in $D_L(s)$. Selecting the cardinal axis with largest variance to define our hyperplane, we split the collection of points into two parts at their median value, associating the smaller values with s_L and larger values with s_R. If $D_L(s)$ contains an odd number of points, we simply split according to some deterministic convention, such as placing the extra point in the left-hand set. We may then repeat this procedure by recursing on each subtree.

This procedure takes $\mathcal{O}(N \log_2 N)$ time, where N is the number of points stored in the KD-tree. However, it is generally very fast, taking much less time than subsequent operations on the KD-tree. The KD-tree can be thought of as creating a deterministic ordering from any given set of data; computational complexity of $\mathcal{O}(N \log_2 N)$ is typical for deterministic sorting algorithms [12].

■ 2.4.3 Cached Statistics

What statistics are useful to store in a KD-tree is an issue that is highly dependent on the precise application for which the KD-tree is intended. Since we will apply KD-tree structures to represent kernel density estimates, we select a particular set of statistics useful for that task. Specifically, we associate each leaf node of the KD-tree with a point μ_i, weight w_i, and bandwidth, represented by the vector h_i whose elements are the variance of the kernel in each dimension. Each internal node s of the KD-tree describes statistics of the density estimate resulting from the sum of kernels stored by its descendant leaf nodes $D_L(s)$, and these statistics can be used to enable fast computations.

Three potentially useful statistics to store at each node s are

$$\begin{aligned} \text{Weight} \quad w_s \qquad & w_s = w_{s_L} + w_{s_R} \\ \text{Mean} \quad \mu_s \qquad & w_s \mu_s = w_{s_L} \mu_{s_L} + w_{s_R} \mu_{s_R} \\ \text{Bandwidth} \quad h_s \qquad & w_s(h_s + \mu_s^2) = w_{s_L}(h_{s_L} + \mu_{s_L}^2) + w_{s_R}(h_{s_R} + \mu_{s_R}^2). \end{aligned}$$

where again, h_s is a vector whose elements indicate the variance along each of the k cardinal dimensions, and μ_s^2 indicates the element-wise product of μ_s with itself. Of course, it is also possible to store more general estimates of covariance, instead of the bandwidth vector we consider here. Given the structure of the KD-tree, these statistics can be computed efficiently in a bottom-up fashion, from leaves to root. The statistics themselves can be used to compute, at each node s, the Gaussian approximation to

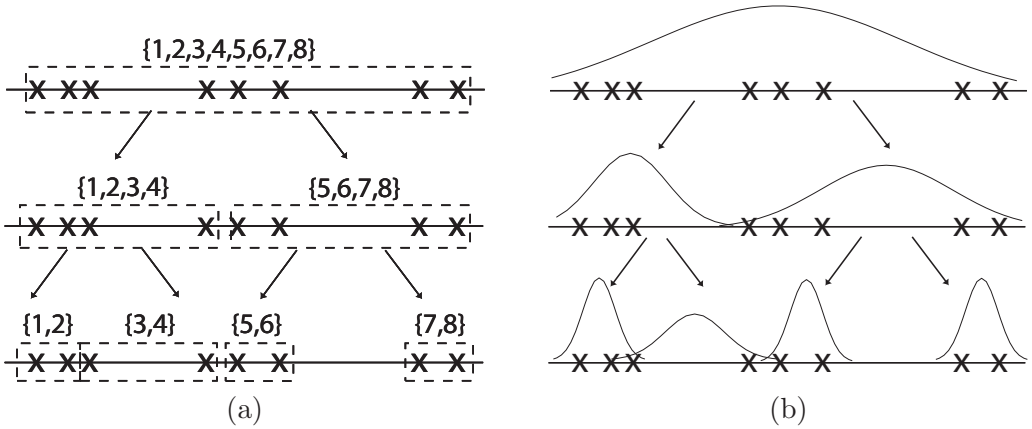


Figure 2.4. Two KD-tree representations of the same one-dimensional point set. (a) Each node maintains a bounding box. (b) Each node maintains mean and bandwidth or variance statistics. (Note: the finest scale, with the points only, is not shown.)

the kernel density estimate contained at the finest scale in $D_L(s)$ [see Figure 2.4(b)]. Another useful statistic to include is a *bounding box* which contains all the points μ_i in $D_L(s)$; the box can be specified by a center and a range (one-half the box width) in each cardinal direction. Again, given two bounding boxes for s_L and s_R, it is easy to find a box center and range for s which contain both child nodes' boxes [see Figure 2.4(a)].

■ 2.4.4 Efficient Computations

The cached statistics at each node s are used to speed up numerous standard algorithms and reduce the number of times the individual data need to be accessed. For example, nearest-neighbor algorithms can make use of the bounding box statistics in order to rule out large sets of samples without examining them directly; this leads to an $\mathcal{O}(\log_2 N)$ nearest-neighbor algorithm [5]. The Expectation-Maximization algorithm for fitting Gaussian mixture models can be sped up using the mean and variance statistics stored at each node [70].

Bounding box statistics can also be used to speed up approximate evaluation of kernel density estimates, using a *dual-tree* algorithm [32]. As it will become relevant in Chapter 3, we describe this procedure in more detail.

Suppose that we wish to compute the kernel density estimate

$$p(y_j) = \sum_i w_i K(x_i - y_j) \tag{2.15}$$

for two sets of N points $\{x_i\}$ and $\{y_j\}$ with $\sum_i w_i = 1$, up to some fractional error tolerance ϵ. In other words, we wish for the true $p(y_j)$ and our estimate $\hat{p}(y_j)$ to differ by at most $\epsilon \cdot p(y_j)$ for each point y_j. We begin by forming each collection of points $\{x_i\}$ and $\{y_j\}$ into KD-trees, and creating bounding box statistics for each internal

node of both trees. Then, beginning with the topmost node of each KD-tree, we follow a recursive divide-and-conquer strategy.

Given a node s in the tree containing the $\{x_i\}$ (the "source" tree) and another node t in the tree containing the $\{y_j\}$ (the "target" tree), we may consider a simple approximation scheme. Using the bounding boxes stored at s and t, we can easily compute the minimum and maximum distances D_{min} and D_{max} between any point in $D_L(s)$ and $D_L(t)$; these distances are depicted in Figure 2.5. Define the kernel function evaluated at these distances by $K_{min} = K(D_{max})$ and $K_{max} = K(D_{min})$. We may then evaluate the error caused by approximating every term in an entire block of the summation using a constant:

$$K(x_i - y_j) \approx C_{st} \qquad \forall i \in D_L(s),\ \forall j \in D_L(t).$$

Taking $C_{st} = (K_{max} + K_{min})/2$, we have that the error in this simple approximation is at most $\Delta_{st} = (K_{max} - K_{min})/2$.

If this approximation is deemed sufficiently accurate (a point we return to momentarily), we may simply approximate all interactions between $D_L(s)$ and $D_L(t)$ using this constant, i.e., since

$$\sum_{i \in D_L(s)} w_i K(x_i - y_j) \approx \left(\sum_{i \in D_L(s)} w_i \right) C_{st} \qquad \forall j \in D_L(t),$$

and the statistic $w_s = \sum_{i \in D_L(s)} w_i$ is also cached at node s, we can add $w_s C_{st}$ to our estimate $\hat{p}(y_j)$ for each $j \in D_L(t)$, and not consider the individual points x_i stored below node s.

If C_{st} is not sufficiently accurate, we may instead consider performing the same type of approximation for each of the four interactions defined by nodes s and t's children: s_L and t_L, s_L and t_R, s_R and t_L, and s_R and t_R. This entire procedure is then repeated in a recursive fashion. Since each child's bounding box is strictly smaller than its parent's box, recursing on the child nodes results in a more accurate approximation. For tolerance $\epsilon = 0$, we will eventually reach the leaf nodes and compute each term $K(x_i - y_j)$ exactly; for $\epsilon > 0$ we expect that the approximation at some earlier node should be sufficiently accurate, and we can avoid computing many of the terms in the sum.

The required level of accuracy $\epsilon\, p(y_j)$ for a given set of locations $\{y_j\}$ depends on the unknown quantity $p(y_j)$; however, it may be bounded in the following manner. At all times, we keep track of a lower bound $p^-(y_j)$ on $p(y_j)$, constructed simultaneously with our approximation $\hat{p}(y_j)$. Each time an approximation is deemed acceptable, in addition to adding $w_s\, C_{st}$ to our estimate $\hat{p}(y_j)$, we also add the value $w_s\, K_{min}$ to our lower bound $p^-(y_j)$. This procedure is guaranteed to provide a lower bound on $p(y_j)$ because $w_s K_{min}$ is itself a lower bound on the partial sum's true contribution. The lower bound $p^-(y_j)$ can can then be used to assess the required precision for any

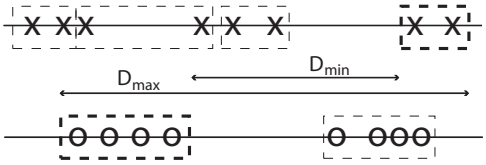


Figure 2.5. Two KD-tree representations may be combined to efficiently bound the maximum ($D_{\max}$) and minimum ($D_{\min}$) pairwise distances between subsets of the summarized points (bold) using the statistics stored in each tree.

DualTree(s, t)

1. Use the bounding boxes stored at s and t to compute
   (a) $K_{max} \geq \max_{i \in D_L(s), j \in D_L(t)} K_{\sigma_i}(x_i - y_j)$
   (b) $K_{min} \leq \min_{i \in D_L(s), j \in D_L(t)} K_{\sigma_i}(x_i - y_j)$
2. If $\Delta_{st} = \frac{1}{2}(K_{max} - K_{min}) \leq \epsilon \cdot p^-(y_j)$ for all $j \in D_L(t)$, approximate the sum over $D_L(s)$ for all of $D_L(t)$:
   (a) $C_{st} = \frac{1}{2}(K_{max} + K_{min})$
   (b) For all $j \in D_L(t)$, $\hat{p}(y_j) = \hat{p}(y_j) + w_s C_{st}$
   (c) For all $j \in D_L(t)$, $p^-(y_j) = p^-(y_j) + w_s K_{min}$
3. Otherwise, refine both trees by calling the procedure recursively on all four subtree pairs:
   (a) DualTree(s_L, t_L)
   (b) DualTree(s_L, t_R)
   (c) DualTree(s_R, t_L)
   (d) DualTree(s_R, t_R)

Figure 2.6. Recursive dual-tree algorithm for approximately evaluating a kernel density estimate $p(y)$ represented by a KD–tree. The values $p^-(y_j)$ denote running lower bounds on each $p(y_j)$, while the $\hat{p}(y_j)$ denote our current estimates. Initialize $p^-(y_j) = \hat{p}(y_j) = 0$ for all j.

subsequent approximation, by ensuring that every approximation satisfies

$$\Delta_{st} = \frac{1}{2}(K_{max} - K_{min}) \leq \epsilon\, p^-(y_j) \leq \epsilon\, p(y_j)$$

for each $j \in D_L(t)$. Then, the error due to using $w_s C_{st}$ is less than $w_s \epsilon\, p(y_j)$, and since $\sum_i w_i = 1$, the total error from all such approximations is less than our tolerance level $\epsilon p(y_j)$. The complete algorithm is described in Figure 2.6. Additional computational tricks can be used to avoid direct, sequential comparison to the lower bounds $p^-(y_j)$ for each $j \in D_L(t)$.

While published proofs of the computation time [32] appear to be flawed, experimentally this procedure appears to take only $\mathcal{O}(N \log N)$ or even $\mathcal{O}(N)$ time to evaluate

the (nominally) N^2 interactions to some fixed tolerance level ϵ. It is comparable in many ways to other low-rank approximation methods such as the fast Gauss transform [33, 35, 92].

■ 2.5 Graphs and Graphical Models

Graphical models provide a rich framework for describing structure in problems of inference and learning. The graph formalism specifies conditional independence relations between variables, allowing exact or approximate global inference using only local computations. This is essential in sensor network applications, where global consolidation and fusion of the sensors' observations may be intractable. We provide a brief introduction to undirected graphs and their uses in modeling the structure of probability distributions.

■ 2.5.1 Undirected Graphs

Graph theory has deep roots in mathematics, originating with Euler's solution to the Königsberg bridge problem in the mid-eighteenth century [36]. Though much of this prior work is not directly pertinent to the use of graphs for statistical modeling, we require a few basic definitions in order to discuss the concepts.

A graph G consists of a set of *vertices* (or *nodes*) $\mathcal{V} = \{v_s\}$ and *edges* $\mathcal{E} = \{(v_s, v_t)\}$ between them; undirected graphs have the property that $(v_s, v_t) \in \mathcal{E} \Rightarrow (v_t, v_s) \in \mathcal{E}$. We focus our discussion on undirected graphs. The vertices v_s and v_t are said to be *adjacent* if there is an edge connecting them, i.e., $(v_s, v_t) \in \mathcal{E}$, and the set of nodes adjacent to v_s are called its *neighbors*, and denoted by $\Gamma(s)$. The *degree* of v_s is the number of incident edges; if a graph has no self-connecting edges (v_s, v_s), which is always the case for the statistical graphs considered in this thesis, this equals the neighborhood size $|\Gamma(s)|$.

When every pair of nodes in a set $C \subseteq \mathcal{V}$ is connected by an edge, C is called fully-connected. Sets of nodes which are fully-connected are called *cliques*, and a clique is called *maximal* when no other node may be added such that the set remains a clique, i.e. , $\nexists C' \subseteq \mathcal{V} : C \subset C'$ and C' a clique.

It is also useful to discuss interconnections between more distant vertices. A *walk* is a series of nodes $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$, each of which is adjacent to the next. A *path* is a special kind of walk which has no repeated vertices ($m \neq n \Rightarrow v_{i_m} \neq v_{i_n}$); if there exists a path between every pair of nodes, G is called *connected*. A *cycle* is a walk which begins and ends with the same vertex ($v_{i_1} = v_{i_k}$) but has no other repeated vertices, thus forming a single loop.

Finally, a graph with no cycles is called a *tree*, or *tree-structured*. The concept of a tree is useful since for a connected tree-structured graph, the path between any two nodes is unique. In many problems, including inference over models defined on a graph, this structure can be used to derive particularly efficient or provably optimal solutions. A *chain* or *chain-structured* graph is a connected tree in which each node has at most two neighbors, and thus can be drawn in a linear fashion.

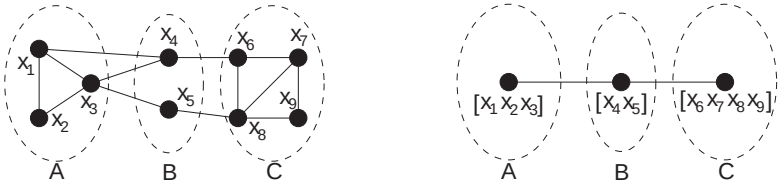


Figure 2.7. Graph separation and grouping variables: (a) shows the set B separating A from C, implying $p(x_A, x_C|x_B) = p(x_A|x_B)p(x_C|x_B)$. This relation is also visible in the graph created by grouping variables within the same sets (b), though some of the detailed structure has been lost.

■ 2.5.2 Undirected Graphical Models

A graphical model associates a random variable x_s with each vertex v_s. The structural properties of the graph describe the statistical relationships among the associated variables. Specifically, the graph encodes the Markov properties of the random variables through graph separation. For a more complete discussion of graphical models, see [60].

Let B be a set of vertices $\{v_s\}$, and define x_B to be the set of random variables associated with those vertices: $x_B = \{x_s : v_s \in B\}$. If every path connecting any two nodes v_t, v_u passes through the set B, B is said to *separate* the nodes v_t and v_u, and the probability density function of the variables x_t, x_u conditioned on the separating set x_B factors as:

$$p(x_t, x_u|x_B) = p(x_t|x_B)p(x_u|x_B) \tag{2.16}$$

This relation generalizes to sets, as well. Figure 2.7(a) shows the nodes of a graph partitioned into three sets, such that $p(x_A, x_C|x_B) = p(x_A|x_B)p(x_C|x_B)$. A particularly well-known instance of this is a temporal Markov chain, where the variables $\{x_i\}$ are ordered according to a discrete time index i, and the edge set $\mathcal{E} = \{(v_i, v_{i+1})\}$. This gives (2.16) the interpretation of decoupling the state at future and past times given its present value: $p(x_i, x_k|x_j) = p(x_i|x_j)p(x_k|x_j)$ for $i < j < k$.

For any set of random variables X, there may be many ways to describe their conditional independence with a graph structure. For example, if we define new random variables $\bar{X}$ by grouping elements of X, a graph which describes the independence relations of $\bar{X}$ also tells us something about the independence relations of X. Figure 2.7(b) shows an example of this, where variables from the graph in Figure 2.7(a) are grouped according to the sets A, B, C. Variables are sometimes grouped such that they obey the Markov properties of a graph with a particular kind of structure, for instance a chain or tree—a tree-structured graph created in this manner is known as a *junction tree* [60]. However, by grouping variables some of the structure present in the original graph is lost; e.g. from Figure 2.7(b) it is no longer obvious that $p(x_5|x_1 \ldots x_9) = p(x_5|x_3, x_8)$. Additionally, the difficulty of performing inference can be increased considerably by the resulting higher-dimensional variables associated with the new vertices.

The Hammersley-Clifford theorem [11] gives us a convenient way of relating the independence structure specified by a graph to the distribution of the random variables

x_s. It says that a distribution $p(x) > 0$ may be written as

$$p(x) = \frac{1}{Z} \prod_{\text{cliques } C} \psi_C(x_C) \tag{2.17}$$

for some choice of positive functions ψ_C, called the *clique potentials* (sometimes called *compatibility functions*), and Z a normalization constant.

When the density (2.17) can be written using only sets of size ≤ 2 (including, but not limited to tree-structured graphs), it becomes possible to associate the clique potentials with either a node ($|C| = 1$) or an edge ($|C| = 2$). In fact, any graph may be converted to one with only pairwise clique potentials by variable augmentation in a manner similar to creating a junction tree. In order to simplify our discussion of inference methods, we assume that the distributions in question may be expressed using only pairwise and single-node potentials. This permits us to denote the clique potential between x_s and x_t by $\psi_{st}(x_s, x_t)$, and the local potential at x_s by $\psi(x_s)$:

$$p(\mathbf{x}) = \prod_{(s,t)\in\mathcal{E}} \psi_{st}(x_s, x_t) \prod_s \psi_s(x_s); \tag{2.18}$$

Up to this point, we have not implied that any of the random variables are observed; however, the typical scenario is that the distribution of interest is actually the posterior distribution $p(\mathbf{x}|\mathbf{y})$, where $\mathbf{y}$ is a set of variables for which we have observed values.

Let us assume that our observed random variables $\mathbf{y}$ take the form of a collection of observations (a) y_s of individual node variables x_s, corrupted by uncertainty independent of $\mathbf{x}$ and other components of $\mathbf{y}$, and (b) y_{ts} of pairs of variables x_s and x_t, again corrupted by independent uncertainty, where s and t are connected by an edge. In this case, no additional statistical dependencies are introduced between the unobserved, or hidden, variables x_s by conditioning on the $\mathbf{y}$, and the conditional distribution $p(\mathbf{x}|\mathbf{y})$ has the same form (2.18), where the potentials are themselves functions of the observed variables, i.e.,

$$p(\mathbf{x}|\mathbf{y}) = \prod_{(s,t)\in\mathcal{E}} \psi_{st}(x_s, x_t, y_{st}) \prod_s \psi_s(x_s, y_s). \tag{2.19}$$

It is worth noting the special case that arises when there are no observations which couple two variables x_s and x_t and we may write $\mathbf{y} = \{y_s\}$, giving

$$p(\mathbf{x}|\mathbf{y}) = \prod_{(s,t)\in\mathcal{E}} \psi_{st}(x_s, x_t) \prod_s \psi_s(x_s, y_s). \tag{2.20}$$

In this situation we say that the observations y_s are *local* to their associated hidden variables x_s.

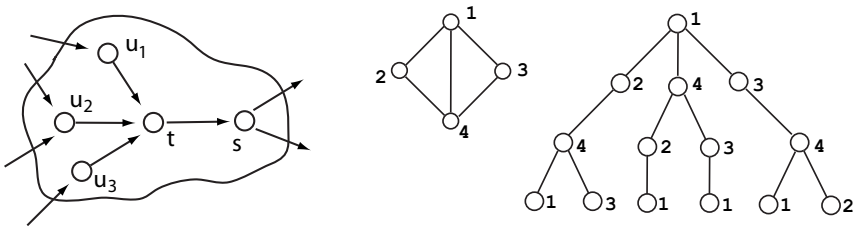


Figure 2.8. (a) BP propagates information from t and its neighbors u_i is to s by a simple message-passing procedure; this procedure is exact on a tree, but approximate in graphs with cycles. (b) For a graph with cycles, one may show an equivalence between n iterations of loopy BP and the depth-n computation tree (shown here for $n = 3$ and rooted at node 1; example from [95]).

■ 2.6 Belief Propagation

In this thesis, we are primarily concerned with a specific inference goal—the problem of computing the posterior marginal distributions

$$p(x_s|\mathbf{y}) = \int_{\mathbf{x}\setminus x_s} p(\mathbf{x}|\mathbf{y})\, d\mathbf{x}$$

for each x_s. These distributions can be used to calculate estimates of the x_s given all observations $\mathbf{y}$ which are optimal with respect to any of a number of criteria, as well as the uncertainty associated with such an estimate.

Exact inference on tree-structured graphs can be described succinctly by the equations of the belief propagation (BP) algorithm [79]. When specialized to particular problems, BP is equivalent to other algorithms for exact inference, for example Kalman filtering / RTS smoothing on Gaussian time-series and the forward-backward algorithm on discrete hidden Markov models.

The goal of belief propagation, also called the sum-product algorithm, is to compute the marginal distribution $p(x_t)$ at each node t. BP takes the form of a message-passing algorithm between nodes, the most common form of which is as a parallel update algorithm, where each node calculates outgoing messages to its neighbors simultaneously. Each iteration can be expressed in terms of an update to the outgoing message at iteration i from each node t to each neighbor s in terms of the previous iteration's incoming messages from t's neighbors Γ_t, not including s itself [see Figure 2.8(a)],

$$m_{ts}^i(x_s) \propto \int \psi_{ts}(x_t, x_s)\psi_t(x_t) \prod_{u\in\Gamma_t\setminus s} m_{ut}^{i-1}(x_t)dx_t. \tag{2.21}$$

In certain special cases involving continuous-valued variables, not every message is guaranteed to be finitely integrable. However, if a message m_{ts} is finitely integrable,

we follow the convention that it is normalized so as to integrate to unity. For discrete-valued random variables, of course, the integral is replaced by a summation. At any iteration, one may calculate the *belief* at node t by

$$M_t^i(x_t) \propto \psi_t(x_t) \prod_{u \in \Gamma_t} m_{ut}^i(x_t). \tag{2.22}$$

It is also useful to define the *partial* belief by the product of all incoming messages except that from a single neighbor,

$$M_{ts}^i(x_t) \propto \psi_t(x_t) \prod_{u \in \Gamma_t \setminus s} m_{ut}^i(x_t). \tag{2.23}$$

When possible, we also normalize the belief M_t and partial belief M_{ts} so as to integrate to one.

For tree-structured graphical models, belief propagation can be used to perform exact marginalization efficiently. Specifically, the iteration (2.21) converges in a finite number of iterations (at most the length of the longest path in the graph), after which the belief (2.22) equals the correct marginal $p(x_t)$. However, as observed by [79], one may also apply belief propagation to arbitrary graphical models by following the same *local* message passing rules at each node and ignoring the presence of cycles in the graph; this procedure is typically referred to as "loopy" BP.

For loopy BP, the sequence of messages defined by (2.21) is not guaranteed to converge to a fixed point after any number of iterations. Under relatively mild conditions, one may guarantee the existence of fixed points [112]. However, they may not be unique, nor do the fixed points correspond to the true marginal distributions $p(x_t)$). In practice however the procedure often arrives at a reasonable set of approximations to the correct marginal distributions [99, 106, 111].

■ 2.6.1 Implementations of BP

As described, the operations of BP consist of taking the point-wise product of collections of messages as in (2.22), and the convolution with a pairwise potential function as given in (2.21). Performing these operations analytically for general, continuous messages and potential functions is intractable. However, there exist special cases for which efficient, exact methods can be derived. These include discrete-valued random variables (in which each x_s takes on one of a finite set of values), and jointly Gaussian distributions.

In the case of discrete-valued random variables, the belief $M_t(x_t)$ takes the form of a vector of probabilities, corresponding to an estimate of the discrete probability mass function for x_t. Since x_s and x_t are both discrete-valued variables, the potential function ψ_{st} may be written as a matrix, and the convolution (2.21) as a matrix-vector product [79]. When the variables are all jointly Gaussian, it can be shown that the messages also have a quadratic form similar to a Gaussian distribution, and have a finite parameterization as a mean vector and covariance matrix [107].

■ 2.6.2 Computation Trees

It is sometimes convenient to think of loopy BP in terms of its *computation tree* [95, 104]. It can be shown that the effect of n iterations of loopy BP at any particular node s is equivalent to exact inference on a tree-structured "unrolling" of the graph from s. A small graph, and its associated 4-level computation tree rooted at node 1, are shown in Figure 2.8(b).

The computation tree with depth n consists of all length-n paths emanating from s in the original graph which do not immediately backtrack (though they may eventually repeat nodes).[1] We draw the computation tree as consisting of a number of *levels*, corresponding to each node in the tree's distance from the root, with the root node at level 0 and the leaf nodes at level n. Each level may contain multiple replicas of each node, and thus there are potentially many replicas of each node in the graph. The root node s has replicas of all neighbors Γ_s in the original graph as children, while all other nodes have replicas of all neighbors except their parent node as children.

Each edge in the computation tree corresponds to both an edge in the original graph *and* an iteration in the BP message-passing algorithm. Specifically, assume an equivalent initialization of both the loopy graph and computation tree—i.e., the initial messages m_{ut}^0 in the loopy graph are taken as inputs to the leaf nodes. Then, the upward messages from level n to level $n-1$ match the messages m_{ut}^1 in the first iteration of loopy BP, and more generally, a upward message m_{ut}^i on the computation tree which originates from a node u on level $n-i+1$ to its parent node t on level $n-i$ is identical to the message from node u to node t in the i^{th} iteration of loopy BP (out of n total iterations) on the original graph. Thus, the incoming messages to the root node (level 0) correspond to the messages in the n^{th} iteration of loopy BP.

■ 2.7 Particle Filtering

Let us now consider inference in the special case of a Markov chain with local observations $\{y_t\}$, so that each y_t is independent of all other variables when conditioned on the value of its associated hidden variable x_t. Since on a Markov chain the nodes $\{v_t\}$ may be sequentially ordered, we will label the neighbors of node v_t by v_{t-1} and v_{t+1}.

Particle filters [3, 19, 31, 49] provide a stochastic method of approximating the BP update equation (2.21) for the forward pass ($v_{t-1} \rightarrow v_t \rightarrow v_{t+1}$) on Markov chains involving general continuous distributions. The goal of particle filtering is thus to estimate the posterior marginal distributions $p(x_t|y_t, y_{t-1}, \ldots, y_1)$ for each t.

In particular, these distributions are not assumed to have any closed, parametric form. Because of this, uncertainty at each node v_t is represented nonparametrically by a collection of weighted particles, which serve as independent samples drawn from the distribution $p(x_t|\{y_s : s \leq t\})$ in a manner that we make precise soon. The basic idea behind particle filtering is to approximate the posterior distributions sequentially, by

[1] Thus in Figure 2.8(b), the computation tree includes the sequence $1-2-4-1$, but not the sequence $1-2-4-2$.

first creating a set of weighted samples which represent the distribution $p(x_1|y_1)$, then using these samples to construct a new set of weighted samples which represents the distribution $p(x_2|y_2, y_1)$, and iterating this procedure to estimate each of the desired posterior marginal distributions in turn.

■ 2.7.1 Particles and Importance Sampling

We begin by considering what it means to "represent" a distribution using a collection of weighted samples. Let $p(x)$ be some arbitrary distribution, and suppose that we are able to draw a set of N i.i.d. samples $\{x_p^i\}$ from $p(x)$. We may approximate the expectation of an arbitrary function $f(x)$ over the distribution p using the following Monte Carlo estimate,

$$\int f(x)p(x)\,dx \approx \frac{1}{N}\sum_i f(x_p^i). \tag{2.24}$$

We can thus use the samples $\{x_p^i\}$ to create an unbiased estimator of any number of useful statistics of the distribution $p(x)$, for example its mean, variance, or higher-order moments. In this sense, the samples $\{x_p^i\}$ can be thought of as representing the distribution $p(x)$.

However, it may be computationally difficult to obtain a collection of i.i.d. samples from $p(x)$ directly, for example when $p(x)$ is not specified in a convenient, closed parametric form. One way to address these situations is instead to create a collection of *weighted* samples which serve the same purpose, through a method called importance sampling [19, 65].

Suppose that, although it is computationally difficult to draw samples directly from $p(x)$, we are able to evaluate $p(x)$ easily up to a normalization constant. Let us define a *proposal distribution* $q(x)$, for which both sampling and evaluation are computationally feasible. We will assume that $q(x)$ is *absolutely continuous* with respect to $p(x)$, which means that if $q(\bar{x}) = 0$ for some $\bar{x}$, then $p(\bar{x}) = 0$ as well. Given a collection of N i.i.d. samples $\{x_q^i\}$ drawn from the proposal distribution $q(x)$, we can compute the relative likelihood of having drawn each sample x_q^i from $p(x)$ versus $q(x)$ and assign it as a relative weight for that sample,

$$w^i \propto \frac{p(x_q^i)}{q(x_q^i)},$$

and normalize the w^i so that $\sum_i w^i = 1$. Notice that, if $p(x) = q(x)$, the w^i are all equal to $\frac{1}{N}$. Also, note that to compute the $\{w^i\}$ we do *not* need to be able to evaluate either distribution $p(x)$ or $q(x)$ individually, but instead only need to evaluate their ratio $p(x)/q(x)$ up to some proportionality constant.

This weighted collection of samples $\{w^i, x_q^i\}$ may now be though of as representing $p(x)$ in a manner similar to before. Specifically, the expectation over $p(x)$ of an arbitrary function $f(x)$ may be estimated using

$$\int f(x)p(x)\,dx \approx \sum_i w^i f(x_q^i). \tag{2.25}$$

The assumption of absolute continuity ensures that for every state x with non-zero probability in $p(x)$, we also have a non-zero probability of drawing x using the proposal $q(x)$. The selection of a "good" proposal distribution is often application dependent, for example depending on the function $f(\cdot)$, and is an area of open research. In particle filtering, the typical goal is to obtain samples which represent $p(x)$ well enough to be reasonably effective for a large class of functions $f(\cdot)$; importance sampling in particle filtering is thus most effective when the discrepancies between $q(x)$ and $p(x)$ are small.

Instead of a collection of weighted samples $\{w^i, x_q^i\}$, it is sometimes desirable to create a collection of "unweighted", or all equal-weight, samples which also represent $p(x)$ in the manner described. Given our collection of weighted samples, we can generate a collection of representative equal-weight samples using a simple *resampling* procedure. Define $\bar{p}(i)$ to be the discrete distribution which assigns weight w^i to state i, with $i \in \{1 \dots N\}$. Then, we can create a new collection of samples $\{x_p^j\}$ as follows. For each $j \in \{1 \dots N\}$, we draw a label l from the discrete distribution, $l \sim \bar{p}(i)$, then assign $x_p^j = x_q^l$. The resulting collection of equally weighted particles $\{\frac{1}{N}, x_p^j\}$ also represents $p(x)$. Because there is the possibility that the new samples $\{x_p^j\}$ may repeat values, we say that this procedure draws samples from the $\{x_q^i\}$ by weight with replacement.

■ 2.7.2 Graph Potentials

We now return to the problem of particle filtering. It is first helpful to select a particular, concrete choice for the pairwise and single-node potential functions ψ. A convenient parameterization is given by the (forward) conditional distributions and likelihood functions

$$\begin{aligned} \psi(x_t, x_{t+1}) &= p(x_{t+1}|x_t) \quad \forall t \\ \psi(x_t, y_t) &\propto p(y_t|x_t) \qquad \forall t > 1 \end{aligned} \tag{2.26}$$

and $\psi(x_1, y_1) = p(x_1|y_1)$. The "forward" beliefs and BP messages then have the nice interpretation that

$$M_{t,t+1}(x_t) = p(x_t|y_t, y_{t-1}, \dots, y_1) \tag{2.27}$$

$$m_{t,t+1}(x_{t+1}) = p(x_{t+1}|y_t, y_{t-1}, \dots, y_1). \tag{2.28}$$

In particle filtering, we choose to represent each message using a collection of samples drawn from the distribution in (2.27) or (2.28) corresponding to that message. In particular, since the posterior distribution at each time t is not in general readily available in a convenient, closed form, we will use the collection of samples created to represent the distribution at $t-1$, along with the conditional distribution and likelihood function given in (2.26), to create a new collection of samples at time t via importance sampling.

■ 2.7.3 Likelihood-weighted Particle Filtering

The most basic form of particle filtering [3] begins by drawing samples from the prior distribution $x_1^j \sim p(x_1)$, for which we assume that direct sampling can be done efficiently. Then, to obtain a collection of weighted samples which represents the posterior

$p(x_1|y_1)$, we simply weight each sample by

$$w_1^j \propto \frac{p(x_1^j|y_1)}{p(x_1^j)} \propto p(y_1|x_1^j),$$

the likelihood of the observation y_1 given the state x_1^j. Using this collection of particles, we can obtain samples which represent the distribution $p(x_2|y_1)$ by propagating each particle x_1^j through the transition probability distribution $p(x_2|x_1)$, i.e., sampling from the transition probability distribution conditioned on x_1 taking on the value x_1^j,

$$x_2^j \sim p(x_2|x_1^j).$$

Then, the collection of particles $\{w_1^j, x_2^j\}$ represents the distribution $p(x_2|y_1)$. By iterating this procedure, we can obtain particle representations of each message and forward belief in the Markov chain.

Let us put this recursive iteration in terms of the messages $m_{t,t+1}$ and beliefs $M_{t,t+1}$. Given a collection of weighted samples $\{w_t^j, x_t^j\}$ from the forward belief $M_{t,t+1}$ at time t, we create a collection of weighted samples $\{w_t^j, x_{t+1}^j\}$ which represent $m_{t,t+1}$ by sampling from the conditional

$$x_{t+1}^j \sim p(x_{t+1}|x_t^j) \tag{2.29}$$

for each particle j. The new information due to the observation y_{t+1} is then incorporated by weighting the samples by their likelihood,

$$w_{t+1}^j \propto w_t^j \; p(y_{t+1}|x_{t+1}^j). \tag{2.30}$$

At every step, the weights are normalized so that $\sum_j w_t^j = 1$.

■ 2.7.4 Sample Depletion

The procedure outlined in Section 2.7.3 does not always work as well as might be desired. One common issue that can arise is *sample depletion*. The samples are said to be depleted when one, or a few, of the weights w_t^j are much larger than the rest. This means that any sample-based estimate such as (2.25) will be unduly dominated by the influence of a few of the particles.

One way to avoid sample depletion is to perform the resampling procedure described in Section 2.7.1 occasionally, since after resampling each of the N new particles has equal weight. Resampling itself does not result in a more diverse collection of particles, since it can only draw values which were in the original collection of samples; resampling on a depleted sample set is likely to result in many copies of identically-valued particles. However, when multiple copies of the same particle at node t are propagated through the forward dynamics (2.29), they will in general result in different values, and thus add sample diversity in the collection at node $t+1$.

Sometimes, when the forward dynamics are close to being deterministic, we may wish to add *artificial* diversity into the collection of samples. One way of doing so

is to simply add a small amount of random noise to each of the samples after each resampling operation. This is known as *regularized* particle filtering. Although this type of regularization has the undesirable effect that estimates of the form (2.25) are no longer unbiased, it has been known to improve performance in many applications.

■ 2.7.5 Links to Kernel Density Estimation

Kernel density estimates also use samples to represent and estimate arbitrary distributions; see Section 2.3.1. The operations of particle filtering can also be though of in terms of kernel density estimates. Traditional particle filtering corresponds to selecting a "density estimate" $\hat{p}(x_t|y_t, \ldots, y_1)$ defined by

$$\hat{p}(x_t|y_t, \ldots, y_1) = \sum_j w_t^j \delta(x_t - x_t^j),$$

where δ is the Dirac delta-function. The procedure of resampling with replacement described previously is then equivalent to drawing N samples independently from the density estimate $\hat{p}$. Regularized particle filtering, in which a small amount of noise is added during this procedure, can be thought of as instead drawing N samples independently from the (generalized) kernel density estimate

$$\hat{p}(x_t|y_t, \ldots, y_1) = \sum_j w_t^j K(x_t - x_t^j),$$

where the kernel $K(\cdot)$ is defined by the distribution of the noise added to each particle after resampling.

Chapter 3

Nonparametric Belief Propagation

MUCH of this thesis is concerned with the problem of computing exact or approximate posterior marginal distributions for a set of random variables, a problem addressed by the belief propagation (BP) algorithm as described in Section 2.6. When the variables under consideration are either jointly Gaussian or when each takes on a finite (and relatively small) number of discrete values, the BP operations (2.21)–(2.22) can be solved efficiently. However, in many problems the objects of interest are most naturally expressed as continuous–valued variables and possess non-linear relationships and non-Gaussian uncertainty. In these cases, Gaussian approximations may be unacceptable, and discretization of the state space may result in undesirable artifacts if the discretization is too coarse, or computational difficulty due to the large state space otherwise.

In filtering problems defined on Markov chains, the particle filtering methods described in Section 2.7 make use of sample-based representations to construct Monte Carlo approximations to the required integrals efficiently [3, 19, 31, 49]. In this chapter, we describe how the sample-based representations used in particle filtering may be extended to approximate the operations of belief propagation. The resulting *nonparametric belief propagation* (NBP) algorithm retains both the ability of particle filtering to capture arbitrary continuous distributions, and the applicability of belief propagation to inference problems on general graphical model structures.

We begin by justifying our choice of Gaussian mixture distributions, or more specifically Gaussian–kernel density estimates, to represent the sample–based message approximations used in NBP. To simplify the presentation, we initially assume that all quantities in the graphical model are represented by Gaussian mixtures, and discuss in Sections 3.3–3.4 how the operations required by NBP may be performed in this case. We then relax this assumption in Section 3.5. We mention one useful modification to the NBP algorithm in Section 3.6, and provide a brief recap and summary of NBP in Section 3.7. An important sub-problem in NBP is the efficient drawing of samples from products of several Gaussian mixture distributions; we consider this problem in

This chapter is based on the conference papers [46, 93], and represents work performed in collaboration with Erik Sudderth.

some depth in Section 3.8. Finally, we give a few experimental examples of the NBP algorithm in Section 3.9.

■ 3.1 Message Normalization

A sample-based representation is typically only meaningful when the BP message m_{ts} is finitely integrable. In standard particle filtering, the graphical model is parameterized in such a way that the (forward) messages at each time step are the posterior distributions. However, in general BP has no such guarantees; in fact, the BP messages are more typically likelihood functions $m_{ts}(x_s) \propto p(\mathbf{y}_{ts}|x_s)$ [101]. This is due to the fact that the inclusion of the prior $p(x_s)$ in more than one of the incoming messages complicates the fusion step, making it no longer a simple product operation. Thus, the BP messages are not necessarily finitely integrable, for example when x_s and $\mathbf{y}_{ts}$ are independent and x_s is not confined to a finite range. To guarantee that every message is, in fact, finitely integrable, and thus can be normalized to integrate to unity (*normalizable*), we assume for the moment that the potentials of our graphical model satisfy

$$\begin{aligned} \int \psi_{st}(x_s, x_t = \bar{x})\, dx_s < \infty \qquad & \forall (s,t) \in \mathcal{E} \\ \int \psi_s(x_s, y_s = \bar{y})\, dx_s < \infty \qquad & \forall s \in \mathcal{V} \end{aligned} \tag{3.1}$$

for any values of $\bar{x}$ and $\bar{y}$. A simple induction argument then shows that all messages are normalizable. Intuitively, the conditions (3.1) require each potential to be "informative" about the neighboring variables, so that observing the value of one random variable constrains the likely locations of the other. In most applications, this assumption is easily satisfied by constraining all variables to a (possibly large) bounded range. We will also relax these conditions in Section 3.5.

■ 3.2 Sample-Based Messages

Recall the form of the BP message update equation

$$m_{ts}^{i+1}(x_s) \propto \int \psi_{ts}(x_t, x_s)\psi_t(x_t) \prod_{u \in \Gamma_t \setminus s} m_{ut}^i(x_t) dx_t \tag{3.2}$$

and belief (or estimated marginal)

$$M_t^i(x_t) \propto \psi_t(x_t) \prod_{u \in \Gamma_t} m_{ut}^i(x_t) \tag{3.3}$$

from Section 2.6. A direct application of the sample- (or particle-) based messages such as are used in standard particle filtering, i.e., messages of the form

$$\hat{m}_{ut}(x_t) = \sum_j w_{ut}^j \delta(x_t - x_{ut}^j) \tag{3.4}$$

where the $\{w_{ut}^j, x_{ut}^j\}$ are weighted samples drawn from the continuous BP message $m_{ut}(x_t)$, immediately encounters difficulty, as the product of the incoming messages may not be well-defined. Assuming the true continuous BP messages are smooth, any two incoming messages $\hat{m}_{ut}$, $\hat{m}_{st}$ to node t of the form (3.4) are virtually guaranteed (i.e., with probability one) to have *no* samples which are exactly the same, and thus their product will be everywhere zero.

To avoid this dilemma, we use the samples $\{w_{ut}^j, x_{ut}^j\}$ to define a *smooth* and *strictly positive* function $\hat{m}_{ut}(x_t)$. While we eventually consider more general message approximation forms, we begin by examining one straightforward and intuitive method of enforcing these conditions, namely to smooth the function (3.4) by convolving it with some strictly positive function $K(x)$. Our message approximations thus have the form of kernel density estimates (Section 2.3, [76, 90]); this representation is reminiscent of that used in regularized particle filtering [3]. We focus solely on the Gaussian kernel

$$K_h(x) = \mathcal{N}(x\,;\,0, \mathrm{diag}(h))$$

since it has a number of desirable properties. First, it has infinite support (i.e., is strictly positive), ensuring that the product of two Gaussian mixtures will always be normalizable. Secondly, it is self-reproducing: the product of two Gaussian distributions has a simple closed form, which is also a Gaussian distribution. As will be seen in Section 3.8, this fact contributes to the tractability of many of the necessary computations.[1]

Of course, there is an inherent degree of freedom in the selection of the smoothing parameter (or bandwidth) h. As a basic rule, one may always simply apply any of the automatic bandwidth estimation methods described in Section 2.3.1 to select h given a collection of samples. Alternatively, however, there are also a number of interesting, more sophisticated possibilities for selecting h by taking into account the potential function ψ_{st}, which we discuss in more detail in Section 3.4.3.

The BP update equation (3.2) can be decomposed into two operations, a message product operation

$$M_{ts}^i(x_t) \propto \psi_t(x_t) \prod_{u \in \Gamma_t \setminus s} m_{ut}^i(x_t) \tag{3.5}$$

and a convolution operation

$$m_{ts}^{i+1}(x_s) \propto \int \psi_{ts}(x_t, x_s) M_{ts}^i(x_t) dx_t. \tag{3.6}$$

We examine each of these operations in turn. We begin by assuming that *all* these functions, i.e., the messages m_{ut} and potential functions ψ_t and ψ_{ts}, are represented by Gaussian mixtures. This assumption serves to simplify the development of the NBP algorithm; we discuss the relaxation of this assumption in Section 3.5.

[1] Thrun et al [96] deal with the same problem of ill-defined message products by applying a different strictly positive density estimation technique, called a *density tree*. However, as we discuss in Section 3.8, our kernel-based estimate leads to a number of efficient algorithms for approximating the involved message products, only one of which (Section 3.8.1) is also applicable to density trees.

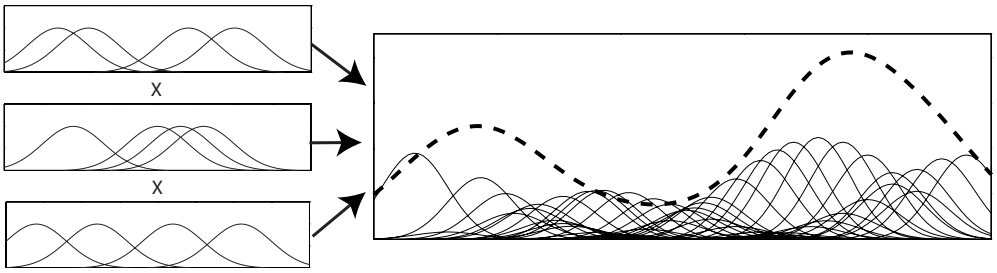


Figure 3.1. Although the product of $d = 3$ mixtures of $N = 4$ Gaussian components each (left) produces a Gaussian mixture of $N^d = 64$ components (right), the resulting distribution itself (dashed) rarely appears as complex as its sheer number of components would suggest, and may often be well-approximated using far fewer components.

■ 3.3 The message product operation

Using smooth and positive sample-based message approximations[2] of the form

$$m_{ut}(x_t) = \sum_j w_{ut}^j \, K_{h_i}(x_t - x_{ut}^j) \tag{3.7}$$

the message product is guaranteed to be well-defined and normalizable. Moreover, the fact that $K(\cdot)$ is chosen to be Gaussian means that a message product operation which multiplies d Gaussian mixtures, each containing N components, will produce another Gaussian mixture with N^d components. In principle, this means that the BP message update operation (3.2) could be performed exactly. In practice, however, the exponential growth of the number of mixture components forces approximations to be made.

In particular, we would like to approximate the N^d-component mixture using some smaller number of Gaussian components. Often, the functional shape of the message product is relatively simple, indicating that it should be possible to approximate the message product function accurately using only, say, N components. For an anecdotal example, see the product of messages depicted in Figure 3.1.

If the N^d-component mixture could be efficiently constructed and manipulated, we could apply any of a number of direct approximation methods to reduce the complexity of the Gaussian mixture [2, 30]. However, for most values of N and d, dealing explicitly with the full N^d-component mixture is generally computationally infeasible. Another possibility is to approximate the product successively, factor by factor, so that we compute the product of two messages $m_1(x) \cdot m_2(x)$, approximate the product using some simpler description, then multiply this approximation by $m_3(x)$, and so on [23]. However, this is often sensitive to the order in which the product is taken, and can thus lead to considerable error in the final product estimate.

[2] Although technically the Gaussian-sum based messages are some approximation $\hat{m}(x)$ of the true, continuous message function $m(x)$, we abuse notation slightly by making no distinction between the two, also using $m(x)$ to refer to the sample-based message estimates.

NBP does something considerably simpler. We assume that N is chosen to be sufficiently large so that a kernel density estimate made up of N independent samples from the product distribution can be used to represent it accurately. As we shall see in Section 3.8, this sampling operation may be considerably easier to perform than the direct exponential enumeration. Given a collection of weighted samples, we may simply select the bandwidth parameters H_{ts}^j, or parameter if all the kernel bandwidths are constrained to be equal for all j, using any automatic method such as those described in Section 2.3 to construct a Gaussian mixture approximation to the message product

$$M_{ts}(x_s) = \sum_j W_{ts}^j K_{H_{ts}^j}(x_s - X_{ts}^j),$$

and can construct a similar approximation to the belief $M_t(x_t)$. Here we have extended our use of capitalization to differentiate between messages and message products, using $\{w_{ts}^j, x_{ut}^j, h_{ts}^j$ to indicate the j^{th} weight, sample value, and bandwith in the representation of the message $m_{ut}(x_t)$ and the capitalized variables $\{W_{ts}^j, H_{ts}^j, X_{ts}^j\}$ to indicate the same quantities in the representation of the message product M_{ts}. As discussed in Section 2.3, for computational reasons it is common to select the bandwidth parameters H_{ts}^j or h_{ts}^j to be diagonal matrices.

■ 3.4 The convolution operation

The second operation required for belief propagation is the convolution

$$m_{ts}^{i+1}(x_s) \propto \int \psi_{ts}(x_t, x_s) M_{ts}^i(x_t) dx_t.$$

In order to approximate the convolution operation via sampling, we require the decomposition of the pairwise potential function ψ_{st} which separates its *marginal* influence on x_t from the conditional relationship it defines between x_s and x_t.

■ 3.4.1 The marginal influence function

In standard particle filtering, the graphical model may be parameterized in such a way that the pairwise potential ψ_{ts} is the conditional distribution $p(x_s|x_t)$, in which case samples $\{w_{ts}^j, x_{ts}^j\}$ which represent m_{ts} are easy to generate given a collection of samples $\{W_{ts}^j, X_{ts}^j\}$ from M_{ts} by drawing at random $x_{ts}^j \sim p(x_s|X_{ts}^j)$ and taking $w_{ts}^j = W_{ts}^j$ for each j. However, in more general graphical models, the pairwise potentials are often not expressed in such a conditional form, making the situation somewhat more complicated. As mentioned previously, we have assumed that the pairwise potential $\psi_{st}(x_s, x_t)$ is informative, in the sense that for any given value of x_t it is finitely integrable in the variable x_s. However, this does not mean that ψ_{st} has no influence on the likely states of the variable x_t. The conditional distribution is by definition agnostic about the possible states of the conditioned variable, i.e.,

$$\int p(x_s|x_t)\, dx_s = 1 \qquad \forall x_t.$$

A general potential function, however, may not have this property. To quantify this difference, we define the *marginal influence function* $\zeta(x_t)$ by

$$\zeta_{ts}(x_t) = \int \psi_{st}(x_s, x_t)\, dx_s \tag{3.8}$$

The NBP algorithm accounts for this marginal influence by incorporating $\zeta(x_t)$ into the product operation described in Section 3.3, i.e., drawing samples from the product

$$\zeta_{ts}(x_t) M_{ts}^i(x_t) \propto \zeta_{ts}(x_t)\psi_t(x_t) \prod_{u \in \Gamma_t \setminus s} m_{ut}^i(x_t) \tag{3.9}$$

rather than (3.5).

The marginal influence function $\zeta(x_t)$ may, of course, be difficult to compute for arbitrarily defined potential functions ψ_{st}; but there are many cases in which it is relatively simple. If, as we have initially assumed, ψ_{st} is specified via a Gaussian mixture, for example a kernel density estimate, the marginalization (3.8) is easily accomplished and simply acts to add another Gaussian mixture to the product operation. Another common case is that in which the pairwise potential ψ_{st} depends only on the difference between its arguments, so that (with a slight abuse of notation) we may write $\psi_{st}(x_s, x_t) = \psi_{st}(x_s - x_t)$. In this case it is easy to show that $\zeta(x_t)$ must be constant, since

$$\int_{-\infty}^{\infty} \psi_{st}(x_s - x_t)\, dx_s = \int_{-\infty}^{\infty} \psi_{st}(x)\, dx \qquad \text{where } x = x_s - x_t$$

and thus can be ignored.

■ 3.4.2 Conditional sampling

Given that the marginal influence of the pairwise potential ψ_{st} has been properly accounted for, the samples $\{W_{ts}^j, X_{ts}^j\}$ which represent the message product $\zeta_{ts} M_{ts}$ given by (3.9) can be used to provide a stochastic approximation to the integration (3.6). We construct particles to represent m_{ts} by sampling from the distributions defined by conditioning the pairwise potential ψ_{st} on each of the samples X_{ts}^j, i.e., for each $j \in \{1 \dots N\}$, drawing one sample x_{ts}^j according to

$$x_{ts}^j \sim f(x_s | X_{ts}^j) \propto \psi_{st}(x_s, X_{ts}^j). \tag{3.10}$$

Taking $w_{ts}^j = W_{ts}^j$ gives a collection of weighted samples $\{w_{ts}^j, x_{ts}^j\}$ which represents the message $m_{ts}(x_s)$. Just as described for particle filtering in Section 2.7.4, after a number of iterations of NBP the weights w_{ts}^j can become severely non-uniform, leading to undesirable dominance of one or a few samples in the message approximation. This sample depletion can be combatted in the same way used in particle filtering, by first resampling the collection $\{W_{ts}^j, X_{ts}^j\}$, i.e., selecting each sample X_{ts}^j with probability W_{ts}^j. This resampling procedure produces a collection of N equally probable samples from the product $\zeta_{ts} M_{ts}$ in (3.9), some of which may have identical values; we then rely on the diffusion effect of the conditional sampling operation (3.10) to provide sample diversity.

■ 3.4.3 Bandwidth selection

Given a collection of samples $\{w_{ts}^j, x_{ts}^j\}$ from the message m_{ts}, it remains to determine the bandwidth or bandwidths h_{ts}^j. As with the message product, we could simply elect to use any of a number of automatic bandwidth selection methods such as those described in Section 2.3. However, these methods can often be improved upon by using information about the form of ψ_{st}.

Let us begin with a simple example. Suppose that the pairwise potential $\psi_{st}(x_s, x_t)$ in (3.10) is Gaussian, so that

$$\psi(x_s, x_t) = \mathcal{N}\left(\begin{bmatrix} x_s \\ x_t \end{bmatrix}; \begin{bmatrix} \mu_s \\ \mu_t \end{bmatrix}, \begin{bmatrix} \Lambda_{ss} & \Lambda_{st} \\ \Lambda_{ts} & \Lambda_{tt} \end{bmatrix}\right)$$

Then, the marginal influence function $\zeta_{ts}(x_t)$ is Gaussian, $\zeta_{ts}(x_t) = \mathcal{N}\left(X_{ts}^j; \mu_t, \Lambda_{tt}\right)$, and is easily incorporated into the product of messages (3.9). If this product is represented as a Gaussian mixture,

$$\zeta_{ts}(x_s) M_{ts}(x_s) = \sum_i W_{ts}^j K_{H_{ts}^j}(x_s - X_{ts}^j),$$

the convolution 3.6 has a simple closed form, namely

$$m_{ts}(x_s) \propto \sum_j W_{ts}^j \mathcal{N}\left(x_s; \mu_{s|t}^j, \Lambda_{s|t} + H_{ts}^j\right) \tag{3.11}$$

where the conditional quantities are given by

$$\begin{aligned} \mu_{s|t}^j &= \mu_s + \Lambda_{st}\left(\Lambda_{tt}\right)^{-1}\left(X_{ts}^j - \mu_t\right) \\ \Lambda_{s|t} &= \Lambda_{ss} + \Lambda_{st}\left(\Lambda_{tt}\right)^{-1}\Lambda_{ts}. \end{aligned}$$

Notice that the form of (3.11) is precisely that of a kernel density estimate, with weight $w_{ts}^j = W_{ts}^j$, bandwidth $h_{ts}^j = \Lambda_{s|t} + H_{ts}^j$, and kernel centers $x_{ts}^j = \mu_{s|t}^j$. Applying this choice of parameters, we can avoid the issue of bandwidth selection for the message m_{ts} entirely.

Furthermore, when $\Lambda_{s|t}$ is much greater than H_{ts}^j, the bandwidth h_{ts}^j selected for the particles of m_{ts} can be approximated by $\Lambda_{s|t}$. Applying this approximation, we no longer require the bandwidth H_{ts}^j, either. We could thus avoid selecting the bandwidth for the message product M_{ts}, as well, avoiding both the complexity of deciding how to choose these bandwidths and computational overhead of computing them. Unfortunately, without determining the value of H_{ts}^j, we cannot determine whether this approximation is appropriate. There is thus some danger in blindly using $\Lambda_{s|t}$ as the smoothing parameter, without ever estimating H_{ts}^j from our samples. However, since most automatically selected bandwidths H_{ts}^j are decreasing functions of the number of particles N, the approximation typically works well when the value of N has been chosen to be sufficiently large.

If the potential $\psi_{st}(x_s, x_t)$ in (3.10) is described by a mixture of Gaussian components, i.e.,

$$\psi_{st}(x_s, x_t) = \sum_{k=1}^{K} V_{st}^k \mathcal{N}\left(\begin{bmatrix} x_s \\ x_t \end{bmatrix}; \begin{bmatrix} \mu_s^k \\ \mu_t^k \end{bmatrix}, \begin{bmatrix} \Lambda_{ss}^k & \Lambda_{st}^k \\ \Lambda_{ts}^k & \Lambda_{tt}^k \end{bmatrix} \right)$$

and the message product $\zeta_{ts}(x_s)M_{ts}(x_s)$ is again represented by a Gaussian mixture with N components, then the convolution 3.6 also has a relatively simple closed form as a Gaussian mixture [48]. Unfortunately, the number of components in the resulting message m_{ts} is then $K \cdot N$, rather than only N, as desired.

There are a number of ways we can go about reducing the number of components back to only N. Typically, many of these components have small relative weight; thus it is usually sufficient to preserve only a few of them. The difficulty, then, is which N components to select.

One alternative is to construct the KN mixture components explicitly, and then create a new collection of only N components by including each component with probability proportional to its weight. For example, if we select components *with* replacement, we have a procedure much like the resampling process in particle filtering, in which we allow multiple copies of the same component to be included if drawn more than once in the sampling process. If this replication is not desirable, we can draw samples *without* replacement—beginning with the collection of all KN components, each time a component is selected by our sampling process, we remove it from the collection and renormalize the weights of the remaining components, thus ensuring that each component is drawn only once. The components which are drawn are assigned their original weight in the new collection, and normalized so that their sum is equal to unity.

At times, we may not wish to compute and store all KN components explicitly. As a reasonable and efficient approximation, we could alternatively select only *one* of the K components to use for each incoming sample X_{ts}^j. For example, conditioned on the value of X_{ts}^j, we may compute the relative weights of each of the K mixture components, and sample from them randomly with probability proportional to their weight. This procedure typically works well when ψ is a kernel density estimate, and very few of the K components have non-negligible weight given X_{ts}^j.

Another possibility is to reduce the size of our collection of particles first. If we draw $\frac{N}{K}$ particles from the collection X_{ts}^j with probability proportional to W_{ts}^j, we may then exactly convolve the particles with the K–component Gaussian mixture, producing N components in the final kernel density estimate of m_{ts}. This works well when N is very large and K is much less than N, so that the smaller collection of $\frac{N}{K}$ particles remains a good approximation to the message product M_{ts}.

All of the methods described result in a collection of Gaussian mixture *components*, each of which has an associated analytically determined bandwidth. Thus we have again, in some sense, sidestepped the issue of bandwidth selection. However, we must be careful if we decide to apply this choice of bandwidth blindly, since it is analytically correct only when *all* KN components are included, and is not necessarily the best

choice when only N of the components are retained. Nevertheless, use of the analytically determined bandwidth often results in a relatively good message approximation, since most of the discarded components have very low weight, and thus have little impact in the overall density estimate.

In practice, each of the described methods has situations for which it works well. When the pairwise potentials are specified by kernel density estimates with $K \approx N$, such as in [93], the first or second methods, i.e., constructing all KN components explicitly or selecting one of the K components for each X_{ts}^j work quite well. For very small mixtures of Gaussians, Sigal et al. [88, 89] found it effective to use the third method, i.e., use all K mixtures but only a few of the N original samples. For many problems, of course, the potentials are expressed either analytically or as a single Gaussian (as in the tracking problem of Sudderth et al. [94] and the localization of Chapter 6), in which case the issue of subsampling does not arise.

■ 3.5 Analytic messages and potential functions

When not all potential functions and messages are Gaussian mixtures, we can apply a slightly modified version of the previously described procedure. Again, we describe the modifications in terms of the two operations, the product of incoming messages and the convolution with the pairwise potential ψ_{st}.

■ 3.5.1 The message product operation

The inclusion of messages which are not Gaussian mixtures to the message product operation can be performed using importance sampling [19]. Let us assume that the form of the messages m_{ut} are either Gaussian mixtures, or are of some analytic form which we are able to evaluate efficiently.

We first define the *mixture product* to be the product of those messages which are Gaussian mixtures. We may draw a collection of samples from this mixture product, using any of the methods we describe in Section 3.8. These samples are then weighted by evaluating the product of the remaining, analytically–specified messages at each sample location, and normalizing the resulting weights. This is quite similar to the way importance sampling is used in particle filters, described in Section 2.7.1. Using importance sampling to account for the effect of analytic messages typically works well so long as the analytic messages are smooth and vary relatively slowly over the set of sample locations.

■ 3.5.2 The convolution operation

When the pairwise potential functions ψ_{ts} are not Gaussian mixtures, we may still be able to perform the convolution operation in a manner similar to that described in Section 3.4. For a pairwise potential ψ_{ts} available in an analytic form, it may become more difficult to compute the marginal influence function ζ_{ts}, and to draw samples from the induced conditional distribution $f(x_s|X_{ts}^j) \propto \psi_{st}(x_s, X_{ts}^j)$. However, if these

two operations can be accomplished, we can construct the message m_{ts} in precisely the same way as before—construct samples $\{W_{ts}^j, X_{ts}^j\}$ which represent the product $\zeta_{ts} M_{ts}$, then stochastically propagate them through the pairwise relationship, giving

$$w_{ts}^j = W_{ts}^j \qquad\qquad x_{ts}^j \sim f(x_s | X_{ts}^j) \propto \psi_{st}(x_s, X_{ts}^j).$$

The message m_{ts} can then be represented as a kernel density estimate using the weighted samples $\{w_{ts}^j, x_{ts}^j\}$.

In doing so, we have assumed that $\psi_{st}(x_s, X_{ts}^j)$ is finitely integrable, as we described in Section 3.1. However, by employing a similar analysis to that given for Gaussian mixtures in Section 3.4.3, it turns out that we can represent the outgoing messages m_{ts} even for some potential functions which are *not* finitely integrable.

Take, for example, the potential function

$$\psi_{st}(x_s, x_t) \propto 1 - (2\pi)^{D/2} |\Lambda|^{1/2} \mathcal{N}(x_s; x_t, \Lambda) \tag{3.12}$$

where D is the dimension of x_s, $|\Lambda|$ is the determinant of Λ, and the variables x_s and x_t are not restricted to a bounded domain. This and other, similar potential functions arise naturally in the localization problem considered in Chapter 6. However, the potential (3.12) does not satisfy the normalization requirement (3.1), since ψ_{st} approaches unity as x_s and x_t grow far apart. Intuitively, information about x_t constrains x_s only in the sense that they are required to be far apart, which is not a very informative statement about x_s.

However, we may still use samples which indicate the uncertainty about x_t given its other incoming messages to construct a sample-based message from node t to node s. Since (3.12) is a function of the difference $x_s - x_t$, its marginal influence function ζ_{ts} is constant and can be neglected. Now, suppose that the message product M_{ts} is represented by the samples $\{W_{ts}^j, X_{ts}^j, H_{ts}^j\}$. Then, as when the potential function is Gaussian, the convolution can be computed in closed form, giving

$$m_{ts}(x_s) = 1 - (2\pi)^{D/2} |\Lambda|^{1/2} \sum_j W_{ts}^j \mathcal{N}\left(x_s; X_{ts}^j, \Lambda + H_{ts}^j\right) \tag{3.13}$$

Although this message is *not* a Gaussian mixture, m_{ts} is both smooth and strictly positive, and thus satisfies all the requirements to be included in the message product operation described in Section 3.3. Moreover, it is relatively easy to evaluate for any value of x_s, and thus can be included in the message product using importance sampling as described in Section 3.5.1.

When Λ is much greater than H_{ts}^j, we can *approximate* this message in a manner similar to that employed in Section 3.4.3, using the simpler form

$$m_{ts}(x_s) = 1 - (2\pi)^{D/2} |\Lambda|^{1/2} \sum_j W_{ts}^j \mathcal{N}\left(x_s; X_{ts}^j, \Lambda\right). \tag{3.14}$$

We may further generalize this approximation to cases in which the function is neither a Gaussian mixture nor does it have the form of a Gaussian mixture. For example, we could approximate a more generic potential function $\psi_{st}(x_s - x_t)$ by

$$m_{ts}(x_s) = \sum_j W_{ts}^j \, \psi_{st}(x_s - X_{ts}^j), \tag{3.15}$$

and assuming ψ_{st} is easy to evaluate, incorporate it into message products via importance sampling as described. This approximation assumes implicitly that the "spread" of ψ_{st}, or the amount of smoothness that it imparts, is large compared to the bandwidth H_{ts}^j, so that the convolution of ψ_{st} with the Gaussian centered at X_{ts}^j is approximately equal to $\psi_{st}(x_s - X_{ts}^j)$.

■ 3.6 Belief sampling

It has been suggested by a number of authors [48, 56] that it is possible to improve the performance of stochastic approximations to the BP messages[3] by "focusing" samples using the estimated marginal distributions; we examine some of the merits of this technique ourselves in experiments described in Section 6.7. In essence, the approximation follows from writing an equivalent form for the BP update equation,

$$\begin{aligned} m_{ts}^{i+1}(x_s) &\propto \int \psi_{ts}(x_t, x_s)\psi_t(x_t) \prod_{u \in \Gamma_t \setminus s} m_{ut}^i(x_t) dx_t \\ &\propto \int \psi_{ts}(x_t, x_s) \frac{M_t^i(x_t)}{m_{st}^i(x_t)} \, dx_t \end{aligned}$$

where M_t^i is the belief, defined by (3.3). If the messages are computed exactly, this manner of rewriting the definition does not change the messages themselves. However, if the messages are approximated using samples, it suggests a different procedure for drawing those samples from that which we described in Section 3.3. Specifically, we may instead draw samples from the belief $M_t^i(x_t)$, and then modify their relative weights so as to represent the product of messages $M_{ts}^i(x_t)$. For example, we may use a simple importance weighting procedure, weighting each sample X_t^j by $1/m_{st}^i(X_t^j)$. We refer to this procedure as *belief sampling*.

In terms of NBP, belief sampling has a number of clear advantages. First of all, it can have a computational advantage—it is often easier to draw N samples from $M_t(x_t)$, which is a product of d incoming messages, than to draw N samples from each of the d combinations of $d-1$ incoming messages. A naive, direct sampling procedure might suggest otherwise, requiring $\mathcal{O}(N^d)$ operations for the former, which is considerably more than the $\mathcal{O}(dN^{d-1})$ operations required by the latter for most practical values

[3]The method described by Isard [48] is similar to NBP and was developed contemporaneously; the method of Koller et al. [56] describes the concept of stochastic approximations in general but applies it specifically to inference over discrete–valued random variables.

of d and N. However, a deeper exploration of sampling procedures for products of Gaussian mixtures, which we consider in Section 3.8, reveals that the cost of drawing N samples is typically more like $\mathcal{O}(dN)$ or $\mathcal{O}(dN^2)$ for the former, and $\mathcal{O}(d^2N)$ or $\mathcal{O}(d^2N^2)$ for the latter. Even when the number of neighbors d is moderate (a two–dimensional nearest-neighbor grid has $d = 4$) the savings involved can be non-trivial.

Another advantage is an improvement in communications costs when NBP messages must be transmitted from one sensor, performing inference for node t, to others performing inference for each of the neighbors of t. We explore the cost of communicating the particle–based representations involved in Chapter 5. However, if the *same* samples are used for all outgoing messages from a node, it turns out that one may communicate *all* outgoing messages simultaneously by transmitting the estimated marginal $M_t^i(x_t)$. Each neighbor $u \in \Gamma_t$ then uses its own, previously computed belief $M_u^{i-1}(x_u)$, along with the pairwise potential ψ_{ut} shared with node t, to compute the incoming message $m_{tu}(x_u)$ from node t. This means that d communications, one to each of the neighbors of t, may be performed for the cost of only a single transmission. Again, this can be a considerable improvement, especially in wireless sensor networks, in which communications (energy and bandwidth) often comprise the most precious resource.

What is less clear is whether or not belief sampling is also statistically helpful. In Section 6.7 we explore this empirically, showing that when the number of particles N is small, the bias inherent in belief sampling can combine with random errors to degrade performance, and can produce worse message estimates than the corresponding N–sample messages computed without belief sampling. When sufficiently many particles are used, however, belief sampling improves the quality of the estimated messages by using early, rough estimates to refine the placement of particles in later messages. It is difficult to say definitively how well and at what values of N belief sampling works, or whether its advantages uniformly outweigh its possible drawbacks. Exploring the relative merits of belief sampling, and considering whether there exist better variants of the belief sampling procedure than the one we have described here, comprise two interesting areas of future research.

■ 3.7 Discussion

Let us recap briefly with a simple comparison between the methodology behind NBP and that common in particle filtering. NBP first requires that we use a class of sample–based approximations to the BP messages which give strictly positive and smooth functions, rather than representing the messages as collections of delta–functions as is commonly done in particle filtering. We do so by using a kernel density estimate for each message, similar to the messages used in "regularized" particle filtering; this serves to ensure that the product of multiple messages will never be everywhere zero and thus will always be normalizable.

The only other fundamental requirement of NBP is that we possess some means of generating particles which represent the product of incoming messages, possibly incor-

porating the marginal influence function ζ if non-constant, as given in (3.9). Just as in particle filtering, this is accomplished using importance sampling—we assume that there is some subset of messages, specifically those represented as Gaussian mixtures, from whose product we may generate samples with relative ease. We use this " mixture product" as a proposal distribution, drawing samples from the product of all the Gaussian mixtures, and then weighting by the influence of the remaining messages to obtain a collection of weighted samples from the full product.

Our description thus far has begged the question of how, precisely, we generate samples from the product of a collection of Gaussian mixtures. This is in itself a difficult task, and we devote the next section to exploring several possible methods for drawing these samples, both exactly and approximately.

As a final note, however, just as with particle filtering it is sometimes possible to apply additional domain knowledge to improve the quality of a proposal distribution. In other words, if anything is known about the messages and potentials which are *not* Gaussian mixtures, we may be able to improve our proposal distributions by incorporating this information explicitly. However, formulating such domain knowledge and successfully incorporating it into the proposal distribution is a highly application–dependent process, and thus we do not explore this possibility further here.

■ 3.8 Products of Gaussian Mixtures

In general, the most computationally difficult part of the NBP algorithm is the procedure for drawing samples from the product of several Gaussian mixtures. In this section, we describe several sampling procedures, including two multi-scale algorithms. We then provide a brief empirical comparison of the methods, suggesting which techniques may be most appropriate under various circumstances. For interested readers, MATLAB code to perform all the described sampling methods is available as part of the KDE Toolbox [47].

Specifically, let $\{p_1(x), \ldots, p_d(x)\}$ denote a set of d mixtures of N Gaussian densities, where

$$p_i(x) = \sum_{l_i \in \mathbb{L}_i} w_{l_i} \mathcal{N}(x\,;\, \mu_{l_i}, \Sigma_i) \tag{3.16}$$

Here, l_i indexes the set $\mathbb{L}_i$, which contains an (abstract) label for each of the N mixture components in the "input" Gaussian mixture $p_i(x)$. As usual, the weights w_{l_i} are normalized to sum to unity (for each mixture i). For notational simplicity, we assume that all mixtures are of equal size N, and that the covariances Σ_i are uniform within each mixture, although the algorithms which follow may be readily extended to problems where this is not the case. In practice with NBP, the covariances Σ_i are generally taken to be diagonal and specified by a bandwidth vector h_i, but this is not required for the algorithms in this section. Our goal is to draw samples from the N^d component mixture density $p(x) \propto \prod_{i=1}^{d} p_i(x)$ efficiently; we assume that N samples are to be drawn.

We may divide the sampling algorithms we describe into two broad categories:

fine-scale and *multi-scale*. We use the term *fine-scale* to describe methods which do not attempt to use agglomerative statistics in order to guide the sampling process, in contrast to the multi-scale methods of Section 3.8.2.

■ 3.8.1 Fine-scale methods

We begin by describing a procedure for direct, exact sampling from the product of Gaussian mixtures. Although this procedure is in general too costly computationally to be of practical use, it serves to establish the notation used in the subsequent sampling algorithms. We then consider generic approaches to importance sampling, and two methods based on Gibbs sampling.

For the fine-scale methods described in this section, we can simply label the N components of the i^{th} mixture as $\{1_i, \ldots N_i\}$, though it will be convenient to use a slightly different labeling convention based on the KD-tree structure for the multi-scale methods in Section 3.8.2. When it is unambiguous, we sometimes abuse notation by writing e.g. $l_i = 1$ to indicate $l_i = 1_i$. In other words, since the variable l_i always refers to labels in the i^{th} mixture, $l_i = 1$ indicates the first label in the i^{th} mixture.

Direct sampling

Sampling from the product density can be decomposed into two steps: first, randomly select one of the product density's N^d components, then draw a sample from the corresponding Gaussian. Let each product density component be labeled as $L = [l_1, \ldots, l_d]$, where l_i labels one of the N components of $p_i(x)$. For our discussion of sampling from products of Gaussian mixtures, we use the convention that lowercase letters such as l_i label components in an input mixture density, while capital letters $L = [l_1, \ldots, l_d]$ label the corresponding product mixture components, i.e., the set $\mathbb{L}_1 \times \ldots \times \mathbb{L}_d$. The relative weight of component L is given by

$$w_L = \frac{\prod_{i=1}^{d} w_{l_i} \mathcal{N}(x\,;\, \mu_{l_i}, \Sigma_i)}{\mathcal{N}(x\,;\, \mu_L, \Sigma_L)} \qquad \Sigma_L^{-1} = \sum_{i=1}^{d} \Sigma_i^{-1} \qquad \Sigma_L^{-1}\mu_L = \sum_{i=1}^{d} \Sigma_i^{-1}\mu_{l_i} \quad (3.17)$$

where μ_L, Σ_L are the mean and covariance of product component L. The weight w_L is a constant, and does not actually depend on the value of x; evaluating the formula at the mean of the denominator, $x = \mu_L$, may be numerically convenient. To form the product density, these weights are normalized by the *weight partition function* $Z = \sum_L w_L$.

Determining Z exactly takes $\mathcal{O}(N^d)$ time, and given this constant we can draw N samples from the distribution in $\mathcal{O}(N^d)$ time and $\mathcal{O}(N)$ storage. This is done by drawing and sorting N uniform random variables $\{u_j\}$ on the interval $[0, 1]$, and then computing the cumulative distribution $P(L)$ of $p(L) = w_L/Z$ to determine which, if any, samples are drawn from each label L. For each sample u_j which falls between $P(L-1)$ and $P(L)$, we draw a sample x_j from the Gaussian distribution determined by the parameters μ_L and Σ_L. This algorithmic procedure, also listed in Figure 3.2, results in N samples $\{x_j\}$ drawn from the product distribution.

DIRECT SAMPLING:

1. For each label $L = [l_1, \ldots, l_d], l_i \in \{1 \ldots N\}$, compute w_L according to (3.17) and sum: $Z = Z + w_L$.
2. Draw and sort N values R uniformly between $[0, 1)$.
3. Assign C=0; for each label $L = [l_1, \ldots, l_d]$,
   (a) Compute $c_L = \frac{w_L}{Z}$
   (b) For each element of R between C and $C + c_L$, draw a sample x_j from the product of Gaussians identified by the labels $l_1, \ldots, l_d$.
   (c) Accumulate: $C = C + c_L$

Figure 3.2. Direct sampling from products of Gaussian mixtures; requires $\mathcal{O}(N^d)$ time and $\mathcal{O}(N)$ storage.

Importance Sampling

Importance sampling, also described in more detail in Section 2.7.1, is a Monte Carlo method for approximately sampling from an intractable distribution $p(x)$, using a proposal distribution $q(x)$ for which sampling is feasible [19, 65]. Here we describe how importance sampling may be used to obtain samples representing the product of d Gaussian mixtures.

Assume that both $p(x)$ and $q(x)$ may be evaluated up to a normalization constant; to draw N samples from $p(x)$, an importance sampler draws $kN \geq N$ samples $x_j \sim q(x)$, and assigns a weight to the j^{th} sample given by $w_j \propto p(x_j)/q(x_j)$. The weights are then normalized by their sum, $Z = \sum_j w_j$, and N samples are drawn with replacement from the discrete distribution $\bar{p}(x_j) = w_j/Z$, meaning that the value x_j is selected with probability w_j/Z, with the possibility that the same value x_j will be drawn multiple times.

Although there are a limitless number of possible proposal distributions to choose from, for the products of Gaussian mixtures considered here we limit ourselves to the following two possibilities. The first, which we refer to as *mixture importance sampling*, draws each sample by randomly selecting one of the d input mixtures, and sampling from its N components. Alternatively, this procedure is equivalent to drawing a sample from the mixture average

$$q(x) = \frac{1}{d} \sum_i p_i(x).$$

The importance weight for each sample is then given by

$$w = \frac{\prod_i p_i(x)}{\sum_i p_i(x)}.$$

This approach is similar to the method used to combine density trees in [96]. Another alternative is to approximate each input mixture $p_i(x)$ by a single Gaussian density

IMPORTANCE SAMPLING:

1. Define the proposal distribution $q(x)$ by:
   (a) $q(x) = \frac{1}{d}\sum_i p_i(x)$ (MIXTURE I.S.)
   (b) $q(x) = \prod_i \mathcal{N}(x; \mathrm{E}[p_i(x)], \mathrm{Var}[p_i(x)])$ (GAUSSIAN I.S.)
2. Draw kN (where $k > 1$) samples x_j from $q(x)$, and weight by $w_j = \frac{\prod_i p_i(x_j)}{q(x_j)}$
3. Resample from the $\{x_j\}$ proportionally to their weight $\{w_j\}$ N times (with replacement)

Figure 3.3. Two possible methods of importance sampling from a product of Gaussian mixtures; either requires $\mathcal{O}(dkN)$ time and storage.

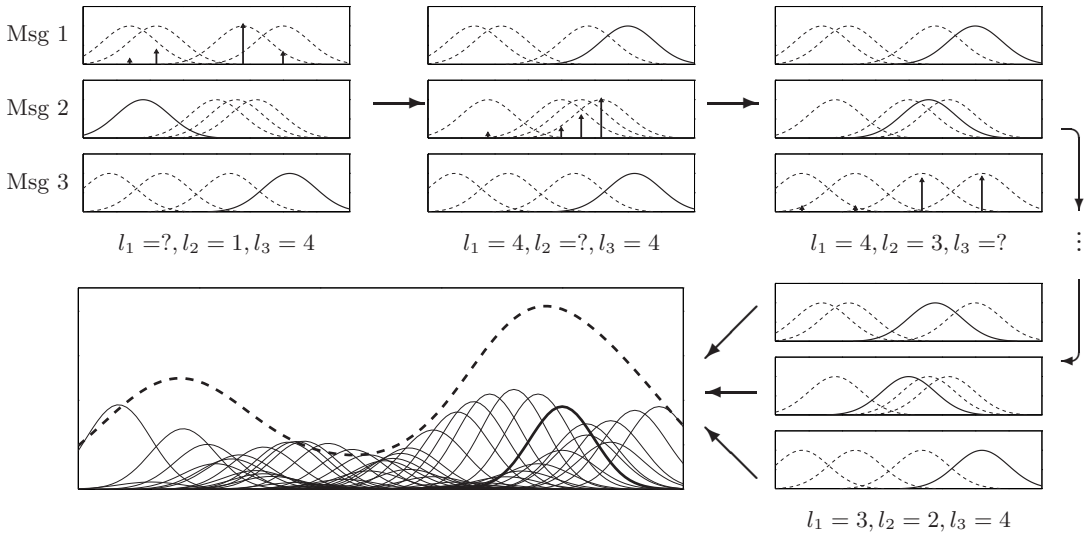


Figure 3.4. *Top row:* Sequential Gibbs sampler for a product of 3 Gaussian mixtures, with 4 components each. New indices are sampled according to weights (arrows) determined by the two fixed components (solid). The Gibbs sampler cycles through the different messages, drawing a new mixture label for one message conditioned on the currently labeled Gaussians in the other messages. *Bottom row:* After κ_s iterations through all the messages, the final labeled Gaussians for each message (right, solid) are multiplied together to identify one (left, solid) of the 4^3 components (left, thin) of the product density (left, dashed).

$q_i(x)$, and choose $q(x) \propto \prod_i q_i(x)$. We call this latter procedure *Gaussian importance sampling*. Pseudocode for both procedures appears in Figure 3.3.

Gibbs Sampling

Sampling from Gaussian mixture products is difficult precisely because the joint distribution over product density labels, as defined by (3.17), is complicated. However,

conditioned on the labels of all but one mixture, we can compute the conditional distribution over the remaining label and draw a sample in only $\mathcal{O}(N)$ operations. Since it is tractable to determine the conditional distribution of each mixture label given the value of the other mixture labels, we may apply a Gibbs sampler [26] to draw asymptotically unbiased samples from the product density. At each iteration, the labels $\{l_k\}_{k \neq j}$ for $d-1$ of the input mixtures are fixed, and the j^{th} label is sampled from the corresponding conditional density. The newly chosen l_j is then fixed, and another label is updated. This procedure continues for a fixed number of iterations κ_s; more iterations lead to more accurate samples, but require greater computational cost. Following the final iteration, a single sample is drawn from the product mixture component identified by the final labels. This iterative procedure is illustrated in Figure 3.4. Since each iteration of the Gibbs sampler requires $\mathcal{O}(dN)$ operations and we apply κ_s iterations to draw each sample, to draw N approximate[4] samples from the product density, the Gibbs sampler requires $\mathcal{O}(d\kappa_s N^2)$ operations.

Although formal verification of the Gibbs sampler's convergence is difficult, our empirical results indicate that accurate Gibbs sampling typically requires far fewer computations than direct sampling. Note that NBP uses the Gibbs sampling method differently from classic simulated annealing algorithms [26]. In simulated annealing, the Gibbs sampler updates a single Markov chain whose state is the value of *all* nodes in the graph, and thus has dimension proportional to the size of the graph. In contrast, the Gibbs sampling methods described here are *local*, with each Gibbs sampler involving only a few nodes.

The previously described *sequential Gibbs sampler* defines an iteration over the labels of the input mixtures. Another possibility uses the fact that, given a data point $\bar{x}$ in the product density space, the d input mixture labels are conditionally independent [39]. Thus, one can also define a *parallel Gibbs sampler* which alternates between sampling a data point conditioned on the current input mixture labels, and parallel sampling of the mixture labels given the current data point, as illustrated in Figure 3.5. The number of iterations κ_p to continue the parallel Gibbs sampling process is once again a parameter of the algorithm. Since drawing the sample $\bar{x}$ can be done in constant time, and sampling from the d labels independently conditioned on $\bar{x}$ requires $\mathcal{O}(dN)$ operations, the complexity of the parallel Gibbs sampler is also $\mathcal{O}(d\kappa_p N^2)$. Pseudocode outlining both Gibbs sampling algorithms is listed in Figure 3.6.

■ 3.8.2 Multi-scale methods

Multi-scale methods of sampling use the statistics of subsets of the data to focus computational effort more effectively. In order to cache and apply these statistics efficiently, we make use of KD ("k-dimensional") tree data structures [5, 17, 71, 75]; KD-trees are described in detail in Section 2.4.

Each Gaussian mixture distribution $p_i(x)$ is formed into a KD-tree, and within

[4]The samples are only approximately from the product distribution, due to the fact that κ_s iterations may be insufficient to reach the steady-state distribution.

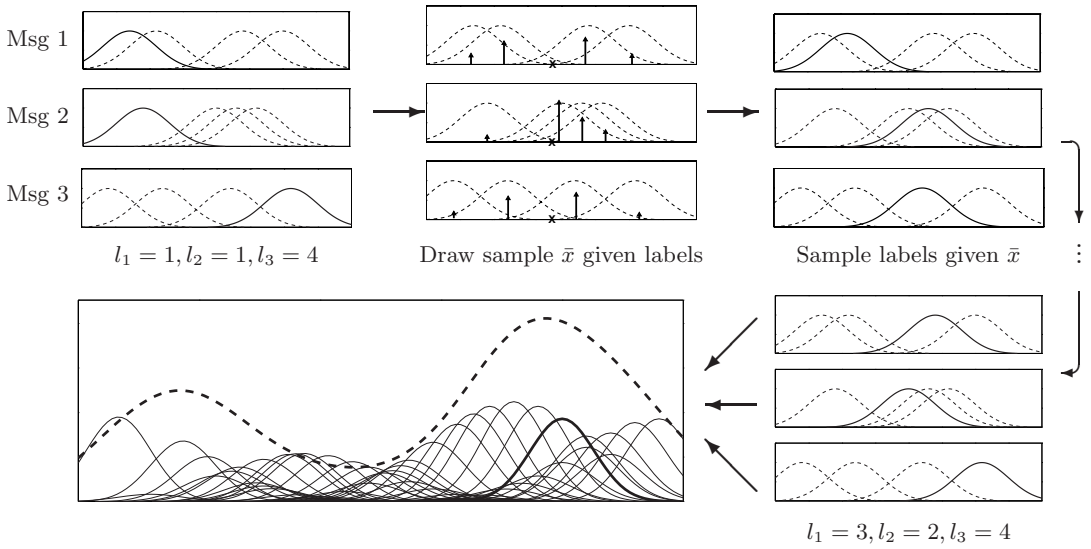


Figure 3.5. *Top Row:* Parallel Gibbs sampler for a product of 3 Gaussian mixtures, with 4 components each. Given a label for each input density selecting one component from each, a sample $\bar{x}$ (whose location s indicated by the "X") is drawn from their product; conditioned on the sampled value, one then computes the conditional distribution over labels independently for each distribution. *Bottom row:* After κ_p iterations, one again obtains a final set of labels identifying one component in the product and draws a sample from this component.

each KD-tree, each node s_i is associated with a collection of the leaf node labels $D_L(s_i)$ located below s_i in the KD-tree. Each node s_i also stores statistics which summarize the Gaussian mixtures indicated by the labels in $D_L(s_i)$. In this section we employ the two types of KD-trees described in Section 2.4. The first type caches mean and covariance information at each node to construct a set of multi-scale Gaussian approximations, as shown in Figure 3.7(b). We use these Gaussian approximations to define a multi-scale version of the Gibbs samplers described previously. The second type of KD-tree caches bounding box information, shown in Figure 3.7(a); using these statistics, along with branch and bound techniques common in the KD-tree literature, we are able to draw approximate samples efficiently using an "ϵ-exact" sampler.

Because our components are now elements of a KD-tree structure, it is convenient to have the label sets $\mathbb{L}_i$ reflect this tree structure. In particular, we use the numbering of the nodes within the KD-tree, so that 1_i refers to the root node of the i^{th} KD-tree, and so forth; note that for $N > 1$, the root node 1_i is *not* one of the original components, and thus not a member of the set $\mathbb{L}_i$. In fact, in the KD-tree notation of Section 2.4, the sets $\mathbb{L}_i$ are precisely the sets of leaf labels for the i^{th} KD-tree, so that $\mathbb{L}_i = D_L(1_i)$.

Multi-scale Gibbs Sampling

Although the pair of fine–scale Gibbs samplers discussed previously are often effective, they sometimes require a very large number of iterations to produce accurate samples.

GIBBS SAMPLING: For each sample required,

1. Choose initial values for the labels $l_1, \ldots, l_d$, e.g. independently by weight.
2. Iterate κ times:

   SEQUENTIAL:

   For each mixture i,

   i. Fix the value of all labels except l_i and compute w_L using (3.17) for each possible value $l_i \in [1 : N]$.
   ii. Sample a new value for l_i proportionally to the weights w_L

   PARALLEL:

   Draw a value $\bar{x}$ from the distribution $\prod_i \mathcal{N}(x\,;\,\mu_{l_i}, \Sigma_i)$

   For each mixture i, sample l_i giving $l_i \in [1 : N]$ weight $\mathcal{N}(\bar{x}\,;\,\mu_{l_i}, \Sigma_i)$

3. Draw the sample $x \sim \prod_i \mathcal{N}(x\,;\,\mu_{l_i}, \Sigma_i)$

Figure 3.6. Two Gibbs sampling procedures for drawing samples from the product of d Gaussian mixtures; either requires $\mathcal{O}(d\kappa N^2)$ time and $\mathcal{O}(dN)$ storage.

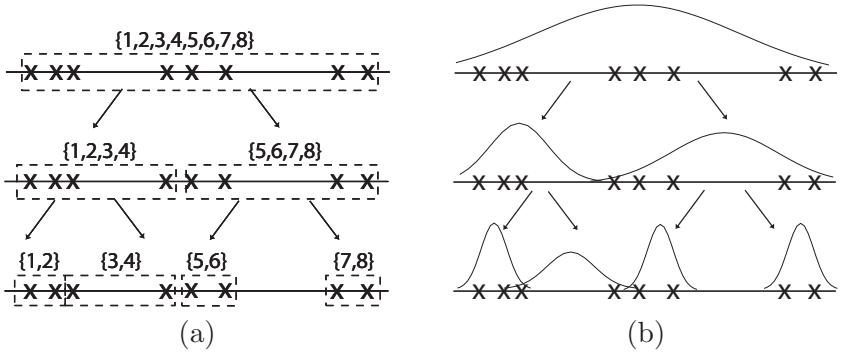


Figure 3.7. Two KD-tree representations of the same one-dimensional point set (finest scale not shown). (a) Each node s maintains a bounding box surrounding the means of all components associated with the node; the label sets indicating which of the original mixture components are summarized by each node s are also shown in braces. (b) Each node s maintains mean and variance statistics for its associated mixture components, giving rise to a collection of Gaussian approximations at each scale.

The most difficult densities are those for which there are several widely separated modes, each of which is associated with disjoint subsets of the input mixture labels. In this case, conditioned on a set of labels corresponding to one mode, it is very unlikely that a label or data point corresponding to a different mode will be sampled, leading to slow mixing between these modes, and thus may require many iterations to obtain accurate samples.

Similar problems have been observed with Gibbs samplers on Markov random fields [26]. In these cases, convergence can often be accelerated by constructing a

series of "coarser–scale" approximate models in which the Gibbs sampler can move between modes more easily [62]. The primary challenge in developing these algorithms is to determine procedures for constructing accurate coarse–scale approximations. For Gaussian mixture products, KD-trees provide a simple, intuitive, and easily constructed set of coarser–scale models.

As in Figure 3.7(b), each node s_i of the i^{th} KD-tree stores two statistics, the mean μ_{s_i} and a covariance Σ_{s_i}, which are used to represent the Gaussian sum defined by the leaf nodes located below s in the tree. Let $q_{i,s_i}(x) = \mathcal{N}(x; \mu_{s_i}, \Sigma_{s_i})$ be the Gaussian distribution defined by these parameters. Beginning at the same coarse scale for all input mixtures, indicated by depth $k = 1$ (the children of the root node), we perform standard Gibbs sampling (either parallel or sequential) on that scale's summary Gaussian components as though we were drawing a sample from the product

$$q_k(x) = \prod_{i=1}^{d} \left(\sum_{s_i \,:\, \mathrm{depth}(s_i)=k} q_{i,s_i}(x) \right)$$

After some number of iterations κ_k of Gibbs sampling, we draw a data sample $\bar{x}$ from the Gaussian defined by our current labels, so that $\bar{x}$ is sampled approximately from $q_k(x)$. We then condition on the value of $\bar{x}$ as in the parallel Gibbs sampler of Section 3.8.1 to sample from the labels at the next finest scale, $k + 1$. Repeating this process, we eventually arrive at the finest scale and obtain a data sample. To simplify the number of parameters in the algorithm, we typically choose the number of iterations κ_k to be equal for all depths k.

Intuitively, by gradually moving from coarse to fine scales, multi-scale sampling can better explore all of the product density's important modes. As the number of sampling iterations approaches infinity, multi-scale samplers have the same asymptotic properties as standard Gibbs samplers. Unfortunately, there is no guarantee that multi-scale sampling will improve performance. However, our simulation results indicate that it is usually very effective (see Section 3.8.3).

ϵ-Exact sampling

In this section, we use KD-trees to compute an efficient approximation to the partition function Z, in a manner similar to the dual tree evaluation algorithm of Gray and Moore [32] described in some detail in Section 2.4. This leads to an ϵ-*exact* sampler for which a label $L = [l_1, \ldots, l_d]$ in the product density, with true probability p_L, is guaranteed to be sampled with some probability $\hat{p}_L \in [p_L - \epsilon, p_L + \epsilon]$.

If we again let s_i denote a node of the i^{th} KD-tree, we can define the label set $\mathfrak{l}_i = D_L(s_i)$ to be the indices of the Gaussian mixture components associated with s_i, i.e., those nodes of the KD-tree which are (leaf) descendants of node s_i. Then, a set of labels in the product density can be written as $\mathfrak{L} = \mathfrak{l}_1 \times \cdots \times \mathfrak{l}_d$. This label set is implicitly a function of the nodes $\mathbf{s} = [s_1, \ldots, s_d]$ in each KD-tree.

The approximate sampling procedure follows a similar structure to the exact sampler

described in Section 3.8.1, but uses branch and bound approximations much like those applied in the dual–tree algorithm of Section 2.4. First we give a high–level sketch of the algorithm, providing the details of each part subsequently. Given a KD-tree representation of each input density which caches bounding box statistics, as illustrated in Figure 3.7(a), we use a *multi–tree* recursion to find sets of labels $\mathfrak{L}$ which have nearly identical weight, i.e., for which we may use a single constant to approximate the weight w_L for each label L in the set $\mathfrak{L}$. These sets, the approximate weights $\hat{w}_L$, and their totals $\hat{w}_{\mathfrak{L}} = \sum_{L \in \mathfrak{L}} \hat{w}_L$ are used to efficiently approximate the weight partition function as $\hat{Z} = \sum \hat{w}_L$. We can then draw samples efficiently by first determining from which *set* $\mathfrak{L}$ each sample originates (using a procedure similar to that of exact sampling), then determining a label $L \in \mathfrak{L}$ within that set. We proceed to describe each of these steps in greater detail.

Approximate Evaluation of the Weight Partition Function, $Z = \sum w_L$. We first note that the weight computation (3.17) can be rewritten using terms which involve only pairs of distributions (i, j), as

$$w_L = C_\Sigma \cdot \Big(\prod_{j=1}^{d} w_{l_j} \Big) \cdot \prod_i \prod_{j>i} \mathcal{N}\left(\mu_{l_i} ; \mu_{l_j}, \Sigma_{(i,j)}\right) \qquad \text{where} \quad \Sigma_{(i,j)} = \Sigma_i \Sigma_L^{-1} \Sigma_j \quad (3.18)$$

where C_Σ is a constant[5] which depends only on the covariances $\{\Sigma_i\}$; when these covariances are uniform within each mixture p_i, as we have assumed, this constant can be ignored. The equation (3.18) may be divided into two parts: a weight contribution $\prod_{i=1}^{d} w_{l_i}$, and a distance contribution, denoted K_L, which is expressed in terms of the pairwise distances between component means. For a particular collection of nodes $\mathbf{s}$ in the KD-trees, we use the bounding boxes stored at each s_i to compute upper and lower bounds on each of the pairwise distance terms for a collection of labels $\mathfrak{L} = \mathfrak{l}_1 \times \cdots \times \mathfrak{l}_d$. The product of the upper (lower) pairwise bounds is itself an upper (lower) bound on the total distance contribution for any label L within the set $\mathfrak{L}$; denote these bounds by $K_{\mathfrak{L}}^+$ and $K_{\mathfrak{L}}^-$, respectively.[6]

By using the mean $K_{\mathfrak{L}}^* = \frac{1}{2}\left(K_{\mathfrak{L}}^+ + K_{\mathfrak{L}}^-\right)$ to approximate K_L, we incur a maximum error $\frac{1}{2}\left(K_{\mathfrak{L}}^+ - K_{\mathfrak{L}}^-\right)$ for any label $L \in \mathfrak{L}$. Let δ be a small tolerance parameter, whose relationship to ϵ we quantify shortly. If the error $|K_{\mathfrak{L}}^* - K_L|$ is less than $Z\delta$, which we ensure by comparing to a running lower bound Z_{min} on Z, we treat it as constant over the set $\mathfrak{L}$ and approximate the contribution to Z by a sum of the approximate weights

[5]Specifically, it is the ratio of normalization constants, so that

$$C_\Sigma \propto |\Sigma_L| \prod_i \prod_{j>i} |\Sigma_{(i,j)}| / \prod_i |\Sigma_i|.$$

[6]We could also use multipole methods such as the fast Gauss transform [34, 35, 92] to efficiently compute alternate, potentially tighter bounds on the pairwise values.

$\hat{w}_L$, namely

$$\sum_{L\in\mathfrak{L}} w_L \approx \sum_{L\in\mathfrak{L}} \hat{w}_L = K^*_{\mathfrak{L}} \sum_{L\in\mathfrak{L}} \left(\prod_i w_{l_i}\right) = K^*_{\mathfrak{L}} \prod_i \left(\sum_{l_i\in\mathfrak{l}_i} w_{l_i}\right). \tag{3.19}$$

This quantity can be easily calculated using cached statistics of the weight contained in each set. If the error is larger than $Z\delta$, our approximation is not sufficiently accurate. The approximation is a function of the KD-tree nodes $\mathbf{s} = [s_1, \ldots, s_d]$ and the bounding boxes stored by each node; it can be made more accurate by using smaller bounding boxes, which brings the upper and lower bounds closer together. We therefore refine at least one of the label sets, by splitting one of the nodes s_i into its left and right children. We use a simple heuristic to make this choice, first finding the pair of trees with the largest discrepancy between upper and lower pairwise bounds and then, of these two, dividing the tree with the larger bounding box. It is possible that some other selection method, if able to select label sets with approximately constant weight more rapidly, might perform better; the existence of an optimal refinement strategy remains an open question. However, in practice the described heuristic appears to perform well. The full procedure is summarized in the pseudocode in Figure 3.8. Note that all of the quantities required by this algorithm may be stored within the KD-trees, avoiding the need for any direct searches over the sets $\mathfrak{l}_i$. At the algorithm's termination, the total error in our estimate of the partition function is bounded by

$$|Z - \hat{Z}| \leq \sum_L |w_L - \hat{w}_L| \leq \sum_L \tfrac{1}{2}\left(K^+_{\mathfrak{L}} - K^-_{\mathfrak{L}}\right) \prod w_{l_i} \leq Z\delta \sum_L \prod w_{l_i} \leq Z\delta \tag{3.20}$$

where the last inequality follows because each input mixture's weights are normalized. This guarantees that our estimate $\hat{Z}$ is within a fractional tolerance δ of its true value.

Approximate Sampling from the Cumulative Distribution: We now show how our partition function estimate $\hat{Z}$ may be used for approximate sampling, and relate the tolerance δ to an ϵ-tolerance on sample probability. To perform approximate sampling, we repeat the process of approximating the weights w_L with $\hat{w}_L$, while following a procedure similar to the exact sampler. We draw N uniform random variables u_j, and sort them in ascending order. We then create the cumulative distribution of the *sets* of labels $\mathfrak{L}$ (in the order these sets were originally identified when computing $\hat{Z}$), giving each set weight $w_{\mathfrak{L}}/\hat{Z}$, and locate each of the samples u_j in this cumulative distribution. This determines which set of labels $\mathfrak{L} = \mathfrak{l}_1 \times \cdots \times \mathfrak{l}_d$ is associated with each sample u_j.

Now, given that u_j came from the set of labels $\mathfrak{L}$, it remains to select an individual label $L \in \mathfrak{L}$. Since each label $L \in \mathfrak{L}$ within this block has an approximately equal distance contribution $K_L \approx K^*_{\mathfrak{L}}$, we can select one of the labels $L \in \mathfrak{L}$ by independently sampling a label l_i within each set $\mathfrak{l}_i$ proportionally to the weight w_{l_i}, for each input density i.

This procedure is shown in Figure 3.9. Note that, to be consistent about when approximations are made and thus produce weights $\hat{w}_{\mathfrak{L}}$ which still sum to $\hat{Z}$, we repeat

MultiTree($[s_1, \ldots, s_d]$)

1. Denote the fine-scale label sets $\mathfrak{l}_i = D_L(s_i)$ for each mixture i
2. For each pair of distributions $(i, j > i)$, use their bounding boxes to compute
   (a) $K_{max}^{(i,j)} \geq \max_{l_i \in \mathfrak{l}_i, l_j \in \mathfrak{l}_j} \mathcal{N}\left(x_{l_i} - x_{l_j} \,;\, 0, \Sigma_{(i,j)}\right)$
   (b) $K_{min}^{(i,j)} \leq \min_{l_i \in \mathfrak{l}_i, l_j \in \mathfrak{l}_j} \mathcal{N}\left(x_{l_i} - x_{l_j} \,;\, 0, \Sigma_{(i,j)}\right)$
3. Find $K_{\mathfrak{L}}^{+} = \prod_{(i,j>i)} K_{max}^{(i,j)}$ and $K_{\mathfrak{L}}^{-} = \prod_{(i,j>i)} K_{min}^{(i,j)}$
4. If $\frac{1}{2}\left(K_{\mathfrak{L}}^{+} - K_{\mathfrak{L}}^{-}\right) \leq Z_{min}\delta$, approximate this combination of label sets:
   (a) $\hat{w}_{\mathfrak{L}} = \frac{1}{2}\left(K_{\mathfrak{L}}^{+} + K_{\mathfrak{L}}^{-}\right)\left(\prod w_{\mathfrak{l}_i}\right)$, where $w_{\mathfrak{l}_i} = \sum_{l_i \in \mathfrak{l}_i} w_{l_i}$ is cached by the KD-trees
   (b) $Z_{min} = Z_{min} + K_{\mathfrak{L}}^{-}\left(\prod w_{\mathfrak{l}_i}\right)$
   (c) $\hat{Z} = \hat{Z} + \hat{w}_{\mathfrak{L}}$
5. Otherwise, refine one of the label sets:
   (a) Find $\arg\max_{(i,j)} K_{max}^{(i,j)} / K_{min}^{(i,j)}$ such that $\text{range}(s_i) \geq \text{range}(s_j)$.
   (b) Call recursively:
      i. MultiTree($[s_1, \ldots, \text{Nearer}(\text{Left}(s_i), \text{Right}(s_i), s_j), \ldots, s_d]$)
      ii. MultiTree($[s_1, \ldots, \text{Farther}(\text{Left}(s_i), \text{Right}(s_i), s_j), \ldots, s_d]$)

   where Nearer(Farther) returns the nearer (farther) of the first two arguments to the third.

Figure 3.8. Recursive multi-tree algorithm for approximately evaluating the partition function Z of the product of d Gaussian mixture densities represented by KD–trees. Z_{min} denotes a running lower bound on the partition function, while $\hat{Z}$ is the current estimate. Initialize $Z_{min} = \hat{Z} = 0$.

Given the final partition function estimate $\hat{Z}$, repeat the algorithm in Figure 3.8 with the following modifications:

4.(c) If $\hat{c} \leq \hat{Z}u_j < \hat{c} + \hat{w}_{\mathfrak{L}}$ for any j, draw $L \in \mathfrak{L}$ by sampling $l_i \in \mathfrak{l}_i$ independently for each mixture i with weight $w_{l_i}/w_{\mathfrak{l}_i}$

4.(d) $\hat{c} = \hat{c} + \hat{w}_{\mathfrak{L}}$

Figure 3.9. Recursive multi-tree algorithm for approximate sampling. $\hat{c}$ denotes the cumulative sum of weights $\hat{w}_{\mathfrak{L}}$. Initialize by sorting N uniform $[0, 1]$ samples $\{u_j\}$, and set $Z_{min} = \hat{c} = 0$.

the procedure for computing $\hat{Z}$ exactly, including recomputing the running lower bound Z_{min}. This algorithm is guaranteed to sample each label L with probability $\hat{p}_L \in [p_L - \epsilon, p_L + \epsilon]$, where

$$|\hat{p}_L - p_L| = \left|\frac{\hat{w}_L}{\hat{Z}} - \frac{w_L}{Z}\right| \leq \frac{2\delta}{1-\delta} \doteq \epsilon \tag{3.21}$$

We can show (3.21) using the following argument. Using our accuracy bound on $\hat{w}_L$ we have

$$\begin{aligned}\left|\frac{w_L}{Z} - \frac{\hat{w}_L}{Z}\right| &= \frac{|K_L - K_L^*|}{Z}\prod w_{l_i} \\ &\leq \delta(\prod w_{l_i}) \\ &\leq \delta.\end{aligned}$$

Furthermore, we can show that

$$\begin{aligned}\left|\frac{\hat{w}_L}{Z} - \frac{\hat{w}_L}{\hat{Z}}\right| &= \frac{\hat{w}_L}{Z}\left|1 - \frac{1}{\hat{Z}/Z}\right| \\ &\leq \frac{\hat{w}_L}{Z}\left|1 - \frac{1}{1-\delta}\right|\end{aligned}$$

since $|\hat{Z} - Z| \leq \delta Z \Rightarrow \frac{\hat{Z}}{Z} \in [1-\delta, 1+\delta]$. Then by simple algebra,

$$\begin{aligned}&= \frac{\hat{w}_L}{Z}\frac{\delta}{1-\delta} \\ &\leq \frac{1+\delta}{1-\delta}\delta\end{aligned}$$

Thus, the estimated probability of choosing label L has at most error

$$\begin{aligned}\left|\frac{w_L}{Z} - \frac{\hat{w}_L}{\hat{Z}}\right| &\leq \left|\frac{w_L}{Z} - \frac{\hat{w}_L}{Z}\right| + \left|\frac{\hat{w}_L}{Z} - \frac{\hat{w}_L}{\hat{Z}}\right| \\ &\leq \frac{2\delta}{1-\delta}\end{aligned}$$

which matches our definition of ϵ in (3.21).

■ 3.8.3 Empirical Comparisons

In order to gain some intuition about which methods of drawing samples from the product distribution may be more or less appropriate under various conditions, we provide some experimental evidence comparing their performance. To be precise, we perform a Monte Carlo analysis to evaluate the quality of a set of N samples drawn from each method described in Sections 3.8.1–3.8.2 as a function of the required computation time. Sample quality is measured by constructing a kernel density estimate using the samples and computing the Kullback-Leibler (KL) divergence from the true product distribution.

Unfortunately, evaluating this KL-divergence is not easy; in general it requires either a discretized estimate of the product distribution or a large number of exact samples. The latter are difficult to provide in general, since as discussed, exact sampling is

computationally expensive for large N. A discretized estimate, on the other hand, is only tractable for low–dimensional problems. In order to construct examples with reasonably large values of N, we consider several synthetic one–dimensional example products, for which a direct, discretized evaluation is feasible.

To this end, we create d one–dimensional distributions expressed as the sum of $N = 100$ equal–weight, equal–bandwidth Gaussian kernels. The three examples we consider are shown in Figure 3.10(a-c), which have $d = 3$, 5, and 2 Gaussian mixtures, respectively. For each sampling method to be evaluated, we then draw N samples from the product of these distributions, estimate a kernel bandwidth using the likelihood cross-validation method described in Section 2.3.1, and evaluate the KL-divergence between the true product and its sampled estimate. Note that we are comparing the true, N^d component Gaussian mixture with a kernel density estimate constructed by drawing N samples; thus, even a density estimate constructed using N exact samples will have some nonzero divergence on average. This average divergence provides a lower bound on the achievable error for comparison. We show the input mixtures, product mixtures, and average performance versus time over 250 Monte Carlo trials for three different scenarios in Figure 3.10.

Exact sampling is extremely slow; except for the example with $d = 2$, shown in Figure 3.10(c), the time required was too far beyond the scale of the other methods to be shown on the same plot. In these cases, corresponding to Figures 3.10(a-b), we simply show a horizontal line indicating the average KL-divergence of a kernel density estimate constructed using N exact samples, toward which all our approximate sampling methods approach asymptotically as the available computational resources increase. We list the time required to *draw* such a set of N samples using exact sampling in each figure caption.

Figure 3.10 illustrates a few important points. The first is that for relatively small numbers of input mixtures, such as the product of three mixtures in Figure 3.10(a), the ϵ-exact method of Section 3.8.2 performs very well. Its theoretical guarantees allow a relatively principled choice of parameter settings, and it shows significant computational gains over exact sampling (0.05 seconds versus 2.75 seconds).

However, for larger numbers of input mixtures such as the product of five densities Figure 3.10(b), ϵ-exact is simply too slow. Although it is still much faster than exact sampling (requiring less than one minute as compared to 7.6 hours), only with very large settings of ϵ, and thus very poor approximation quality, do we manage to draw samples within the same time scale required by the Gibbs-based methods (0.3 seconds). In our experiments, both multi-scale Gibbs sampling methods out–perform their single–scale counterparts. This difference in performance can be attributed to the bimodal nature of the product density. In addition, we see that sequential Gibbs sampling is more accurate than parallel Gibbs sampling.

Notably, in the first two examples [Figures 3.10(a)-(b)], mixture importance sampling (IS) described in Section 3.8.1 is nearly as accurate as the best multi-scale methods, although Gaussian IS seems ineffective. However, in cases where the regions of the

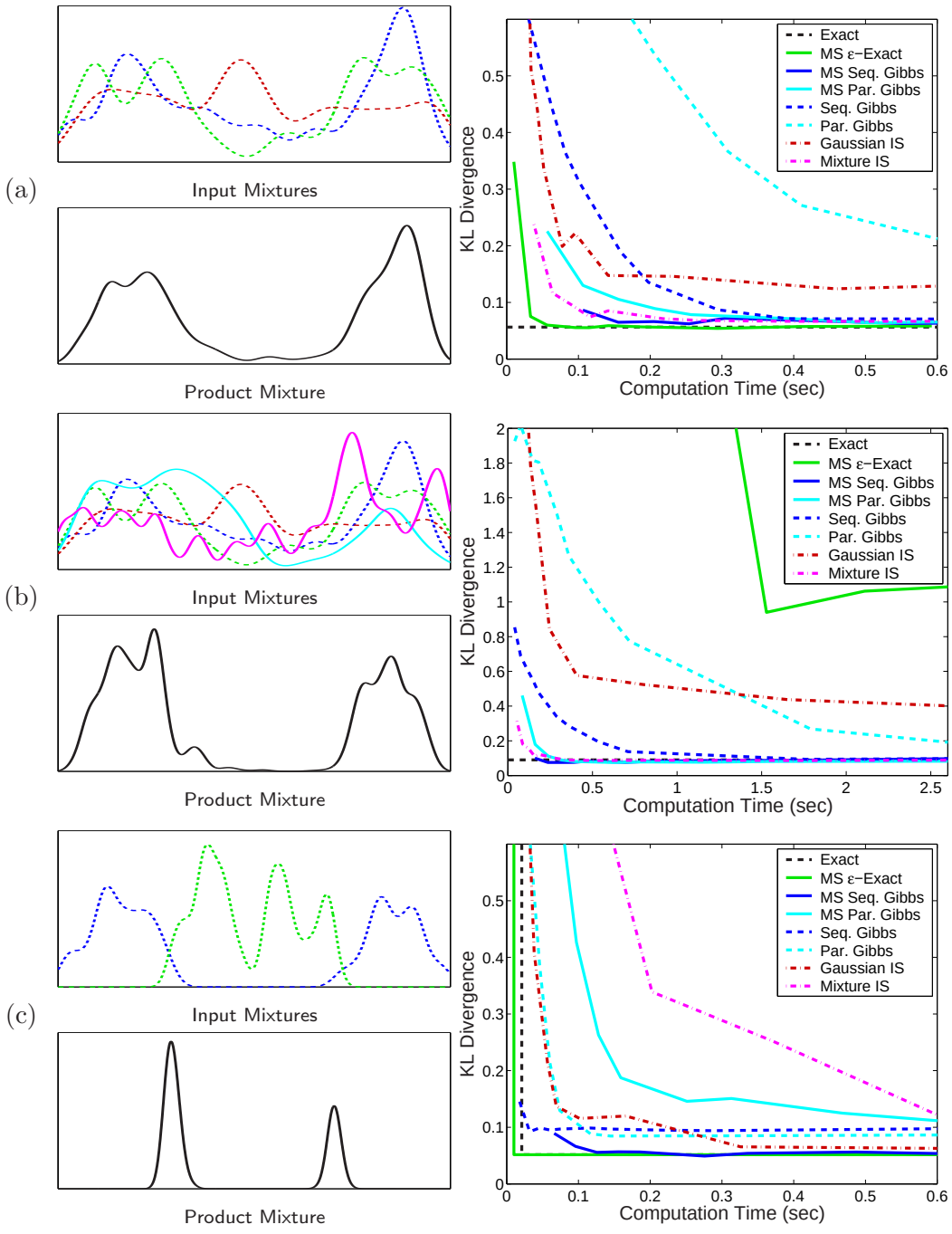


Figure 3.10. Comparison of average sampling accuracy versus computation time for different algorithms (see text). (a) Product of 3 mixtures (exact requires 2.75 sec). (b) Product of 5 mixtures (exact requires 7.6 hours). (c) Product of 2 mixtures (exact requires 0.02 sec). Computation times were measured on a Pentium III 800MHz workstation.

state space with high probability in the product density have relatively low probability in each of the input densities [i.e., the input densities have little overlap with one another, as in Figure 3.10(c)], mixture IS performs very poorly. In contrast, multi-scale samplers perform very well in such situations, because they can discard large numbers of low weight product density kernels. These types of situations plague importance sampling methods in general, and are more likely to arise and cause problems in high dimensional problems [65].

■ 3.9 Experimental Demonstrations

In this section, we apply NBP to a few relatively simple inference problems as a demonstration of its utility. We first use a jointly Gaussian problem to provide a simple validation of NBP's ability to approximate the correct continuous BP messages without prior knowledge of the distribution's parametric form, by comparing NBP to an exact implementation of BP specialized to Gaussian problems. We then apply NBP to a target tracking problem, in which the individual targets are constrained not to approach too closely to one another, illustrating one of the ways in which NBP can be used to augment traditional particle filtering approaches to improve robustness without significantly impacting efficiency.

■ 3.9.1 Gaussian Graphical Models

Gaussian graphical models provide one of the few continuous distributions for which the BP algorithm may be implemented exactly [107]. For this reason, Gaussian models may be used to test the accuracy of the nonparametric approximations made by NBP. Note that we cannot hope for NBP to outperform algorithms, like Gaussian BP, designed to take advantage of the linear structure underlying Gaussian problems. Instead, our goal is to verify NBP's performance in a situation where exact comparisons are possible.

We examine NBP's performance on a 5×5 nearest–neighbor grid as in Figure 3.11(a), with randomly chosen inhomogeneous potentials. Qualitatively similar results have also been observed in experiments on tree–structured and chain–structured graphs. Each potential is thus specified by the parameters of a Gaussian distribution; however, in the interest of generality we do not use the analytic message convolution form described in Section 3.4.3, as this is applicable only to Gaussian potentials. Instead, we use the more general NBP procedure, drawing samples from the conditional defined by ψ_{ts} and selecting the bandwidth automatically; in particular we use the likelihood cross-validation method described in Section 2.3.1.

For each node $t \in \mathcal{V}$, Gaussian BP converges to a steady–state estimate of the marginal mean μ_t and variance σ_t^2 after about 10 iterations. To evaluate NBP, we performed 10 iterations of the NBP message updates using several different particle set sizes $N \in [10, 800]$. We then found the mean $\hat{\mu}_t$ and variance $\hat{\sigma}_t^2$ of the approximate marginal distributions obtained via NBP. For each tested particle set size, the NBP comparison was repeated 50 times.

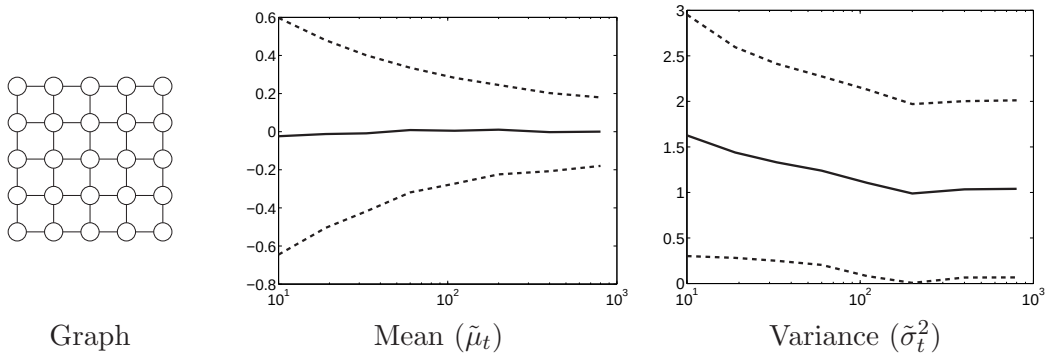


Figure 3.11. NBP performance on a 5×5 grid with Gaussian potentials. Plots show the mean (solid) and standard deviation (dashed) of the normalized error measures of equation (3.22), for different particle set sizes M.

Using the data from each NBP trial, we computed the error in the mean and variance estimates, normalized to represent the relative error in each estimate (as compared to exact Gaussian BP),

$$\tilde{\mu}_t = \frac{\hat{\mu}_t - \mu_t}{\sigma_t} \qquad \tilde{\sigma}_t^2 = \frac{\hat{\sigma}_t^2 - \sigma_t^2}{\sqrt{2}\sigma_t^2}. \tag{3.22}$$

Figure 3.11 shows the mean and standard deviation of these error statistics, across all nodes and trials, for different particle set sizes N. The NBP algorithm always provides unbiased estimates of the mean statistics, and the variance of the error in means decreases fairly rapidly. The estimates of the variance statistics, however, are positively biased, due to the inherent smoothing of kernel–based density estimates. This bias decreases as more particles are used, and the (automatically chosen) kernel size becomes smaller.

■ 3.9.2 Multi-Target Tracking

A classic filtering application is the task of estimating the location of a moving object (the "target"), along with its uncertainty, given a dynamic model and sequence of observations. Particle filtering is frequently brought to bear on this problem when either the dynamics or observation process is nonlinear or non-Gaussian, leading to non-Gaussian posterior distributions. However, even for linear, Gaussian dynamics and observations, the presence of *multiple*, interacting targets can present difficulties, and lead to non-Gaussian (for example, multi-modal) estimates of uncertainty.

We consider the problem of tracking multiple, indistinguishable moving objects using sample-based (particle filter-like) representations. Computationally speaking, it is much simpler to represent the state of each target using separate, independently evolving Markov chains. This is due to the fact that the number of particles required to model a distribution adequately is approximately exponential in the dimension [90]. Thus for

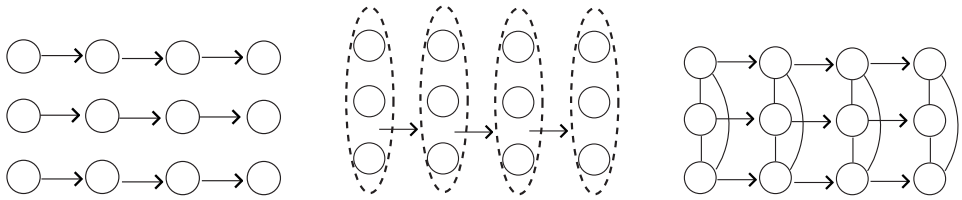


Figure 3.12. Three possible graphical models for multi-target tracking: (a) multiple, independent Markov chains, (b) a single Markov chain defined on the joint state space of all targets, and (c) multiple chains coupled by pairwise interactions at each time step.

m targets, if each is represented by a d-dimensional state variable whose uncertainty can be approximated accurately using N samples, it requires only mN samples to represent the targets independently, but approximately N^m samples to represent the joint uncertainty.

However, treating each target independently often fails. When two targets are in close proximity, with nothing to distinguish between them, independent tracking of each target has the potential to result in *both* trackers following the same target, typically whichever one happens to have higher likelihood under the dynamics and observation noise models. This effect is generally avoided by imposing a *data association* condition, that a given observation (or portion of the observation, for example in video-based tracking) be assigned to one and only one target, resulting in a highly non-linear and non-Gaussian relationship [64]. In order to capture this interaction exactly, it is typically necessary to model the joint state of all targets collectively. Seeking representational simplicity, many multi-target tracking applications construct locality–based approximations, in which targets which are sufficiently far from one another are treated independently, under the assumption that these targets are unlikely to be assigned the same observations [61].

We apply a different constraint in order to avoid degeneracy in the target tracks. Instead of constraining the association of observations, we simply enforce a condition that no two targets are allowed to occupy the same region of space. This can be expressed as a simple potential function between each pair of targets; we use the repulsive potential given by (3.12). Then the uncertainty in each target position can be modeled independently, and the interaction between targets captured using NBP on the resulting loopy graph, depicted in Figure 3.12.

Figure 3.13 shows a simulation of a multi-target tracking problem, in which five targets are to be tracked using either five independent Markov chains as in Figure 3.12(a), or five Markov chains coupled by repulsive pairwise potentials and estimated using NBP as in Figure 3.12(c). The solid lines indicate the true path of each target up to the current time, and the state estimate of each tracker is indicated by a cluster of sample locations. As can be seen, the independent Markov chains quickly suffer from track degeneracy as several targets pass through the same region; by $t = 34$ it has lost track

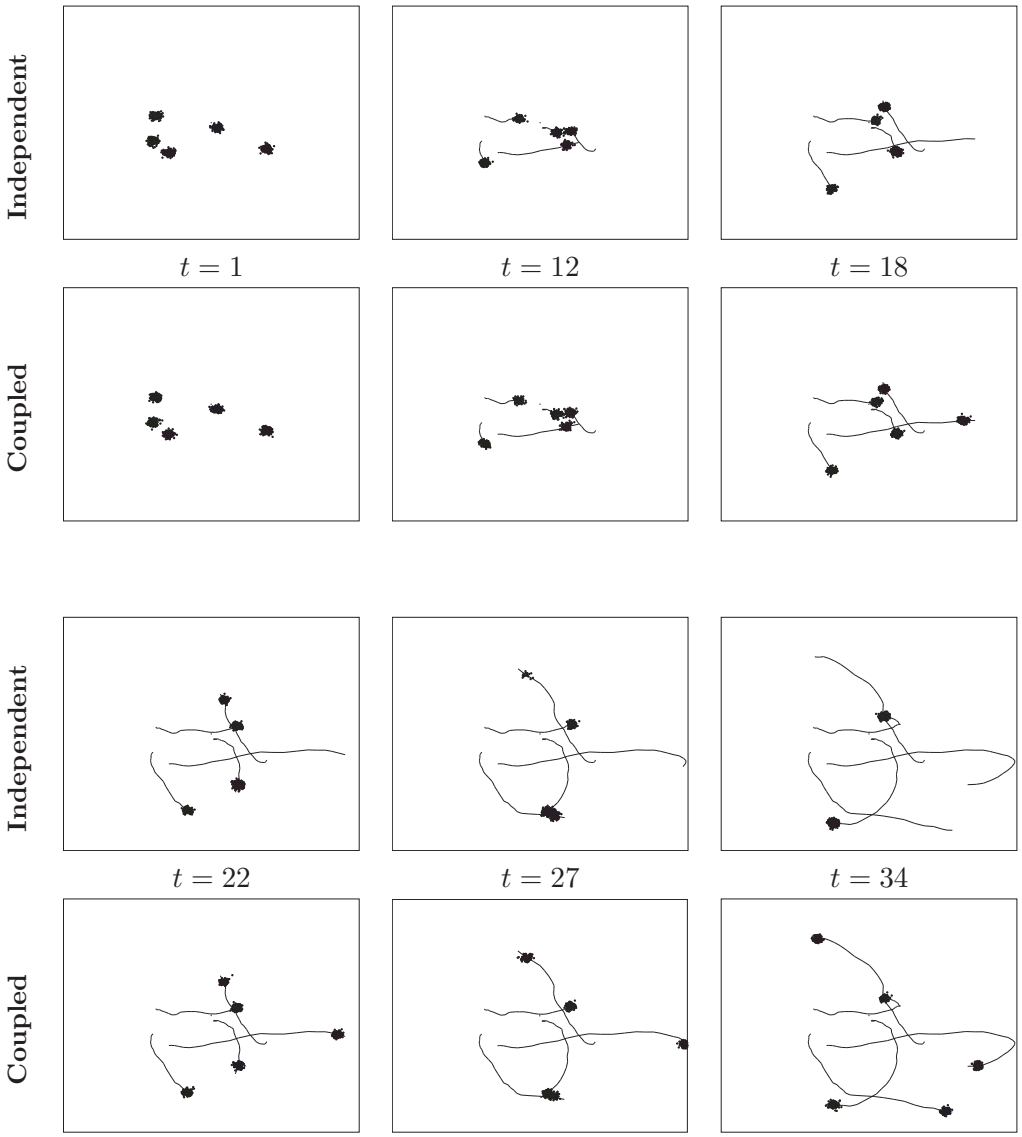


Figure 3.13. Using NBP for multi-target tracking. At time $t = 1$, the estimated uncertainty (dots) corresponds closely to the true position of each of the five targets, whose paths over time are indicated by lines. However, as several targets pass close by to one another at $t = 12$, the independent tracker (upper row) begins to lose track of some targets, and follows only the "best", losing one target completely by $t = 18$. As targets continue to move near one another, more tracks are lost until by $t = 34$ only two remain correctly localized.

of all but two targets. The NBP-based tracker, on the other hand, is able to maintain accurate estimates of the location of each target and its uncertainty.

Chapter 4

Message Approximation

SENSOR networks are by their nature subject to constraints which can often prevent exact inference from being feasible. In particular, communications constraints prevent the network from aggregating all observations at a single, central location, making a distributed implementation of exact or approximate inference algorithms a necessity. For the task of estimating the local posterior marginal distributions at each sensor, belief propagation (BP) provides an efficient and potentially distributed approach. One part of the appeal of BP lies in its optimality for tree-structured graphical models (models which contain no loops); however, its is also widely applied to graphical models with cycles ("loopy" BP). In these cases belief propagation may not converge, and if it does its solution is approximate; however in practice these approximations are often good. Recently, some additional justifications for loopy belief propagation have been developed, including a handful of convergence results for graphs with cycles [37, 95, 105].

The approximate nature of loopy belief propagation is often a more than acceptable price for performing efficient inference; in fact, it is sometimes desirable to make *additional* approximations. There may be a number of reasons for this—for example, when exact message representation is computationally intractable, the messages may be approximated stochastically [56] or deterministically by discarding low-likelihood states [13]. For belief propagation involving continuous, non-Gaussian potentials, some form of approximation is required to obtain a finite parameterization for the messages [48, 69, 93]. Additionally, simplification of complex graphical models through edge removal, quantization of the potential functions, or other forms of distributional approximation may be considered in this framework. Finally, one may wish to approximate the messages and reduce their representation size for another reason—to decrease the communications required for distributed inference applications. In distributed message passing, one may approximate the transmitted message to reduce its representational cost (see Chapter 5), or discard it entirely if it is deemed "sufficiently similar" to the previously sent version [10]. Through such means one may significantly reduce the amount of communications required.

Given that message approximation may be desirable, we would like to know what effect the errors introduced have on our overall solution. In order to characterize the approximation effects in graphs with cycles, we analyze the deviation from the solution given by "exact" loopy belief propagation (*not*, as is typically considered, the deviation

of loopy BP from the true marginal distributions). As a byproduct of this analysis, we also obtain some results on the convergence of loopy belief propagation.

We apply the formulation of loopy belief propagation as described in Section 2.6, for pairwise graphical models using a parallel update schedule, and we describe the notion of *approximate* messages in Section 4.1. Section 4.3 then examines the consequences of measuring a message error by its dynamic range. In particular, we explain the utility of this measure and its behavior with respect to the operations of belief propagation. This allows us to derive conditions for the convergence of traditional loopy belief propagation, and bounds on the distance between any pair of BP fixed points (Sections 4.4.1-4.4.2), and these results are easily extended to many approximate forms of BP (Section 4.4.3). If the errors introduced are independent (a typical assumption in, for example, quantization analysis [28, 109]), tighter estimates of the resulting error can be obtained (Section 4.4.5).

It is also instructive to examine other measures of message error, in particular ones which emphasize more average-case (as opposed to pointwise or worst-case) differences. To this end, we consider a KL-divergence based measure in Section 4.5. While the analysis of the KL-divergence measure is considerably more difficult and does not lead to strict guarantees, it serves to give some intuition into the behavior of perturbed BP under an average-case difference measure.

■ 4.1 Message Approximations

Let us consider the concept of *approximate* BP messages. We begin by assuming that the "true" messages $m_{ts}(x_s)$ are some fixed point of BP, so that $m_{ts}^i = m_{ts}^{i+1}$. We may ask what happens when these messages are perturbed by some (perhaps small) error function $e_{ts}(x_s)$. Although there are certainly other possibilities, the fact that BP messages are combined by taking their product makes it natural to consider multiplicative message deviations (or additive in the log-domain):

$$\hat{m}_{ts}^i(x_s) = m_{ts}(x_s)e_{ts}^i(x_s)$$

To facilitate our analysis, we split the message update operation (2.21) into two parts. In the first, we focus on the message *products*

$$\hat{M}_{ts}^i(x_t) \propto \psi_t(x_t) \prod_{u \in \Gamma_t \setminus s} \hat{m}_{ut}^i(x_t) \qquad \hat{M}_t^i(x_t) \propto \psi_t(x_t) \prod_{u \in \Gamma_t} \hat{m}_{ut}^i(x_t) \tag{4.1}$$

where the proportionality constant is chosen to normalize $\hat{M}$. The second operation, then, is the message *convolution*

$$\hat{m}_{ts}^{i+1}(x_s) \propto \int \psi_{ts}(x_s, x_t)\hat{M}_{ts}^i(x_t)dx_t \tag{4.2}$$

where again $\hat{M}$ is a normalized message or product of messages.

We use the convention that lowercase quantities ($m_{ts}, e_{ts}, \ldots$) refer to messages and message errors, while uppercase ones ($M_{ts}, E_{ts}, M_t, \ldots$) refer to their products—at node t, the product of all incoming messages and the local potential is denoted $M_t(x_t)$, its approximation $\hat{M}_t(x_t) = M_t(x_t)E_t(x_t)$, with similar definitions for M_{ts}, $\hat{M}_{ts}$, and E_{ts}.

■ 4.2 Overview of Chapter Results

To orient the reader, we lay out the order and general results which are obtained in this chapter. We begin in Section 4.3 by examining a *dynamic range* measure $d(e)$ of the variability of a message error $e(x)$ (or more generally of any function) and show how this measure behaves with respect to the BP belief and message update equations. Specifically, we show in Section 4.3.2 that the measure $\log d(e)$ is sub-additive with respect to the product operation (4.1), and contractive with respect to the convolution operation (4.2).

Applying these results to traditional belief propagation results in a new sufficient condition for BP convergence (Section 4.4.1), specifically

$$\max_{s,t} \sum_{u \in \Gamma_t \backslash s} \frac{d(\psi_{ut})^2 - 1}{d(\psi_{ut})^2 + 1} < 1; \tag{4.3}$$

and this condition may be further improved in many cases. The condition (4.3) can be shown to be slightly stronger than the sufficient condition given in [95], and empirically appears to be stronger than that of [37]. More importantly, however, the *method* in which it is derived allows us to generalize to many other situations:

1. Using the same methodology, we may demonstrate that any two BP fixed points must be within a ball of a calculable diameter; the condition (4.3) is equivalent to this diameter being zero (Section 4.4.2).

2. Both the diameter of the bounding ball and the convergence criterion (4.3) are easily improved for graphical models with irregular geometry or potential strengths, leading to better conditions on graphs which are more "tree-like" (Section 4.4.3).

3. The same analysis may also be applied to the case of quantized or otherwise approximated messages and models (potential functions), yielding bounds on the resulting error (Section 4.4.4).

4. If we regard the message errors as a stochastic process, a similar analysis with a few additional, intuitive assumptions gives alternate, tighter estimates (though not necessarily bounds) of performance (Section 4.4.5).

Finally, in Section 4.5 we perform the same analysis for a less strict measure of message error (i.e. disagreement between a message $m(x)$ and its approximation $\hat{m}(x)$), namely the Kullback-Leibler divergence. This analysis shows that, while failing to

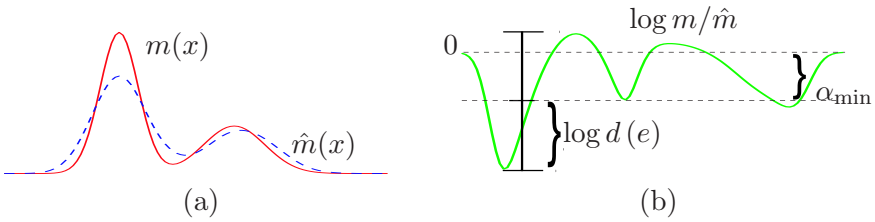


Figure 4.1. (a) A message $m(x)$ and an example approximation $\hat{m}(x)$; (b) their log-ratio $\log m(x)/\hat{m}(x)$, and the error measure $\log d\,(e)$.

provide strict bounds in several key ways, one is still able to obtain some intuition into the behavior of approximate message passing under an average-case difference measure.

In the next few sections, we first describe the dynamic range measure and discuss some of its salient properties (Section 4.3). We then apply these properties to analyze the behavior of loopy belief propagation (Section 4.4). Almost all proofs are given in an in-line fashion, as they frequently serve to give intuition into the method and meaning of each result.

■ 4.3 Dynamic Range Measure

In order to discuss the effects and propagation of errors, we first require a measure of the difference between two messages. In this section, we examine the following measure on $e_{ts}(x_s)$: let $d\,(e_{ts})$ denote the function's *dynamic range*[1], specifically

$$d\,(e_{ts}) = \sup_{a,b} \left(e_{ts}(a)/e_{ts}(b)\right)^{\frac{1}{2}} \tag{4.4}$$

Then, we have that $m_{ts} \equiv \hat{m}_{ts}$ (i.e., the pointwise equality condition $m_{ts}(x) = \hat{m}_{ts}(x)$ for all x) if and only if $\log d\,(e_{ts}) = 0$. Figure 4.1 shows an example of $m(x)$ and $\hat{m}(x)$ along with their associated error $e(x)$; $\log d\,(e)$ is shown as $\frac{1}{2}\sup_{a,b} \log e(a)/e(b)$.

■ 4.3.1 Motivation

We begin with a brief motivation for this choice of error measure. It has a number of desirable features; for example, it is directly related to the pointwise log error between the two distributions.

Lemma 4.3.1. *The dynamic range measure* (4.4) *may be equivalently defined by*

$$\log d\,(e_{ts}) = \inf_{\alpha} \sup_{x} |\log \alpha m_{ts}(x) - \log \hat{m}_{ts}(x)| = \inf_{\alpha} \sup_{x} |\log \alpha - \log e_{ts}(x)|$$

[1]This measure has also been independently investigated to provide a stability analysis for the max-product algorithm in Bayes' nets (acyclic, directed graphical models) [9]. While similar in some ways, the analysis for acyclic graphs is considerably simpler; loopy graphs require demonstrating a rate of contraction, which we show is possible for the sum-product algorithm (Theorem 4.3.4).

Proof. The minimum is given by $\log \alpha = \frac{1}{2}(\sup_a \log e_{ts}(a) + \inf_b \log e_{ts}(b))$, and thus the right-hand side is equal to $\frac{1}{2}(\sup_a \log e_{ts}(a) - \inf_b \log e_{ts}(b))$, or $\frac{1}{2}(\sup_{a,b} \log e_{ts}(a)/e_{ts}(b))$, which by definition is $\log d\,(e_{ts})$. □

The scalar α serves the purpose of "zero-centering" the function $\log e_{ts}(x)$ and making the measure invariant to simple rescaling. This invariance reflects the fact that the scale factor for BP messages is essentially arbitrary, defining a class of equivalent messages. Although the scale factor cannot be completely ignored, it takes on the role of a nuisance parameter. The inclusion of α in the definition of Lemma 4.3.1 acts to select particular elements of the equivalence classes (with respect to rescaling) between which to measure distance—specifically, choosing the closest such messages in a log-error sense. The log-error, dynamic range, and the minimizing α are depicted in Figure 4.1.

Lemma 4.3.1 allows the dynamic range measure to be related directly to an approximation error in the log-domain when both messages are normalized to integrate to unity, using the following theorem:

Theorem 4.3.1. *The dynamic range measure can be used to bound the approximation error in the log-domain,*

$$|\log m_{ts}(x) - \log \hat{m}_{ts}(x)| \leq 2 \log d\,(e_{ts}) \qquad \forall x.$$

Proof. We first consider the magnitude of $\log \alpha$:

$$\begin{aligned}
\forall x, \qquad & \left| \log \frac{\alpha m_{ts}(x)}{\hat{m}_{ts}(x)} \right| \leq \log d\,(e_{ts}) \\
\Rightarrow \qquad & \frac{1}{d\,(e_{ts})} \leq \frac{\alpha m_{ts}(x)}{\hat{m}_{ts}(x)} \leq d\,(e_{ts}) \\
\Rightarrow \qquad & \int \hat{m}_{ts}(x)dx \, \frac{1}{d\,(e_{ts})} \leq \alpha \int m_{ts}(x)dx \leq \int \hat{m}_{ts}(x)dx \, d\,(e_{ts})
\end{aligned}$$

and since the messages are normalized, $|\log \alpha| \leq \log d\,(e_{ts})$. Then by the triangle inequality,

$$|\log m_{ts}(x) - \log \hat{m}_{ts}(x)| \leq |\log \alpha m_{ts}(x) - \log \hat{m}_{ts}(x)| + |\log \alpha| \leq 2 \log d\,(e_{ts})\,. \qquad \square$$

In this light, our analysis of message approximation (Section 4.4.4) may be equivalently regarded as a statement about the required quantization level for an accurate implementation of loopy belief propagation. Interestingly, it may also be related to a floating-point precision on $m_{ts}(x)$.

Lemma 4.3.2. *Let $\hat{m}_{ts}(x)$ be an F-bit mantissa floating-point approximation to $m_{ts}(x)$. Then,* $\log d\,(e_{ts}) \leq 2^{-F} + \mathcal{O}(2^{-2F})$.

Proof. For an F-bit mantissa, we have $|m_{ts}(x) - \hat{m}_{ts}(x)| < 2^{-F} \cdot 2^{\lfloor \log_2 m_{ts}(x) \rfloor} \leq 2^{-F} \cdot m_{ts}(x)$. Then, using the Taylor expansion of $\log\left[1 + (\frac{\hat{m}}{m} - 1)\right] \approx (\frac{\hat{m}}{m} - 1)$ we have that

$$\begin{aligned}
\log d\left(e_{ts}\right) &\leq \sup_x \left| \log \frac{\hat{m}_{ts}(x)}{m_{ts}(x)} \right| \\
&\leq \sup_x \frac{\hat{m}_{ts}(x) - m_{ts}(x)}{m_{ts}(x)} + \mathcal{O}\left(\left(\sup_x \frac{\hat{m}_{ts}(x) - m_{ts}(x)}{m_{ts}(x)} \right)^2 \right) \\
&\leq 2^{-F} + \mathcal{O}\left(2^{-2F}\right)
\end{aligned}$$

□

Thus our measure of error is, to first order, similar to the typical measure of precision in floating-point implementations of belief propagation on microprocessors. We may also relate $d\left(e\right)$ to other measures of interest, such as the Kullback-Leibler (KL) divergence:

Lemma 4.3.3. *The KL-divergence satisfies the inequality* $D(m_{ts} \| \hat{m}_{ts}) \leq 2 \log d\left(e_{ts}\right)$

Proof. By Theorem 4.3.1, we have

$$D(m_{ts} \| \hat{m}_{ts}) = \int m_{ts}(x) \log \frac{m_{ts}(x)}{\hat{m}_{ts}(x)} dx \leq \int m_{ts}(x) \left(2 \log d\left(e_{ts}\right)\right) dx = 2 \log d\left(e_{ts}\right)$$

□

Finally, a bound on the dynamic range or the absolute log-error can also be used to develop confidence intervals for the maximum and median of the distribution.

Lemma 4.3.4. *Let* $\hat{m}(x)$ *be an approximation of* $m(x)$ *with* $\log d\left(\hat{m}/m\right) \leq \epsilon$*, so that*

$$\hat{m}^+(x) = \exp(2\epsilon)\hat{m}(x) \qquad\qquad \hat{m}^-(x) = \exp(-2\epsilon)\hat{m}(x)$$

are upper and lower pointwise bounds on $m(x)$*, respectively. Then we have a confidence region on the maximum of* $m(x)$ *given by*

$$\arg\max_x m(x) \in \{x : \hat{m}^+(x) \geq \max_y \hat{m}^-(y)\}$$

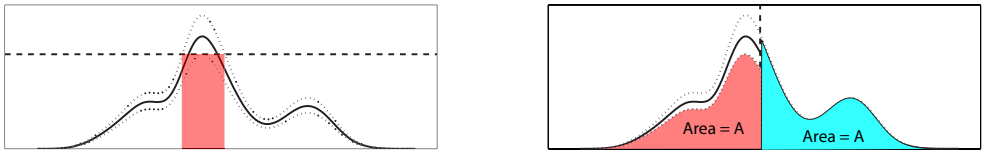
and an upper bound μ *on the median of* $m(x)$*, i.e. ,*

$$\int_{-\infty}^{\mu} m(x) \geq \int_{\mu}^{\infty} m(x) \qquad \textit{where} \qquad \int_{-\infty}^{\mu} \hat{m}^-(x) = \int_{\mu}^{\infty} \hat{m}^+(x)$$

with a similar lower bound.

Proof. The definitions of $\hat{m}^+$ and $\hat{m}^-$ follow from Theorem 4.3.1. Given these bounds, the maximum value of $m(x)$ must be larger than the maximum value of $\hat{m}^-(x)$, and this is only possible at locations x for which $\hat{m}^+(x)$ is also greater than the maximum of $\hat{m}^-$. Similarly, the left integral of $m(x)$ ($-\infty$ to μ) must be larger than the integral of $\hat{m}^-(x)$, while the right integral (μ to ∞) must be smaller than for $\hat{m}^+(x)$. Thus the median of $m(x)$ must be less than μ. □

These bounds and confidence intervals are illustrated in Figure 4.2: given the approximate message $\hat{m}$ (solid black), a bound on the error yields $\hat{m}^+(x)$ and $\hat{m}^-(x)$ (dotted lines), which yield confidence regions on the maximum and median values of $m(x)$.



Conf. Region on Maximum
(a)

(Right boundary of) Conf. Region on Median
(b)

Figure 4.2. Using the error measure (4.4) to find confidence regions on maximum and median locations of a distribution. The distribution estimate $\hat{m}(x)$ is shown in solid black, with $|\log m(x)/\hat{m}(x)| \leq \frac{1}{4}$ bounds shown as dotted lines. Then, the maximum value of $m(x)$ must lie above the shaded region, and the median value is less than the dashed vertical line; a similar computation gives a lower bound.

■ 4.3.2 Additivity and Error Contraction

We now turn to the properties of our dynamic range measure with respect to the operations of belief propagation. First, we consider the error resulting from taking the product (4.1) of a number of incoming approximate messages.

Theorem 4.3.2. *The log of the dynamic range measure is sub-additive:*

$$\log d\left(E^i_{ts}\right) \leq \sum_{u \in \Gamma_t \setminus s} \log d\left(e^i_{ut}\right) \qquad \log d\left(E^i_{t}\right) \leq \sum_{u \in \Gamma_t} \log d\left(e^i_{ut}\right)$$

Proof. We show the left-hand sub-additivity statement; the right follows from a similar argument. By definition, we have

$$\log d\left(E^i_{ts}\right) = \log d\left(\hat{M}^i_{ts}/M^i_{ts}\right) = \frac{1}{2} \log \sup_{a,b} \prod e^i_{ut}(a) / \prod e^i_{ut}(b)$$

Increasing the number of degrees of freedom gives

$$\leq \frac{1}{2} \log \prod \sup_{a_u, b_u} e^i_{ut}(a_u)/e^i_{ut}(b_u) = \sum \log d\left(e^i_{ut}(x)\right) \square$$

Theorem 4.3.2 allows us to bound the error resulting from a combination of the incoming approximations from two different neighbors of the node t. It is also important that $\log d\,(e)$ satisfy the triangle inequality, so that the application of two successive approximations results in an error which is bounded by the sum of their respective errors.

Theorem 4.3.3. *The log of the dynamic range measure satisfies the triangle inequality:*

$$\log d\,(e_1 e_2) \leq \log d\,(e_1) + \log d\,(e_2)$$

Proof. This follows from the same argument as Theorem 4.3.2. □

We may also derive a minimum rate of contraction occurring with the convolution operation (4.2). We characterize the strength of the potential ψ_{ts} by extending the definition of the dynamic range measure:

$$d\left(\psi_{ts}\right)^2 = \sup_{a,b,c,d} \frac{\psi_{ts}(a,b)}{\psi_{ts}(c,d)} \tag{4.5}$$

When this quantity is finite, it represents a minimum rate of *mixing* for the potential, and thus causes a contraction on the error. This fact is exhibited in the following theorem:

Theorem 4.3.4. *When $d\left(\psi_{ts}\right)$ is finite, the dynamic range measure satisfies a rate of contraction:*

$$d\left(e_{ts}^{i+1}\right) \leq \frac{d\left(\psi_{ts}\right)^2 d\left(E_{ts}^{i}\right) + 1}{d\left(\psi_{ts}\right)^2 + d\left(E_{ts}^{i}\right)}. \tag{4.6}$$

Proof. See Appendix 4.7. □

Two limits are of interest. First, if we examine the limit as the potential strength $d\left(\psi\right)$ grows, we see that the error cannot increase due to convolution with the pairwise potential ψ. Similarly, if the potential strength is finite, the outgoing error cannot be arbitrarily large (independent of the size of the incoming error).

Corollary 4.3.1. *The outgoing message error $d\left(e_{ts}\right)$ is bounded by*

$$d\left(e_{ts}^{i+1}\right) \leq d\left(E_{ts}^{i}\right) \qquad\qquad d\left(e_{ts}^{i+1}\right) \leq d\left(\psi_{ts}\right)^2$$

Proof. Let $d\left(\psi_{ts}\right)$ or $d\left(E_{ts}^{i}\right)$ tend to infinity in Theorem 4.3.4. □

The contractive bound (4.6) is shown in Figure 4.3, along with the two simpler bounds of Corollary 4.3.1, shown as straight lines. Moreover, we may evaluate the asymptotic behavior by considering the derivative

$$\frac{\partial}{\partial d\left(E\right)} \left. \frac{d\left(\psi\right)^2 d\left(E\right) + 1}{d\left(E\right) + d\left(\psi\right)^2} \right|_{d(E)\to 1} = \frac{d\left(\psi\right)^2 - 1}{d\left(\psi\right)^2 + 1} = \tanh(\log d\left(\psi\right))$$

The limits of this bound are quite intuitive: for $\log d\left(\psi\right) = 0$ (independence of x_t and x_s), this derivative is zero; increasing the error in incoming messages m_{ut}^{i} has no effect on the error in m_{ts}^{i+1}. For $d\left(\psi\right) \to \infty$, the derivative approaches unity, indicating that for very large $d\left(\psi\right)$ (strong potentials) the propagated error can be nearly unchanged.

We may apply these bounds to investigate the behavior of BP in graphs with cycles. We begin by examining loopy belief propagation with exact messages, using the previous results to derive a new sufficient condition for BP convergence to a unique fixed point. When this condition is not satisfied, we instead obtain a bound on the relative distances between any two fixed points of the loopy BP equations. This allows us to consider the effect of introducing additional errors into the messages passed at each iteration, showing sufficient conditions for this operation to converge, and a bound on the resulting error from exact loopy BP.

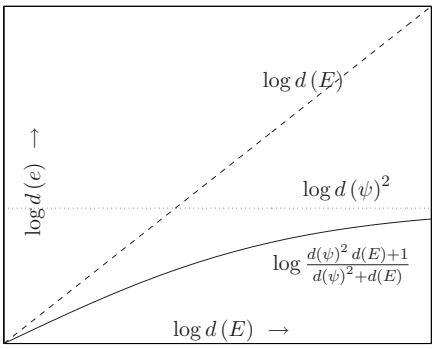


Figure 4.3. Three bounds on the error output $d(e)$ as a function of the error on the product of incoming messages $d(E)$.

■ 4.4 Applying Dynamic Range to Graphs with Cycles

In this section, we apply the framework developed in Section 4.3, along with the computation tree formalism of [95], to derive results on the behavior of traditional belief propagation (in which messages and potentials are represented exactly). We then use the same methodology to analyze the behavior of loopy BP for quantized or otherwise approximated messages and potential functions.

■ 4.4.1 Convergence of Loopy Belief Propagation

The work of [95] showed that the convergence and fixed points of loopy BP may be considered in terms of a Gibbs measure on the graph's computation tree. In particular, this led to the result that loopy BP is guaranteed to converge if the graph satisfies Dobrushin's condition [27]. Dobrushin's condition is a global measure, and difficult to verify; given in [95] is the easier to check sufficient condition (often called Simon's condition),

Theorem 4.4.1 (Simon's condition). *Loopy belief propagation is guaranteed to converge if*

$$\max_t \sum_{u \in \Gamma_t} \log d(\psi_{ut}) < 1 \tag{4.7}$$

where $d(\psi)$ *is defined as in* (4.5).

Proof. See [95]. □

Using the previous section's analysis, we obtain the following, stronger condition, and (after the proof) show analytically how the two are related.

Theorem 4.4.2 (BP convergence). *Loopy belief propagation is guaranteed to converge if*

$$\max_{(s,t)\in\mathcal{E}} \sum_{u\in\Gamma_t\setminus s} \frac{d\left(\psi_{ut}\right)^2 - 1}{d\left(\psi_{ut}\right)^2 + 1} < 1 \tag{4.8}$$

Proof. By induction. Let the "true" messages m_{ts} be any fixed point of BP, and consider the incoming error observed by a node t at level $n-1$ of the computation tree (corresponding to the first iteration of BP), and having parent node s. Suppose that the total incoming error $\log d\left(E_{ts}^1\right)$ is bounded above by some constant $\log \epsilon^1$ for all $(t,s) \in \mathcal{E}$. Note that this is trivially true (for any n) for the constant $\log \epsilon^1 = \max_t \sum_{u\in\Gamma_t} \log d\left(\psi_{ut}\right)^2$, since the error on any message m_{ut} is bounded above by $d\left(\psi_{ut}\right)^2$.

Now, assume that $\log d\left(E_{ut}^i\right) \leq \log \epsilon^i$ for all $(u,t) \in \mathcal{E}$. Theorem 4.3.4 bounds the maximum log-error $\log d\left(E_{ts}^{i+1}\right)$ at any replica of node t with parent s, where s is on level $n-i$ of the tree (which corresponds to the i^{th} iteration of loopy BP) by

$$\log d\left(E_{ts}^{i+1}\right) \leq g_{ts}(\log \epsilon^i) = G_{ts}(\epsilon^i) = \sum_{u\in\Gamma_t\setminus s} \log \frac{d\left(\psi_{ut}\right)^2 \epsilon^i + 1}{d\left(\psi_{ut}\right)^2 + \epsilon^i} \tag{4.9}$$

We observe a contraction of the error between iterations i and $i+1$ if the bound $g_{ts}(\log \epsilon^i)$ is smaller than $\log \epsilon^i$ for every $(t,s) \in \mathcal{E}$, and asymptotically achieve $\log \epsilon^i \to 0$ if this is the case for any value of $\epsilon^i > 1$.

Defining $z = \log \epsilon$, we may equivalently show $g_{ts}(z) < z$ for all $z > 0$. This can be guaranteed by the conditions $g_{ts}(0) = 0$, $g'_{ts}(0) < 1$, and $g''_{ts}(z) \leq 0$ for each t, s. The first is easy to verify, as is the last (term by term) using the identity $g''_{ts}(z) = \epsilon^2 G''_{ts}(\epsilon) + \epsilon G'_{ts}(\epsilon)$; the second ($g'_{ts}(0) < 1$) can be rewritten to give the convergence condition (4.8). □

We may relate Theorem 4.4.2 to Simon's condition by expanding the set $\Gamma_t \setminus s$ to the larger set Γ_t, and observing that $\log x \geq \frac{x^2-1}{x^2+1}$ for all $x \geq 1$ with equality as $x \to 1$. Doing so, we see that Simon's condition is sufficient to guarantee Theorem 4.4.2, but that Theorem 4.4.2 may be true (implying convergence) when Simon's condition is not satisfied. The improvement over Simon's condition becomes negligible for highly-connected systems with weak potentials, but can be significant for graphs with low connectivity. For example, if the graph consists of a single loop then each node t has at most two neighbors. In this case, the contraction (4.9) tells us that the outgoing message in either direction is *always* as close or closer to the BP fixed point than the incoming message. Thus we easily obtain the result of [105], that (for finite-strength potentials) BP always converges to a unique fixed point on graphs containing a single loop. Simon's condition, on the other hand, is too loose to demonstrate this fact. The form of the condition in Theorem 4.4.2 is also similar to a result shown for binary spin models; see [27] for details.

We provide a more empirical comparison between our condition, Simon's condition, and the recent work of [37] shortly. Similarly to [37], we shall see that it is possible to use the graph geometry to improve our bound (Section 4.4.3); but perhaps more importantly (and in contrast to both other methods), when the condition is *not* satisfied, we still obtain useful information about the relationship between any pair of fixed points (Section 4.4.2), allowing its extension to quantized or otherwise distorted versions of belief propagation (Section 4.4.4).

■ 4.4.2 Distance of multiple fixed points

Theorem 4.4.2 may be extended to provide not only a sufficient condition for a unique BP fixed point, but an upper bound on distance between the beliefs generated by successive BP updates and any BP fixed point. Specifically, the proof of Theorem 4.4.2 relied on demonstrating a bound $\log \epsilon^i$ on the distance from some arbitrarily chosen fixed point $\{M_t\}$ at iteration i. When this bound decreases to zero, we may conclude that only one fixed point exists. However, even should it decrease only to some positive constant, it still provides information about the distance between any iteration's belief and the fixed point. Moreover, applying this bound to another, different fixed point $\{\tilde{M}_t\}$ tells us that all fixed points of loopy BP must lie within a sphere of a given diameter (as measured by $\log d\left(M_t/\tilde{M}_t\right)$). These statements are made precise in the following two theorems:

Theorem 4.4.3 (BP distance bound). *Let $\{M_t\}$ be any fixed point of loopy BP. Then, after $n > 1$ iterations of loopy BP resulting in beliefs $\{\hat{M}_t^n\}$, for any node t and for all x*

$$\log d\left(M_t/\hat{M}_t^n\right) \leq \sum_{u\in\Gamma_t} \log \frac{d\left(\psi_{ut}\right)^2 \epsilon^{n-1} + 1}{d\left(\psi_{ut}\right)^2 + \epsilon^{n-1}}$$

where ϵ^i is given by $\epsilon^1 = \max_{s,t} d\left(\psi_{st}\right)^2$ and

$$\log \epsilon^{i+1} = \max_{(s,t)\in\mathcal{E}} \sum_{u\in\Gamma_t\setminus s} \log \frac{d\left(\psi_{ut}\right)^2 \epsilon^i + 1}{d\left(\psi_{ut}\right)^2 + \epsilon^i}$$

Proof. The result follows directly from the proof of Theorem 4.4.2. □

We may thus infer a distance bound between any two BP fixed points:

Theorem 4.4.4 (Fixed-point distance bound). *Let $\{M_t\}$, $\{\tilde{M}_t\}$ be the beliefs of any two fixed points of loopy BP. Then, for any node t and for all x*

$$\left|\log M_t(x)/\tilde{M}_t(x)\right| \leq 2\log d\left(M_t/\tilde{M}_t\right) \leq 2\sum_{u\in\Gamma_t} \log \frac{d\left(\psi_{ut}\right)^2 \epsilon + 1}{d\left(\psi_{ut}\right)^2 + \epsilon} \tag{4.10}$$

where ϵ is the largest value satisfying

$$\log \epsilon = \max_{(s,t)\in\mathcal{E}} G_{ts}(\epsilon) = \max_{(s,t)\in\mathcal{E}} \sum_{u\in\Gamma_t\setminus s} \log \frac{d\left(\psi_{ut}\right)^2 \epsilon + 1}{d\left(\psi_{ut}\right)^2 + \epsilon} \tag{4.11}$$

Proof. The inequality $|\log M_t(x)/\tilde{M}_t(x)| \leq 2\log d\left(M_t/\tilde{M}_t\right)$ follows directly from Theorem 4.3.1. The rest follows from Theorem 4.4.3—taking the "approximate" messages to be any other fixed point of loopy BP, we see that the error cannot decrease over any number of iterations. However, by the same argument given in Theorem 4.4.2, $g''_{ts}(z) < 0$, and for z sufficiently large, $g_{ts}(z) < z$. Thus (4.11) has at most one solution greater than unity, and $\epsilon^{i+1} < \epsilon^i$ for all i with $\epsilon^i \to \epsilon$ as $i \to \infty$. Letting the number of iterations $i \to \infty$, we see that the message "errors" $\log d\left(M_{ts}/\tilde{M}_{ts}\right)$ must be at most ϵ, and thus the difference in M_t (the belief of the root node of the computation tree) must satisfy (4.10). □

Thus, if the value of $\log \epsilon$ is small (the sufficient condition of Theorem 4.4.2 is nearly satisfied) then although we cannot guarantee convergence to a unique fixed point, we can still make a strong statement: that the set of fixed points are all mutually close (in a log-error sense), and reside within a ball of diameter described by (4.10). Moreover, even though it is possible that loopy BP does not converge, and thus even after infinite time the messages may not correspond to *any* fixed point of the BP equations, we are guaranteed by Theorem 4.4.3 that the resulting belief estimates *will* asymptotically approach the same bounding ball (achieving distance at most (4.10) from *all* fixed points).

■ 4.4.3 Path-counting

If we are willing to put a bit more effort into our bound-computation, we may be able to improve it further, since the bounds derived using computation trees are very much "worst-case" bounds. In particular, the proof of Theorem 4.4.2 assumes that, as a message error propagates through the graph, repeated convolution with *only* the strongest set of potentials is possible. But often even if the worst potentials are quite strong, every cycle which contains them may also contain several weaker potentials. Using an iterative algorithm much like belief propagation itself, we may obtain a more globally aware estimate of how errors can propagate through the graph.

Theorem 4.4.5 (Non-uniform distance bound). *Let $\{M_t\}$ be any fixed point belief of loopy BP. Then, after $n \geq 1$ iterations of loopy BP resulting in beliefs $\{\hat{M}_t^n\}$, for any node t and for all x*

$$|\log M_t(x)/\hat{M}_t^n(x)| \leq 2\log d\left(M_t/\hat{M}_t^n\right) \leq 2\sum_{u\in\Gamma_t} \log \upsilon_{ut}^n$$

where υ_{ut}^i *is defined by the iteration*

$$\log \upsilon_{ts}^{i+1} = \log \frac{d\left(\psi_{ts}\right)^2 \epsilon_{ts}^i + 1}{d\left(\psi_{ts}\right)^2 + \epsilon_{ts}^i} \qquad \log \epsilon_{ts}^i = \sum_{u \in \Gamma_t \setminus s} \log \upsilon_{ut}^i \tag{4.12}$$

with initial condition $\upsilon_{ut}^1 = d\left(\psi_{ut}\right)^2$.

Proof. Again we consider the error $\log d\left(E_{ts}^i\right)$ incoming to node t with parent s, where t is at level $n-i+1$ of the computation tree. Using the same arguments as Theorem 4.4.2 it is easy to show by induction that the error products $\log d\left(E_{ts}^i\right)$ are bounded above by ϵ_{ts}^i, and the individual message errors $\log d\left(e_{ts}^i\right)$ are bounded above by υ_{ts}^i, and . Then, by additivity we obtain the stated bound on $d\left(E_t^n\right)$ at the root node. □

The iteration defined in Theorem 4.4.5 can also be interpreted as a (scalar) message-passing procedure, or may be performed offline. As before, if this procedure results in $\log \epsilon_{ts} \to 0$ for all $(t,s) \in \mathcal{E}$ we are guaranteed that there is a unique fixed point for loopy BP; if not, we again obtain a bound on the distance between any two fixed-point beliefs. When the graph is perfectly symmetric (every node has identical neighbors and potential strengths), this yields the same bound as Theorem 4.4.3; however, if the potential strengths are inhomogeneous Theorem 4.4.5 provides a strictly better bound on loopy BP convergence and errors.

This situation is illustrated in Figure 4.4—we specify two different graphical models defined on a 5×5 grid in terms of their potential strengths $\log d\left(\psi\right)^2$, and compute bounds on the dynamic range $d\left(M_t / \tilde{M}_t\right)$ of any two fixed point beliefs M_t, $\tilde{M}_t$ for each model. (Note that, while potential strength does not completely specify the graphical model, it is sufficient for all the bounds considered here.) One grid (a) has equal-strength potentials $\log d\left(\psi\right)^2 = \omega$, while the other has many weaker potentials $(\omega/2)$. The worst-case bounds are the same (since both have a node with four strong neighbors), shown as the solid curve in (c). However, the dashed curves show the estimate of (4.12), which improves only slightly for the strongly coupled graph (a) but considerably for the weaker graph (b). All three bounds give considerably more information than Simon's condition (dotted vertical line).

Having shown how our bound may be improved for irregular graph geometry, we may now compare our bounds to two other known uniqueness conditions [37, 95]. First, for certain special cases such as graphs with binary-valued state and pairwise potentials, Simon's condition can be further strengthened by a factor of two [27, 37]. Thus for these special cases, Simon's condition may give more information than the one presented here. Additionally, the recent work of [37] takes a very different approach to uniqueness based on analysis of the minima of the Bethe free energy, which directly correspond to stable fixed points of BP [112]. This leads to an alternate sufficient condition for uniqueness. As observed in [37] it is unclear whether a unique fixed point necessarily implies convergence of loopy BP. In contrast, our approach gives a sufficient condition

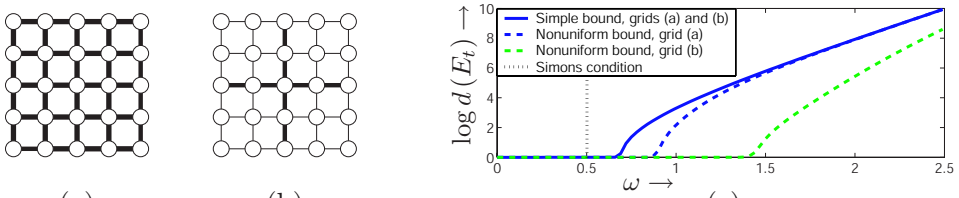
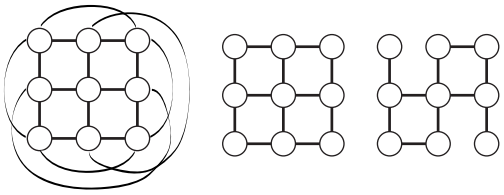


Figure 4.4. (a-b) Two small (5×5) grids. In (a), the potentials are all of equal strength ($\log d(\psi)^2 = \omega$), while in (b) several potentials (thin lines) are weaker ($\log d(\psi)^2 = .5\omega$). The methods described may be used to compute bounds (c) on the distance $d(E_t)$ between any two fixed point beliefs as a function of potential strength ω.



Method	(a)	(b)	(c)
Simon's condition, [95]	.56	.56	.56
(Improved for binary)	.62	.62	.62
Heskes' condition, [37]	.55	.58	.65
This work	.59	.69	.76
Empirical	.67	.79	.88

η_{crit}

Figure 4.5. Comparison of various BP uniqueness bounds. For binary potentials parameterized by η, we find the predicted η_{crit} at which a fixed point of loopy BP can no longer be guaranteed to be unique. For these simple problems, the η_{crit} at which the trivial (correct) solution becomes unstable may be found empirically. Examples and empirical values of η_{crit} from [37].

for the convergence of BP to a unique solution, which implies uniqueness of the fixed point.

Showing an analytic relation between all three approaches does not appear straightforward; to give some intuition, we show the three example binary graphs compared in [37], whose structures are shown in Figure 4.5(a-c) and whose potentials are parameterized by a scalar $\eta > .5$, namely

$$\psi = \begin{bmatrix} \eta & 1-\eta \\ 1-\eta & \eta \end{bmatrix}$$

(so that $d(\psi)^2 = \frac{\eta}{1-\eta}$). The trivial solution $M_t = [.5; .5]$ is always a fixed point, but may not be stable; the precise η_{crit} at which this fixed point becomes unstable (implying the existence of other, stable fixed points) can be found empirically for each case [37]; the same values may also be found algebraically by imposing symmetry requirements on the messages [112]. This value may then be compared to the uniqueness bounds of [95], its strengthened version for binary potentials, the bound of [37], and this work; these are shown in Figure 4.5.

While the strengthened version of Simon's condition exceeds the performance of our condition for the perfectly symmetric case of Figure 4.5(a), as the problem becomes more asymmetric (and tree-like), methods which account for the graph structure begin

to perform better. Notice that our bound is always better than the unstrengthened (typical) version of Simon's condition, though for the perfectly symmetric graph the margin is not large (and decreases further with increased connectivity, for example a cubic lattice). Additionally, in all three examples our method appears to outperform that of [37], though without analytic comparison it is unclear whether this is always the case. None of the sufficient conditions manage to approach the empirical η_{crit} and are thus clearly not necessary conditions for convergence.

On this last point, however, our method also allows the bounds to be generalized to situations in which we are not interested in true uniqueness of the fixed point, but rather only equivalence of all fixed points up to some numerical precision. For example, we may find the η_{crit} below which all pairs of fixed points $\{M_t\}$, $\{\tilde{M}_t\}$ satisfy $\log d\left(M_t/\tilde{M}_t\right) < 10^{-3}$, obtaining the values $\{.65, .76, .85\}$ for the grids in Figure 4.5(a), (b), and (c), respectively.

■ 4.4.4 Introducing intentional message errors and censoring

As discussed in the introduction, we may wish to introduce or allow *additional* errors in our messages at each stage, in order to improve the computational or communication efficiency of the algorithm. This may be the result of an actual distortion imposed on the message (perhaps to decrease its complexity, for example quantization), or the result of censoring the message update (reusing the message from the previous iteration) when the two are sufficiently similar. Errors may also arise from quantization or other approximation of the potential functions. Such additional errors may be easily incorporated into our framework.

Theorem 4.4.6. *If at every iteration of loopy BP, each message is further approximated in such a way as to guarantee that the additional distortion has maximum dynamic range at most δ, then for any fixed point beliefs $\{M_t\}$, after $n \geq 1$ iterations of loopy BP resulting in beliefs $\{\hat{M}_t^n\}$ we have*

$$\log d\left(M_t/\hat{M}_t^n\right) \leq \sum_{u \in \Gamma_t} \log \upsilon_{ut}^n$$

where υ_{ut}^i is defined by the iteration

$$\log \upsilon_{ts}^{i+1} = \log \frac{d\left(\psi_{ts}\right)^2 \epsilon_{ts}^i + 1}{d\left(\psi_{ts}\right)^2 + \epsilon_{ts}^i} + \log \delta \qquad \log \epsilon_{ts}^i = \sum_{u \in \Gamma_t \setminus s} \log \upsilon_{ut}^i$$

with initial condition $\upsilon_{ut}^1 = \delta\, d\left(\psi_{ut}\right)^2$.

Proof. Using the same logic as Theorems 4.4.3 and 4.4.5, apply additivity of the log dynamic range measure to the additional distortion $\log \delta$ introduced to each message. □

As with Theorem 4.4.5, a simpler bound can also be derived (similar to Theorem 4.4.3). Either gives a bound on the maximum total distortion from any true fixed point which will be incurred by quantized or censored belief propagation. Note that (except on tree-structured graphs) this does *not* bound the error from the true marginal distributions, only from the loopy BP fixed points.

It is also possible to interpret the additional error as arising from an approximation to the correct single-node and pairwise potentials ψ_t, ψ_{ts}.

Theorem 4.4.7. *Suppose that $\{M_t\}$ are a fixed point of loopy BP on a graph defined by potentials ψ_{ts} and ψ_t, and let $\{\hat{M}_t^n\}$ be the beliefs of n iterations of loopy BP performed on a graph with potentials $\hat{\psi}_{ts}$ and $\hat{\psi}_t$, where $d\left(\hat{\psi}_{ts}/\psi_{ts}\right) \leq \delta_1$ and $d\left(\hat{\psi}_t/\psi_t\right) \leq \delta_2$. Then,*

$$\log d\left(M_t/\hat{M}_t^n\right) \leq \sum_{u \in \Gamma_t} \log \upsilon_{ut}^n + \log \delta_2$$

where υ_{ut}^i is defined by the iteration

$$\log \upsilon_{ts}^{i+1} = \log \frac{d\left(\psi_{ts}\right)^2 \epsilon_{ts}^i + 1}{d\left(\psi_{ts}\right)^2 + \epsilon_{ts}^i} + \log \delta_1 \qquad \log \epsilon_{ts}^i = \log \delta_2 + \sum_{u \in \Gamma_t \setminus s} \log \upsilon_{ut}^i$$

with initial condition $\upsilon_{ut}^1 = \delta_1 \, d\left(\psi_{ut}\right)^2$.

Proof. We first extend the contraction result given in Appendix 4.7 by applying the inequality

$$\frac{\int \psi(x_t, a) \frac{\hat{\psi}(x_t,a)}{\psi(x_t,a)} M(x_t) E(x_t) dx_t}{\int \psi(x_t, b) \frac{\hat{\psi}(x_t,b)}{\psi(x_t,b)} M(x_t) E(x_t) dx_t} \leq \frac{\int \psi(x_t, a) M(x_t) E(x_t) dx_t}{\int \psi(x_t, b) M(x_t) E(x_t) dx_t} \cdot d\left(\hat{\psi}/\psi\right)^2$$

Then, proceeding similarly to Theorem 4.4.6 yields the definition of υ_{ts}^i, and including the additional errors $\log \delta_2$ in each message product (resulting from the product with $\hat{\psi}_t$ rather than ψ_t) gives the definition of ϵ_{ts}^i. □

Incorrect models $\hat{\psi}$ may arise when the exact graph potentials have been estimated or quantized; Theorem 4.4.7 gives us the means to interpret the (worst-case) overall effects of using an approximate model. As an example, let us again consider the model depicted in Figure 4.5(b). Suppose that we are given *quantized* versions of the pairwise potentials, $\hat{\psi}$, specified by the value (rounded to two decimal places) $\eta = .65$. Then, the true potential ψ has $\eta \in .65 \pm .005$, and thus is within $\delta_1 \approx 1.022 = \frac{(.35)(.655)}{(.345)(.65)}$ of the known approximation $\hat{\psi}$. Applying the recursion of Theorem 4.4.7 allows us to conclude that the solution obtained using the approximate model $\hat{\psi}$ and true model ψ are within $\log d\left(e\right) \leq .36$, or alternatively that the beliefs found using the approximate model are correct to within a multiplicative factor of about 1.43. The same $\hat{\psi}$, with η assumed correct to three decimal places, gives a bound $\log d\left(e\right) \leq .04$, or multiplicative factor of 1.04.

■ 4.4.5 Stochastic Analysis

Unfortunately, the bounds given by Theorem 4.4.7 are often pessimistic compared to actual performance. We may use a similar analysis, coupled with the assumption of uncorrelated message errors, to obtain a more realistic estimate (though no longer a strict bound) on the resulting error.

Proposition 4.4.1. *Suppose that the errors* $\log e_{ts}$ *are random and uncorrelated, so that at each iteration* i*, for* $s \neq u$ *and any* x*,* $\mathrm{E}\left[\log e^i_{st}(x) \cdot \log e^i_{ut}(x)\right] = 0$*, and that at each iteration of loopy BP, the additional error (in the log domain) imposed on each message is uncorrelated with variance at most* $(\log \delta)^2$*. Then,*

$$\mathrm{E}\left[\left(\log d\left(E^i_t\right)\right)^2\right] \leq \sum_{u \in \Gamma_t} \left(\sigma^i_{ut}\right)^2 \tag{4.13}$$

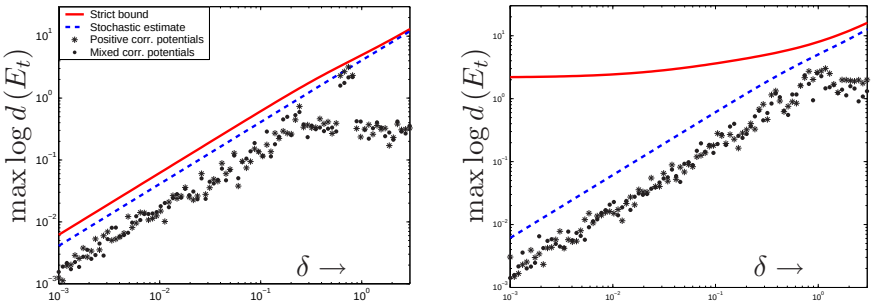
where $\sigma^1_{ts} = \log d\left(\psi_{ts}\right)^2$ *and*

$$\left(\sigma^{i+1}_{ts}\right)^2 = \left(\log \frac{d\left(\psi_{ts}\right)^2 \lambda^i_{ts} + 1}{d\left(\psi_{ts}\right)^2 + \lambda^i_{ts}}\right)^2 + (\log \delta)^2 \qquad \left(\log \lambda^i_{ts}\right)^2 = \sum_{u \in \Gamma_t \setminus s} \left(\sigma^i_{ut}\right)^2$$

Proof. Let us define the (nuisance) scale factor $\alpha^i_{ts} = \arg\min_\alpha \sup_x |\log \alpha e^i_{ts}(x)|$ for each error e^i_{ts}, and let $\zeta^i_{ts}(x) = \log \alpha^i_{ts} e^i_{ts}(x)$. Now, we model the error function $\zeta^i_{ts}(x)$ (for each x) as a random variable with mean zero, and bound the standard deviation of $\zeta^i_{ts}(x)$ by σ^i_{ts} at each iteration i; under the assumption that the errors in any two incoming messages are uncorrelated, we may assert additivity of their variances. Thus the variance of $\sum_{\Gamma_t \setminus s} \zeta^i_{ut}(x)$ is bounded by $(\log \lambda^i_{ts})^2$. The contraction of Theorem 4.3.4 is a non-linear relationship; we estimate its effect on the error variance using a simple sigma-point quadrature ("unscented") approximation [54], in which the standard deviation σ^{i+1}_{ts} is estimated by applying Theorem 4.3.4's nonlinear contraction to the standard deviation of the error on the incoming product $(\log \lambda^i_{ts})$. □

The assumption of uncorrelated errors is clearly questionable, since propagation around loops may couple the incoming message errors. However, similar assumptions have yielded useful analysis of quantization effects in assessing the behavior and stability of digital filters [109]. It is often the case that empirically, such systems behave similarly to the predictions made by assuming uncorrelated errors. Indeed, we shall see that in our simulations, the assumption of uncorrelated errors provides a good estimate of performance.

Given the bound (4.13) on the variance of $\log d\left(E\right)$, we may apply a Chebyshev-like argument to provide probabilistic guarantees on the magnitude of errors $\log d\left(E\right)$ observed in practice. In our experiments (Section 4.4.6), the 2σ distance was almost always larger than the observed error. The probabilistic bound derived using (4.13) is typically much smaller than the bound of Theorem 4.4.6 due to the strictly sub-additive relationship between the standard deviations. However, the underlying assumption of



(a) $\log d(\psi)^2 = .25$ (b) $\log d(\psi)^2 = 1$

Figure 4.6. Maximum belief errors incurred as a function of the quantization error. The scatterplot indicates the maximum error measured in the graph for each of 200 Monte Carlo runs; this is strictly bounded above by Theorem 4.4.6, solid, and bounded with high probability (assuming uncorrelated errors) by Proposition 4.4.1, dashed.

.

uncorrelated errors makes the estimate obtained using (4.13) unsuitable for deriving strict convergence guarantees.

■ 4.4.6 Experiments

We demonstrate the dynamic range error bounds for quantized messages with a set of Monte Carlo trials. In particular, for each trial we construct a binary-valued 5×5 grid with uniform potential strengths, which are either (1) all positively correlated, or (2) randomly chosen to be positively or negatively correlated (equally likely); we also assign random single-node potentials to each variable x_s. We then run a quantized version of BP, rounding each log-message to discrete values separated by $2 \log \delta$ (ensuring that the newly introduced error satisifies $d(e) \leq \delta$). Figure 4.6 shows the maximum belief error in each of 100 trials of this procedure for various values of δ.

Also shown are two performance estimators—the *bound* on belief error developed in Section 4.4.4, and the 2σ estimate computed assuming uncorrelated message errors as in Section 4.4.5. As can be seen, the stochastic estimate is a much tighter, more accurate assessment of error, but it does not possess the same strong theoretical guarantees. Since (as observed for digital filtering applications [109]) the errors introduced by quantization are typically close to independent, the assumptions underlying the stochastic estimate are reasonable, and empirically we observe that the estimate and actual errors behave similarly.

■ 4.5 KL-Divergence Measures

Although the dynamic range measure introduced in Section 4.3 leads to a number of strong guarantees, its performance criterion may be unnecessarily (and undesirably) strict. Specifically, it provides a *pointwise* guarantee, that m and $\hat{m}$ are close for every

possible state x. For continuous-valued states, this is an extremely difficult criterion to meet—for instance, it requires that the messages' tails match almost exactly. In contrast, typical measures of the difference between two distributions operate by an average (mean squared error or mean absolute error) or weighted average (Kullback-Leibler divergence) evaluation. To address this, let us consider applying a measure such as the Kullback-Leibler (KL) divergence,

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

The pointwise guarantees of Section 4.3 are necessary to bound performance even in the case of "unlikely" events. More specifically, the tails of a message approximation can become important if two parts of the graph strongly disagree, in which case the tails of each message are the only overlap of significant likelihood. One way to discount this possibility is to consider the graph potentials themselves (in particular, the single node potentials ψ_t) as a realization of random variables which "typically" agree, then apply a probabilistic measure to estimate the typical performance. From this viewpoint, since a strong disagreement between parts of the graph is unlikely we will be able to relax our error measure in the message tails.

Unfortunately, many of the properties which we relied on for analysis of the dynamic range measure do not strictly hold for a KL-divergence measure of error, resulting in an *approximation*, rather than a bound, on performance. In Appendix 4.8, we give a detailed analysis of each property, showing the ways in which each aspect can break down and discussing the reasonability of simple approximations. In this section, we apply these approximations to develop a KL-divergence based estimate of error.

■ 4.5.1 Local Observations and Parameterization

To make this notion concrete, let us consider a graphical model in which the single-node potential functions are specified in terms of a set of observation variables $\mathbf{y} = \{y_t\}$; in this section we will examine the average (expected) behavior of BP over multiple realizations of the observation variables $\mathbf{y}$. We further assume that both the prior $p(\mathbf{x})$ and likelihood $p(\mathbf{y}|\mathbf{x})$ exhibit conditional independence structure, expressed as a graphical model. Specifically, we assume throughout this section that the observation likelihood factors as

$$p(\mathbf{y}|\mathbf{x}) = \prod_t p(y_t|x_t) \tag{4.14}$$

in other words, that each observation variable y_t is *local* to (conditionally independent given) one of the x_t. As for the prior model $p(\mathbf{x})$, for the moment we confine our attention to tree-structured distributions, for which one may write [100]

$$p(\mathbf{x}) = \prod_{(s,t)\in\mathcal{E}} \frac{p(x_s, x_t)}{p(x_s)p(x_t)} \prod_s p(x_s) \tag{4.15}$$

The expressions (4.14)-(4.15) give rise to a convenient parameterization of the joint distribution, expressed as

$$p(\mathbf{x}, \mathbf{y}) \propto \prod_{(s,t)\in\mathcal{E}} \psi_{st}(x_s, x_t) \prod_s \psi_s^x(x_s)\psi_s^y(x_s) \tag{4.16}$$

where

$$\psi_{st}(x_s, x_t) = \frac{p(x_s, x_t)}{p(x_s)p(x_t)} \quad \text{and} \quad \psi_s^x(x_s) = p(x_s) \quad , \quad \psi_s^y(x_s) = p(y_s|x_s). \tag{4.17}$$

Our goal is to compute the posterior marginal distributions $p(x_s|\mathbf{y})$ at each node s; for the tree-structured distribution (4.16) this can be performed exactly and efficiently by BP. As discussed in the previous section, we treat the $\{y_t\}$ as random variables; thus almost all quantities in this graph are themselves random variables (as they are dependent on the y_t), so that the single node observation potentials $\psi_s^y(x_s)$, messages $m_{st}(x_t)$, *etc.* are random functions of their argument x_s. The potentials due to the prior (ψ_{st} and ψ_s^x), however, are not random variables as they do not depend on any of the observations y_t.

For models of the form (4.16)-(4.17), the (unique) BP message fixed point consists of normalized versions of the likelihood functions $m_{ts}(x_s) \propto p(\mathbf{y}_{ts}|x_s)$, where $\mathbf{y}_{ts}$ denotes the set of all observations $\{y_u\}$ such that t separates u from s. In this section it is also convenient to perform a *prior-weighted* normalization of the messages m_{ts}, so that $\int p(x_s)m_{ts}(x_s) = 1$ (as opposed to $\int m_{ts}(x_s) = 1$ as assumed previously); we again assume this prior-weighted normalization is always possible (this is trivially the case for discrete-valued states $\mathbf{x}$). Then, for a tree-structured graph, the prior-weight normalized fixed-point message from t to s is precisely

$$m_{ts}(x_s) = p(\mathbf{y}_{ts}|x_s)/p(\mathbf{y}_{ts}) \tag{4.18}$$

and the products of incoming messages to t, as defined in Section 4.1, are equal to

$$M_{ts}(x_t) = p(x_t|\mathbf{y}_{ts}) \qquad\qquad M_t(x_t) = p(x_t|\mathbf{y}).$$

We may now apply a *posterior-weighted log-error* measure, defined by

$$\mathcal{D}(m_{ut}\|\hat{m}_{ut}) = \int p(x_t|\mathbf{y}) \log \frac{m_{ut}(x_t)}{\hat{m}_{ut}(x_t)} dx_t; \tag{4.19}$$

and may relate (4.19) to the Kullback-Leibler divergence.

Lemma 4.5.1. *On a tree-structured graph, the error measure $\mathcal{D}(M_t, \hat{M}_t)$ is equivalent to the KL-divergence of the true and estimated posterior distributions at node t:*

$$\mathcal{D}(M_t\|\hat{M}_t) = D(p(x_t|\mathbf{y})\|\hat{p}(x_t|\mathbf{y}))$$

Proof. This follows directly from the definitions of $\mathcal{D}$, and the fact that on a tree, the unique fixed point has beliefs $M_t(x_t) = p(x_t|\mathbf{y})$. □

Again, the error $\mathcal{D}(m_{ut}\|\hat{m}_{ut})$ is a function of the observations $\mathbf{y}$, both explicitly through the term $p(x_t|\mathbf{y})$ and implicitly through the message $m_{ut}(x_t)$, and is thus also a random variable. Although the definition of $\mathcal{D}(m_{ut}\|\hat{m}_{ut})$ involves the *global* observation $\mathbf{y}$ and thus cannot be calculated at node u without additional (non-local) information, we will primarily be interested in the expected value of these errors over many realizations $\mathbf{y}$, which is a function only of the distribution. Specifically, we can see that in expectation over the data $\mathbf{y}$, it is simply

$$\mathrm{E}\left[\mathcal{D}(m_{ut}\|\hat{m}_{ut})\right] = \mathrm{E}\left[\int p(x_t)m_{ut}(x_t)\log\frac{m_{ut}(x_t)}{\hat{m}_{ut}(x_t)}dx_t\right]. \tag{4.20}$$

One nice consequence of the choice of potential functions (4.17) is the locality of prior information. Specifically, if *no* observations $\mathbf{y}$ are available, and only prior information is present, the BP messages are trivially constant ($m_{ut}(x) = 1\ \forall x$). This ensures that any message approximations affect only the data likelihood, and not the prior $p(x_t)$; this is similar to the motivation of [77], in which an additional message-passing procedure is used to create this parameterization.

Finally, two special cases are of note. First, if x_s is discrete-valued and the prior distribution $p(x_s)$ constant (uniform), the expected message distortion with prior-normalized messages, $\mathrm{E}[\mathcal{D}(m\|\hat{m})]$, and the KL-divergence of traditionally normalized messages behave equivalently, i.e.,

$$\mathrm{E}\left[\mathcal{D}(m_{ts}\|\hat{m}_{ts})\right] = \mathrm{E}\left[D\left(\frac{m_{ts}}{\int m_{ts}}\Big\|\frac{\hat{m}_{ts}}{\int \hat{m}_{ts}}\right)\right]$$

where we have abused the notation of KL-divergence slightly to apply it to the normalized likelihood $m_{ts}/\int m_{ts}$. This interpretation leads to the same message-censoring criterion used in [10].

Secondly, when the state x_s is a discrete-valued random variable taking on one of M possible values, a straightforward uniform quantization of the value of $p(x_s)m(x_s)$ results in a bound on the divergence (4.20). Specifically, we have the following lemma:

Lemma 4.5.2. *For an M-ary discrete variable x, the quantization*

$$p(x)m(x) \to \{\epsilon, 3\epsilon, \ldots, 1-\epsilon\}$$

results in an expected divergence bounded by

$$\mathrm{E}\left[\mathcal{D}(m(x)\|\hat{m}(x))\right] \leq (2\log 2 + M)M\epsilon + \mathcal{O}(M^3\epsilon^2)$$

Proof. Define $\mu(x) = p(x)m(x)$, and $\bar{\mu}(x) \in \{\epsilon, 3\epsilon, \ldots, 1-\epsilon\}$ (for each x) to be its quantized value. Then, the prior-normalized approximation $\hat{m}(x)$ satisfies

$$p(x)\hat{m}(x) = \bar{\mu}(x)\ /\ \sum_x \bar{\mu}(x) = \bar{\mu}(x)/C$$

where $C \in [1 - M\epsilon, 1 + M\epsilon]$. The expected divergence

$$\begin{aligned} \mathrm{E}\left[\mathcal{D}(m(x)\|\hat{m}(x))\right] &= \sum_x p(x)m(x)\log\frac{m(x)}{\hat{m}(x)} \\ &\leq \sum_x \mu(x)\log\frac{\mu(x)}{\bar{\mu}(x)} + \sum_x |\log C| \end{aligned}$$

The first sum is at its maximum for $\mu(x) = 2\epsilon$ and $\bar{\mu}(x) = \epsilon$, which results in the value $\sum_x (2\log 2)\epsilon$. Applying the Taylor expansion of the log, the second sum $\sum |\log C|$ is bounded above by $M^2\epsilon + \mathcal{O}(M^3\epsilon^2)$. □

Thus, for example, for uniform quantization of a message with binary-valued state x, fidelity up to two significant digits ($\epsilon = .005$) results in an error $\mathcal{D}$ which, on average, is less than .034.

We now state the approximations which will take the place of the fundamental properties used in the preceding sections, specifically versions of the triangle inequality, sub-additivity, and contraction. Although these properties do *not* hold in general, in practice useful estimates are obtained by making approximations corresponding to each property and following the same development used in the preceding sections. (In fact, experimentally these estimates still appear quite conservative.) A more detailed analysis of each property, along with justification for the approximation applied, is given in Appendix 4.8.

■ 4.5.2 Approximations

Three properties of the dynamic range described in Section 4.3 are important in the error analysis of Section 4.4—(1) a form of the triangle inequality, enabling the accumulation of errors in successive approximations to be bounded by the sum of the individual errors, (2) a form of sub-additivity, enabling the accumulation of errors in the message product operation to be bounded by the sum of incoming errors, and (3) a rate of contraction due to convolution with each pairwise potential. We assume the following three properties for the expected error; see Appendix 4.8 for a more detailed discussion.

Approximation 4.5.1 (Triangle Inequality). *For a true BP fixed-point message m_{ut} and two approximations $\hat{m}_{ut}$, $\tilde{m}_{ut}$, we assume*

$$\mathcal{D}(m_{ut}\|\tilde{m}_{ut}) \leq \mathcal{D}(m_{ut}\|\hat{m}_{ut}) + \mathcal{D}(\hat{m}_{ut}\|\tilde{m}_{ut}) \tag{4.21}$$

Comment. This is not strictly true for arbitrary $\hat{m}$, $\tilde{m}$, since the KL-divergence (and thus $\mathcal{D}$) does not satisfy the triangle inequality.

Approximation 4.5.2 (Sub-additivity). *For true BP fixed-point messages $\{m_{ut}\}$ and approximations $\{\hat{m}_{ut}\}$, we assume*

$$\mathcal{D}(M_{ts}\|\hat{M}_{ts}) \leq \sum_{u\in\Gamma_t\setminus s} \mathcal{D}(m_{ut}\|\hat{m}_{ut}) \tag{4.22}$$

Approximation 4.5.3 (Contraction). *For a true BP fixed-point message product M_{ts} and approximation $\hat{M}_{ts}$, we assume*

$$\mathcal{D}(m_{ts}\|\hat{m}_{ts}) \leq (1-\gamma_{ts})\mathcal{D}(M_{ts}\|\hat{M}_{ts}) \tag{4.23}$$

where

$$\gamma_{ts} = \min_{a,b} \int \min\left[\rho(x_s, x_t = a)\,,\, \rho(x_s, x_t = b)\right] dx_s \quad \rho(x_s, x_t) = \frac{\psi_{ts}(x_s, x_t)\psi_s^x(x_s)}{\int \psi_{ts}(x_s, x_t)\psi_s^x(x_s)dx_s}$$

Comment. For tree-structured graphical models with the parametrization described by (4.16)-(4.17), $\rho(x_s, x_t) = p(x_s|x_t)$, and γ_{ts} corresponds to the rate of contraction described by [6].

■ 4.5.3 Steady-state errors

Applying these approximations to graphs with cycles, and following the same development used for constructing the strict bounds of Section 4.4, we find the following estimates of steady-state error. Note that, other than those outlined in the previous section (and described in Appendix 4.8), this development involves no additional approximations.

Approximation 4.5.4. *After $n \geq 1$ iterations of loopy BP subject to additional errors at each iteration of magnitude (measured by $\mathcal{D}$) bounded above by some constant δ, with initial messages $\{m_{tu}^0\}$ satisfying $\mathcal{D}(m_{tu}\|m_{tu}^0)$ less than some constant C, results in an expected KL-divergence between a true BP fixed point $\{M_t\}$ and the approximation $\{\hat{M}_t^n\}$ bounded by*

$$\mathrm{E}_{\mathbf{y}}\left[D(M_t\|\hat{M}_t^n)\right] = \mathrm{E}_{\mathbf{y}}\left[\mathcal{D}(\mathcal{M}_t\|\hat{\mathcal{M}}_t^n)\right] \leq \sum_{u\in\Gamma_t}((1-\gamma_{ut})\epsilon_{ut}^{n-1} + \delta)$$

where $\epsilon_{ts}^0 = C$ and

$$\epsilon_{ts}^i = \sum_{u\in\Gamma_t\setminus s}((1-\gamma_{ut})\epsilon_{ut}^{i-1} + \delta)$$

Comment. The argument proceeds similarly to that of Theorem 4.4.6. Let ϵ_{ts}^i bound the quantity $\mathcal{D}(\mathcal{M}_{ts}\|\hat{\mathcal{M}}_{ts}^i)$ at each iteration i, and apply Approximations 4.5.1-4.5.3.

We refer to the estimate described in Approximation 4.5.4 as a "bound-approximation", in order to differentiate it from the stochastic error estimate presented next.

Just as a stochastic analysis of message error gave a tighter estimate for the pointwise difference measure, we may obtain an alternate Chebyshev-like "bound" by assuming that the message perturbations are uncorrelated (already an assumption of the KL additivity analysis) and that we require only an estimate which exceeds the expected error with high probability.

Approximation 4.5.5. *Under the same assumptions as Approximation 4.5.4, but describing the error in terms of its variance and assuming that these errors are uncorrelated gives the estimate*

$$\mathrm{E}\left[\mathcal{D}(\mathcal{M}_t \| \hat{\mathcal{M}}_t^n)^2\right] \leq \sum_{u \in \Gamma_t} (\sigma_{ut}^{n-1})^2$$

where $(\sigma_{ts}^0)^2 = C$ *and*

$$(\sigma_{ts}^i)^2 = \sum_{u \in \Gamma_t \setminus s} ((1 - \gamma_{ut})\sigma_{ut}^{i-1})^2 + \delta^2$$

Comment. The argument proceeds similarly to Proposition 4.4.1, by induction on the claim that $(\sigma_{ut}^i)^2$ bounds the variance at each iteration i. This again applies Theorem 4.8.3 ignoring any effects due to loops, as well as the assumption that the message errors are uncorrelated (implying additivity of the variances of each incoming message). As in Section 4.4.5, we take the 2σ value as our performance estimate.

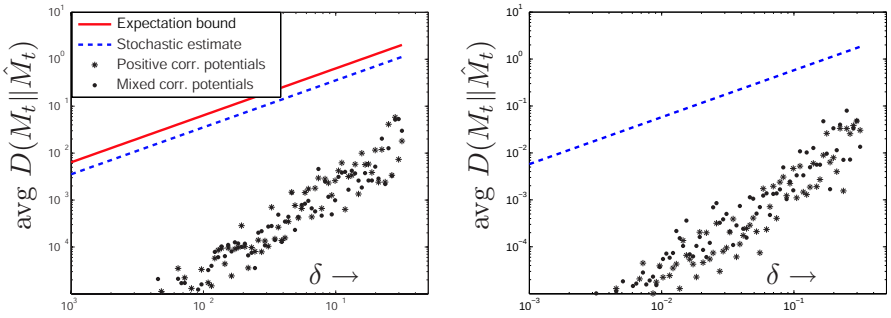
■ 4.5.4 Experiments

Once again, we demonstrate the utility of these two estimates on the same uniform grids used in Section 4.4.6. Specifically, we generate 200 example realizations of a 5×5 binary grid and its observation potentials (100 with strictly attractive potentials and 100 with mixed potentials), and compare a quantized version of loopy BP with the solution obtained by exact loopy BP, as a function of KL-divergence bound δ incurred by the quantization level ϵ (see Lemma 4.5.1).

Figure 4.7(a) shows the maximum KL-divergence from the correct fixed point resulting in each Monte Carlo trial for a grid with relatively weak potentials (in which loopy BP is analytically guaranteed to converge). As can be seen, both the bound (solid) and stochastic estimate (dashed) still provide conservative estimates of the expected error. In Figure 4.7(b) we repeat the same analysis but with stronger pairwise potentials (for which convergence is not guaranteed but occurs in practice). In this case, the bound-based estimate of KL-divergence is trivially infinite—its linear rate of contraction is insufficient to overcome the accumulation rate. However, the greater sub-additivity in the stochastic estimate leads to the non-trivial curve shown (dashed), which still provides a reasonable (and still conservative) estimate of the performance in practice.

■ 4.6 Discussion

We have described a framework for the analysis of belief propagation stemming from the view that the message at each iteration is some noisy or erroneous version of some true BP fixed point. By measuring and bounding the error at each iteration, we may analyze the behavior of various forms of BP and test for convergence to the ideal fixed-point messages, or bound the total error from any such fixed point.



(a) $\log d\,(\psi)^2 = .25$ (b) $\log d\,(\psi)^2 = 1$

Figure 4.7. KL-divergence of the beliefs as a function of the added message error δ. The scatterplots indicates the average error measured in the graph for each of 200 Monte Carlo runs, along with the expected divergence bound (solid) and 2σ stochastic estimate (dashed). For stronger potentials, the upper bound may be trivially infinite; in this example the stochastic estimate still gives a reasonable gauge of performance.

.

In order to do so, we introduced a measure of the pointwise dynamic range, which represents a strong condition on the agreement between two messages; after showing its utility for common inference tasks such as MAP estimation and its transference to other common measures of error, we showed that under this measure the influence of message errors is both sub-additive and measurably contractive. These facts led to conditions under which traditional belief propagation may be shown to converge to a unique fixed point, and more generally a bound on the distance between any two fixed points. Furthermore, it enabled analysis of quantized, stochastic, or other approximate forms of belief propagation, yielding conditions under which they may be guaranteed to converge to some unique region, as well as bounds on the ensuing error over exact BP. If we further assume that the message perturbations are uncorrelated, we obtain an alternate, tighter estimate of the resulting error.

The second measure considered an average case error similar to the Kullback-Liebler divergence, in expectation over the possible realizations of observations within the graph. While this gives no guarantees about any particular realization, the difference measure itself is able to be much less strict (allowing poor approximations in the distribution tails, for example). Analysis of this case is substantially more difficult and leads to approximations rather than guarantees, but explains some of the observed similarities in behavior among the two forms of perturbed BP. Simulations indicate that these estimates remain sufficiently accurate to be useful in practice.

Further analysis of the propagation of message errors has the potential to give an improved understanding of when and why BP converges (or fails to converge), and potentially also the role of the message schedule in determining the performance. Additionally, there are many other possible measures of the deviation between two messages, any of which may be able to provide an alternative set of bounds and estimates on per-

formance of BP using either exact or approximate messages.

■ 4.7 Proof of Theorem 4.3.4

Because all quantities in this section refer to the pair (t, s), we suppress the subscripts. The error measure $d(e)$ is given by

$$d(e)^2 = d(\hat{m}/m)^2 = \max_{a,b} \frac{\int \psi(x_t, a) M(x_t) E(x_t) dx_t}{\int \psi(x_t, a) M(x_t) dx_t} \cdot \frac{\int \psi(x_t, b) M(x_t) dx_t}{\int \psi(x_t, b) M(x_t) E(x_t) dx_t} \quad (4.24)$$

subject to a few constraints: positivity of the messages and potential functions, normalization of the message product M, and the definitions of $d(E)$ and $d(\psi)$. In order to analyze the maximum possible value of $d(e)$ for any functions ψ, M, and E, we make repeated use of the following property:

Lemma 4.7.1. *For f_1, f_2, g_1, g_2 all positive,*

$$\frac{f_1 + f_2}{g_1 + g_2} \leq \max \left[\frac{f_1}{g_1}, \frac{f_2}{g_2} \right]$$

Proof. Assume without loss of generality that $f_1/g_1 \geq f_2/g_2$. Then we have $f_1/g_1 \geq f_2/g_2 \Rightarrow f_1 g_2 \geq f_2 g_1 \Rightarrow f_1 g_1 + f_1 g_2 \geq f_1 g_1 + f_2 g_1 \Rightarrow \frac{f_1}{g_1} \geq \frac{f_1 + f_2}{g_1 + g_2}$. □

This fact, extended to more general sums, may be applied directly to (4.24) to prove Corollary 4.3.1. However, a more careful application leads to the result of Theorem 4.3.4. The following lemma will assist us:

Lemma 4.7.2. *The maximum of $d(e)$ with respect to $\psi(x_t, a)$, $\psi(x_t, b)$, and $E(x_t)$ is attained at some extremum of their feasible function space. Specifically,*

$$\psi(x, a) = 1 + (d(\psi)^2 - 1)\chi_A(x) \qquad E(x) = 1 + (d(E)^2 - 1)\chi_E(x)$$
$$\psi(x, b) = 1 + (d(\psi)^2 - 1)\chi_B(x)$$

where χ_A, χ_B, and χ_E are indicator functions taking on only values 0 and 1.

Proof. We simply show the result for $\psi(x, a)$; the proofs for $\psi(x, b)$ and $E(x)$ are similar. First, observe that without loss of generality we may scale $\psi(x, a)$ so that its minimum value is 1. Now consider a convex combination of any two possible functions: let $\psi(x_t, a) = \alpha_1 \psi_1(x_t, a) + \alpha_2 \psi_2(x_t, a)$ with $\alpha_1 \geq 0$, $\alpha_2 \geq 0$, and $\alpha_1 + \alpha_2 = 1$. Then, applying Lemma 4.7.1 to the left-hand term of (4.24) we have

$$\frac{\alpha_1 \int \psi_1(x_t, a) M(x_t) E(x_t) dx_t + \alpha_2 \int \psi_2(x_t, a) M(x_t) E(x_t) dx_t}{\alpha_1 \int \psi_1(x_t, a) M(x_t) dx_t + \alpha_2 \int \psi_2(x_t, a) M(x_t) dx_t}$$
$$\leq \max \left[\frac{\int \psi_1(x_t, a) M(x_t) E(x_t) dx_t}{\int \psi_1(x_t, a) M(x_t) dx_t}, \frac{\int \psi_2(x_t, a) M(x_t) E(x_t) dx_t}{\int \psi_2(x_t, a) M(x_t) dx_t} \right] \quad (4.25)$$

Thus, $d(e)$ is maximized by taking whichever of ψ_1, ψ_2 results in the largest value—an extremum. It remains only to describe the form of such a function extremum. Any potential $\psi(x,a)$ may be considered to be the convex combination of functions of the form $\left(d(\psi)^2 - 1\right)\chi(x) + 1$, where χ takes on values $\{0,1\}$. This can be seen by the construction

$$\psi(x,a) = \int_0^1 \left(d(\psi)^2 - 1\right)\chi_m^y(x,a) + 1 \; dy$$

where

$$\chi_m^y(x,a) = \begin{cases} 1 & \psi(x,a) \geq 1 + (d(\psi)^2 - 1)\, y \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the maximum value of $d(e)$ will be attained by a potential equal to one of these functions. □

Applying Lemma 4.7.2, we define the shorthand

$$M_A = \int M(x)\chi_A(x) \qquad M_B = \int M(x)\chi_B(x) \qquad M_E = \int M(x)\chi_E(x)$$
$$M_{AE} = \int M(x)\chi_A(x)\chi_E(x) \quad M_{BE} = \int M(x)\chi_B(x)\chi_E(x)$$
$$\alpha = d(\psi)^2 - 1 \qquad \beta = d(E)^2 - 1$$

giving

$$d(e)^2 \leq \max_M \; \frac{1 + \alpha M_A + \beta M_E + \alpha\beta M_{AE}}{1 + \alpha M_B + \beta M_E + \alpha\beta M_{BE}} \cdot \frac{1 + \alpha M_B}{1 + \alpha M_A}$$

Using the same argument outlined by Equation 4.25, one may argue that the scalars M_{AE}, M_{BE}, M_A, and M_B must also be extremum of their constraint sets. Noticing that M_{AE} should be large and M_{BE} small, we may summarize the constraints by

$$0 \leq M_A,\ M_B,\ M_E \leq 1 \qquad M_{AE} \leq \min[M_A,\ M_E]$$
$$M_{BE} \geq \max[0,\ M_E - (1 - M_B)]$$

(where the last constraint arises from the fact that $M_E + M_B - M_{BE} \leq 1$). We then consider each possible case: $M_A \leq M_E$, $M_A \geq M_E$, ... In each case, we find that the maximum is found at the extrema $M_{AE} = M_A = M_E$ and $M_E = 1 - M_B$. This gives

$$d(e)^2 \leq \max_M \; \frac{1 + (\alpha + \beta + \alpha\beta)M_E}{1 + \alpha + (\beta - \alpha)M_E} \cdot \frac{1 + \alpha - \alpha M_E}{1 + \alpha M_E}$$

The maximum with respect to M_E (whose optimum is not an extreme point) is given by taking the derivative and setting it to zero. This procedure gives a quadratic equation;

solving and selecting the positive solution gives $M_E = \frac{1}{\beta}(\sqrt{\beta+1}-1)$. Finally, plugging in, simplifying, and taking the square root yields

$$d(e) \leq \frac{d(\psi)^2 d(E) + 1}{d(\psi)^2 + d(E)} \qquad \square$$

■ 4.8 Properties of the Expected Divergence

We begin by examining the properties of the expected divergence (4.20) on tree–structured graphical models parameterized by (4.16)–(4.17); we discuss the application of these results to graphs with cycles in Appendix 4.8.4. Recall that, for tree-structured models described by (4.16)–(4.17), the prior-weight normalized messages of the (unique) fixed point are equivalent to

$$m_{ut}(x_t) = p(\mathbf{y}_{ut}|x_t)/p(\mathbf{y}_{ut}).$$

and that the message products are given by

$$M_{ts}(x_t) = p(x_t|\mathbf{y}_{ts})M_t(x_t) = p(x_t|\mathbf{y})$$

Furthermore, let us define the *approximate* messages $\hat{m}_{ut}(x)$ in terms of some approximate likelihood function, i.e.,

$$\hat{m}_{ut}(x) = \hat{p}(\mathbf{y}_{ut}|x_t)/\hat{p}(\mathbf{y}_{ut}) \qquad \text{where} \qquad \hat{p}(\mathbf{y}_{ut}) = \int \hat{p}(\mathbf{y}_{ut}|x_t)p(x_t)dx_t.$$

We may then examine each of the three properties in turn: the triangle inequality, additivity, and contraction.

■ 4.8.1 Triangle Inequality

Kullback-Leibler divergence is not a true distance, and in general, it does not satisfy the triangle inequality. However, the following generalization does hold:

Theorem 4.8.1. *For a tree-structured graphical model parameterized as in* (4.16)–(4.17)*, and given the true BP message* $m_{ut}(x_t)$ *and two approximate messages* $\hat{m}_{ut}(x_t)$*,* $\tilde{m}_{ut}(x_t)$*, suppose that* $m_{ut}(x_t) \leq c_{ut}\hat{m}_{ut}(x_t)\ \forall x_t$*. Then,*

$$\mathcal{D}(m_{ut}\|\tilde{m}_{ut}) \leq \mathcal{D}(m_{ut}\|\hat{m}_{ut}) + c_{ut}\mathcal{D}(\hat{m}_{ut}\|\tilde{m}_{ut})$$

and furthermore, if $\hat{m}_{ut}(x_t) \leq c^*_{ut}\tilde{m}_{ut}(x_t)\ \forall x_t$*, then* $m_{ut}(x_t) \leq c_{ut}c^*_{ut}\tilde{m}_{ut}(x_t)\ \forall x_t$*.*

Comment. Since $m, \hat{m}$ are prior-weight normalized ($\int p(x)m(x) = \int p(x)\hat{m}(x) = 1$), for a strictly positive prior $p(x)$ we see that $c_{ut} \geq 1$, with equality if and only if $m_{ut}(x) = \hat{m}_{ut}(x)\ \forall x$. However, this is often quite conservative and Approximation 4.5.1 ($c_{ut} = 1$) is sufficient to estimate the resulting error. Moreover, we shall see that the constants $\{c_{ut}\}$ are also affected by the product operation, described next.

■ 4.8.2 Near-Additivity

For BP fixed-point messages $\{m_{ut}(x_t)\}$, approximated by the messages $\{\hat{m}_{ut}(x_t)\}$, the resulting error is not quite guaranteed to be sub-additive, but is almost so.

Theorem 4.8.2. *The expected error* $\mathrm{E}[\mathcal{D}(M_t\|\hat{M}_t)]$ *between the true and approximate beliefs is nearly sub-additive; specifically,*

$$\mathrm{E}\left[\mathcal{D}(M_t\|\hat{M}_t)\right] \leq \sum_{u\in\Gamma_t} \mathrm{E}\left[\mathcal{D}(m_{ut}\|\hat{m}_{ut})\right] + \left(\hat{\mathrm{I}} - \mathrm{I}\right) \tag{4.26}$$

$$\textit{where} \qquad \mathrm{I} = \mathrm{E}\left[\log\ p(\mathbf{y}) / \prod_{u\in\Gamma_t} p(\mathbf{y}_{ut})\right] \qquad \textit{and} \qquad \hat{\mathrm{I}} = \mathrm{E}\left[\log\ \hat{p}(\mathbf{y}) / \prod_{u\in\Gamma_t} \hat{p}(\mathbf{y}_{ut})\right]$$

Moreover, if $m_{ut}(x_t) \leq c_{ut}\hat{m}_{ut}(x_t)$ *for all* x_t *and for each* $u \in \Gamma_t$*, then*

$$M_t(x_t) \leq \prod_{u\in\Gamma_t} c_{ut} C_t^* \hat{M}_t(x_t) \qquad\qquad C_t^* = \frac{\hat{p}(\mathbf{y})}{\prod_{u\in\Gamma_t}\hat{p}(\mathbf{y}_{ut})} \frac{\prod_{u\in\Gamma_t} p(\mathbf{y}_{ut})}{p(\mathbf{y})} \tag{4.27}$$

Proof. By definition we have

$$\begin{aligned}
\mathrm{E}[\mathcal{D}(M_t\|\hat{M}_t)] &= \mathrm{E}\left[\int p(x_t,\mathbf{y}) \log \frac{M_t(x_t)}{\hat{M}_t(x_t)} dx_t\right] \\
&= \mathrm{E}\left[\int p(x_t|\mathbf{y}) \log \frac{p(x_t)}{p(x_t)} \frac{p(\mathbf{y}|x_t)}{\hat{p}(\mathbf{y}|x_t)} \frac{\hat{p}(\mathbf{y})}{p(\mathbf{y})} dx_t\right]
\end{aligned}$$

Using the Markov property of (4.16) to factor $p(\mathbf{y}|x_t)$, we have

$$= \mathrm{E}\left[\int p(x_t|\mathbf{y}) \sum_{u\in\Gamma_t} \log \frac{p(\mathbf{y}_{ut}|x_t)}{\hat{p}(\mathbf{y}_{ut}|x_t)} + p(x_t|\mathbf{y}) \log \frac{\hat{p}(\mathbf{y})}{p(\mathbf{y})} dx_t\right]$$

and, applying the identity $m_{ut}(x_t) = p(\mathbf{y}_{ut}|x_t)/p(\mathbf{y}_{ut})$ gives

$$\begin{aligned}
&= \sum_{u\in\Gamma_t} \mathrm{E}\left[\int p(x_t|\mathbf{y}) \log \frac{m_{ut}(x_t)}{\hat{m}_{ut}(x_t)}\right] + \mathrm{E}\left[\log \frac{\hat{p}(\mathbf{y})}{\prod_u \hat{p}(\mathbf{y}_{ut})} \frac{\prod_u p(\mathbf{y}_{ut})}{p(\mathbf{y})}\right] dx_t \\
&= \sum_{u\in\Gamma_t} \mathrm{E}\left[\mathcal{D}(m_{ut}\|\hat{m}_{ut})\right] + (\hat{\mathrm{I}} - \mathrm{I})
\end{aligned}$$

where $\hat{\mathrm{I}}$, I are as defined. Here, I is the mutual information (the divergence from independence) of the variables $\{\mathbf{y}_{ut}\}_{u\in\Gamma_t}$. Equation (4.27) follows from a similar argument. □

Unfortunately, it is *not* the case that the quantity $\hat{\mathrm{I}} - \mathrm{I}$ must necessarily be less than or equal to zero. To see how it may be positive, consider the following example. Let $x = [x_a, x_b]$ be a two-dimensional binary random variable, and let y_a and y_b be observations of the specified dimension of x. Then, if y_a and y_b are independent ($\mathrm{I} = 0$), the true messages $m_a(x)$ and $m_b(x)$ have a regular structure; in particular, m_a and m_b have the forms $[p_1 p_2 p_1 p_2]$ and $[p_3 p_3 p_4 p_4]$ for some $p_1, \ldots, p_4$. However, we have placed no such requirements on the message *errors* $\hat{m}/m$; they have the potentially arbitrary forms $e_a = [e_1 e_2 e_3 e_4]$, *etc.*. If either message error e_a, e_b does *not* have the same structure as m_a, m_b respectively (even if they are random and independent), then $\hat{\mathrm{I}}$ will in general not be zero. This creates the *appearance* of information between y_a and y_b, and the KL-divergence will not be strictly sub-additive.

However, this is not a typical situation. One may argue that in most problems of interest, the information I between observations is non-zero, and the types of message perturbations (particularly random errors, such as appear in stochastic versions of BP [48, 56, 93]) tend to degrade this information on average. Thus, is is reasonable to assume that $\hat{\mathrm{I}} \leq \mathrm{I}$.

A similar quantity defines the multiplicative constant C_t^* in (4.27). When $C_t^* \leq 1$, it acts to reduce the constant which bounds M_t by $\hat{M}_t$; if this occurs "typically", it lends additional support for Approximation (4.5.1). Moreover, if $\mathrm{E}[C_t^*] \leq 1$, then by Jensen's inequality, we have $\hat{\mathrm{I}} - \mathrm{I} \leq 0$, ensuring sub-additivity as well.

■ 4.8.3 Contraction

Analysis of the contraction of expected KL-divergence is also non-trivial; however, the work of [6] has already considered this problem in some depth for the specific case of directed Markov chains (in which additivity issues do not arise) and projection-based approximations (for which KL-divergence does satisfy a form of the triangle inequality). We may directly apply their findings to construct Approximation 4.5.3.

Theorem 4.8.3. *On a tree-structured graphical model parameterized as in* (4.16)-(4.17)*, the error measure* $\mathcal{D}(M, \hat{M})$ *satisfies the inequality*

$$\mathrm{E}\left[\mathcal{D}(m_{ts} \| \hat{m}_{ts})\right] \leq (1 - \gamma_{ts})\, \mathrm{E}\left[\mathcal{D}(M_{ts} \| \hat{M}_{ts})\right]$$

$$\gamma_{ts} = \min_{a,b} \int \min\left[p(x_s | x_t = a)\,,\, p(x_s | x_t = b)\right] dx_s$$

Proof. For a detailed development, see [6]; we merely sketch the proof here. First, note that

$$\begin{aligned}
\mathrm{E}\left[\mathcal{D}(m_{ts} \| \hat{m}_{ts})\right] &= \mathrm{E}\left[\int p(x_s | \mathbf{y}) \log \frac{p(\mathbf{y}_{ts} | x_s)}{p(\mathbf{y}_{ts})} \frac{\hat{p}(\mathbf{y}_{ts})}{\hat{p}(\mathbf{y}_{ts} | x_s)}\right] \\
&= \mathrm{E}\left[\int p(x_s | \mathbf{y}_{ts}) \log \frac{p(x_s | \mathbf{y}_{ts})}{\hat{p}(x_s | \mathbf{y}_{ts})}\right] \\
&= \mathrm{E}\left[D(p(x_s | \mathbf{y}_{ts}) \| \hat{p}(x_s | \mathbf{y}_{ts}))\right]
\end{aligned}$$

(which is the quantity considered in [6]), and that $p(x_s|\mathbf{y}_{ts}) = \int p(x_s|x_t)p(x_t|\mathbf{y}_{ts})dx_t$. By constructing two valid conditional distributions $f_1(x_s|x_t), f_2(x_s|x_t)$ such that f_1 has the form $f_1(x_s|x_t) = f_1(x_s)$ (independence of x_s, x_t), and

$$p(x_s|x_t) = \gamma_{ts} f_1(x_s|x_t) + (1 - \gamma_{ts} f_2(x_s|x_t)$$

one may use the convexity of KL-divergence to show

$$D(p(x_s|\mathbf{y}_{ts})\|\hat{p}(x_s|\mathbf{y}_{ts})) \leq \gamma_{ts} D(f_1 * p(x_t|\mathbf{y}_{ts})\|f_1 * \hat{p}(x_t|\mathbf{y}_{ts})) + (1 - \gamma_{ts}) D(f_2 * p(x_t|\mathbf{y}_{ts})\|f_2 * \hat{p}(x_t|\mathbf{y}_{ts}))$$

where "$*$" denotes convolution, i.e., $f_1 * p(x_t|\mathbf{y}_{ts}) = \int f_1(x_s|x_t)p(x_t|\mathbf{y}_{ts})dx_t$. Since the conditional f_1 induces independence between x_s and x_t, the first divergence term is zero, and since f_2 is a valid conditional distribution, the second divergence is less than $D(p(x_t|\mathbf{y}_{ts})\|\hat{p}(x_t|\mathbf{y}_{ts}))$ (see [15]). Thus we have a minimum rate of contraction of $(1 - \gamma_{ts})$. □

It is worth noting that Theorem 4.8.3 gives a *linear* contraction rate. While this makes for simpler recurrence relations than the nonlinear contraction found in Section 4.3.2, it has the disadvantage that, if the rate of error addition exceeds the rate of contraction it may result in a trivial (infinite) bound. Theorem 4.8.3 is the best contraction rate currently known for arbitrary conditional distributions, although certain special cases (such as binary-valued random variables) appear to admit stronger contractions.

■ 4.8.4 Graphs with Cycles

The analysis and discussion of each property (Appendices 4.8.1- 4.8.3) also relied on assuming a tree-structured graphical model, and using the direct relationship between messages and likelihood functions for the parameterization (4.16)-(4.17). However, for BP on general graphs, this parameterization is not valid.

One way to generalize this choice is given by the re-parameterization around some fixed point of loopy BP on the graphical model of the prior. If the original potentials $\tilde{\psi}_{st}, \tilde{\psi}^x_s$ specify the prior distribution (c.f. (4.17)),

$$p(\mathbf{x}) \propto \prod_{(s,t)\in\mathcal{E}} \tilde{\psi}_{st}(x_s, x_t) \prod_s \tilde{\psi}^x_s(x_s) \tag{4.28}$$

then given a BP fixed point $\{\tilde{M}_{st}, \tilde{M}_s\}$ of (4.28), we may choose a new parameterization of the same prior ψ_{st}, ψ^x_s given by

$$\psi_{st}(x_s, x_t) = \frac{\tilde{M}_{st}(x_s)\tilde{M}_{ts}(x_t)\tilde{\psi}_{st}(x_s, x_t)}{\tilde{M}_s(x_s)\tilde{M}_t(x_t)} \qquad \text{and} \qquad \psi^x_s(x_s) = \tilde{M}_s(x_s) \tag{4.29}$$

This parameterization ensures that uninformative messages ($m_{ut}(x_t) = 1 \ \forall x_t$) comprise a fixed point for the graphical model of $p(\mathbf{x})$ as described by the new potentials $\{\psi_{st}, \psi_s\}$. For a tree-structured graphical model, this recovers the parameterization given by (4.17).

However, the messages of loopy BP are no longer precisely equal to the likelihood functions $m(x) = p(\mathbf{y}|x)/p(\mathbf{y})$, and thus the expectation applied in Theorem 4.8.2 is no longer consistent with the messages themselves. Additionally, the additivity and contraction statements were developed under the assumption that the observed data $\mathbf{y}$ along different branches of the tree are conditionally *independent*; in graphs with cycles, this is not the case. In the computation tree formalism, instead of being conditionally independent, the observations $\mathbf{y}$ actually *repeat* throughout the tree.

However, the assumption of independence is precisely the same assumption applied by loopy belief propagation itself to perform tractable approximate inference. Thus, for problems in which loopy BP is well-behaved and results in answers similar to the true posterior distributions, we may expect our estimates of belief error to be similarly incorrect but near to the true divergence.

In short, all three properties required for a strict analysis of the propagation of errors in BP fail, in one sense or another, for graphs with cycles. However, for many situations of practical interest, they are quite close to the real average-case behavior. Thus we may expect that our approximations give rise to reasonable estimates of the total error incurred by approximate loopy BP, an intuition which appears to be borne out in our simulations (Section 4.5.4).

Chapter 5

Communications Cost of Particle–Based Representations

WE next consider the inherent cost of communicating sample–based representations of distributions, such as arise in distributed implementations of particle filtering (Section 2.7) or nonparametric belief propagation (Chapter 3). In particular, we first examine the cost of optimal *lossless* encoding to transmit a collection of particles exactly, and describe some of the necessary characteristics of good suboptimal encoders. Then, applying the analysis of message error effects from Chapter 4, we consider the problem of *lossy* message encoding. In particular, we describe a method of jointly optimizing over a class of Gaussian mixture models defined on a KD-tree to efficiently trade off between communications cost and several measures of error. We finish with a few simulations which provide examples of the role and performance of lossy encoding for applications in sensor networks.

■ 5.1 Introduction

One of the reasons wireless sensor networks have become so attractive is that they require little or no physical infrastructure, enabling a network to be constructed quickly and inexpensively. However, limited battery life poses a serious difficulty, making efficient use of their finite energy resources one of the most important requirements for a wireless sensor network. The high energy cost of communications, relative to the tasks of computation and sensing, makes it desirable to minimize or limit the required inter-sensor communication in the network.

Unfortunately, reducing communications is often in direct conflict with the primary goal of a sensor network—to accumulate and fuse information from the collection of sensors—by restricting the amount of information which can be broadcast or relayed from each sensor node. To some degree, power may be conserved through intelligent routing of messages or data selection [51, 114]; however, it is also possible to trade off the *fidelity* of the information with its communications cost. This is particularly true for potentially redundant representations—for example, messages created by fine-grain discretization or consisting of large collections of samples. This latter compromise falls into the general category of lossy source coding—that the data may be represented

approximately to fit within some communications budget [28]. However, lossy data compression is generally examined from the perspective of minimizing reconstruction error on the data; in contrast, balancing communications with *inferential* utility (our ability to use the data in subsequent tasks) is comparatively unexplored [25, 97].

In this chapter, we explore the tradeoffs between communication cost and error when the data to be communicated is represented in the form of a distribution or likelihood function. Specifically, we use the analysis of the previous chapter to provide measures of loss which capture not only the distortion on the message itself, but also its impact in terms of the amount of error caused in subsequent inference steps. The tradeoff between these loss measures and communications cost has many similarities to standard density approximation methods. For example, employing a naive characterization of communication cost (number of components in a Gaussian mixture, for example) may lead to a number of common density fitting optimizations, including vector quantization for source coding [28] and "reduced-set" density estimation [30], among others. However, when communication resources are dear, a more careful examination of both error measures and communication cost is warranted.

We begin by outlining the details of our problem framework in Section 5.2. Section 5.3 examines the cost of optimal, lossless encoding of particle– or kernel–based messages, and discusses some necessary features of any good encoder for such messages. In Section 5.4 we describe the role of error measures, such as described in Chapter 4, in lossy encoding of distributions and likelihood functions. Section 5.5 introduces a particular encoding method based on KD-trees which can be applied to both lossless and lossy encoding, and describes an efficient algorithm for balancing the encoding cost with any of the loss measures covered in Section 5.4. Section 5.6 discusses the choice of the resolution parameter β, which we assume fixed in previous sections, and Section 5.7 gives several examples and applications. We conclude with Sections 5.8–5.9, which describe some of the problems which remain open in communications for iterative message–passing algorithms and summarize the contributions of the chapter.

■ 5.2 Problem overview

We frame our analysis by first describing a generic inference problem defined on a small graphical model. Suppose that we have three sensors S_1, S_2, and S_3, each of which observes a local random variable y_1, y_2, y_3, respectively. Each sensor S_k uses y_k to infer about a local hidden state variable (denoted x_k). Our goal is for S_1 to encode and transmit to S_2 its information about y_1 so as to assist in computing the posterior marginal $p(x_2|y_1, y_2, y_3)$. A secondary goal is to allow this information to be passed onward from S_2 to S_3 to assist in computing the posterior marginal of x_3, as well. A graphical model which captures the distribution of the $\{x_k, y_k\}$ is shown in Figure 5.1. We apply a graphical model based description of the problem in order to frame the global inference task in terms of only local information and messages. Local sensing, distributed in-network processing, and limited bandwidth combined with finite energy resources

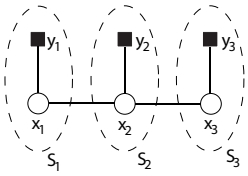


Figure 5.1. A simple yet sufficiently general graphical model description for the transmission problem. A sensor S_1 wishes to send its information y_1 to S_2, who will use it to perform inference on x_2 (or pass it on to S_3).

necessitate a compromise between communication costs and information content.

As in the rest of this thesis, we are concerned with calculating the posterior marginal distributions $p(x_k|\mathbf{y})$. Since our example graph is tree-structured, the task of marginalization may be accomplished using the belief propagation (BP) algorithm (Section 2.6). In regard to Figure 5.1, we analyze the computation of the posterior distribution of x_2:

$$p(x_2|\mathbf{y}) \propto p(x_2)p(y_1|x_2)p(y_2|x_2)p(y_3|x_2).$$

while minimizing communications from S_1, momentarily ignoring the inference tasks of the other sensors. This situation arises, for example, if S_2 were responsible for fusing information or communicating it to an outside user. More symmetric problems, for example when S_2 is also interested in communicating its information to S_1, may be thought of as a generalization of the communication task considered here.

The simple formulation just described can also be easily extended to larger tree–structured graphs, in which case y_1 represents all information separated from sensor S_2 by S_1. Tree structured graphical models have already found application in sensor networks [77]. While certain problems on sensor networks may be described by loopy (non-tree structured) graphical models (for example, the localization problem described in Chapter 6), inference in these situations, and thus the communications/error tradeoff, is considerably more complex and remains a subject of ongoing research. In particular, the error contraction statements described in Chapter 4 are often insufficient to give guarantees about the propagation of errors in inference problems for continuous-valued random variables on loopy graphical models, making it difficult to assess the effects of lossy encoding. Thus for simplicity, in this chapter we concentrate solely on analysis of the communications/error tradeoff in tree-structured graphs.

Describing the inference between the x_k in terms of the messages passed in BP, we assume that the message m_{12} transmitted from S_1 to S_2 is a *function*, and specifically may be either of the local posteriors $p(x_2|y_1)$, $p(x_1|y_1)$ or likelihoods $p(y_1|x_2)$, $p(y_1|x_1)$. We also assume that both S_1 and S_2 share the prior model $p(x_1, x_2)$, in which case all four functions may be considered essentially equivalent, since given any of these functions (and similar information from S_3), it is straightforward for S_2 to calculate $p(x_2|\mathbf{y})$ using Bayes' rule. For concreteness, let us assume that $m_{12} \propto p(y_1|x_2)$, and as is typical normalize each message (function) to integrate to unity for numerical stability.

Although both the transmitter S_1 and receiver S_2 share the joint relationship of

their hidden variables $p(x_1, x_2)$, we assume that the transmitting sensor, S_1, does *not* know the global statistical model. In other words, S_1 does not know the statistics $p(y_2|x_2)$ of S_2's observation, or the statistics $p(x_3, y_3|x_2)$ relating S_3's observation and state to S_2. In order to consider lossless encoding (Section 5.3) we shall assume some global knowledge at the receiver, S_2, but relax this assumption in later sections. We further assume that the communication from transmitter S_1 to receiver S_2 is *open–loop*, i.e., that S_2 provides no feedback or other information to assist S_1 in the encoding and communication process.

A primary assumption of sending a *distribution* message rather than the raw data is that the size of S_1's observation y_1 is larger than the representation size of the likelihood function $p(y_1|x_2)$ (parameterized by x_2). This may be the case for a number of reasons—the observations y_k may be high-dimensional (e.g., high-resolution imagery), or may be a large set of accumulated data (e.g., an entire observation history). The optimal size of a data representation is a statement about the total amount of randomness present, as measured in bits; thus our assumption that $p(y_1|x_2)$ has a smaller representation than y_1 is essentially a statement that some of the uncertainty in y_1 is not relevant to x_2, and that we may reduce costs by being selective about the parts of y_1 communicated. Of course, practically speaking, this tradeoff also involves our ability to encode either the raw observation y_1 or the function $p(y_1|x_2)$ efficiently. The former has been much considered in the source coding literature [28]; here we concentrate on the latter.

■ 5.2.1 Message Representation

There are many possible representations for the inter-sensor messages; common forms include Gaussian distributions, discrete vectors (perhaps resulting from discretization of some complex continuous function), and sample sets. We focus specifically on the latter form. In particular, we assume that each message is described using a kernel density estimate (Section 2.3)

$$m(x) = \sum_i w_i \; K_{h_i}(x - \mu_i) \qquad\qquad K_{h_i}(x - \mu_i) = \mathcal{N}\left(x\,;\, \mu_i, \mathrm{diag}(h_i)\right) \tag{5.1}$$

where the kernel function K is a Gaussian distribution. The bandwidth of K is specified a vector h_i of the same dimension as μ_i and x; the elements of h_i determine the diagonal covariance matrix $\mathrm{diag}(h_i)$ of the Gaussian kernel. While the assumption of a diagonal covariance matrix, and thus vector–valued rather than matrix–valued bandwidth parameter h_i, is not strictly necessary, it is computationally convenient (Section 2.3.1), and furthermore serves to simplify much of the subsequent discussion of *distributions* over the quantities μ_i and h_i.

Gaussian sum–based messages are common in a number of applications. For example, they represent a generalization of the distribution estimates in particle filtering algorithms [3, 19] (in which $h_i = 0$ for all i) and more recently appear in stochastic approximations to belief propagation on general graphical models, such as nonparametric belief propagation (Chapter 3) and the Pampas algorithm [48].

■ 5.3 Lossless Transmission

We begin by examining *lossless* encoding of a kernel density estimate $m(x)$. This means that we would like to transmit an exact copy of the parameters $\{w_i, \mu_i, h_i\}$ from one sensor, S_1, to another, S_2. However, the values of these parameters are continuous, real–valued random variables. Let us therefore instead assume that the parameters $\{w_i, \mu_i, h_i\}$ stored at sensor S_1 have already been quantized to some "very fine" discretization level β, over which we have no control. For example, β might be determined by the resolution, in bits, of sensor S_1's data processing hardware. Lossless encoding means that we transmit the parameters $\{w_i, \mu_i, h_i\}$ up to the specified, arbitrary resolution β without error.

In this section we also assume that the message $m(x)$ from S_1 to S_2 is a simple kernel density estimate as given by (5.1), in which all weights are equal, $w_i = \frac{1}{N}$ for all i, and the samples μ_i are i.i.d. and distributed according to some $p(\mu)$. In Section 5.3.1, to consider the minimal possible cost of communications, we shall assume $p(\mu)$ known at both sender S_1 and receiver S_2. However, this assumption is unrealistic for most situations, and we then discuss some ways in which it can be relaxed in Section 5.3.2. Furthermore, let us assume that the bandwidth h_i has the same value, h, for all i, and that this bandwidth is known at the receiver. The latter requirement can be achieved in any number of ways—the value of h may be deterministically fixed, it may be transmitted separately and its cost neglected for the purpose of our analysis, or it could be chosen automatically from the data $\{\mu_i\}$ in the same manner at both sender and receiver. Finally, the total number of samples, N, is also assumed known at the receiver.

The main consequence of these assumptions is that we may analyze the asymptotic costs involved with the transmission of the collection of i.i.d. samples $\{\mu_i\}$. In particular, we show that the cost of transmitting the *set* $\{\mu_i\}$ is much smaller than the cost of transmitting the *sequence* $[\mu_1, \ldots, \mu_N]$, i.e., that the invariance of our density estimate to reordering of the $\{\mu_i\}$ can lead to significant communication savings.

■ 5.3.1 Optimal Communications

Information theory tells us that the minimum cost of transmitting large volumes of continuous–valued data can be expressed in terms of the data's differential entropy H. We examine the implications of this statement first for the sequence of parameters $[\mu_1, \ldots, \mu_N]$, and then for the set of parameters $\{\mu_i\}$.

Sequence Cost

As discussed in Section 2.2, a *sequence* of N random variables $\boldsymbol{\mu}_N = [\mu_1, \ldots, \mu_N]$ can be sent up to some "high" resolution β, in bits, with expected cost $N\beta + \mathrm{H}(p(\boldsymbol{\mu}_N))$. Of course, for small values of N and β, quantization effects and other factors may influence the actual performance of a source coding scheme [28]. However, for simplicity we focus here on the ideal case.

In particular, this cost is achieved by encoding the symbol μ_i by assigning each possible value of μ_i a codeword whose length is proportional to the negative log–probability of μ_i taking on that value, conditioned on all the information which has been sent so far. Thus, the cost of transmitting the sequence can be decomposed into

$$\beta - \log_2 p(\mu_1) \; + \; \beta - \log_2 p(\mu_2|\mu_1) \; + \ldots + \; \beta - \log_2 p(\mu_N|\mu_{N-1}, \ldots, \mu_1),$$

which, in expectation over the μ_i, is $N\beta + \mathrm{H}(p(\boldsymbol{\mu}_N))$. Since the μ_i are i.i.d. random variables, the conditional distributions simplify, giving

$$\beta - \log_2 p(\mu_1) \; + \ldots + \; \beta - \log_2 p(\mu_N) = N\left(\beta + \mathrm{H}\left(p(\mu_1)\right)\right),$$

or N times the expected cost of sending any one of the μ_i individually.

Set Cost

The problem of communicating a kernel density estimate as in (5.1) is actually considerably simpler—we require only the transmission of the *set* of samples $\{\mu_1, \ldots, \mu_N\}$. In particular, the *ordering* of the data comprises an additional source of uncertainty which we do not require to be transmitted[1]. We can calculate the maximal improvement which may be achieved by analyzing the entropy of the reordered samples.

Let us assume that the resolution β is sufficiently fine that no two samples fall within the same bin (so that all the μ_i differ by more than $2^{-\beta}$). We again denote the complete i.i.d. sample sequence by $\boldsymbol{\mu}_N = [\mu_1, \ldots, \mu_N]$ and its distribution by $p(\boldsymbol{\mu}_N)$. We can then label the resulting samples, which have been deterministically re-ordered (for example, sorted) given the values of the μ_i, by the symbol $\boldsymbol{\mu}^s_N = [\mu_{(1)}, \ldots, \mu_{(N)}]$ and denote the distribution of the full, sorted data by $\rho(\boldsymbol{\mu}_N)$. Throughout this chapter, we will adhere to the convention of using the symbol ρ for distributions of deterministically ordered quantities such as the $\mu_{(i)}$.

It is easy to show [82] that for any deterministic sorting procedure,

$$\rho(\boldsymbol{\mu}_N) = \begin{cases} N!\; p(\boldsymbol{\mu}_N) & \boldsymbol{\mu}_N \text{ “in order”} \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

Essentially, this is because there are $N! = N \cdot (N-1) \cdots 1$ possible values of $\boldsymbol{\mu}_N$ (corresponding to $N!$ possible orderings) which map to the same value of $\boldsymbol{\mu}^s_N$. Thus, the entropy of $\boldsymbol{\mu}^s_N$ is given by

$$\begin{aligned} \mathrm{H}(\rho(\boldsymbol{\mu}_N)) &= -\int \rho(\boldsymbol{\mu}_N) \log_2 \rho(\boldsymbol{\mu}_N) \\ &= -\int \rho(\boldsymbol{\mu}_N) \log_2 p(\boldsymbol{\mu}_N) - \log_2 N! \end{aligned}$$

[1] Interestingly, reordering has also been applied to certain *sequence* coding applications; for example, a reversible reordering procedure is used in the Burrows–Wheeler transform [7] for (discrete–alphabet) source coding to help capture redundancy in non–*i.i.d.* sequences.

and applying (5.2) along with the same counting argument to the integral, we have

$$\mathrm{H}(\rho(\boldsymbol{\mu}_N)) = \mathrm{H}(p(\boldsymbol{\mu}_N)) - \log_2 N! \tag{5.3}$$

indicating savings up to $\log N!$ bits over the cost of sending the sequence naively. Note however that this is no longer accurate if we allow multiple, equal–value (up to the resolution β) samples, since this would result in fewer than $N!$ possible distinct orderings.

Order Statistics

Equation (5.2) is a classic result from the analysis of *order statistics*, defined to be the ascending sorted values for a set of one–dimensional (1-D) random variables. Order statistics provide a natural and well–studied deterministic order in 1-D. We have chosen our notation for the deterministically ordered sequence $\mu_{(i)}$ to be consistent with order statistics; however, the previous analysis does not require any particular method of ordering, and is general to samples of arbitrary dimension.

For 1-D distributions, we have the convenient fact that the optimal conditional distributions $\rho(\mu_{(i)}|\mu_{(i-1)}, \ldots, \mu_{(1)})$ based on an ascending sort are first-order Markov, so that they can be written

$$\rho(\mu_{(i)}|\mu_{(i-1)}, \ldots, \mu_{(1)}) = \rho(\mu_{(i)}|\mu_{(i-1)}).$$

Moreover, these distributions can be computed numerically from the distribution $p(\mu)$ using standard order statistic results [82], via

$$\begin{aligned} p_i^+(\mu) &\propto \begin{cases} 0 & \mu < \mu_{(i)} \\ p(\mu) & \mu \geq \mu_{(i)} \end{cases} \\ P_i^+(\mu) &= \int_{\mu_{(i)}}^{\mu} p_i^+(\bar{\mu})\, d\bar{\mu} \\ \rho(\mu_{(i+1)}|\mu_{(i)}) &\propto \left(1 - P_i^+(\mu_{(i+1)})\right)^{N-i-1} p_i^+(\mu_{(i+1)}) \end{aligned}$$

where the various proportionality constants are chosen to normalize each distribution.

An example illustrating how the sorted samples may be sent using relatively few bits is shown in Figure 5.2. Here, we show five samples distributed uniformly on the interval $[0, 1)$, drawn as arrows. Also shown is the uniform distribution $p(\mu)$ from which they are drawn, shown as the horizontal dotted line; this distribution is optimal for encoding the original, unsorted ordering of the samples. The conditional distributions $\rho(\mu_{(i)}|\mu_{(i-1)})$ for each i, representing the optimal distributions for encoding each sorted sample given those already transmitted, are shown as the dashed lines. Each sorted random variable $\mu_{(i)}$ has much lower entropy than is indicated by $p(\mu)$; this difference translates directly into lower transmission costs for the sample set.

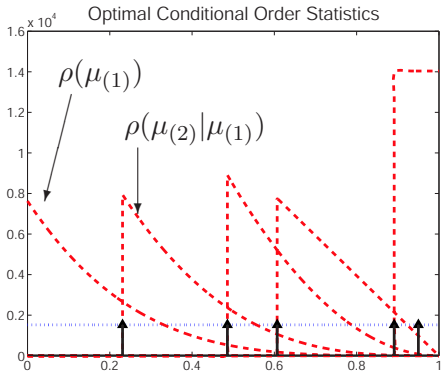


Figure 5.2. Deterministic ordering reduces the entropy of the sample set. Optimal encoding of 1-D samples can be accomplished via the conditional distributions of the order statistics $\rho(\mu_{(i)}|\mu_{(i-1)})$, if known.

■ 5.3.2 Suboptimal Encoding

It is difficult to implement the optimal encoding process for a set of i.i.d. samples $\{\mu_i\}$ in arbitrary dimension for a number of reasons. First of all, the distribution $p(\mu)$ from which the samples are drawn is typically not known at the receiver; it may not even be known at the transmitter. Furthermore, even if $p(\mu)$ is known, the optimal encoding distributions $\rho(\mu_{(i)}|\mu_{(i-1)}, \ldots, \mu_{(1)})$ can become complicated and unwieldy to compute for each sample i. It is therefore desirable to approximate the optimal encoding distribution in some manner; any approximate distribution $\hat{\rho}(\cdot)$ implicitly defines an encoder by assigning each value of μ a symbol whose length is proportional to the log–probability $\hat{\rho}(\mu)$. When $p(\mu)$ is known at both sensors, approximations $\hat{\rho}(\cdot)$ to the optimal $\rho(\cdot)$ can be used to provide computational efficiency. Perhaps more importantly however, when $p(\mu)$ is not known at the receiver, approximating the optimal encoding distributions with a simple parametric description of $\hat{\rho}(\cdot)$ allows the transmitter to encode the samples in a suboptimal but efficient manner, by first describing the parameters of the encoder itself, then describing the sample values.

Stationary and Non-stationary Codes

One important aspect to consider is that the optimal distributions $\rho(\mu_{(i)})$ are *non-stationary*, i.e., the marginal distribution of the sorted variables $\mu_{(i)}$ are *different* for each i. It turns out that this non-stationarity is key to obtaining significant savings due to reordering. Strict–sense stationary encoding distributions, i.e., encoding distributions for which the marginal $\hat{\rho}(\mu_{(i+1)}, \ldots, \mu_{(i+k)})$ is the same for all i, for any value of k, can only achieve encoding cost $N\beta + \mathrm{H}(p(\boldsymbol{\mu}_N))$, rather than the optimal $N\beta + \mathrm{H}(\rho(\boldsymbol{\mu}_N))$.

Let us consider a small example, which illustrates a special case of the more general result. Suppose that we have only two samples μ_1, μ_2, each of which take on one of the

two values $\{0, 1\}$, with probability p_1 and $1 - p_1$, respectively. Let $0 < p_1 < 1$, so that the μ_i are not deterministic. We write these probability mass functions as

$$p(\mu_1) = [p(\mu_1 = 0), p(\mu_1 = 1)] = [p_1, 1 - p_1] \qquad p(\mu_2) = [p_1, 1 - p_1]$$

Sorting μ_1, μ_2 to obtain the random variables $\mu_{(1)} \leq \mu_{(2)}$ we have that

$$\rho(\mu_{(1)}) = [\rho_1 \ , \ 1 - \rho_1] = [p_1^2 + 2p_1(1 - p_1) \ , \ (1 - p_1)^2] \tag{5.4}$$

$$\rho(\mu_{(2)}|\mu_{(1)}) = \begin{cases} [p_1^2/\rho_1 \ , \ 2p_1(1 - p_1)/\rho_1] & \text{for } \mu_{(1)} = 0 \\ [0 \ , \ 1] & \text{for } \mu_{(1)} = 1 \end{cases} \tag{5.5}$$

and marginalizing, we obtain

$$\rho(\mu_{(2)}) = [\rho_2 \ , \ 1 - \rho_2] = [p_1^2 \ , \ (1 - p_1)^2 + 2p_1(1 - p_1)]$$

Since $2p_1(1 - p_1) > 0$, the marginal distributions $\rho(\mu_{(1)})$ and $\rho(\mu_{(2)})$ are not equal, or in other words, the optimal encoding distributions are non-stationary.

Suppose that we attempt to approximate the optimal distributions (5.4)–(5.5) using a stationary encoding distribution, i.e., one for which $\rho_1 = \rho_2$. Such a distribution is parameterized by

$$\hat{\rho}(\mu_{(1)}) = [q_1, 1 - q_1] \qquad \hat{\rho}(\mu_{(2)}|\mu_{(1)}) = \begin{cases} [q_2, 1 - q_2] & \text{for } \mu_{(1)} = 0 \\ [q_3, 1 - q_3] & \text{for } \mu_{(1)} = 1 \end{cases} \tag{5.6}$$

with q_1, q_2, and q_3 all in the interval $[0, 1]$; the condition of stationarity requires that the marginal distribution of $\mu_{(2)}$ match that of $\mu_{(1)}$, i.e.,

$$q_1 q_2 + (1 - q_1)q_3 = q_1 \qquad \Rightarrow \qquad q_3 = \frac{q_1}{1 - q_1}(1 - q_2)$$

making q_3 a deterministic function of q_1 and q_2. We can select q_1 and q_2 so as to minimize the expected cost of transmission, which is given by

$$(p_1^2 + 2p_1(1 - p_1)) \log_2 q_1 + (1 - p_1)^2 \log_2(1 - q_1) + \\ p_1^2 \log_2 q_2 + 2p_1(1 - p_1) \log_2(1 - q_2) + (1 - p_1)^2 \log_2(1 - q_3).$$

Optimizing this over q_1 and q_2, we find that the best possible stationary distribution for encoding the sorted random variables $\mu_{(1)}$, $\mu_{(2)}$ is given by $q_1 = q_2 = q_3 = p_1$, the same distribution as should be used for the unsorted, independent variables μ_1, μ_2.

This result should not be surprising, since the information about the $\mu_{(i)}$ can be thought of in two pieces—one piece, the distribution $p(\mu)$, is identical for all samples and thus stationary in nature, while the second, the fact that the samples are ordered increasingly in i, is a purely non-stationary piece of information. By selecting an encoding distribution which is strict–sense stationary, we remain able to take advantage

of the first piece of information, but lose the latter entirely. Intuitively, the reduced transmission cost of the ordered samples is a result of the distribution of $\mu_{(i)}$ changing in a predictable way, based not only on the value of previous data but also on i itself.

An inductive argument on both the size N of the sample set and the number of discrete bins can be applied to demonstrate that the best strict–sense stationary encoding distribution for the $\mu_{(i)}$ is in fact the distribution of the unsorted samples, $p(\mu)$. Thus, *no* stationary encoding distribution is able to gain any advantage from the reduced entropy of the sorted samples.

With this fact in mind, it behooves us to select a non-stationary encoding process. Unfortunately, this means that most traditional source coding techniques, which are typically based around the assumption of stationary processes, are unable to derive significant advantage from the lower entropy of a set. Furthermore, the encoder should be chosen so that the predictive distributions change in a manner consistent with whatever deterministic order is selected. Thus we cannot simply sort the samples and apply any arbitrary encoder; we must select source coding methods which are in a sense well–matched to both the idea and method of sorting samples.

Linear Predictive Encoding

One possible method of constructing a non-stationary code is to assume that the *conditional* distribution of the $\mu_{(i)}$, given some fixed number k of previous samples, is stationary, in other words, to select $\hat{\rho}(\mu_{(i)}|\mu_{(i-1)}, \ldots, \mu_{(i-k)})$ to be constant for all i. In this section we focus particularly on the simplest case, in which $k = 1$. Of course, the optimal (in 1-D, order statistic) distributions $\rho(\cdot)$ are not conditionally stationary, either; they depend on the value of i and the total number of samples N. However, conditional stationarity provides a simple parameterization of $\hat{\rho}$ which can result in a non-stationary encoding distribution.

Linear predictive coding [28] is one commonly used representation based on conditionally stationary processes. A trivial example of a linear predictive code is a random walk with Gaussian noise; however, as we shall see even this simple model can be quite powerful as a predictive encoder of sample sets. For the moment, we focus solely on 1-D distributions and ascending sample order. We consider constructive methods of encoding collections of samples from higher–dimensional distributions in Section 5.5.

Let us compare the performance obtained using two possible suboptimal encoding methods based on the conditional distributions of random walks. Again, we suppose that the true distribution $p(\mu)$ is uniform on the interval $[0, 1)$. We consider two random walk distributions; the first is a traditional random walk with Gaussian noise,

$$\hat{\rho}(\mu_{(i)}|\mu_{(i-1)}) = \mathcal{N}(\mu_{(i)}; \mu_{(i-1)}, \lambda^2). \tag{5.7}$$

Sending the samples in increasing order and encoding according to (5.7) can obtain performance not far from the optimal encoder. To be concrete, we choose λ^2 equal to the variance of $p(\mu)$ divided by N, e.g. for uniform $p(\mu)$ we take $\lambda^2 = \frac{1}{12N}$.

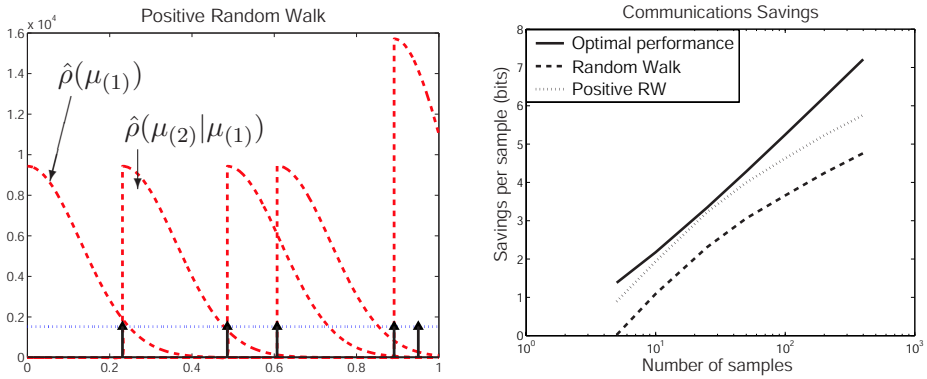


Figure 5.3. (a) Using a positive–only random walk distribution $\hat{\rho}$ to encode a set of deterministically ordered samples; the optimal encoding distributions ρ are shown in Figure 5.2. (b) Simplified, suboptimal encoders $\hat{\rho}$ can be described more succinctly than ρ, but result in a slight reduction of the per–symbol communications savings over encoding according to the distribution $p(\mu)$ of the unsorted samples.

Our second random walk distribution is a simple variant of this encoding procedure, obtained by noting that $\mu_{(i)} \geq \mu_{(i-1)}$ and using only the positive half of the random walk. This gives

$$\hat{\rho}(\mu_{(i)}|\mu_{(i-1)}) = \begin{cases} 2\mathcal{N}(\mu_{(i)} \;;\; \mu_{(i-1)}, \lambda^2) & \mu_{(i)} \geq \mu_{(i-1)} \\ 0 & \text{otherwise.} \end{cases} \tag{5.8}$$

Figure 5.3(a) shows the conditional distributions given by (5.8) for five samples, and can be compared to the optimal encoding distributions ρ shown in Figure 5.2.

Furthermore, we may measure the performance of any suboptimal encoding algorithm by examining the per–sample savings (in bits) achieved over encoding according to the unsorted distribution $p(\mu)$ as a function of the total number of samples N. Optimal encoding results in per–sample savings of $\frac{1}{N}\log_2 N!$, while for any stationary code the savings are, on average, zero. Figure 5.3(b) shows curves indicating the per–sample savings, in bits, of three encoders: the optimal encoder $\rho(\cdot)$, the zero–mean random walk given by (5.7), and the positive–only random walk given by (5.8). Although the random walk distributions (5.7)–(5.8) are suboptimal, they come quite close to the optimal performance.

■ 5.4 Message Approximation

Let us now turn our attention to *lossy* encoding. Typically, source coding problems are examined from the perspective of minimizing a reconstruction error on the original data. However, for inference problems we do not need to reconstruct the original data $\mathbf{y}$, but instead merely wish to minimize the impact of mistakes on our final distribution

estimates. Thus, the "correct" error measure to use with respect to any transmitted message depends on the desired measure of error for the estimated posterior distribution. Ideally, we desire *local* rules which, when followed at all sensors, lead to *global* bounds or estimates on the error at each sensor.

■ 5.4.1 Maximum Log–Error

One possible measure which has these properties is the maximum log–error

$$\Delta(m, \hat{m}) = \max_x \left| \log m(x) \,/\, \hat{m}(x) \right|, \tag{5.9}$$

which is closely related to the the similar "dynamic range" measure described in detail in Chapter 4:

$$d(m, \hat{m}) = \max_x \min_\alpha \left| \alpha + \log m(x) \,/\, \hat{m}(x) \right|.$$

For normalized messages $m, \hat{m}$, the two are related by the inequalities

$$d(m, \hat{m}) \leq \Delta(m, \hat{m}) \leq 2d(m, \hat{m})$$

which makes it possible both to control $d(m, \hat{m})$ through the easier to evaluate quantity $\Delta(m, \hat{m})$, and bound its influence using the analysis of Chapter 4.

Unfortunately, the measure $\Delta(\cdot)$ is a very strict requirement for distribution agreement. Its implication for continuous distributions is that both distributions must have nearly *identical* tail behavior. In portions of the state space where $m(x)$ is very small, we see that $\hat{m}(x)$ must stay within some constant factor of $m(x)$ in order to have bounded error Δ, and thus the *rate* at which $m(x)$ and $\hat{m}(x)$ approach zero must at some point become identical.

For a pair of Gaussian mixture distributions, this behavior can be ensured by any of several means. If the mixture components which determine the tail behavior can be identified (for example the left– and right–most components in a one–dimensional distribution) they may be preserved to high precision. This is particularly easy if a *single* mixture component determines the tail behavior of the overall distribution; for example, one very broad (possibly low weight) component which dominates the rest of the distribution in very low likelihood regions. Such components are sometimes added to model outlier processes [42, 48], and may be either deterministically added at the receiver or transmitted with high precision.[2]

Computationally, evaluating the error $\Delta(\cdot)$ between two Gaussian mixtures is also non-trivial, but may be performed either by discretization and direct evaluation in relatively low (1–2) dimensions, or via gradient search. Although gradient search can be susceptible to local maxima, the form of (5.9) as the ratio of Gaussian mixtures leads one to expect that the global maximum may be found with relative ease by local

[2] A similar solution involves the addition of a small but non-zero constant to the distribution estimate (or imposition of a threshold away from zero), essentially modeling the inclusion of a small uniform (rather than Gaussian) outlier component.

optimization from each of the mixture centers, using a procedure similar to that outlined in [8] for mode–finding in Gaussian mixtures. Thus, evaluating $\Delta(m, \hat{m})$, where m and $\hat{m}$ are both Gaussian mixtures with N or fewer components, requires $\mathcal{O}(N)$ operations. A reasonable *estimate* of $\Delta(m, \hat{m})$ when m is a kernel density estimate and $\hat{m}$ consists of a *single* Gaussian (a task which becomes important in the next section) is given by simply evaluating the log–ratio at each kernel center and taking the maximum absolute value, as

$$\hat{\Delta}(m, \hat{m}) = \max_i \left| \log m(\mu_i)/\hat{m}(\mu_i) \right|,$$

where the μ_i are the kernel centers of $m(x)$; this estimate also requires $\mathcal{O}(N)$ work, and in practice tends to be much faster than finding the modes of Δ using gradient ascent.

■ 5.4.2 Kullback–Leibler Divergence

We may also consider another common measure of error between two distribution–based messages, the Kullback–Leibler (KL) divergence

$$D(m(x) \,\|\, \hat{m}(x)) = \int m(x) \log m(x) \,/\, \hat{m}(x) \; dx. \tag{5.10}$$

Compared to the maximum log–error, the KL-divergence tends to be a much more forgiving measure of error between two messages $m(x)$ and $\hat{m}(x)$. Intuitively, while the maximum log–error focuses on the largest deviation between the log–messages, the KL-divergence *weights* that deviation by the importance assigned to that portion of the state by the message $m(x)$. This emphasizes errors which occur in regions of the state space currently thought to have high probability. The reduced emphasis on low–probability regions means that strict matching of tail behavior, such as described in Section 5.4.1, is no longer required.

We can estimate the KL-divergence between a kernel density estimate $m(x)$ and a general Gaussian mixture $\hat{m}(x)$ fairly easily, using the plug–in methods described in Section 2.3.2. Specifically, we use

$$\hat{D}(m \| \hat{m}) = \sum_i w_i \log m(\mu_i)/\hat{m}(\mu_i)$$

where $\{w_i, \mu_i\}$ are the weights and kernel centers of $m(x)$. This estimate is again $\mathcal{O}(N)$, where N is the number of kernels in $m(x)$.

If the messages $m(x)$ communicated between sensors are chosen to be posterior distributions, e.g., $p(x_2|y_1)$, then the KL-divergence between messages (5.10) is in many ways similar to the posterior–weighted log–error discussed in Chapter 4. In Chapter 4, we used this error measure to describe the expected or average behavior of the KL-divergence over many realizations of the observed random variables $\mathbf{y}$. In this chapter, we consider what can be done to control the tradeoff between the communications cost and error given a *single* realization of $\mathbf{y}$. However, there may be situations in which we may, in fact, be interested in the average behavior of a system. For example, consider

a sensor network which regularly takes measurements to infer some characteristic of the environment, say once per day. Each day, then, provides a new realization $\mathbf{y}$, and we may be interested in minimizing our average energy use and average inference error over many days, rather than the energy required or error obtained for any particular day's measurements.

In prolonged, repetitive inference problems such as these, there are two aspects to controlling the average behavior of errors, as measured by the expected KL-divergence of Chapter 4. First, each new realization of observations $\mathbf{y}$ forces us to balance the KL-divergence (5.10) with the communications cost required for inference using those particular observations. Second, in addition to the tradeoff given a particular $\mathbf{y}$, there is also a resource allocation problem—how to assign resources between various realizations (in our example, from day to day) so as to control the average behavior of both communications cost and inferential error. The second half of this problem is left for future research; here we concentrate only on the former problem, in which we have a single, fixed set of observations $\mathbf{y}$ and assume that we have a predetermined set of resources available, expressed either as a maximum communications cost or as a maximum allowable error.

■ 5.4.3 Other Measures of Error

There exist many other measures of error between distributions which are commonly applied in density approximation literature. While a link between applying these measures of error to BP messages and estimating the level of error in subsequent distribution estimates and inference tasks has yet to be established, their popularity makes it worthwhile to consider whether methods of lossy encoding may be developed which incorporate these other measures instead. We mention two measures explicitly, the L_1 or absolute integrated error,

$$L_1(m, \hat{m}) = \int |m(x) - \hat{m}(x)| \, dx \tag{5.11}$$

and the L_2 or integrated squared error,

$$L_2(m, \hat{m}) = \int (m(x) - \hat{m}(x))^2 \, dx. \tag{5.12}$$

We shall see that the optimization procedures developed for lossy encoding in Section 5.5 can be easily modified to accommodate both of these error measures, as well.

■ 5.5 KD-tree Codes

In many data compression applications, multi–scale descriptions have proven extremely useful [55, 87]. Multi–scale descriptions first capture broad, large scale phenomena, then encode a series of "refinements" which capture further details. In general, the multi–scale description is hand selected (for example wavelet decompositions, Fourier

coefficients, etc.), and the refinement information forms a tree–like structure which can then be easily optimized and truncated to trade off representation size with reconstruction quality.

Similarly, we can create an encoder by adopting a particular multi–scale description of the kernel density estimate, the KD-tree (Section 2.4). We use the KD-tree for three separate but related purposes. We first use the KD-tree to provide a class of approximations to the original message $m(x)$. We then apply the same tree structure to define an encoder for any particular member of this class. The KD-tree provides both a deterministic ordering, similar to the sorting process in 1-D but applicable to distributions in arbitrary dimension, and a convenient choice for the encoding distributions $\hat{\rho}(\cdot)$. Third, we apply the KD-tree structure to efficiently *select* one of the approximations $\hat{m}(x)$ in order to balance the resulting communication cost and message error.

We begin by constructing a KD-tree structure for our kernel density estimate $m(x)$ in the manner described in Section 2.4, creating Gaussian approximations to the portions of $m(x)$ at each level of the KD-tree. To remind the reader, this means that the sufficient statistics stored at each node s in the KD-tree are the mean and variance in each dimension of the Gaussian sum defined by the node's children, along with a weight w_s representing the total weight contained in the subtree (i.e., for an equal weight kernel density estimate, the number of leaf nodes stored below node s, divided by the total number of leaf nodes in the KD-tree). These statistics are easily computed recursively for each node s:

$$\begin{aligned} w_s &= w_{s_L} + w_{s_R} \\ \mu_s &= \frac{w_{s_L}}{w_s}\mu_{s_L} + \frac{w_{s_R}}{w_s}\mu_{s_R} \\ h_s &= \frac{w_{s_L}}{w_s}(h_{s_L} + \mu_{s_L}^2) + \frac{w_{s_R}}{w_s}(h_{s_R} + \mu_{s_R}^2) - \mu_s^2 \end{aligned} \tag{5.13}$$

where the square of a vector μ is computed element-wise. As mentioned in Section 2.4, there are many methods of constructing KD-trees, any of which may be applied here. In practice, we use the simple procedure described in Section 2.4, dividing the data into equal or nearly equal sets along a cardinal axis chosen according to the covariance of the data. We assume that the weights of the original kernel density estimate (5.1) are all equal, i.e., $w_i = 1/N$. This fact, along with our choice of construction method, means that the weights w_s within the KD-tree are *deterministic* given the total number of kernels N, and thus it is sufficient to represent only the means and bandwidths within the KD-tree.

In the following sections, we describe how the Gaussian statistics of the KD-tree can be used to define a class of approximations $\hat{m}(x)$ to the original message $m(x)$. We then consider the communication cost inherent in transmitting any particular member of this class of approximations, by describing one possible constructive encoding algorithm. Finally, we turn to the problem of efficiently selecting a member of this class which has both low communications cost and low error.

■ 5.5.1 KD-tree Gaussian Mixtures

KD-trees are typically applied to perform locality–based computations rapidly on large sets of continuous–valued points. For example, KD-trees have been applied to improve the speed of EM for learning mixture models [70]. In this section, we describe how to use the same KD-tree structure *instead* to define a class of Gaussian–mixture approximations to the message $m(x)$.

In particular, each node s of the KD-tree describes a Gaussian approximation $\hat{m}_s(x)$ to the normalized sum of Gaussian kernels $m_s(x)$ stored by its associated leaves, i.e.,

$$m_s(x) = \sum_{l \in D_L(s)} \frac{w_l}{w_s} \mathcal{N}\left(x\,;\,\mu_l, \mathrm{diag}(h_l)\right) \qquad \hat{m}_s(x) = \mathcal{N}\left(x\,;\,\mu_s, \mathrm{diag}(h_s)\right) \tag{5.14}$$

where again the set $D_L(s)$ indicates the leaf nodes descended from node s. We can use these Gaussian components to define a class of Gaussian mixture models, parameterized as follows. Define an *admissible* density set S to be any set of nodes in the KD-tree such that, for every node $s \in S$, S contains neither descendants nor ancestors of s, and for every *leaf* node l, either l or some ancestor of l is contained in S. The Gaussian sum defined by any such S, namely

$$\hat{m}_S(x) = \sum_{s \in S} w_s \hat{m}_s(x) = \sum_{s \in S} w_s \, \mathcal{N}\left(x\,;\,\mu_s, \mathrm{diag}(h_s)\right)$$

yields an approximation to the original kernel density estimate

$$m(x) = \sum_{s \in S} w_s m_s(x) = \sum_{\text{leaves } l} w_l \, \mathcal{N}\left(x\,;\,\mu_l, \mathrm{diag}(h_l)\right)$$

of varying degrees of coarseness depending on the selection of nodes in S. For example, when S consists of only the root node, i.e. $S = \{1\}$, this gives a Gaussian approximation to $m(x)$; when $S = \{2, 3\}$, we have a two-component approximation, and so forth. A simple one–dimensional example KD-tree, along with several Gaussian sum approximations parameterized by different choices of S, are shown in Figure 5.4. The admissibility conditions essentially require that each leaf node be represented by one and only one Gaussian mixture element.

Suppose that we elect to approximate our message $m(x)$ using the Gaussian mixture $\hat{m}_S(x)$, for some S. Clearly, smaller sets S translate into lower communication costs for the approximation $\hat{m}_S(x)$. On the other hand, smaller sets S also have the property that they approximate the density $m(x)$ more coarsely, and are thus likely to have increased error. Let us first consider how, given a particular set S, the density estimate $\hat{m}_S(x)$ may be encoded efficiently, after which we will consider the process of *selecting* the set S.

■ 5.5.2 Encoding KD-tree Mixtures

Suppose that we are given the KD-tree representation of a kernel density estimate $m(x)$ with parameters $\{w_i, \mu_i, h_i\}$, with $w_i = 1/N$, and we would like to encode the Gaussian

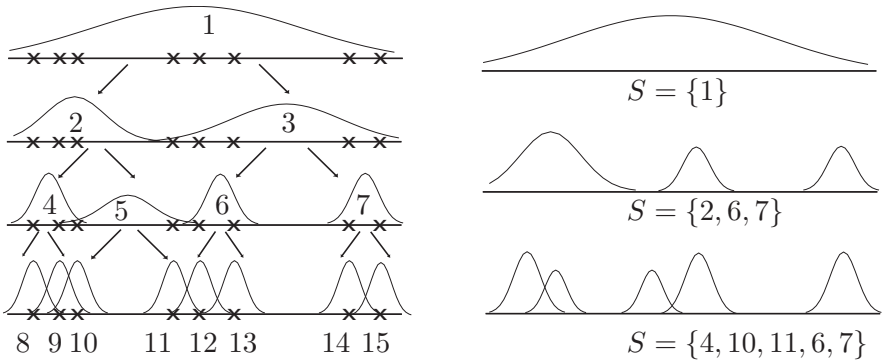


Figure 5.4. A one–dimensional KD-tree representing a mixture of 8 Gaussian kernels, and caching means and variances at each level resulting in a hierarchy of Gaussian approximations; the nodes have been labeled by the numbers $1 \ldots 15$ for the discussion in the text. These Gaussian components can be used to define a class of Gaussian mixture approximations parameterized by the set S; several choices of S and the resulting approximations are shown.

mixture approximation $\hat{m}_S(x)$, parameterized by some particular choice of admissible set S. Both $m(x)$ and the set S are unknown at the receiver. When S is taken to be the collection of all leaf nodes in the KD-tree, this is a generalization of the lossless encoding problem discussed in Section 5.3; when S consists of some smaller set, communicating $\hat{m}_S(x)$ represents a *lossy* encoding of the original message $m(x)$.

The KD-tree can be used to establish a partial ordering on the elements of the set S (e.g., left–to–right order in the binary tree structure). Moreover, since each parent node within the KD-tree is constructed deterministically from its children, there is no more randomness in the parameters of the nodes of S and their ancestors together (the entire KD-tree in or above the set S) than there is in the parameters of S alone. In other words, since the ancestors of each node in S can be deterministically constructed using the statistics stored at the nodes of S, an optimal representation of both S and its ancestors should cost no more than the optimal representation of just the statistics in S. By sending the finest scale information of interest (S), the coarser scales are effectively free.

On the other hand, we prefer to send the approximations in a coarse–to–fine manner, i.e., transmitting the statistics stored at the root node, then the "refinement" information necessary to reconstruct the statistics stored at its children, and so forth. The refinement of a node s consists of the information necessary to reconstruct the statistics of the children of s, given the statistics at s itself. In particular, we describe how the statistics of s provide information and constraints on the possible values of its children's statistics. This leads to a general procedure for encoding $\hat{m}_S(x)$ for any fixed choice of an admissible set S.

With regard to the original message $m(x)$, we have again assumed that the weights w_i are all equal, but relax the assumption that h_i is the same for all i. The encoder we describe is sufficiently general to deal with nonuniform h_i, though we also describe

how an agreement between transmitter and receiver to choose equal valued h_i can sometimes be used to reduce the KD-tree representation cost. As mentioned previously, the assumption of uniform weights w_i, along with our choice of KD-tree construction algorithm, means that the weights w_s at each node of the KD-tree are predetermined given the total number of samples N, and thus it suffices to represent the means and bandwidths at each node.

There are two inputs to our encoding process. The first is a KD-tree created in the manner described in Section 2.4 and with the statistics given by (5.13) stored at each internal node s. At the finest scale these statistics are the stored values of the original parameters $\{w_i, \mu_i, h_i\}$ of the kernel density estimate $m(x)$. At any internal node s, the statistics provide a Gaussian approximation $\hat{m}_s(x)$, with parameters μ_s and h_s, to the kernels stored below s in the KD-tree. The second input is any admissible set S in the KD-tree, as defined in the previous section.

To encode the KD-tree representation of $\hat{m}_S(x)$, we first represent the root node. As we assume that, initially, no information about the distribution $p(\mu)$ is available to the receiver, we simply represent μ_1 and h_1 using a direct fixed–point representation. We also transmit a single bit indicating whether or not the root node is in the set S. If it is, we are done, since the only admissible set containing the root node is the root node itself; if not, we must describe the refinement information necessary to reconstruct each of its two children.

Now, suppose that we have transmitted the statistics of a particular node s, meaning that the receiver is able to decode the coarse, single Gaussian approximation $\hat{m}_s(x) = \mathcal{N}(x\,;\,\mu_s, \mathrm{diag}(h_s))$ representing the subtree rooted at node s, along with a bit indicating whether $s \in S$. If $s \in S$, we have no need for the statistics stored below s, and can stop; otherwise, we need to transmit more information about the subtree below s. We use the distribution $\hat{m}_s(x)$ as a form of prior information to encode the parameters of the left and right children s_L and s_R of s. To refine the KD-tree description at node s, the receiver must be able to recover the mean vectors μ_{s_L}, μ_{s_R} and bandwidth vectors h_{s_L}, h_{s_R} stored at the children of node s. The recursive relationship (5.13) yields two equations with four unknowns; thus it is sufficient[3] to transmit μ_{s_R} and h_{s_R}, then recover the left–hand values by algebra. In other words, by encoding either child of s we have implicitly encoded the other. We again represent whether each of the nodes s_L and s_R is in the set S, which is done using one bit each.

The exact sequence in which to visit the nodes of the KD-tree is arbitrary, so long as it is deterministic. Perhaps the most intuitive refinement sequence is to follow a breadth–first order, e.g., refine both s_L and s_R, then refine each of their four children, and so forth until reaching the set S.

We have chosen to indicate which nodes are in S, i.e., have no further refinement information in the representation, using one bit per node, or $2|S| - 1$ bits total. While it is possible that this information can be represented in some more compact form, this

[3]With a slight caveat: due to the averaging process, given μ_s, μ_{s_R} to precision β, μ_{s_L} is computed to precision $\beta - 1$; thus we may require an additional bit of information for μ_{s_L}, and similarly for h_{s_L}.

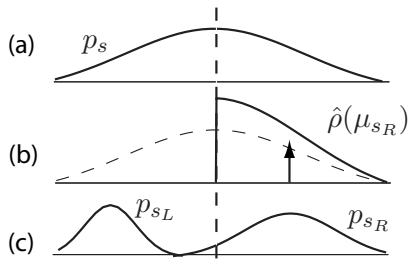


Figure 5.5. Transmitting a KD-tree in top-down fashion. (a) Given the mean μ_s and bandwidth h_s of a coarse–scale estimate, (b) we encode the right–hand mixture component according to (5.15)–(5.16); the encoding distribution $\hat{\rho}(\mu_{s_R})$ is shown as solid, while the transmitted value of μ_{s_R} is indicated by the arrow. (c) Having decoded the right–hand component, the receiver may simply solve for the left–hand component using (5.13).

provides a very reasonable approximation since of the total $2|S| - 1$ nodes in or above S, just over half are in S.

To provide a complete encoder requires making one further design choice, specifying precisely *how* the the right– (or left–) hand mean and covariance statistics are encoded. Clearly, the summarization statistics from the parent node μ_s, h_s can be used to construct an encoding distribution known at both sender and receiver. These summarization statistics also provide ordering information about the children—for example, in 1-D, the right child is always greater than the left child. In higher dimensions this ordered relationship holds for one of the cardinal dimensions (which dimension can be determined from the parent node's statistics). For example, a simple choice for 1-D samples reminiscent of Section 5.3.2's random walk encoder is given by

$$\hat{\rho}(\mu_{s_R}|\mu_s, h_s) = \begin{cases} 2\mathcal{N}\left(\mu_{s_R}\,;\, \mu_s, \mathrm{diag}(h_s)\right) & \mu_{s_R} \geq \mu_s \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

$$\hat{\rho}(h_{s_R}|h_s, \mu_s, \mu_{s_R}) = \mathcal{N}\left(h_{s_R}\,;\, \hat{h}_{s_R}, \mathrm{diag}(\hat{h}_{s_R}/2)\right) \tag{5.16}$$

where

$$\hat{h}_{s_R} = h_s + \mu_s^2 - \frac{w_{s_R}}{w_s}\mu_{s_R}^2 - \frac{w_{s_L}}{w_s}\mu_{s_L}^2.$$

This encoder is depicted in Figure 5.5.

This procedure generalizes easily to densities in arbitrary dimension; the KD-tree provides a natural ordering of the points, just as the sorted order did in one dimension. Positivity requirements such as appear in (5.15), if present, are only included for the dimension along which the data has been split at each level, since this is the only dimension in which the KD-tree provides ordering information. Otherwise, equation (5.15) may remain the same, with summations and squaring operations performed elementwise. Were we to elect to store general, non-diagonal covariance matrices at each node in our KD-tree instead of the bandwidth vectors h_s, we could apply a similar algorithm;

however, the required encoding distribution for the covariance statistics would become considerably more complex.

The algorithm just described does not require that the bandwidths h_i be equal for each leaf node i. However, it is quite common that the bandwidths *are* uniform. If this fact is known at the receiver, and (as assumed in Section 5.3) the value of this bandwidth is also known, we should be able to encode the message more efficiently. We can easily incorporate this information in two ways.

First, if S has elements which are leaf nodes of the tree, we can improve the cost of refining those nodes' parents. This is simply a consequence of the deterministic relationships dictated by (5.13). For example, consider the process of refining a particular node s when *both* children of s are leaf nodes. In this case, the process of determining μ_{s_R} and μ_{s_L}, with $h_{s_R} = h_{s_L} = h$ known, involves solving two equations with only two unknowns and is thus essentially free.

Another improvement can be made to the encoder of the mean value, (5.15). The value μ_s is the mean of a set of kernel centers μ_i at the finest scale; but the bandwidth h_s stored at s is the variance of those kernel centers, increased by the bandwidth h_i of the kernels at the finest scale. Thus, if the finest–scale bandwidths are uniform with known value h, their influence can be subtracted off to provide a more accurate characterization of the variance of the means μ, and thus a better encoder of their values. For example, instead of using (5.15) to encode μ_{s_R} we may instead define $\bar{h}_s = h_s - h$ and use

$$\hat{\rho}(\mu_{s_R}|\mu_s, h_s) = \begin{cases} 2\mathcal{N}\left(\mu_{s_R}\,;\, \mu_s, \mathrm{diag}(\bar{h}_s)\right) & \mu_{s_R} \geq \mu_s \\ 0 & \text{otherwise.} \end{cases} \tag{5.17}$$

where again, the positivity constraint is used for only one of the dimensions of μ, with the others being encoded according to, say, a typical Gaussian distribution.

Given a KD-tree representation for the N-component kernel density estimate $m(x)$, we may pre-compute the communications cost of all $N-1$ potential refinement actions, i.e., the cost of encoding the child nodes of each parent node, and store these costs within the KD-tree as well. It is then easy to determine the communications cost associated with transmitting any particular admissible density set S for the tree–structure, by simply summing the costs stored by the ancestors of S, plus the cost of transmitting the root node. Equivalently, we can define the cost of the root node to be $B(1)$, and the cost $B(s)$ for any node s to be one–half the cost of its parent plus one–half the cost of refining its parent. Then, the cost of transmitting the set S is simply the sum over the values of $B(s)$ for each element $s \in S$.

The KD-tree based encoder described here is, of course, only *one* possible code choice; there may exist many other, equally good methods of encoding the density estimate. Further investigation of methods for encoding kernel density estimates remains an open area of research.

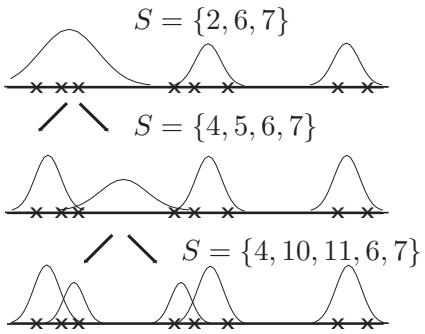


Figure 5.6. A sequence of approximations S for the KD-tree in Figure 5.4. The majority of the error in the approximation $S = \{2, 6, 7\}$ comes from node 2, but refining node 2 does not necessarily lead to an immediate reduction of error; the approximation $S = \{4, 5, 6, 7\}$ may actually be worse. However, further refinement, e.g., to $S = \{4, 10, 11, 6, 7\}$, eventually improves the error.

■ 5.5.3 Choosing among admissible sets

We now have a class of potential message approximations, parameterized by a choice of an admissible set S of nodes on the KD-tree. Furthermore, given a particular set S, we have both a means of encoding the estimate $\hat{m}_S(x)$ and the means of efficiently computing the cost, in bits, of its description. We now consider the problem of *selecting* the set S to balance the cost of communications and approximation error.

Choosing a good set S is certainly difficult; we begin by considering the task somewhat abstractly. Suppose that our goal is to obtain a "small" set S which has approximation error smaller than some constant ϵ. One sensible way to find S is to begin with the smallest possible set, namely the root node, then successively refine the density estimate by increasing the size of S, until $\hat{m}_S(x)$ is deemed sufficiently accurate.

If the current set S does not yet give an acceptable approximation, one way to refine the density estimate is to select and refine some node s in S, i.e., replace s with its two children s_L and s_R. In order for the new set to remain admissible, both children s_L and s_R must be included and the parent, s, excluded.

This procedure leaves two issues undecided—first, determining when the density estimate associated with S is sufficiently accurate, and second, selecting the node $s \in S$ to be refined at each stage. Unfortunately, the second issue is quite complicated. First, there is a communications cost associated with refinement which may not be identical for each node in S. Secondly, the error between $m(x)$ and $\hat{m}_S(x)$ is a complex function of the set S. Let us consider this latter point.

Replacing a particular node s with its children s_L and s_R may not reduce our overall error significantly; in fact, it may even increase the error. Yet this substitution of s with its children may be a necessary step in order to arrive eventually at a sufficiently accurate estimate. In other words, we may need to refine certain nodes even though the estimate does not show any immediate improvement. Let us consider an example—in

Figure 5.6 we show a sequence of refinements using the KD-tree from Figure 5.4, which represents a (tri-modal) message $m(x)$. The set $S_1 = \{2, 6, 7\}$ gives a reasonable, but not excellent, tri-modal approximation $\hat{m}_{S_1}(x)$. Most of the error is clearly attributable to the approximation at node 2, which summarizes the leftmost four samples. However, refining node 2 to obtain $S_2 = \{4, 5, 6, 7\}$ actually appears to result in a worse approximation; for example, it has an extra mode. In order to improve significantly over S_1, we must further refine S_2 to obtain $S_3 = \{4, 10, 11, 6, 7\}$.

This example illustrates a fundamental weakness of a greedy, "largest improvement" approach—such an approach would never refine node 2, yet this is exactly what needs to be done to improve the error significantly. We elected to refine node 2 not because the subsequent approximation was considerably better, but rather because the approximation at node 2 did not accurately fit the points stored below it. This can be made precise in the following way.

Suppose that we could somehow *attribute* the error between $m(x)$ and $\hat{m}_S(x)$ to the individual nodes $s \in S$. Then, instead of selecting the member of S resulting in the greatest immediate improvement, we could instead select the member of S most in *need* of improvement. While this is still a "greedy" approach, it does not suffer from the same kind of failure just described. It turns out that we may construct an upper bound on the error between $m(x)$ and $\hat{m}_S(x)$ which has precisely this quality—it decomposes into parts, which measure error attributable to each of the nodes $s \in S$.

■ 5.5.4 KD-tree Approximation Bounds

Given a message $m(x)$, and a Gaussian mixture approximation $\hat{m}_S(x)$ parameterized by the admissible subset S, it is relatively straightforward to estimate the error between $m(x)$ and $\hat{m}_S(x)$ under any of the measures discussed in Section 5.4. As outlined in that section, this typically requires $\mathcal{O}(N)$ operations, where N is the number of kernels in the message $m(x)$.

However, as outlined in the previous section, it may be useful to construct an assessment of the error between $m(x)$ and $\hat{m}_S(x)$ which informs us about which of the elements $\bar{s} \in S$ is most culpable for the current level of error. In particular, for any admissible density set S, with KD-tree Gaussian mixture approximation $\hat{m}_S(x)$ to the true kernel density estimate $m(x)$, we construct an upper bound on the error between $\hat{m}_S(x)$ and $m(x)$ which is computed *only* in terms of the individual elements $s \in S$, by measuring the error between their single–Gaussian approximations $\hat{m}_s(x)$ and the portion of the density summarized by s, i.e., $m_s(x)$, as given in (5.14). This allows us to determine if the sub-tree below node s has already been approximated "sufficiently well" by the statistics at s, or whether we need to refine the approximation further, eventually improving the quality of our approximate density estimate in that sub-region.

These bounds are also relatively fast to compute. Each of the error measures described below require $\mathcal{O}(N_s)$ operations to estimate the error between $m_s(x)$ and $\hat{m}_s(x)$, where N_s is the number of leaf nodes below s in the tree. Thus, to pre-compute the error bound contribution of every node in the tree takes a total of $\mathcal{O}(N + 2 \cdot \frac{N}{2} + 4 \cdot \frac{N}{4} + \ldots) =$

$\mathcal{O}(N \log_2 N)$ operations, and given these values it is trivial to compute an upper bound on the error for any particular set S.

The exact nature of the bound used to decompose error along the KD-tree structure depends on which error measure we select to evaluate the difference between $\hat{m}_S(x)$ and $m(x)$. We proceed to create bounds for use with each of the error measures considered in Section 5.4.

Maximum Log–Error

For the maximum log–error $\Delta(m, \hat{m}_S)$ we may use the inequality shown as Lemma 4.7.1 in Section 4.7. This result gives

$$\Delta\left(m(x), \hat{m}_S(x)\right) = \Delta\left(\sum_{s\in S} w_s m_s\,,\, \sum_{s\in S} w_s \hat{m}_s\right) \leq \max_{s\in S} \Delta(m_s(x)\,,\, \hat{m}_s(x)),$$

or equivalently, that the error between $m(x)$ and the Gaussian mixture $\hat{m}_S(x)$ is bounded above by the maximum error incurred by any of the single–Gaussian approximations at each node $s \in S$.

As mentioned in Section 5.4, for the error as measured by Δ to be finite, we require exact agreement between the tails of $m(x)$ and $\hat{m}_S(x)$; this same concern applies to the elements of the error bound $\Delta(m_s(x)\,,\, \hat{m}_s(x))$. We described several methods which can be applied to make (5.9) finite for a given Gaussian mixture, for example addition of a single broad component, flat threshold, or preservation of tail–dominant components. Whichever method is selected, it should also be applied to the approximation at each node s, ensuring that $m_s(x)$ and $\hat{m}_s(x)$ also have finite error. This is easily done by adding the same common modes to each subset approximation, i.e., if $m(x) = \sum_{s\in S} w_s m_s(x)$ and we include an outlier component $m_0(x)$, we assign m_0 to each node s in S in proportion to the weight w_s, so that

$$m(x) + m_0(x) = \sum_{s\in S} w_s \left(m_s(x) + m_0(x)\right),$$

and do the same for the approximations stored at each node, e.g.,

$$\hat{m}_S(x) + m_0(x) = \sum_{s\in S} w_s \left(\hat{m}_s(x) + m_0(x)\right).$$

This practice ensures that each node s matches its associated collection of kernels $m_s(x)$ sufficiently well to yield a finite error bound.

Kullback–Leibler Divergence

The KL-divergence $D(m\|\hat{m}_S)$ may be bounded using well-known convexity results [15]. Specifically, we have

$$D\left(m(x)\,\|\hat{m}_S(x)\right) = D\left(\sum_{s\in S} w_s m_s \,\Big\|\, \sum_{s\in S} w_s \hat{m}_s\right) \leq \sum_{s\in S} w_s D(m_s(x)\,\|\,\hat{m}_s(x)),$$

or equivalently, that the error between $m(x)$ and $\hat{m}_S(x)$ is bounded by the weighted sum of errors incurred by each of the single–Gaussian approximations stored at the nodes in S.

Since the KL-divergence is less strict than the maximum log–error, it is unnecessary to ensure that the tails of each approximation $\hat{m}_s(x)$ match those of $m_s(x)$. However, if an outlier component $m_0(x)$ is present in the original message $m(x)$, it may still be reasonable to assign $m_0(x)$ to each node in proportion to that node's weight, in the same way discussed for the maximum log–error.

Other Error Measures

Several other error measures may also be optimized using the same methods which we use for the maximum log–error and KL-divergence. All that is required is to specify a tree–structured bound which can be used to separate the contributions of errors from each node $s \in S$ to the error in the overall approximation $m_S(x)$. This can be done relatively easily for both the L_1 and L_2 error measures.

The L_1 error $L_1(p,q) = \int |p(x) - q(x)|\, dx$ satisfies a simple convex inequality similar to that of the KL-divergence, specifically,

$$L_1\left(m(x)\,,\,\hat{m}_S(x)\right) = L_1\left(\sum_{s\in S} w_s m_s\,,\,\sum_{s\in S} w_s \hat{m}_s\right) \leq \sum_{s\in S} w_s L_1(m_s\,,\,\hat{m}_s).$$

Bounding the L_2 error, $L_2(p,q) = \int |p(x) - q(x)|^2\, dx$, is slightly more complex. However, we may use the fact that $(a-b)^2 \geq 0 \Rightarrow a^2 + b^2 \geq 2ab$ to show that

$$\begin{aligned}(a_1 - b_1 + a_2 - b_2)^2 &= (a_1 - b_1)^2 + (a_2 - b_2)^2 + 2(a_1 - b_1)(a_2 - b_2) \\ &\leq 2(a_1 - b_1)^2 + 2(a_2 - b_2)^2\end{aligned}$$

Applying this inequality recursively to each node until we reach the elements of S gives the bound

$$L_2\left(m(x)\,,\,\hat{m}_S(x)\right) = L_2\left(\sum_{s\in S} w_s m_s\,,\,\sum_{s\in S} w_s \hat{m}_s\right) \leq \sum_{s\in S} 2^{\text{depth}(s)}\; w_s^2\; L_2(m_s\,,\,\hat{m}_s).$$

Here, the function depth(s) indicates the depth of node s in the KD-tree, i.e., the number of edges between s and the root node, so that the root has depth zero, its children depth one, and so forth.

■ 5.5.5 Optimization Over Subsets

We now return to the problem of selecting the admissible set S which parameterizes our choice of approximate message $\hat{m}_S(x)$. Given a KD-tree representation of the message $m(x)$, where $m(x)$ is a kernel density estimate with N components, we pre-compute the cost of communicating each refinement action (Section 5.5.2) and the component of

the error bound (Section 5.5.4) at each node of the KD-tree. These computations take $\mathcal{O}(N)$ and $\mathcal{O}(N \log_2 N)$ operations, respectively.

We then select the set S as follows. Suppose that we have some function $f(B, E)$ which represents the "cost" of selecting a representation requiring B bits and with error E, making precise the tradeoff between bits and error. We assume that the cost $f(B, E)$ is increasing in both B and E. We use $\bar{S}$ to indicate a temporary set variable; beginning with $\bar{S} = \{1\}$, we compute the communications cost $B(\bar{S})$ of $\bar{S}$ and an assessment of the error $E(\bar{S})$ associated with using $\bar{S}$ (which we return to momentarily), giving the total cost $f(B(\bar{S}), E(\bar{S}))$. We then choose the element $\bar{s} \in \bar{S}$ which has largest contribution to the error bound for $\bar{S}$, as described in Section 5.5.4. Replacing $\bar{s}$ with its children $\bar{s}_L$, $\bar{s}_R$ in the set $\bar{S}$, we repeat the procedure, until finally $\bar{S}$ consists of only the leaf nodes. To define our approximation $\hat{m}_S(x)$ we take S to be the set $\bar{S}$ which had the best combined cost $f(B(\bar{S}), E(\bar{S}))$. This is easily found by initializing the procedure with $S = \bar{S} = \{1\}$ and taking $S = \bar{S}$ whenever $f(B(\bar{S}), E(\bar{S}))$ is less than $f(B(S), E(S))$.

The only aspect remaining to be specified is the assessment of error $E_{\bar{S}}$ for a given set $\bar{S}$. Estimating the actual error between $m(x)$ and $\hat{m}_{\bar{S}}(x)$ directly is feasible, and requires $\mathcal{O}(N)$ operations per estimate. Then, the optimization procedure just described requires $\mathcal{O}(N^2)$ operations in total.

However, this cost can be reduced significantly if we are willing to substitute the actual error between $m(x)$ and $\hat{m}_{\bar{S}}$ with the upper bound on the error given in Section 5.5.4. In this case the entire procedure requires at most $\mathcal{O}(N \log_2 N)$ operations, the cost of computing the upper bound. For $N = 1000$, the computational improvement is over two orders of magnitude, and experimentally has only a slight impact on the approximation obtained, since the bound appears relatively tight when the error is small. Thus, in practice we typically elect to use the bound, rather than recompute the error for each set $\bar{S}$. Pseudocode for the optimization procedure is given in Figure 5.7.

Empirically, the optimization procedure is often faster than this might suggest. If we have a maximum acceptable communications cost B_{max}, so that we can never select a set with greater communications cost, we can stop the iterative refinement procedure as soon as $\bar{S}$ exceeds this cost. This means that the procedure stops at some maximum number of components, or alternatively at some maximum depth in the KD-tree, which is a function of only B_{max} and independent of N. Thus for large N and relatively small communications budgets B_{max}, the algorithm requires only $\mathcal{O}(N)$ operations total. Similar behavior is observed when minimizing communications such that the error is less than some value value E_{max}; intuitively, we can think of this procedure as similarly refining to a maximum number of components (or depth in the KD-tree), where the number of components is determined by the complexity of the underlying distribution and the error tolerance E_{max}, rather than a maximum communications cost.

The optimization scheme we have described here is a greedy method, in which we select the node whose contribution to the error is highest to be improved at each step. However, it is interesting to note that, if we measure the error E_S for a set S using the upper bound of Section 5.5.4 on the maximum log–error, the same procedure results

Construct KD-tree representation of $m(x)$ and initialize $S = \bar{S} = \{1\}$.
Compute node 1's communications $B(1)$, and error $E(1)$.
While total communications $B(\bar{S}) = \sum_{s \in \bar{S}} B(s) \leq B_{max}$,

- Find $\bar{s} = \arg\max_{s \in \bar{S}} E(s)$.
- Exclude $\bar{s}$ and include its children: $\bar{S} = \bar{S} \setminus \bar{s} \cup \bar{s}_L \cup \bar{s}_R$
- For left and right child nodes $\bar{s}_L$, $\bar{s}_R$, compute
  - Error $E(\bar{s}_L)$, $E(\bar{s}_R)$ associated with each node
  - Communications $B(\bar{s}_L) = B(\bar{s}_R) = \frac{1}{2}(B(\bar{s}) +$ cost of refining node $\bar{s})$
- Assess the total error $E(\bar{S})$, by direct evaluation or using the upper bounds of Section 5.5.4.
- If the cost $f(B(\bar{S}), E(\bar{S})) < f(B(S), E(S))$, set $S = \bar{S}$.

Return $\hat{m}_S$, the density associated with the set S.

Figure 5.7. Greedy algorithm for approximating a kernel density estimate $m(x)$ subject to maximum communications B_{max}, with cost function $f(B, E)$ describing the relative importance of communication costs B and errors E, by optimizing over Gaussian mixtures defined by a KD-tree.

in an exact optimization. This is because, for the maximum log–error Δ, our bound is dominated by the maximum error over any of the nodes $s \in S$, i.e., the node we select for refinement. If we did *not* refine this node, we would never improve the error bound. Choosing any other node to refine only increases the communications B; since $f(\cdot)$ is increasing in both B and E, this can only result in a greater cost. Thus, under these conditions, the same greedy procedure is guaranteed to select the best possible set S.

■ 5.6 Adaptive Resolution

In the analysis up to this point, we have assumed a fixed, known resolution β for all means and bandwidths. However, when encoding relatively smooth portions of a distribution, for example components which have large bandwidths, it is far less important to transmit the parameter values to a high precision. Slightly varying the mean of a wide Gaussian, for instance, has very little overall effect. Ideally, we would like to reduce communications by not sending the superfluous bits.

We may entertain a number of possible modifications to the KD-tree encoder of Section 5.5. While a fully adaptive bit resolution, with separate resolution requirements for each mean and covariance value, is very flexible, *specifying* the resolution for each value may add enough overhead to negate or even overwhelm any benefit we might obtain. A more conservative option is to determine a single bit resolution β which is sufficiently high for all variables; once specified, our problem reverts to the same situation discussed previously. However, the joint optimization over the resolution β and the selection of components S may be somewhat complicated. At worst case,

of course, we may simply perform the optimization of Section 5.5 for each possible resolution β, and select the best combination.

An adaptive or floating–point resolution may be particularly appropriate for the bandwidths h_i, since for a hierarchical representation the initial, coarse–scale approximations and final, fine–scale components have very different requirements. In particular, if we elect to use the same fixed–point resolution at all scales of the tree, we must be careful to select a sufficiently fine resolution to accurately represent the smallest values of the h_i. This is because the effect of rounding the bandwidth h_i to zero is essentially catastrophic for any of the error measures considered, as it creates an ill–defined Gaussian mixture. If a minimal value of h_i is known, we could elect to quantize $\log_2 h_i$, rather than h_i itself, to avoid some of these difficulties. If floating–point or logarithmic representations for h_i cannot be used, another possibility is to round small bandwidth values upward by convention.

■ 5.7 Experiments

This section describes a few example applications of KD-tree based density approximation and the tradeoff between error and communications. First, we show the process of approximation on a single message, then examine the performance in two example multi-sensor systems: a distributed particle filtering application, and estimation of a spatially dependent non-Gaussian random process.

■ 5.7.1 Single Message Approximation

We begin by taking a fixed kernel density estimate and showing the sequence of approximations which are made as communication constraints are relaxed. The original kernel density estimate is made up of 100 samples, and we transmit all parameters up to $\beta = 16$ bits of resolution; thus the most naive approach to lossless encoding, direct representation, requires 1600 bits. Figure 5.8 shows the first eight Gaussian sums in the sequence of approximations (dashed) to the original density (solid) made by our KD-tree splitting algorithm. Communications cost and max–log error are listed for each approximation. In this case, the sequence of refinements to optimize the max–log error and KL-divergence was exactly the same up to 15 components; for simplicity of presentation we have not listed the numerical values of the KL-divergence or its bound.

We also show the sequence of improvements in our error bound (and in the actual resulting error) as a function of the number of bits required for transmission. Figure 5.9(a) shows the rate at which the resulting max–log error (solid) and its bound (dashed) decline as communications increase; Figure 5.9(b) shows the same trend for the KL-divergence measure. It is clear that beyond a certain communications level, we gain very little for any additional expenditure of energy.

Finally, we can also consider changing and optimizing over the resolution β of the quantization. However, there are several issues with doing so. As discussed, if β is chosen too small we require special precautions when representing the bandwidth. Also,

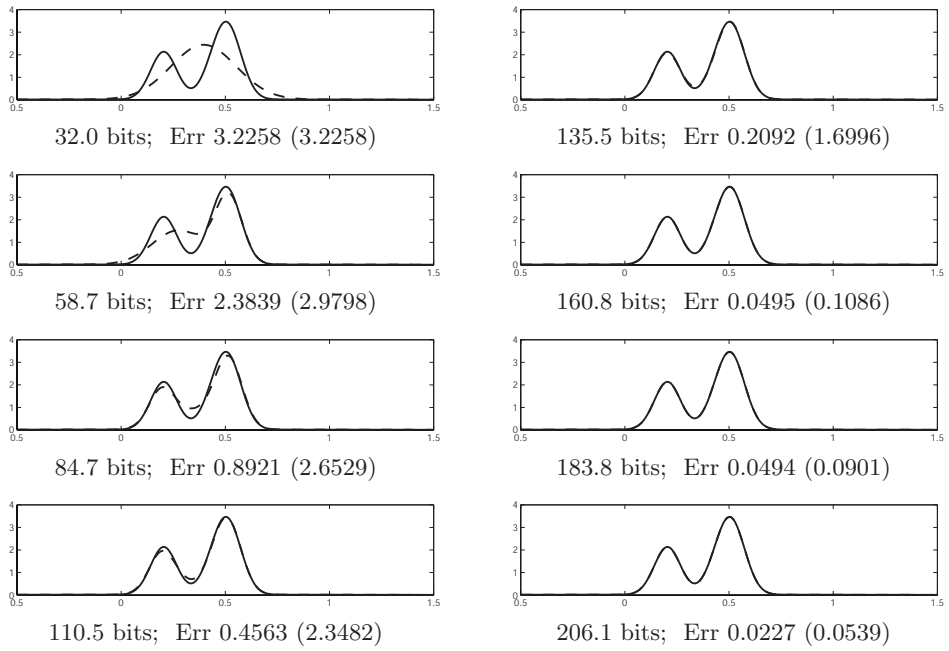


Figure 5.8. Sequence of KD-tree based approximations (dashed) to a 100–kernel density estimate (solid) of decreasing error and increasing communications cost (with $\beta = 16$ bits). Listed are the transmit cost in bits, and the actual error and tree–decomposed bound on Δ.

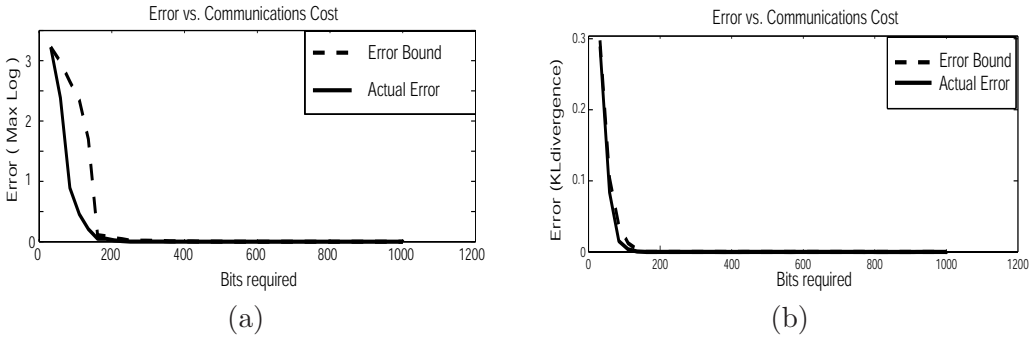


Figure 5.9. Comparing transmitted density error (both the tree–structured bound and actual error) versus total communications cost (in bits) for both (a) max–log error and (b) KL-divergence. In both cases, very few bits are required to transmit most of the density's information.

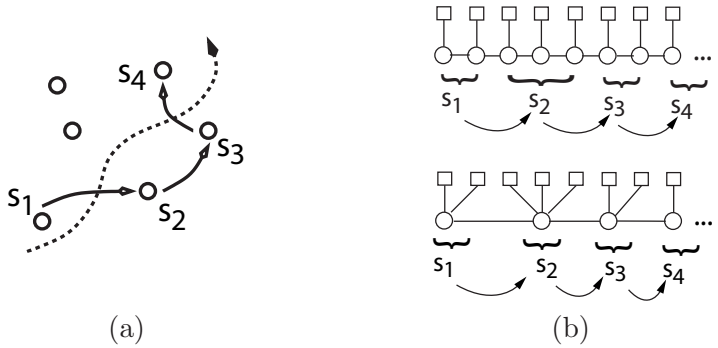
the two optimizations, over β and the retained mixture components S, are coupled. For example when measuring error using the maximum log–error measure, if β is decreased to only 8 bits, we improve performance at moderate error levels—sending 6 Gaussians costs about 62 bits, with error bound of 2.23 and actual error of .330. We may compare this performance to the similar communications cost of 2 Gaussians totalling 59 bits, with error bounds 2.98 and actual error 2.38 at $\beta = 16$ bits. However, at some point the quantization error begins to dominate—i.e., for $\beta = 8$ bits, sending more than 6 Gaussian components never improved the resulting maximum log–error.

■ 5.7.2 Distributed Particle Filtering

Particle filtering is often used for single– or multi–target tracking involving highly non-Gaussian observation likelihoods and potentially non-linear dynamics. When sensors are myopic, i.e., only observe objects which are nearby to their own location, and constrained by a limited power budget, it is typical to perform the operations of particle filtering at some sensor which is nearby to the object itself. Using a local representation removes the need to export data from the network and thus reduces the distance over which sensor observations must be communicated [114]. The sensor in charge of data fusion has been called the "leader" node [113]. At each time t, the leader node and potentially a few other sensors collect observations, for example measurements of the range of the object of interest. These measurements are then transmitted to the leader, who uses them to update the current estimate of the posterior distribution.

Because the object is moving, however, the most appropriate leader node is also a function of time. Therefore, at each time t, the leader also uses its estimate of the posterior distribution to select a new leader node at time $t + 1$. The old leader may, of course, select itself as the new leader; but if not, it must also communicate the current model of the posterior distribution to the new leader. This procedure is depicted in Figure 5.10(a). If the distribution is estimated using a nonparametric representation, the naive cost of this communication can be hundreds or even thousands of times larger than any single measurement communication.

There are any number of possible protocols for selecting when, and to which sensor, the leader should transfer control; for one example, see [114]. Another related issue is the decision of which sensors should collect and transmit measurements to the leader at each time step. However, we shall treat both of these questions as fixed aspects of the leadership protocol, and focus only on minimizing the communications cost inherent in any given strategy. Since the leadership sequence is assumed to be fixed, we may also ignore the distance–dependent aspect of communications cost, i.e., the fact that transmitting information to a nearby sensor is often cheaper than the same transmission to a distant sensor, and concentrate on applying the preceding sections' analysis to minimize the representation size as measured in bits. We assume that the selected leader changes at each time step; as depicted in Figure 5.10(b), this can be assumed without any loss of generality by simply grouping all measurements associated with each leader.



(a) (b)

Figure 5.10. (a) Repeated transfer of "leadership", along with the model of state uncertainty, while tracking in an ad-hoc sensor network. (b) Markov chain representation of the sequential state estimation problem; without loss of generality we may assume that the model is transmitted at each time step.

An important aspect which differentiates this problem from typical lossy data compression tasks is that the approximated data is being *used* at the next time step to construct a new density estimate, and that we are interested in minimizing not only the error in the *transmitted* density, but also its effect on subsequent estimates. In other words, we are interested not only in the error introduced in the distribution to reduce communications cost (which is directly calculable by the sender), but *also* and perhaps more importantly the error that this difference induces in the updated posterior estimates at each subsequent time step. The error measures of Chapter 4 come with theoretical assessments, in the form of bounds and estimates, on the subsequent inference errors which can result from a sequence of approximation steps; by controlling these measures of error we can obtain meaningful statements about the level of error in future inference.

We examine the tradeoff between communications and error experimentally for this problem by considering a simple two–dimensional particle filtering application. We simulate an object moving in two dimensions, with dynamics

$$x_t = x_{t-1} + v_0[\cos\theta_t; \sin\theta_t] + \omega_t \tag{5.18}$$

where ω_t is Gaussian and θ_t uniform:

$$\omega_t \sim \mathcal{N}(\omega\,;\,0, \sigma_\omega^2 I) \qquad\qquad \theta_t \sim U\left[-\frac{\pi}{4}, \frac{\pi}{4}\right] \tag{5.19}$$

At each time step t a single sensor (the leader) updates the estimated distribution using a range measurement from its location s_t:

$$y_t = \|x_t - s_t\| + d_t \qquad\qquad d_t \sim N(d; 0, \sigma_d^2) \tag{5.20}$$

The leader node is changed after each update (i.e., at each time step), and the updated distribution estimate communicated to a new sensor. In these experiments, the distribution $p(x_t|y_t, \ldots, y_1)$ at each time step t is typically unimodal but non-Gaussian. Since

at each time step t, the leader node must communicate its particle–based distribution estimate to another sensor, we may compare methods of compressing the messages.

We first create an estimate of the true posterior distributions at each time step by performing particle filtering using $N = 1000$ samples. At each step, this particle filter sends all N samples exactly, under no communications constraints. We refer to these posterior distributions as "exact", although they are technically estimates. In order to determine what level of accuracy we can expect from these estimates, we perform the same filtering process several times using multiple initializations (i.e., different random number seeds); by computing the error between the resulting density estimates we obtain an estimate of the level of error which is attributable to our choice of a finite value of N. This estimate indicates a lower bound on the error which is achievable by any of the three particle filters we compare to the "exact" filter, each of which use lossy approximations to the distributions at each communication step.

Each of these three approximate particle filters works in the same basic manner. At each time t, a message representing $p(x_t|y_{t-1}, \ldots, y_1)$ is compressed by the leader node at time $t-1$ and communicated to the leader at time t. Because it has been compressed, this message is in general not a collection of N particles, but rather some smaller mixture of Gaussian components. The leader at time t re-creates a collection of N particles by drawing i.i.d. samples from the message, and weights these samples according to the likelihood information given by the observation y_t. The node then samples from this collection N times with replacement to obtain N equally–weighted particles, and propagates these particles through the forward dynamics $p(x_{t+1}|x_t)$. This procedure results in a particle representation of $p(x_{t+1}|y_t, \ldots, y_1)$, which is then compressed and communicated to the leader at time $t + 1$ in the same manner. We consider three possible methods of message approximation, each parameterized by the total number of bits B required per message.

Subsampling—one way to approximate the N–particle density estimate $m(x)$ is to draw some $K_s < N$ particles from the distribution $m(x)$, and transmit those K_s particles exactly. We use an optimistic estimate of the cost of this transmission, by taking the minimum possible expected cost of sending the sample set (i.e., $K_s \, \mathrm{H}(p) - \log_2 K_s!$, as detailed in Section 5.3). The number of particles, K_s, is selected so that this minimum expected cost is less than B bits.

Expectation–Maximization (EM)—the EM algorithm [2] is a common method of fitting Gaussian mixture models to a collection of samples. Given a collection of equal–weight particles $\{x_t^j\}$ and a kernel bandwidth h_t which specify $m(x)$, and the number of desired mixture components K_{EM} in $\hat{m}(x)$, an iterative update procedure is used to determine the means, weights, and covariances of each component of $\hat{m}(x)$. However, since efficient encoding of such a mixture model is an open question, we simply choose the number of components K_{EM} to require fewer than B bits in a naive, direct fixed–point representation. For consistency with the rest of the Gaussian mixtures used in this chapter, we require that the covariance of each mixture component be diagonal, and represent it using a bandwidth parameter h_i.

Given the number of components K_{EM}, and an initial value of their means μ_i, weights w_i, and kernel bandwidths h_i for $i \in \{1 \dots K_{EM}\}$, we update the parameters $\{\mu_i, w_i, h_i\}$ by first finding the relative probability that each sample x_t^j was generated by each of the K_{EM} components, i.e.,

$$v_{ij} \propto w_i \mathcal{N}\left(x_t^j\,;\, \mu_i, \mathrm{diag}(h_i)\right)$$

with $\sum_i v_{ij} = 1$, then updating the parameters of the K_{EM} components by taking their maximum likelihood estimates given these relative probabilities, i.e.,

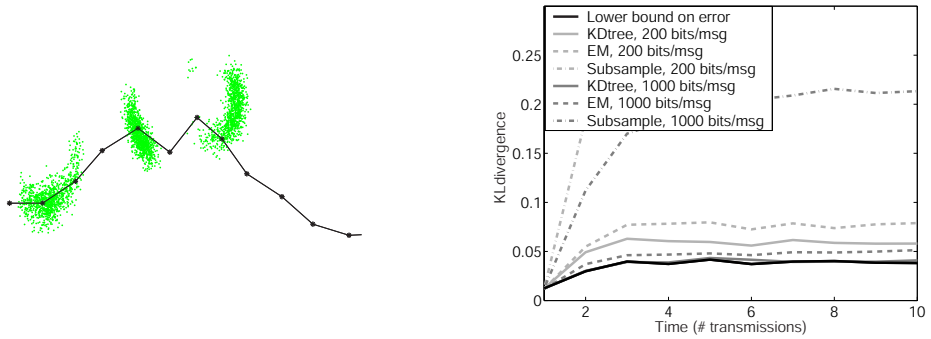
$$w_i = \sum_j v_{ij} \qquad \mu_i = \frac{1}{w_i} \sum_j v_{ij} x_t^j$$
$$h_i = \frac{1}{w_i} \sum_j v_{ij}\, \mathrm{diag}\left((x_t^j - \mu_i)(x_t^j - \mu_i)^T\right) + h_t$$

where h_t is the bandwidth of $m(x)$. Repeating this procedure, we eventually converge to a locally optimal estimate of the mixture parameters $\{\mu_i, w_i, h_i\}$. This procedure can be regarded as acting to minimize the KL-divergence between the original kernel density estimate and our Gaussian mixture approximation.

KD-tree optimization—we also compare to the KD-tree optimization methods described in Section 5.5. To obtain a fair comparison to EM, we use the algorithm described in Section 5.5.5 to minimize the KL-divergence between our original density estimate and the approximation, subject to the communications cost (as defined by the encoder of Section 5.5.2) being less than B bits.

In all cases, the parameters of the Gaussian mixtures are communicated up to resolution $\beta = 16$ bits. We compare the resulting posterior distributions at each time step to the "exact" posteriors obtained by our original particle filter. In Figure 5.11 we plot the resulting average KL-divergence of the estimated posteriors obtained using approximate messages from those obtained using our "exact" particle filter over 500 Monte Carlo trials, for all three approximation methods and $B \in \{200, 1000\}$ bits. Also shown is the lower bound which estimates the amount of error attributable to our choice of a finite N. For a given bit budget, smart approximation of the distribution (using either EM or KD-tree based methods) performs considerably better than simple subsampling. The KD-tree based method performs the best; given a bit budget of 1000 bits its performance is nearly indistinguishable, on average, from the lower bound.

The EM–based approximation also does well, though not quite as well as the KD-tree method. This is likely due to two factors: first, being less constrained, EM may be more prone to finding local maxima while fitting; second, the KD-tree's additional constraints are used to define an efficient encoding procedure, so that for a given bit budget B the KD-tree typically allows more mixture components to be used than the naive encoding of the mixture model found via EM. Perhaps given a more efficient encoder of arbitrary Gaussian mixtures, the performance of EM would be similarly improved; this is one important direction for further research.



(a) (b)

Figure 5.11. Particle filtering example. (a) One sample path $\{x_t\}$, along with the samples used to estimate the posterior distribution at times $t = 2, 5, 8$. (b) Average KL-divergence at each time step for approximate message–passing. Careful approximations (EM and KD-tree) perform much better than subsampling; the KD-tree approximations (solid) perform best, and compare favorably with the lower bound estimated under unconstrained communications (black).

As a final point, we could instead have optimized our KD-tree structure to minimize the maximum–log error, or any of the other measures discussed. While the maximum log–error has better theoretical guarantees, in practice it appears to be a less accurate gauge of average–case behavior, perhaps due to its more conservative nature. For the experiment above, a maximum–log optimized KD-tree also performed much better than the subsampling approach, and was still comparable to optimal performance at 1000 bits, but at low bit rates ($B = 200$ bits) was outperformed on average by the EM–trained mixture model and the KD-tree optimized for KL-divergence.

■ 5.7.3 Non-Gaussian Field Estimation

We next consider another use of sensor networks—to fuse a collection of spatially separated observations. Suppose that we have a collection of sensors, arranged in an 8×8 regular grid. Each sensor i obtains an observation about a two–dimensional random variable, denoted x_i, which is known to vary slowly in space but possesses a few sharp transitions.

We employ a multi-resolution quad-tree model to capture the interactions between the x_i. Similar models have been used with considerable success for efficient estimation of Gaussian fields [110]. To be precise, we associate each of the x_i to the finest scale (leaf) node of a quad-tree which corresponds, in spatial arrangement, to sensor i. Each non-leaf node of the tree is also associated with a random variable; we use $\gamma^1 x_i$ to indicate the random variable associated with the parent node of node i, $\gamma^2 x_i$ to be the parent of that node, and so forth. For notational consistency we define $\gamma^0 x_i = x_i$.

To capture local smoothness with the possibility of sharp transitions, we model the

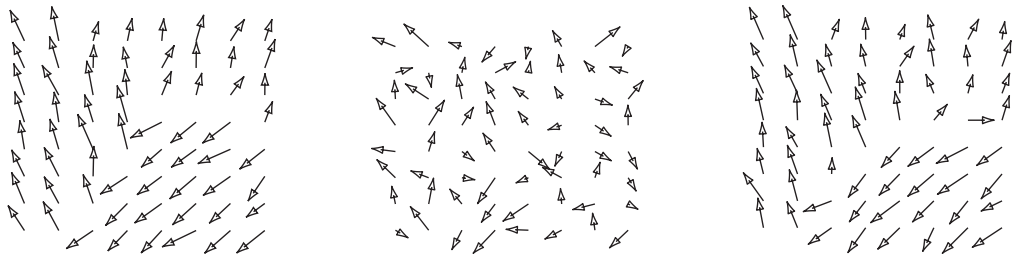


Figure 5.12. Example two–dimensional non-Gaussian field. (a) True state at each of the 64 sensors, indicated by vectors. (b) Mean of the individual observation likelihoods at each sensor. (c) Maximum of each posterior distribution, estimated using NBP.

interactions between variables by a simple mixture of Gaussians:

$$\begin{aligned}\psi(\gamma^a x_i|\gamma^{a-1}x_i) &= \psi(\gamma^{a-1}x_i|\gamma^a x_i) \\ &= .9\mathcal{N}(\gamma^a x_i - \gamma^{a-1}x_i; 0, \sigma_a^2 I) + .1\mathcal{N}(\gamma^a x_i - \gamma^{a-1}x_i; 0, I);\end{aligned} \tag{5.21}$$

where the variance σ_a^2 controls the desired smoothness of the field, and depends on the scale a within the quad-tree; we select $\sigma_1 = .05$, $\sigma_2 = .1$, and $\sigma_3 = .2$. The smaller, high–variance mode allows for the possibility of sharp disagreements between neighboring x_i in the finest–scale grid.

We choose to represent the two–dimensional likelihood messages $p(y_i|x_i)$ as sample–based density estimates, performing fusion of the messages by sampling from their products using nonparametric belief propagation (NBP) as described in Chapter 3. In the quad-tree structure, optimal inference can be performed via a simple two–pass sequence: first, messages are passed upward from the leaf nodes to the root and fused at each level, then the fused results are sent back downwards to the leaves. An example of the true underlying state of each x_i along with the mean value of each individual likelihood $p(y_i|x_i)$ and the estimate computed using NBP for comparison are shown in Figure 5.12. For the NBP estimate shown we have allocated 1000 bits per message, the maximum representation size we consider in this problem.

We impose the statistical quad-tree structure shown in Figure 5.13(a) onto the physical sensor and communications structure by associating each of the "virtual" parent nodes to the same sensor as one of its four children. Thus, at each level in the upward sweep, three nodes transmit, and thus may wish to approximate, their messages to the parent; in the downward sweep, the parent node transmits to the other children. This can be done using a single message, by simultaneously broadcasting its belief to all neighboring nodes (as described in Section 3.6). The upward sweep of the message transmission schedule is depicted in Figure 5.13(b). After the message–passing process concludes, most sensors have sent only one message, while a few (about $\frac{1}{4}$) have sent two.

We may compare the quality of the fusion results as a function of the number of bits

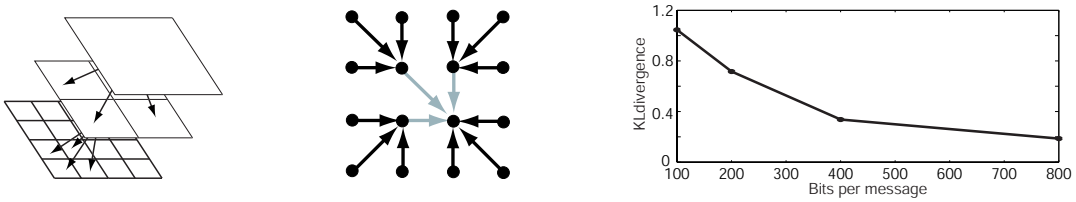


Figure 5.13. Quad-tree structure for inference in a sensor network. (a) An example (4×4) quad-tree structure. (b) Allocating the nodes in the quad-tree to sensors; responsibility for each parent node is assumed by one of the children. Arrows indicate the upward message sweep, from leaf nodes to root. (c) Error, in terms of KL-divergence, of the solution as a function of the allowed number of bits per message.

allocated to each message, applying the KD-tree based approximation of Section 5.5. Figure 5.13(c) shows the resulting KL-divergence between estimated and true posterior distributions as a function of the number of bits, averaged over 500 Monte Carlo trials. As with the particle filtering example, reasonably good results are obtained even for relatively small messages (less than 1000 bits required to represent messages over a two–dimensional state space).

■ 5.8 Some Open Issues

There are a number of open problems in which we have not managed to address in this chapter. Perhaps chief among them is the issue of *iterative* methods of communications in message–passing algorithms for inference. In many sequential message–passing algorithms, we may have already passed a representation of some of the available information and wish to determine whether and how best to update that information so as to aid in the overall goals of inference. This type of scenario may arise in belief propagation on graphs with loops, and even in tree–structured graphs where it is desirable to have sensor nodes make time–critical decisions. In these cases, each node may begin by sending its local information to its neighbors, then refining and updating this message with more transmissions as it receives additional information from neighboring sensors.

For iterative, multi-transmission problems, there exist several additional means of reducing communications. By *censoring*, or opting not to send, certain messages which are "sufficiently close" to their previously transmitted versions, we may be able to reduce the network's required communications considerably [10]. Furthermore, even if the next message transmission is not censored, its previous version provides a type of *prior* for the samples representing the updated message. Using this prior, however, requires some idea of how (and how much) messages change from iteration to iteration, a difficult and potentially application–dependent question.

Non-local statistics can also be used to lower the communications cost. If the transmitter and receiver share additional sources of information, including, for example, access to the full joint distribution represented by the graphical model, they may be

able to use this knowledge to compress the message still further [91]. Feedback from the receiver also provides a potential source of savings. Iterative message–passing methods can make use of simple feedback in their evaluation of error and the impact of approximations. For example, in stochastic message approximation, received messages can be used to focus sampling [48, 56]; we provide an experimental analysis of a similar approach in the next chapter (see Section 6.7).

■ 5.9 Conclusions

Power–limited wireless sensor networks must be able to perform inference in a communications–constrained environment. We consider an important subset of this general task, that of inference on continuous–valued random variables using sample–based representations, the most common example of which is particle filtering. We discuss the cost of transmitting such representations, both exactly and approximately.

For exact (lossless) communications, we showed that the representation's invariance to reordering can be used to reduce the required communications cost, and that to do so we must take advantage of predictable non-stationarity in the distribution of the deterministically ordered samples. We also described a simple sub-optimal linear predictive encoder for one–dimensional samples which provided some of these benefits.

To treat more general problems, including approximate (lossy) representations, we applied the KD-tree data structure to the tasks of both encoding and density approximation, demonstrating how communications cost may be efficiently balanced with errors in inference. We then showed several examples demonstrating lossy encoding for distributed inference, including a distributed implementation of particle filtering and a multi-resolution model for estimating a non-Gaussian random field.

Many important questions remain open for future research, however. It may be possible to improve the way the resolution β of the samples is selected. Additionally, feedback and iterative transmission of updated messages provide important sources of information which we have not exploited. Lastly, and perhaps most importantly, we have only described a few examples of possible encoding methods. We can expect that further investigation may result in additional, perhaps substantially improved schemes for communicating sample–based distributions and their approximations.

Chapter 6

Sensor Self-Localization

IN this chapter, we focus on a particular task inherent to sensor networks, bringing the results of the previous chapters to bear on the problem of self-localization for an ad-hoc deployment of wireless sensor nodes. Sensor localization, i.e., obtaining estimates of each sensor's position as well as accurately representing the uncertainty of that estimate, is a critical step for effective application of large sensor networks to almost all subsequent tasks. Manual calibration[1] of each sensor may be impractical or even impossible, and equipping every sensor with a GPS receiver or equivalent technology may be cost prohibitive. Consequently, methods of *self-localization* are desirable; we can exploit relative information, perhaps obtained from received signal strength measurements via the wireless communication or from measuring time delay between sensors, along with a limited amount of global reference information as might be available to a small subset of sensors, to estimate a location and its uncertainty for each sensor in the network. In the wireless sensor network context, the process of localization is further complicated by the need to minimize inter-sensor communication in order to preserve energy resources.

We present a localization method in which each sensor has available noisy measurements of its distance to several neighboring sensors. In the special case that the noise on distance observations is well modeled by a Gaussian distribution, localization may be formulated as a nonlinear least-squares optimization problem. In [73] it was shown that a relative calibration solution which approached the Cramer-Rao bound could be obtained using an iterative, non-linear least-squares optimization approach.

In contrast, we reformulate sensor localization as an inference problem defined on a graphical model. This allows us to apply nonparametric belief propagation (NBP; Chapter 3) to obtain an approximate solution. This approach has several advantages—it exploits the local nature of the problem, in the sense that a given sensor's location estimate depends primarily on information about nearby sensors. It also naturally allows for a distributed estimation procedure. Furthermore, it is not restricted to Gaussian measurement models, which may be overly restrictive for real–world systems. Finally, it produces both an estimate of the sensor locations and a representation of the location

[1] Within the context of this chapter, we use the term *localization* interchangeably with the more general term *calibration* in sensor networks.

uncertainties. The last point is notable for random sensor deployments, in which multi-modal uncertainty in sensor locations is a frequent occurrence. Furthermore, estimation of the uncertainty in sensor positions, whether multi-modal or not, provides guidance for expending additional resources in order to obtain better, more refined solutions.

In the subsequent sections, we describe the sensor localization problem in more detail and relate it to inference in graphical models. In Sections 6.1–6.2, we formalize the problem and discuss the types of uncertainty which occur in localization. As we show, sensor localization can often have multiple solutions with equal or nearly–equal quality, indicating that the problem, in these cases, is fundamentally ill–posed. In Section 6.3 we examine an idealized version of the localization problem in order to characterize when the task is likely to be well–posed. Section 6.4 re-formulates the localization problem as a graphical model, and presents a solution based on the NBP algorithm of Chapter 3. We show several empirical examples demonstrating the ability of NBP to solve difficult distributed localization problems. We also include three modifications which can improve NBP's performance in practical applications: Section 6.6 shows how NBP may be augmented to include an outlier model in the measurement process, and demonstrates its improved robustness to non-Gaussian measurement errors; Section 6.7 presents an alternative sampling procedure which may improve the performance of NBP in systems with limited computational resources; and Section 6.8 uses the results of Chapter 5 to consider the communication costs inherent in a distributed implementation of NBP, and provides simulations to demonstrate the inherent tradeoff between communication and estimate quality in localization.

■ 6.1 Self-localization of Sensor Networks

We begin by describing a statistical framework for the sensor network self-localization problem, similar but more general than that given in [72]. We restrict our attention to cases in which individual sensors obtain noisy distance measurements of a (usually nearby) subset of the other sensors in the network. This type of problem includes, for example, scenarios in which each sensor is equipped with either a wireless or acoustic transceiver and inter-sensor distances are estimated by measuring the received signal strength or time delay of arrival between sensor locations. Typically, this measurement procedure can be accomplished using a broadcast transmission (acoustic or wireless) from each sensor as all other sensors listen [72, 108].

While the framework we describe is not the most general framework possible, it is sufficiently flexible to be extended to more complex scenarios. For instance, our method may be easily modified to fit cases in which sources are not co-located with a cooperating sensor, to incorporate direction-of-arrival information (which also necessitates estimation of the orientation of each sensor) [72], or to perform simultaneous estimation of other sensor characteristics such as transmitter power [108].

Let us assume that we have K sensors scattered in a planar region, and denote the two-dimensional location of sensor t by x_t. The sensor t obtains a noisy measurement

d_{tu} of its distance from sensor u with some probability $P_o(x_t, x_u)$:

$$d_{tu} = \|x_t - x_u\| + \nu_{tu} \qquad \nu_{tu} \sim p_\nu(x_t, x_u) \tag{6.1}$$

We use the binary random variable o_{tu} to indicate whether this observation is available, i.e. $o_{tu} = 1$ if d_{tu} is observed, and $o_{tu} = 0$ otherwise. Finally, each sensor t has some prior distribution, denoted $p_t(x_t)$. For many of the sensors, the prior $p_t(x_t)$ may be an uninformative one. Then, the joint distribution is given by

$$p(x_1, \ldots, x_K, \{o_{tu}\}, \{d_{tu}\}) = \prod_{(t,u)} p(o_{tu}|x_t, x_u) \prod_{(t,u):o_{tu}=1} p(d_{tu}|x_t, x_u) \prod_t p_t(x_t) \tag{6.2}$$

The typical goal of sensor localization is to estimate the maximum a posteriori (MAP) sensor locations x_t given a set of observations $\{d_{tu}\}$. Of course, there is a distinction between the individual MAP estimates of each x_t versus the MAP estimate of all $\{x_t\}$ jointly. For this work, it is convenient to select the former. In a system of binary–valued random variables, selecting the individual ML estimates would correspond to minimizing the average number of errors, as opposed to minimizing an "all-or-nothing" probability of error, which corresponds to the joint ML estimate.

Technically, the measured distances d_{ut} and d_{tu} may be different, and it is even possible to have $o_{ut} \neq o_{tu}$ (indicating that only one of the sensors u and t can observe the other). However, for the purposes of our development it is convenient to assume that both sensors obtain the same, single observation, i.e., that $d_{ut} = d_{tu}$ and $o_{ut} = o_{tu}$. We include remarks on differences which arise in the more general case, and how we may deal with these differences.[2]

Additionally, the amount of prior location information may be almost nonexistent. In this case, we may wish to solve for a *relative* sensor geometry, as opposed to estimating the sensor locations with respect to some absolute frame of reference [73]. Given only the relative measurements $\{o_{tu}, d_{tu}\}$, the sensor locations x_t may only be solved up to an unknown rotation, translation, and negation (mirror image) of the entire network. We avoid ambiguities in the relative calibration case by assuming priors which enforce known conditions for three sensors (denoted s_1, s_2, s_3):

1. *Translation*: s_1 has known location (taken to be the origin: $x_1 = [0; 0]$)
2. *Rotation*: s_2 is in a known direction from s_1 ($x_2 = [0; a]$ for some $a > 0$)
3. *Negation*: s_3 has known sign ($x_3 = [b; c]$ for some b, c with $b > 0$).

Typically s_1, s_2, and s_3 are taken to be spatially close to each other in the network. When our goal is absolute calibration (calibration with respect to a known coordinate

[2]When d_{tu} and d_{ut} are independent observations of the distance, and there is some possibility that $o_{ut} \neq o_{tu}$, we are first required to symmetrize the observations, i.e., exchange information between any two sensors which observe either d_{tu} or d_{ut}, so that both sensors know the values of all four random variables. This process of exchanging and symmetrizing information may involve multi-hop message routing or other communication protocols which are beyond the scope of this thesis.

reference), we simply assume that the prior distributions $p_t(x_t)$ contain sufficient information to resolve this ambiguity. The sensors with significant prior information (or s_1, s_2, and s_3 for relative calibration) are referred to as *anchor* nodes.

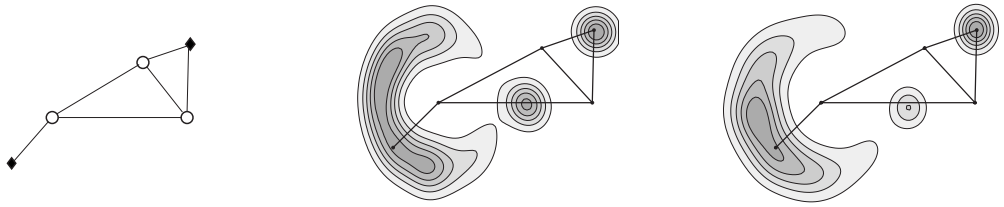
In general finding the MAP sensor locations is a complex nonlinear optimization problem. If the uncertainties p_ν, p_t described previously are Gaussian and the probability of observing a distance $P_o(x_u, x_t)$ is assumed constant (i.e., not a function of x_u and x_t), maximum likelihood joint estimation of the sensor locations $\{x_t\}$ reduces to a nonlinear least-squares optimization [72]. In the case that we observe distance measurements between *all* pairs of sensors (i.e., $P_o(\cdot) \equiv 1$), this optimization problem also corresponds to a well studied distortion criterion (known as the "STRESS" criterion) in multidimensional scaling problems [98]. However, for large-scale sensor networks, it is reasonable to assume that only a subset of pairwise distances will be available, primarily between sensors which are located within the same region. One model, proposed by [73], assumes that the probability of detecting nearby sensors falls off exponentially with squared distance, so that

$$P_o(x_t, x_u) = \exp\left(-\frac{1}{2}\left\|x_t - x_u\right\|^2 / R^2\right). \tag{6.3}$$

We use (6.3) in our example simulations, though alternative forms are equally simple to incorporate into our framework. This flexibility leaves open the possibility of estimating the probability of detection P_o from training data, if available; experiments to estimate these probabilities have already been performed for certain models of wireless sensors and measurement methods [108].

A large number of methods have been previously proposed to estimate sensor locations [18, 59, 78, 81, 85, 108]. An exhaustive categorization of all the methods which have been applied to localization is beyond the scope of this chapter; here we briefly describe only a few. For better or worse, many of these methods eschew a statistical interpretation of the sensor localization task in favor of algorithmic or computational simplicity. Some examples include using the distances which have been observed in the network to approximate the distances between each pair of sensors which did not measure their distance, and then estimating the positions by applying classical multidimensional scaling [98], multi-lateration [85], or other techniques [59]. Other approaches search for sensor locations which satisfy rigid convex distance constraints [18]. Yet another method heuristically minimizes the rank of the matrix of squared distances, while preserving the fidelity of the distances which have been observed [22].

However, these algorithms often lack a direct statistical interpretation, and as one consequence rarely provide an estimate of the remaining uncertainty in each sensor location. Iterative least-squares methods [72, 78, 81, 85] *do* have a straightforward statistical interpretation, but assume a Gaussian model for all uncertainty, which may be questionable in practice. As we discuss in Section 6.2, non-Gaussian uncertainty is a common occurrence in sensor localization problems. Often, since the posterior uncertainty is *not* Gaussian and in general has no convenient closed form, the Cramer-Rao



(a) (b) (c)

Figure 6.1. Example sensor network. (a) Sensor locations are indicated by symbols and distance measurements by connecting lines. Calibration is performed relative to the three sensors drawn as circles. (b) Marginal uncertainties for the two remaining sensors (one bimodal, the other crescent-shaped), indicating that their estimated positions may not be reliable. (c) Estimates of the same marginal distributions using NBP.

bound is used to characterize the residual uncertainty given a set of measurements. However, the Cramer-Rao bound may be an overly optimistic characterization of the actual uncertainty in sensor location, particularly if the true posterior distribution is multi-modal. Estimating which, if any, sensor positions are unreliable is an important task when parts of the network are under-determined. Simulations in Section 6.3 suggest that under-determined networks of sensors may be surprisingly common. Furthermore, Gaussian noise models may be inadequate for real-world noise, which often possesses some fraction of highly erroneous (outlier) measurements.

In this chapter we pose the sensor localization problem as an inference task defined on a graphical model, and propose an approximate solution making use of the non-parametric belief propagation (NBP) algorithm. NBP allows us to apply the general, flexible statistical formulation described above, and can capture the complex uncertainties which occur in localization of sensor networks, described next.

■ 6.2 Uncertainty in sensor location

The sensor localization problem as described in the previous section involves the optimization of a complex nonlinear likelihood function (6.2). However, it is often desirable to also have some measure of confidence in the estimated locations. Even for Gaussian measurement noise ν, the nonlinear relationship of inter-sensor distances to sensor positions results in highly non-Gaussian uncertainty of the sensor location estimates.

For sufficiently small networks it is possible to use Gibbs sampling [26] to obtain samples from the joint distribution of the sensor locations. In Figure 6.1(a), we show an example network with five sensors. Calibration is performed relative to measurements from the three sensors marked by open circles, with the remaining two sensors marked by filled diamonds. A line is shown connecting each pair of sensors which obtain a distance measurement. Contour plots of the marginal distributions for the two remaining sensors are given in Figure 6.1(b); these sensors do not have sufficient information to be well-localized, and in particular have highly non-Gaussian, multi-modal uncertainty,

suggesting the utility of a nonparametric representation. Although we defer the details of the NBP-based solution to localization until Section 6.4.3, for comparison an estimate of the same marginal uncertainties performed using NBP is displayed in Figure 6.1(c).

■ 6.3 Uniqueness

The example network in Figure 6.1 is suggestive of the fact that the residual sensor uncertainty may be multi-modal. In fact, there may be more than one set of estimated locations which fit the relative measurements equally well, i.e., the problem may not have a unique solution [72]. It is useful to know how often this type of situation occurs in practice. To address this question, we examine an idealized situation in which the existence of a uniquely determined solution is more readily quantified. Let us take K sensors which are distributed at random (in a spatially uniform manner) within a planar circle of radius R_0, and let

$$P_o(x_t, x_u) = \begin{cases} 1 & \text{for } \|x_t - x_u\| \leq R_1 \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

so that sensors t and u obtain a measurement of their distance d_{tu} if and only if $d_{tu} \leq R_1$. We consider the problem of relative calibration, and thus assume that no prior location information is available to any sensor so that for all t, $p_t(x_t)$ is uninformative. We further assume that the uncertainty due to noise ν_{tu} present in each measurement d_{tu} is negligibly small. An example of sensors distributed in this manner is given in Figure 6.2(a).

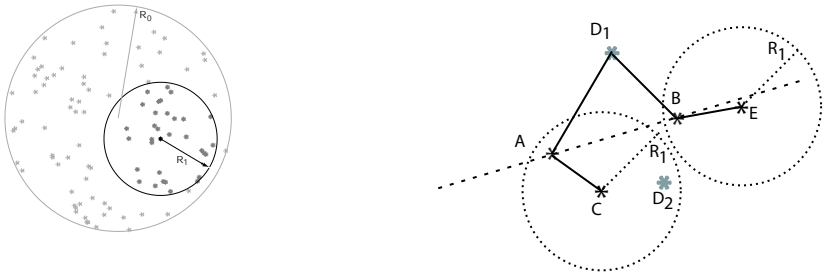
As previously discussed, without prior knowledge of the absolute location of sensors in the network this problem can only be solved up to an unknown rotation, translation, and negation [74]. Therefore, we assume a minimal set of known values; in the negligible-noise case and assuming these sensors are mutually co-observing this is equivalent to assuming known locations for three sensors. Without loss of generality, we take $x_1 = [0; 0]$, $x_2 = [0; d_{12}]$, and $x_3 = [b; c]$, where

$$b = \sqrt{d_{13}^2 - c^2} \qquad\qquad c = \frac{d_{12}^2 + d_{13}^2 - d_{23}^2}{2d_{12}}$$

■ 6.3.1 A sufficient condition for uniqueness

We now derive a sufficient condition for all sensors to be localizable, i.e., to have a uniquely determined relative location given the measurements. Some subtleties arise if any sensors are perfectly co-linear; however, under our model for sensor dispersal this occurs with probability zero and we proceed to describe conditions which are sufficient for uniqueness with probability one. This same sufficient condition (called a *trilateration graph*) has also recently been investigated by Eren et al. [21].

Let S be the set of nodes which are localizable (with probability one), and let "$\sim$" denote the binary, symmetric relation of observing an inter-sensor distance. It is



(a) (b)

Figure 6.2. (a) K sensors distributed uniformly within radius R_0, each sensor seeing its neighbors within radius R_1. (b) The two potential locations of sensor D (denoted D_1 and D_2) given distance measurements from sensors A and B is resolved by the *lack* of observation at sensor C, while sensor E has no additional information about the position of D.

straightforward to show that

$$s_A, s_B, s_C \in S \quad \text{and} \quad s_D \sim s_A, \quad s_D \sim s_B, \quad s_D \sim s_C \quad \Rightarrow \quad s_D \in S \tag{6.5}$$

We then define S recursively as the minimal set which satisfies (6.5) with $\{s_1, s_2, s_3\} \subseteq S$; all sensor locations are uniquely determined if $|S| = K$. In practice we may evaluate this condition by initializing $S = \{s_1, s_2, s_3\}$ and iteratively adding to S all nodes with at least three neighbors in S. This condition also has the nice property that it is computable using only the binary observation variables o_{tu}, and never requires us to calculate the estimated position x_t of any sensor.

While this condition is sufficient to determine all sensor locations uniquely, it is not necessary. A useful source of information which is not used in (6.5) arises from the *lack* of distance measurement between two sensors.

Let us define a pair of nodes s and t to be "1-step" neighbors of one another if they observe their pairwise distance d_{st}. We then define the "2-step" neighbors of s to be all nodes t such that we do *not* observe d_{st}, but do observe both d_{su} and d_{ut} for some node u. We can follow the same pattern to define the "3-step" neighbors, and so forth. In our visual depictions of sensor networks up to this point, we have drawn a line between pairs of nodes which are 1-step neighbors; for example, in Figure 6.2(b), nodes A and C are 1-step neighbors.

Our first sufficient condition for a unique solution of sensor locations consisted solely of using 1-step information—a sensor was guaranteed to be localizable (in the set S) if three of its 1-step neighbors were in S, as in (6.5). However, 2-step and other neighbors also have information useful in localizing the sensors. Specifically, the lack of measurement d_{tu} between sensors t and u (so that $o_{tu} = 0$) implies $\|x_t - x_u\| > R_1$. Thus, to draw a parallel to (6.5), two sensors $s_A \sim s_D, s_B \sim s_D$ and a third $s_C \not\sim s_D$ may be able to localize s_D, or may not, depending on the precise locations of the sensors involved.

An example of each case is shown in Figure 6.2(b); given the positions of sensors A and B, sensor C, which is a 2-step neighbor of sensor D, is able to resolve the location of D. However, the combination of sensors A and B with sensor E (also a 2-step neighbor of D) is not able to resolve the location of D.

This yields a second sufficient condition to (6.5), specifically that $s_D \in S$ if it has two 1-step neighbors in S and either a third 1-step neighbor *or* another node which precludes one of the two possible locations for s_D, as sensor C did in Figure 6.2(b). Note that this condition requires us to actually solve for the position of each sensor when it is included in the set S; we cannot use only the connectivity information o_{tu}. Again, if the resulting set S has $|S| = K$, the locations are uniquely determined with probability one. We investigate the behavior of both these sufficient conditions.

■ 6.3.2 Probability of uniqueness

The existence of a unique solution to our idealized problem may now be addressed, in terms of how often a collection of sensors generated in the manner described satisfies either of the given sufficient conditions (using only 1-step neighbor information, or information from *all* other sensors) as a function of the parameters K and $\frac{R_1}{R_0}$. We use Monte Carlo trials to investigate the frequency with which the conditions are true. In doing so, we note a number of interesting observations—first, that almost all information useful for localizing sensor s_t is in the 1- and 2-step neighbors of s_t; and second, that in order to have high probability that a random network is uniquely determined, we require a surprisingly high average connectivity (i.e., the average number of 1-step neighbors required is significantly greater than its minimum possible value).

The probability of a random graph having a unique solution as a function of $\frac{R_1}{R_0}$ is shown in Figure 6.3 for several values of K. The solid lines indicate the probability when *all* sensors contribute information, i.e., we also utilize information between sensors which do not obtain a distance measurement (the second sufficient condition discussed in Section 6.3.1). The dashed lines, on the other hand, illustrate the comparative loss in performance when only the information from sensors which are 1-step neighbors is used. Both follow the same trend in the number of sensors K, and demonstrate a kind of "threshold" behavior in which the probability of uniqueness changes rapidly from zero to one. This threshold behavior is also predicted by the asymptotic analysis of random graphs presented in [21].

Notably, most of the information for computing a sensor position is local to the sensor. From Figure 6.3, we see that a substantial portion of the information, though not all, is already captured by the 1-step neighbors; using more distant sensors (2-step and beyond) reduced the radius R_1 required to achieve a given probability of uniqueness by about 10%. Furthermore, in 500 Monte Carlo trials at each setting of K and $\frac{R_1}{R_0}$, every network which was uniquely determined given all the observed data was also uniquely determined using only the 1- and 2-step neighbors. This equivalence cannot be guaranteed theoretically; it is possible to construct examples in which it is not the case. However, they are equivalent with very high probability, which indicates that the

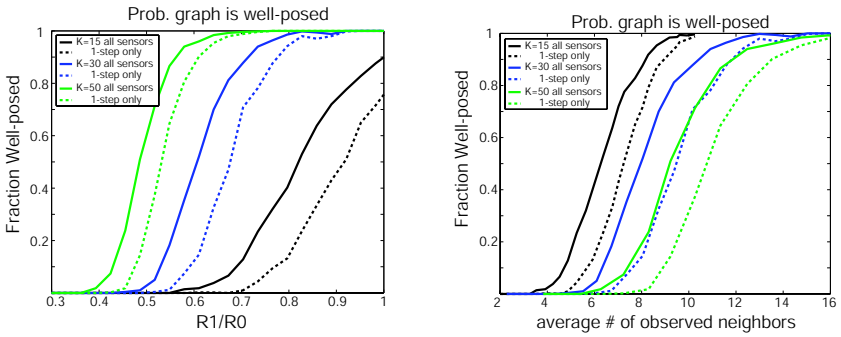


Figure 6.3. Probability of satisfying the uniqueness condition for various K, as a function of (a) R_1/R_0; (b) Expected number of 1-step neighbors given R_1/R_0 and K. Solid lines use information from all sensors (equivalent to using only 2-step neighbors); the dashed lines use only the 1-step neighbor constraints.

important information for sensor localization is *local*, and this locality of information plays an important part of creating a distributed algorithm for localization.

It is also interesting to note the relationship between how frequently we obtain a unique solution and the average number of neighboring sensors which observe a distance. Clearly a minimal value is three (or two, with the possibility that a sensor which does not observe its distance may assist); but we find that the average is quite high—10 or more for even relatively small networks. This high average connectivity is also predicted by the theoretical results of [21], and is indicative of the fact that the minimum connectivity is the driving factor in uniqueness. The implication of this statement is that in practical networks, there may be a number of under-determined sensors, and suggests that having an estimate of the uncertainty associated with each sensor position may be of great importance.

The nonparametric methods studied in this thesis are appealing for characterizing highly non-Gaussian uncertainties. Particle-based representations are able to provide reasonable approximations of many distributions which would otherwise be difficult to express in closed form. To apply the nonparametric belief propagation algorithm of Chapter 3 to the problem of sensor localization first requires that we describe the statistical model of sensor locations and observations and its associated optimization problem within the framework of graphical models.

■ 6.4 Graphical Models for Localization

Graphical models, described in Section 2.5, provide one means of characterizing the factorization of a probability distribution. Expressing the distribution over sensor locations as a graphical model allows us in principle to apply any of number of simple, general algorithms for exact or approximate inference [79, 101, 112]. Of these, the belief propagation (BP) algorithm described in Section 2.6 is perhaps the best-known. In

practice, however, we shall see that for the localization problem the typical, discrete implementation of BP has an unacceptably high computational cost. However, using a particle-based approximation to BP such as nonparametric belief propagation (NBP) results in a more tractable algorithm.

■ 6.4.1 Graphical Models

Recall from Section 2.5 that a graphical model associates each variable x_t with a vertex (or node) v_t in a graph, and describes the conditional independence relations among the x_t via graph connectivity. The relationship between the graph and joint distribution may be quantified in terms of potential functions ψ which are defined over each of the graph's cliques,

$$p(x_1, \ldots, x_K) \propto \prod_{\text{cliques } C} \psi_C(\{x_i : i \in C\}) \tag{6.6}$$

Taking x_t to be the location of sensor t, we may immediately define potential functions which express the joint distribution (6.2), or equivalently (up to a normalization constant) the posterior distribution $p(x_1, \ldots, x_K | \{o_{ut}, d_{ut}\})$, in the form (6.6). Notably, this only requires functions defined over variables associated with single nodes and pairs of nodes. Take

$$\psi_t(x_t) = p_t(x_t) \tag{6.7}$$

to be the single-node potential at each node v_t, and define the pairwise potential between nodes v_t and v_u as

$$\psi_{tu}(x_t, x_u) = \begin{cases} P_o(x_t, x_u)\ p_\nu(d_{tu} - \|x_t - x_u\|) & \text{if } o_{tu} = 1 \\ 1 - P_o(x_t, x_u) & \text{otherwise} \end{cases} \tag{6.8}$$

We make no distinction between ψ_{tu} and ψ_{ut}, only one of which[3] appears in the product (6.6). The joint posterior likelihood of the x_t is then

$$p(x_1, \ldots, x_K | \{o_{tu}, d_{tu}\}) \propto \prod_t \psi_t(x_t) \prod_{t,u} \psi_{tu}(x_t, x_u) \tag{6.9}$$

Notice also that for non-constant P_o *every* sensor t has some information about the location of each sensor u, i.e., there is some information contained in the fact that two

[3]The definition of ψ is slightly more complicated for asymmetric measurements, since to obtain a self-consistent undirected graphical model we require both t and u to know and agree on $\psi_{tu} = \psi_{ut}$, which will thus involve all four quantities $o_{tu}, o_{ut}, d_{tu}, d_{ut}$, so that

$$\psi_{tu}(x_t, x_u) = \begin{cases} P_o^2(x_t, x_u)\ p_\nu(d_{tu} - \|x_t - x_u\|)\ p_\nu(d_{ut} - \|x_t - x_u\|) & \text{if } o_{tu} = o_{ut} = 1 \\ (1 - P_o(x_t, x_u))P_o(x_t, x_u)\ p_\nu(d_{tu} - \|x_t - x_u\|) & \text{if } o_{tu} = 1,\ o_{ut} = 0 \\ (1 - P_o(x_t, x_u))P_o(x_t, x_u)\ p_\nu(d_{ut} - \|x_t - x_u\|) & \text{if } o_{tu} = 0,\ o_{ut} = 1 \\ (1 - P_o(x_t, x_u))^2 & \text{if } o_{tu} = o_{ut} = 0 \end{cases}$$

sensors do not observe a distance between them, namely that they should prefer to be far from each other. However, uncertainty in the measurement process such as physical barriers, multipath, and interference result in the fact that sometimes, nearby sensors may still not observe each other. The model of observing a distance P_o provides a *probabilistic* description of the measurement process, modeling o_{tu} as a random variable, and these kinds of situations can be accounted for using even simple models in which the probability P_o is never exactly one. The overall effect of incorporating the model P_o and its influence on the positions of sensors which do not share a distance measurement is similar to, but less strict than, that achieved by approximating unobserved distances by shortest paths[4] [85], and to the non-convex constraints mentioned (though not actually used) in [18]. Probabilistic constraints have the additional benefit of being less vulnerable to the distortion effects caused by shortest–path methods and observed by [85] when the sensor configuration is not entirely convex.

Unfortunately, fully connected graphs are very difficult for most inference algorithms, and thus it behooves us to approximate the exact model in (6.9). Let us define the *observed* edges to be those edges for which we have observed d_{tu} (and thus $o_{tu} = 1$); the *unobserved* edges refer to all other edges in the graphical model. Given the set of observations o_{tu} within the network, we can construct approximate graphical models on which inference is more tractable. We construct a sequence of such approximations, using notation derived from the 1-step and 2-step neighbors discussed in Section 6.3.

Suppose that we create an approximate graphical model which has been constructed by including *only* the observed edges in the original graph; we call this the "1-step" graph. The "2-step" graph, then, is created by also including an edge between each 2-step neighbor, i.e., including the edge between t and u if we observe d_{tv} and d_{vu} for some sensor v, but not d_{tu}; these edges we call the "2-step" edges. We can also extend these definitions to "3-step" graphs, and so forth. Building these approximate graphical models is, of course, an adaptive process, in the sense that the definition of the "k-step" graph is determined by the observations $\{o_{tu}\}$ in the network.

From the experiments described in Section 6.3 it appears that there is little loss in information when we discard the interactions between nodes which are far apart, in the sense that they are k-step neighbors for a high value of k. Those experiments indicated that most of the necessary information for localization is present in the 1-step graph, and almost all information is present in the 2-step graph. Note also that the 1-step graph exactly represents the distribution (6.2) if P_o is a constant, i.e., is not a function of x_t and x_u, since in this case the unobserved edges offer no additional information about sensor location.

There is also a convenient relationship between the *statistical* and *communications* graph in localization. Specifically, since the observations are obtained via acoustic

[4]Specifically, one may "approximate" the distance between two nodes t, s which do not observe a distance d_{ts} by taking the sum of the distances along the shortest path between t and s along edges which have observed distances. This often causes distortion in the final location estimates, due to its inherent over-estimation of the distance between t and s.

or wireless exchange, distance measurements are only obtained for sensor pairs which have some form of communications link. Thus, messages along observed edges may be communicated directly, while messages along unobserved edges may require some form of multi-hop forwarding protocol, with 2-step edges requiring at most 2 hops, and so forth. Technically, of course, the time-varying nature of these wireless or acoustic links means that communications may not be entirely reliable. However, we ignore this subtlety and assume that, over the short period of time in which localization is performed, the communications graph is static.

■ 6.4.2 Belief Propagation

Having defined a graphical model for sensor localization, we can now estimate the sensor locations by applying the belief propagation (BP) algorithm, described in Section 2.6. In particular, each sensor t is responsible for performing the computations associated with node v_t in the graph and computing its "belief", or estimated marginal distribution of the sensor location x_t. The form of BP as an iterative, local message passing algorithm makes this procedure trivial to distribute among the wireless sensor nodes [16].

By applying BP to sensor localization, we can estimate the posterior marginal distributions $p(x_t|\{o_{uv}\},\{d_{uv}\})$ of each sensor location variable x_t. Alternatively, we might like to find the joint MAP configuration of sensor locations. While there exists an algorithm, called the *max-product* or belief revision algorithm [79], for estimating the joint MAP configuration of a discrete-valued graphical model, this technique is computationally difficult to apply to high-dimensional, continuous-valued graphical models. However, determining a likely configuration with the MAP location of each posterior marginal, as estimated via BP, is a common practice in graphical models [24]. In fact, investigation of the performance of both max- and sum-product algorithms in iterative decoding schemes have shown that the latter may even be preferable in some situations [104]. Thus, we apply BP to estimate each sensor's posterior marginal, and use the maximum of this marginal and its associated uncertainty to characterize sensor placements.

To remind the reader, we repeat the equations for BP; for more detail see Section 2.6. In integral form, each node v_t computes its belief about x_t, an estimate of the posterior marginal distribution of x_t, at iteration i by taking a product of its local potential ψ_t with the messages from its set of neighbors Γ_t,

$$M_t^i(x_t) \propto \psi_t(x_t) \prod_{u\in\Gamma_t} m_{ut}^i(x_t) \tag{6.10}$$

The messages m_{tu} from node v_t to node v_u are computed in a similar fashion, by

$$\begin{aligned} m_{tu}^i(x_u) &\propto \int \psi_{tu}(x_t,x_u)\psi_t(x_t) \prod_{v\in\Gamma_t\setminus u} m_{vt}^{i-1}(x_t)\,dx_t \\ &\propto \int \psi_{tu}(x_t,x_u)\frac{M_t^{i-1}(x_t)}{m_{ut}^{i-1}(x_t)}\,dx_t. \end{aligned} \tag{6.11}$$

Sensor Self-Localization with BP
Initialization:

- Each sensor obtains local information $p_t(x_t)$, if available.
- Obtain distance estimates:
  - Broadcast sensor id & listen for other sensor broadcasts
  - Estimate distance d_{tu} for any received sensor IDs
  - Communicate with observed neighbors to symmetrize distance estimates
- Initialize $m_{ut} \equiv 1$ and $M_t^0 = p_t$ for all u, t.

Belief Propagation: for each sensor t

- Broadcast $M_t^i(x_t)$ to neighbors; listen for neighbors' broadcasts
- Compute m_{ut}^{i+1} from m_{tu}^i and $M_t^i(x_u)$ using (6.11)
- Compute new marginal estimate $M_t^{i+1}(x_t)$ via (6.10)
- Repeat until sufficiently converged (see Section 6.8)

Figure 6.4. Belief propagation for sensor self-localization.

One appealing consequence of using a message–passing inference method and assigning each vertex of the graph to a sensor in the network is that computation is naturally distributed among the sensors. Each node v_t, embodied by sensor t, performs computations using information sent by its neighbors, and disseminates the results, as described in the pseudocode in Figure 6.4. This process is repeated until some convergence criterion is met, after which each sensor is left with an estimate of its location and uncertainty.

The pseudocode in Figure 6.4 uses the idea of "belief sampling", described in Section 3.6. This expresses the message update (6.11) in terms of a ratio of the belief at the previous iteration, M_t^{i-1}, and the incoming message m_{ut}^{i-1}. When the BP messages and beliefs are computed exactly, the two definitions in (6.11) are identical. However, when they are approximated, it may be to some advantage to use one form over the other; in Section 6.7 we describe how the information in M_t^{i-1} can be used to improve estimation in some cases. Perhaps more importantly, expressing the message update in terms of the ratio (6.11) has a significant communication benefit, in that all messages from t to its neighbors Γ_t may be communicated *simultaneously* via a broadcast step. The message m_{tu}^i from sensor t to each neighbor $u \in \Gamma_t$ is a function of the belief $M_t^{i-1}(x_t)$, the previous iteration's message from u to t, and the compatibility ψ_{tu}, which depends only on the observed distance between t and u. Since the latter two quantities (m_{ut}^{i-1} and ψ_{tu}) are also known at sensor u, sensor t may simply communicate its belief $M_t^i(x_t)$ to all its neighbors, and allow each neighbor u to deduce the rest.

■ 6.4.3 Nonparametric Belief Propagation

As described in Section 2.6, the BP update and belief equations (6.10)–(6.11) are easily computed in closed form for discrete or Gaussian likelihood functions; unfortunately neither discrete nor Gaussian BP is well-suited to localization. For discrete BP, this is because even the two–dimensional space in which the x_t reside is too large to accommodate an efficient discretized estimate—in general, to obtain acceptable spatial resolution for the sensors the discrete state space must be made too large for BP to be computationally feasible. The presence of nonlinear relationships and potentially highly non-Gaussian uncertainties in sensor localization makes Gaussian BP undesirable as well. However, using particle-based representations via nonparametric belief propagation, enables the application of BP to inference in sensor networks. Chapter 3 presented the general theory and methods behind NBP; in this chapter we describe more precisely how that material can be applied to the sensor localization problem.

In NBP, each message is represented using either a sample-based density estimate (as a mixture of Gaussians) or as an analytic function. Both types are necessary for the sensor localization problem. Messages along observed edges are represented by samples, while messages along unobserved edges must be represented as analytic functions since their potentials have the form $1 - P_o(x_t, x_u)$, which is typically not normalizable. Most reasonable models have the characteristic that P_o tends to 0, and thus $1 - P_o$ to 1, as the distance $\|x_t - x_u\|$ becomes large; thus, messages along unobserved edges are poorly approximated by any finite set of samples. The belief and message update equations (6.10)–(6.11) are performed using stochastic approximations, in two stages: first, drawing samples from the belief $M_t^i(x_t)$, then using these samples to approximate each outgoing message m_{tu}^i. We discuss each of these steps in turn, and summarize the procedure with pseudocode in Figure 6.5.

Given N weighted samples $\{W_t^j, X_t^j\}$ from the belief $M_t^i(x_t)$ obtained at iteration i, computing a Gaussian mixture estimate of the outgoing BP message from t to u is relatively simple. We first consider the case of observed edges. The distance measurement d_{tu} provides information about how far sensor u is from sensor t, but no information about its relative direction. To draw a sample from the state x_u of sensor u given the sample X_t^j representing the position of sensor t, we simply select a direction θ at random, uniformly in the interval $[0, 2\pi)$. We then shift X_t^j in the direction of θ by an amount which represents our information about the distance between x_u and x_t, i.e., the observed distance d_{tu} plus a sample realization of the noise ν on the distance measurement. This gives

$$x_{tu}^j = X_t^j + (d_{tu} + \nu^j)[\sin(\theta^j); \cos(\theta^j)] \qquad \theta^j \sim U[0, 2\pi) \qquad \nu^j \sim p_\nu. \tag{6.12}$$

where $U[0, 2\pi)$ indicates the uniform distribution on the interval from zero to 2π. The samples are then weighted by the remainder of (6.11), $w_{tu}^j = W_t^j \cdot P_o(x_{tu}^j)/m_{ut}(X_t^j)$, and we assign a single bandwidth vector h_{tu} to all samples to construct a kernel density estimate.[5] There are a number of possible techniques for choosing the bandwidth h_{tu}.

[5] If $o_{ut} = o_{tu} = 1$ but $d_{ut} \neq d_{tu}$, the potential ψ_{tu} involves both distance measurements and may

The simplest is to apply the rule of thumb estimate [90] described in Section 2.3, given by computing the (weighted) variance of the samples

$$h_{tu} = N^{-\frac{1}{3}} \operatorname{Var}[\{x_{tu}^j\}] = N^{-\frac{1}{3}} \sum_{j,k} w_{tu}^k w_{tu}^j (x_{tu}^k - \bar{x})(x_{tu}^j - \bar{x})^T \tag{6.13}$$

where $\bar{x}$ is the mean, $\bar{x} = \sum_j w_{tu}^j x_{tu}^j$.

A small modification to this procedure can be used to improve the approximation when N is sufficiently large and the uncertainty added by p_ν is Gaussian or nearly Gaussian. In these cases, instead of drawing samples ν^j from p_ν and using their randomness to model the uncertainty in the distance d_{tu}, we can model this uncertainty explicitly. To be precise, an excellent approximation to the message can be obtained by taking the mean value of the noise, i.e., $\nu^j = 0$ for all j, and using the variance of the Gaussian uncertainty p_ν as the bandwidth, so that the vector $h_{tu} = [\sigma_\nu^2; \sigma_\nu^2]$. This is a simple variation on the procedure described in Section 3.4.3; however, as noted in that section, N and σ_ν must be sufficiently large so as not to result in an undersmoothed representation.

As stated previously, messages along unobserved edges are represented using an analytic function. Using the probability of detection P_o and samples from the belief M_t^{i-1} at x_t, an estimate of the outgoing message to u is given by

$$m_{tu}^i(x_u) = 1 - \sum_j w_{tu}^j P_o(x_u - X_t^j) \qquad\qquad w_{tu}^j \propto 1/m_{ut}^{i-1}(X_t^j) \tag{6.14}$$

which is easily evaluated for any analytic model of P_o; in our simulations, we use the form (6.3). See Section 3.5 for a more complete discussion of analytic potentials and messages in NBP.

To estimate the belief $M_t^i = \psi_t \prod m_{ut}^i$, we draw samples from the product of several Gaussian mixture and analytic messages using the methods described in Chapter 3. Specifically, in this chapter we make use of the mixture importance sampling method described in detail in Section 3.8.1. We give a brief algorithmic review here.

Denote the set of neighbors of t having *observed* edges to t by Γ_t^o. In order to draw N samples, we create a collection of $k \cdot N$ weighted samples (where $k \geq 1$ is a parameter of the sampling algorithm) by drawing $\frac{kN}{|\Gamma_t^o|}$ samples from each message m_{ut} with $u \in \Gamma_t^o$ and assigning each sample a weight equal to the ratio $\prod_{v \in \Gamma_t} m_{vt} / \sum_{v \in \Gamma_t^o} m_{vt}$. We then draw N values independently from this collection with probability proportional to their weight, i.e., resampling with replacement, yielding equal-weight samples drawn from the product of all incoming messages. Computationally, this requires $\mathcal{O}(k\,|\Gamma_t|\,N)$ operations per marginal estimate.

be more difficult to draw samples from ψ. For identical Gaussian observation noise p_ν at each sensor, we can simply average the two distance measurements at both sensors, and adjust the variance of the model's likelihood function to account for the fact that we have averaged two independent observations of the true distance. If p_ν is non-Gaussian the procedure is slightly more difficult, but we can instead draw some samples according to each of $p(x_u|x_t, d_{tu})$ and $p(x_u|x_t, d_{ut})$ and weight by the influence of the other observation.

Compute NBP messages:
Given N weighted samples $\{W_t^j, X_t^j\}$ from the belief $M_t^i(x_t)$, construct an approximation to $m_{tu}^{i+1}(x_u)$ for each neighbor $u \in \Gamma_t$:

- If $o_{tu} = 1$ (we observe inter-sensor distance d_{tu}), approximate with a Gaussian mixture:
  - Draw random values for $\theta^j \sim U[0, 2\pi)$ and $\nu^j \sim p_\nu$
  - Means: $x_{tu}^j = X_t^j + (d_{tu} + \nu^j)[\sin(\theta^j); \cos(\theta^j)]$
  - Weights: $w_{tu}^j = P_o(x_{tu}^j)\ W_t^j / m_{ut}^i(X_t^j)$
  - Bandwidth: $h_{tu} = N^{-\frac{1}{3}} \cdot \mathrm{Var}[\{x_{tu}^j\}]$
- Otherwise, use the analytic function:
  - $m_{tu}^{i+1}(x_u) = 1 - \sum_j w_{tu}^j P_o(x_u - X_t^j)$
  - $w_{tu}^j \propto 1/m_{ut}^i(X_t^j)$

Compute NBP marginals:
Given several Gaussian mixture messages $m_{ut}^i = \{x_{ut}^j, w_{ut}^j, h_{ut}\}, u \in \Gamma_t^o$, compute samples from $M_t^i(x_t)$:

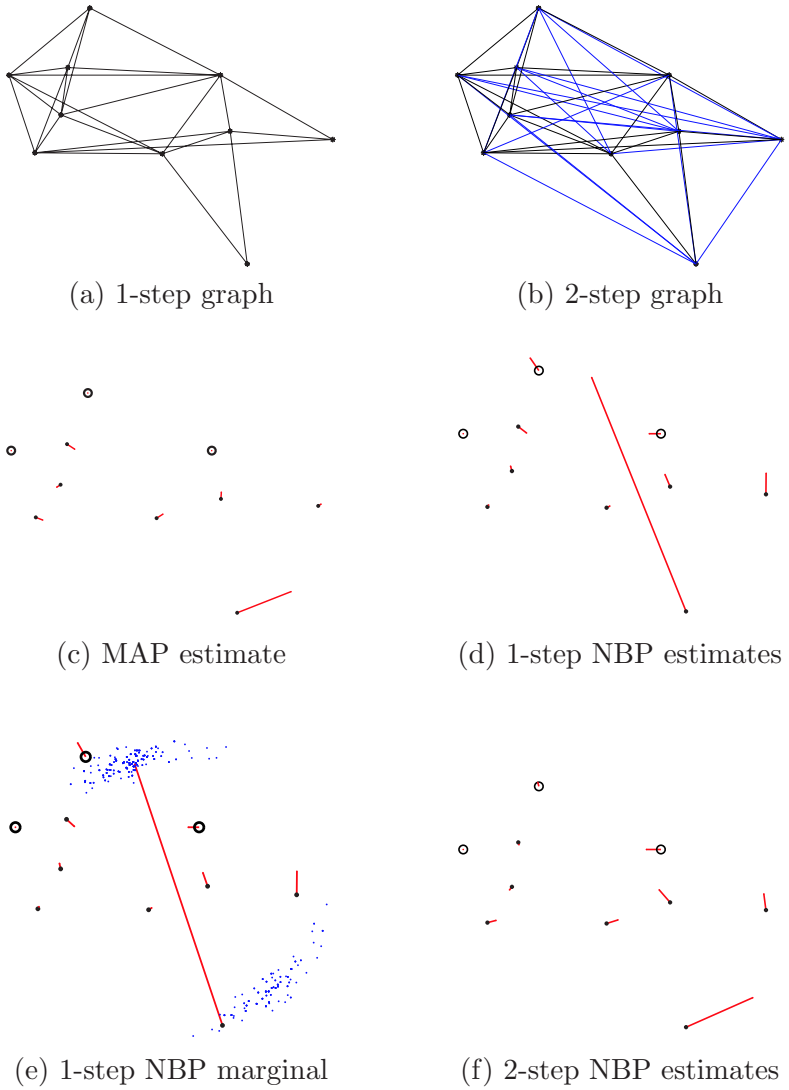
- For each observed neighbor $u \in \Gamma_t^o$,
  - Draw $\frac{kN}{|\Gamma_t^o|}$ samples $\{X_t^j\}$ from each message m_{ut}^i
  - Weight by $W_t^j = \prod_{v \in \Gamma_t} m_{vt}^i(X_t^j) / \sum_{v \in \Gamma_t^o} m_{vt}^i(X_t^j)$
- From these kN locations, re-sample by weight (with replacement) N times to produce N equal-weight samples.

Figure 6.5. Using NBP to compute messages and marginals for sensor localization.

■ 6.5 Empirical Calibration Examples

We show two example sensor networks to demonstrate NBP's utility. All the networks in this section have been generated by placing K sensors at random with spatially uniform probability in an $L \times L$ area, and letting each sensor observe its distance from another sensor (corrupted by Gaussian noise with variance σ_ν^2) with probability given by (6.3). We investigate the *relative* calibration problem, in which the sensors are given no absolute location information; the anchor nodes are indicated by open circles. These simulations used $N = 200$ particles and underwent three iterations of the sequential message schedule described in Section 6.8; each iteration took less than 1 second per node on a P4 workstation.

The first example, shown in Figure 6.6(a), consists of a small graph of 10 sensors which was generated using $R/L = .2$ and noise $\sigma_\nu/L = .02$; in this graph the average measured distance was about $.33L$, and each sensor observed an average of 5 neighbors. One sensor (the bottommost) has significant multi-modal location uncertainty, due to



(a) 1-step graph (b) 2-step graph

(c) MAP estimate (d) 1-step NBP estimates

(e) 1-step NBP marginal (f) 2-step NBP estimates

Figure 6.6. (a) A small (10-sensor) graph with edges denoting observed pairwise distances; (b) the same network with 2-step edges indicating the lack of a distance measurement also shown. Calibration is performed relative to the sensors drawn as open circles. (c) A centralized estimate of the MAP solution shows generally similar errors (lines) to (d), NBP's approximate (marginal maximum) solution. However, NBP's estimate of uncertainty (e) for the poorly-resolved sensor displays a clear bi-modality. Adding 2-step potentials (f) results in a reduction of the spurious mode and an improved estimate of location.

the fact that it is involved in only two distance measurements. The true, joint MAP configuration is shown in Figure 6.6(c), and the 1-step NBP estimate (NBP performed on the 1-step approximate graphical model) is shown in Figure 6.6(d). Comparison of the error residuals would indicate that NBP has significantly larger error on the sensor in question than the true MAP. However, this is mitigated by the fact that

NBP has a representation of the marginal uncertainty, shown in Figure 6.6(e), which accurately captures the bi-modality of the sensor location and which could be used to determine that the location estimate is questionable. Additionally, exact MAP uses more information than 1-step NBP. We approximate this information by including some of the unobserved edges (2-step NBP). The result is shown in Figure 6.6(f); the error residuals are now comparable to the exact MAP estimate.

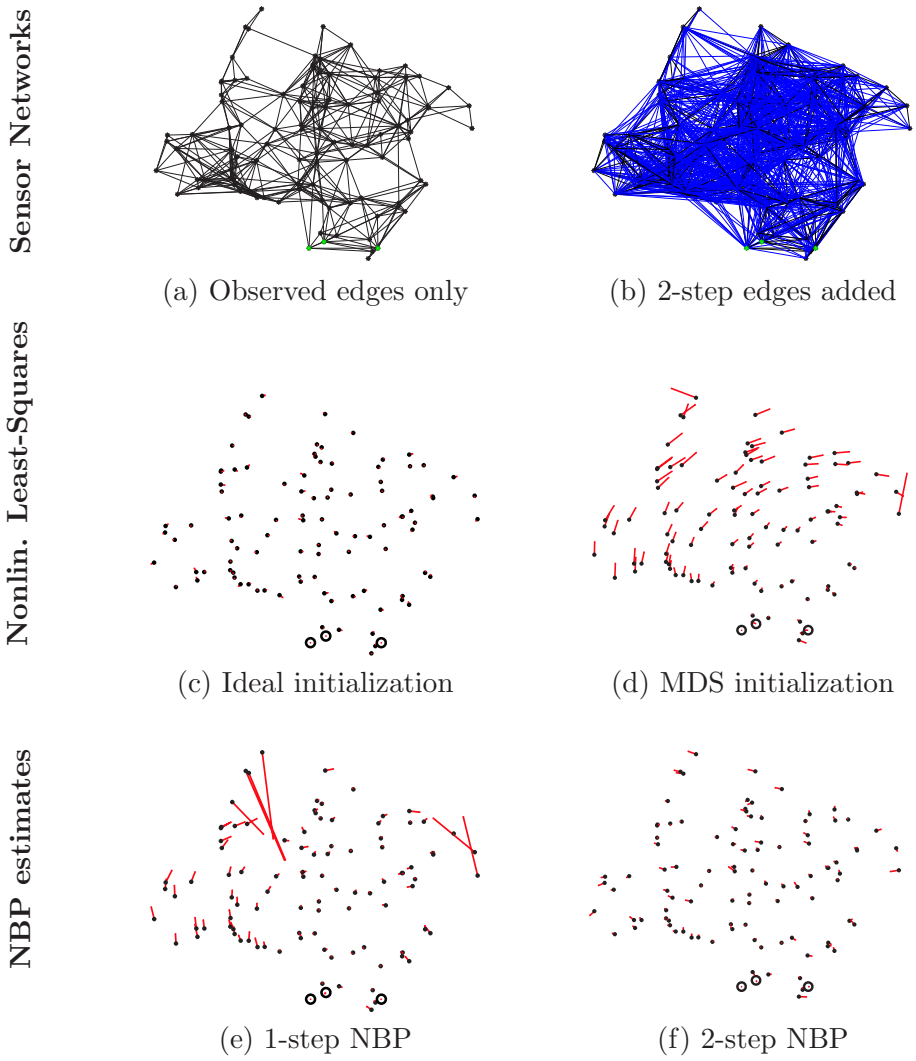
While the previous example illustrates some important details of the NBP approach, our primary interest is in automatic calibration of moderate- to large-scale sensor networks with sparse connectivity. We examine a graph of a network with 100 sensors generated with $R/L = .08$ (giving an average of about 9 observed neighbors) and $\sigma_\nu/L = .005$, shown in Figure 6.7. For problems of this size, computing the true MAP locations is considerably more difficult. The iterative nonlinear minimization of [73] converges slowly and is highly dependent on initialization. As a benchmark to illustrate the best possible performance, an idealized estimate in which we initialize using the *true* locations is shown in Figure 6.7(c). In practice, we cannot expect to perform this well; starting from a more realistic value (initialization given by classical MDS [98]) finds the alternate local minimum shown in Figure 6.7(d). The 1-step and 2-step NBP solutions are shown in Figure 6.7(e)-(f). Errors due to multi-modal uncertainty similar to those discussed in the previous 10–sensor example arise for a few sensors in the 1-step case. Examination of the 2-step solution shows that the errors are comparable to the nonlinear least-squares estimate with an idealized initialization.

In the 2-step examples above, we have included *all* 2-step edges, but this is often not required. The sensors which require this additional information are typically those with too few observed neighbors, often those sensors located near the edge of the sensor field. We could achieve similar results by including only 2-step edges which are incident on a node with fewer than, for example, four observed edges.

■ 6.6 Modeling Non-Gaussian Measurement Noise

In an NBP–based solution to sensor localization, it is straightforward to change the form of the noise distribution p_ν so long as sampling remains tractable. This may be used to accommodate alternative noise models for the distance measurements, for example the log-normal model of [78] which might arise when distance between sensor pairs is estimated using the received signal strength, or models which have been learned from data [108].

Although this fact can also be used to model the presence of a broad outlier process, the form of NBP's messages as Gaussian mixtures provides a more elegant solution. We augment the Gaussian mixtures in each message by a single, high-variance Gaussian to approximate an outlier process in the uncertainty about d_{tu}, in a manner similar to [48]. To be precise, we add an extra particle to each outgoing message along an observed edge, centered at the mean of the other particles and with weight and variance chosen to model the expected outliers, e.g., weight equal to the probability of an outlier and



(a) Observed edges only (b) 2-step edges added

(c) Ideal initialization (d) MDS initialization

(e) 1-step NBP (f) 2-step NBP

Figure 6.7. Large (100-node) example sensor network. (a-b) 1- and 2-step edges. Even in a centralized solution we can at best hope for (c) the local minimum closest to the true locations; a more realistic initialization (d) yields higher errors. NBP (e-f) provides similar or better estimates, along with uncertainty, and is easily distributed. Calibration is performed relative to the three sensors shown as open circles.

standard deviation sufficiently large to cover the expected support of P_o. Since outlier samples by definition occur rarely, good estimates of the tail regions of the noise may take many samples. Direct approximation of the outlier process requires fewer particles to adequately represent the message, and thus is more computationally efficient.

Figure 6.8(a) shows the same small (10 sensor) 1-step network examined in Figure 6.6 but with several additional distance measurements; the complete set of distance measurements are indicated by lines. We also introduce a single outlier measurement,

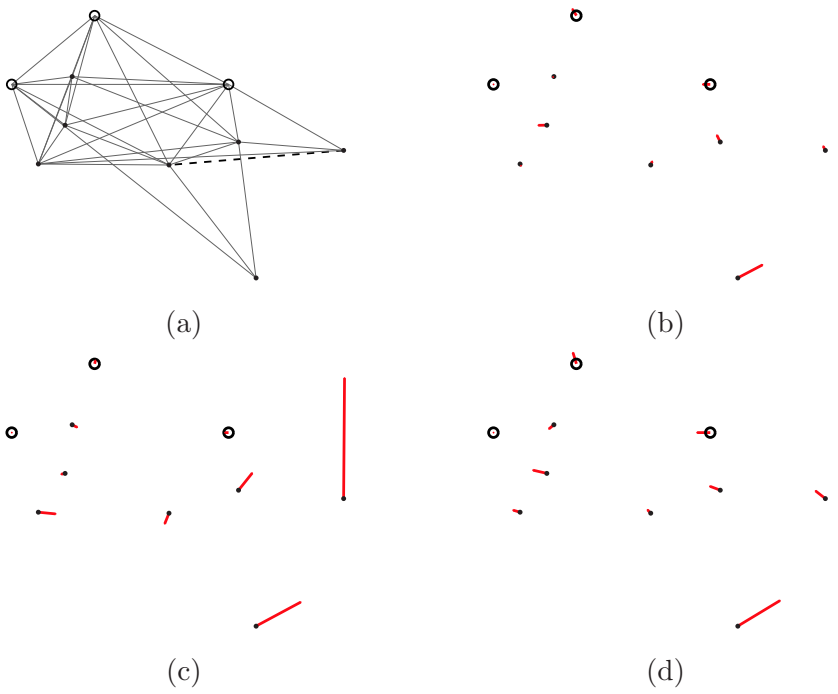


Figure 6.8. (a) A small (10-sensor) graph and the observable pairwise distances; calibration is performed relative to the location of the sensors shown in green. One distance (shown as dashed) is highly erroneous, due to a measurement outlier. (b) The MAP estimate of location, discarding the erroneous measurement. (c) A nonlinear least-squares estimate of location is highly distorted by the outlier; (d) NBP is robust to the error by inclusion of a measurement outlier process in the model.

shown as the dashed line. We again perform calibration relative to the three sensors shown as circles. If we possessed an oracle which allowed us to detect and discard the erroneous measurement, the optimal sensor locations can be found using an iterative nonlinear least-squares optimization [73]; the residual errors after this procedure (for a single noise realization) are shown in Figure 6.8(b). However, with the outlier measurement present, the same procedure results in a large distortion in the estimates of some sensor locations, as shown in Figure 6.8(c). NBP, by virtue of the measurement outlier process discussed in Section 6.4.3, remains robust to this error and produces the near-optimal estimate shown in Figure 6.8(d).

In order to provide a measure of the robustness of NBP in the presence of non-Gaussian (outlier) distance measurements, we perform Monte Carlo trials, keeping the same sensor locations and connectivity used in Figure 6.8(a) but introducing different sets of observation noise and outlier measurements. In each trial, each distance measurement is replaced with probability .05 by a value drawn uniformly in $[0, L]$. As there are 37 measurements in the network, on average approximately two outlier measurements are observed in each trial. We then measure the number of times each sensor's estimated location is within distance r of its true location, as a function of r/L. We

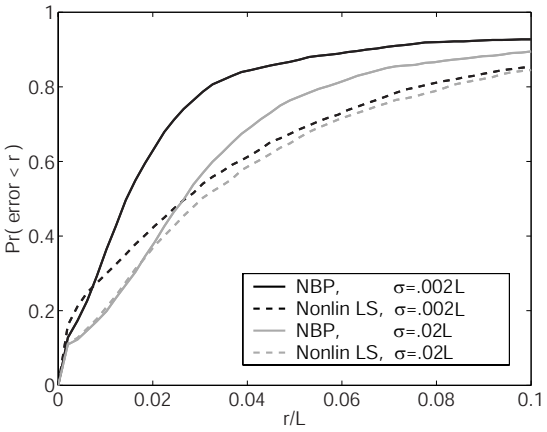


Figure 6.9. Monte Carlo localization trials on the sensor network in Figure 6.8(a). We measure the probability of a sensor's estimated location being within a radius r of its true location (normalized by the region size L), with noise $\sigma_\nu = .02L$ and $.002L$ for both NBP and nonlinear least-squares, indicating NBP's superior performance in the presence of outlier measurements.

repeat the same experiments for two noise levels, $\sigma_\nu/L = .02$ and $\sigma_\nu/L = .002$. The curves are shown in Figure 6.9 for both NBP and nonlinear least-squares estimation. As can be seen, NBP provides an estimate which is more often "nearby" to the true sensor location, indicating its increased robustness to the outlier noise; this becomes even more prominent as σ_ν becomes small and the outlier process begins to dominate the total noise variance. Both methods asymptote around 90%, indicating the probability that the outlier process completely overwhelms the information at one or more nodes.

However, Figure 6.9 understates the advantages of NBP for this scenario. NBP also provides an estimate of the *uncertainty* in sensor position; trials resulting in large errors also display highly uncertain (often bimodal) estimates for the sensor locations in question, as in Figure 6.1. Thus, in addition to providing a more robust estimate of sensor location, NBP also provides a measure of the reliability of each estimate.

■ 6.7 Parsimonious Sampling

We may also apply techniques from importance sampling [3, 19] in order to improve the small-sample performance of NBP, which may play an important part in reducing its computational burden. In the algorithm of Figure 6.5, the outgoing messages are computed via an importance sampling procedure to estimate (6.11). In particular, samples are drawn from an approximation to (6.11), the proposal distribution, and then re-weighted so as to asymptotically represent the target distribution (6.11).

As outlined in Section 2.7.1, so long as the proposal distribution f is absolutely continuous with respect to the target distribution g (meaning $g(x) > 0 \Rightarrow f(x) > 0$), we are guaranteed that, for a sufficiently large sample size N we can obtain samples which

are representative of g by drawing samples from f and weighting by g/f. However, the sample size N is limited by computational power, and as is well-known in particle filtering the low-sample performance of any such approximation is strongly influenced by the quality of the proposal distribution [3, 19]. Typically, one takes f to be as close as possible to g while remaining tractable for sampling. We accomplished this for (6.11) by drawing samples from the belief (6.10), weighting by the remaining terms of (6.11), and moving the particles in a direction θ by the observed distance d_{tu} plus noise, where θ was chosen at random and uniformly on $[0, 2\pi)$ since we do not have any information about the direction from sensor t to u.

However, in the context of belief propagation, the goal is to accurately estimate the product $M_u = \prod_s m_{su}$ in the regions of the state space in which it has significant probability mass. Thus, a good proposal distribution is one which allows us to accurately estimate the portions of the message m_{tu} which overlap these regions of the state space, i.e., the regions m_{tu} has in common with the other incoming messages. In other words, we would like to use our limited representative power to accurately model the parts of each message which overlap with non-negligible regions of the messages from u's other neighbors, and any additional knowledge of $M_u(x_u)$ may be used to focus samples in the correct regions [56].

One alternative proposal distribution involves utilizing previous iterations' information to determine the angular direction to each of the neighboring sensors. Rather than estimating a ring-like distribution at each iteration (most of which is ignored as it does not overlap with any other rings), successive estimates are improved by estimating smaller and smaller arcs located in the region of interest. A simple procedure implementing this idea is given in Figure 6.10. In particular, we use samples from the marginal distributions computed at the previous iteration to form a density estimate p_θ of the relative direction θ, draw samples from p_θ, and weight them by $\frac{1}{p_\theta}$ so as to cancel the asymptotic effect of drawing samples from p_θ rather than uniformly. The process requires estimating a density which is 2π-periodic; this is accomplished by sample replication [90].

We first demonstrate the potential improvement on a small example of only four sensors. Figures 6.11(a)–(b) show example messages from three sensors to a fourth, with $N = 30$ particles. Using the additional angular information results in the samples being clustered in the same region of the state space in which the product has significant probability mass, effectively similar to having used a larger value of N. To compare both methods' performance, we first construct the marginal estimate using a large-N approximation ($N = 1000$), and compare (in terms of KL-divergence) to the results of running NBP with fewer samples ($10 \leq N \leq 100$) using both naive sampling, i.e., drawing $\theta \sim U[0, 2\pi)$, and the angular proposal distribution as described in Figure 6.10. The results are shown in Figure 6.11(c); as expected, we find that the angular proposal distribution concentrates more samples in the region of interest, reducing the estimate's KL-divergence.

As noted in [56], however, by re-using previous iterations' information we run the

Using previous iterations' angular information:
Perform NBP as described in Figure 6.5, but at iteration $n > 1$, replace $\theta^j \sim U[0, 2\pi)$ by:

- Draw samples $\tilde{X}_t^j \sim M_t^{i-1}(x_t), \quad \tilde{X}_u^j \sim M_u^{i-1}(x_u)$
- Construct a kernel density estimate p_θ using $\tilde{\theta}^j = \arctan(\tilde{X}_u^j - \tilde{X}_t^j) \quad (\tilde{\theta} \in [-\pi, \pi])$
  - To approximate 2π-periodicity, construct p_θ using samples at $\tilde{\theta}^j + \{2\pi, 0, -2\pi\}$
- Draw $\theta^j \sim p_\theta, \theta \in [-\pi, \pi]$
- Calculate w_{tu}^j in a manner similar to that of Figure 6.5, but using importance reweighting to cancel the influence of p_θ:
  - $w_{tu}^j = \frac{P_o(x_{tu}^j)\ W_t^j}{m_{ut}^{i-1}(X_t^j)} \frac{1}{p_\theta(\theta^j)}$

Figure 6.10. Using an alternative angular proposal distribution for NBP. The previous iteration's marginals may be used to estimate their relative angle, and better focus samples on the important regions of the state space. The estimate is made asymptotically equivalent to that described in Figure 6.5 by importance weighting.

risk of biasing our results. The results of a more realistic situation are shown in Figure 6.11(d)—performing the same comparison for a relative calibration of the 10-node sensor network shown in Figure 6.6(b) reveals the possibility of biased estimates. When the number of particles is sufficiently large ($N \geq 100$), we observe the same improvement as seen in the 4-node case. However, for very few particles ($N = 25$), we see that it is possible for our angular proposal distribution to reinforce incorrect estimates, ultimately worsening performance.

■ 6.8 Incorporating communications constraints

Communications constraints are extremely important for battery-powered, wireless sensor networks; communication is one of the primary factors determining sensor lifetime. There are a number of factors which influence the communications cost of a distributed implementation of NBP. These include

1. *Resolution*, β, of all fixed- (or floating-) point values.
2. *Number of iterations* performed
3. *Schedule*—the order in which sensors transmit
4. *Approximation*—the fidelity to which the marginal estimates are communicated between sensors

All these aspects are, of course, interrelated, and also influence the quality of any solution obtained; often their effects are difficult to separate. Note that the number of particles N used for estimating each message and marginal influences only computational

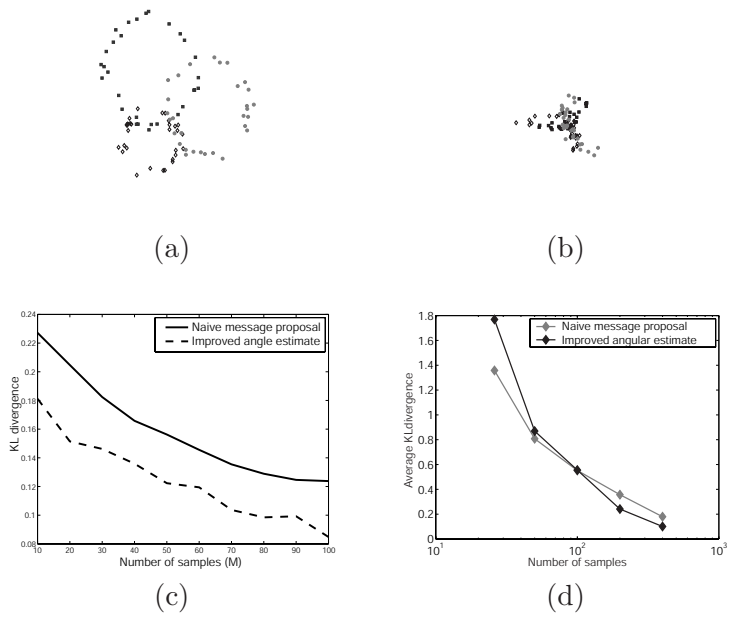


Figure 6.11. By using an alternate proposal distribution during NBP's message construction step, we may greatly improve the fidelity of the messages. (a) Naive (uniform) sampling in angle produces ring-shaped messages; however, (b) using previous iterations' information we may preferentially draw samples from the useful regions. Monte Carlo trials (c) show the improvement in terms of average K-L divergence of the sensor's estimated marginal (from an estimate performed with $N = 1000$ samples) as a function of the number of samples N used. (d) In a larger (10-node) network, we begin to observe the effects of bias: for sufficiently large N performance improves, but for small N we may become overconfident in a poor estimate.

complexity, not communications cost, since we may use approximate representations (Chapter 5) to ensure a fixed communications cost. The following experiments used $N = 200$ samples per message and marginal estimate, with $k = 5$ times oversampling in the product computation.

In the following sections, we consider the effects of changing the number of iterations, the message–passing schedule, and the communications cost (and degree of approximation) of the messages. We leave the resolution fixed, choosing its value to be sufficiently high to avoid quantization artifacts; for example, taking $\beta = 16$ bits is typically more than sufficient.

■ 6.8.1 Schedule and iterations

The message schedule can have a strong influence on the behavior of BP, affecting the number of iterations until convergence and even potentially the quality of the converged solution [67]. We consider two possible BP message schedules, and analyze performance on the 10-node graph shown in Figure 6.6(b). Because we are primarily concerned with the inter-sensor communications required, we enforce a maximum total number of

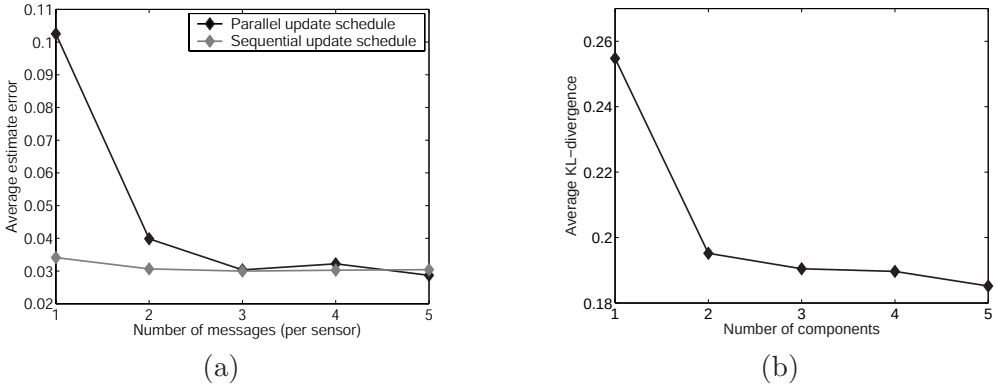


Figure 6.12. Analyzing the communications cost of NBP. (a) The number of iterations required may depend on the message schedule, but is typically very few (1-3). (b) The transmitted marginal estimates may be compressed by fitting a small Gaussian mixture distribution; a few (1-3) components is usually sufficient.

messages per sensor, rather than the actual number of iterations.

The first BP schedule is a "sequential" schedule, in which each sensor takes turns transmitting its message to all its neighbors. We determine the order of transmission by beginning with the anchor nodes, and moving outward in sequence based on the shortest observed distance to any anchor. This has similarities to schedules based on spanning trees [100], though since each sensor is transmitting to *all* neighbors it is not a tree-structured message ordering. For this schedule, one iteration corresponds to one message from each sensor. Strictly speaking, this ordering is only available given global information (the observed distances in the network), but in practice the schedule appears to be robust to small reorderings and thus local or randomized approximations to the sequential schedule could be substituted. Here, however, we will ignore this subtlety.

The second BP schedule we consider is a "parallel" schedule, in which a set of sensors transmit to their neighbors simultaneously. Since initially, large numbers of sensors have no information about their location, we restrict the participating nodes to be those whose belief is well-localized, as determined by some threshold on the entropy of the belief $M_t^i(x_t)$. To provide a fair comparison with the sequential schedule, we limit the number of iterations by allowing each sensor to transmit only a fixed number of messages, terminating when no more sensors are allowed to communicate.

Figure 6.12(a) compares the two schedules' performance over 100 Monte Carlo trials, measured by mean error in the location estimates and as a function of the number of message transmissions allowed by each schedule. As can be seen, both schedules produce reasonably similar results, and neither requires more than a few iterations (inter-sensor communications) to converge. Empirically, we find that the sequential schedule performs slightly better on average.

Faulty communications, i.e., nodes' failure to receive some messages, may also be

considered in terms of small deletions in the BP message schedule. While the exact effect of these changes is difficult to quantify, it is typically not catastrophic to the algorithm.

6.8.2 Message approximation

We may also reduce the communications by approximating each marginal estimate using a relatively small mixture of Gaussians before transmission, rather than communicating the complete set of N particles. Having considered this problem closely in Chapter 5, we make use of the Kullback-Liebler based KD-tree approximation described in that chapter. As we described, the KD-tree method has the advantage of being relatively computationally efficient; however, more traditional methods such as Expectation-Maximization [2] could also be employed. Note that locally, each node still maintains a sample-based density estimate (allowing tests for multimodality, etc.) regardless of how coarsely the messages to its neighbors are approximated.

In order to observe the behavior of NBP with message approximations in a problem with multimodal uncertainty, we performed 100 Monte Carlo trials of NBP with measurement outliers as in Section 6.6, but approximated each message by a fixed number of mixture components before transmitting. We apply the sequential schedule described in the previous section. Figure 6.12(b) shows the resulting errors in the estimated marginal, as measured by KL-divergence from a solution obtained by exact message-passing with 1000 particles, and plotted as a function of the number of retained components. Single Gaussian, and thus unimodal, approximations to the beliefs resulted in a slight loss in performance, while two-component estimates, which have some potential to represent bimodalities, proved better at capturing the uncertainty. As a benchmark, representing each two-dimensional Gaussian component costs at most 4β bits, so that a two-component mixture at $\beta = 16$ is less than 128 bits per message.

6.9 Discussion

In this chapter, we proposed a novel approach to sensor localization, applying a graphical model framework and using the nonparametric message–passing algorithm of Chapter 3 to solve the ensuing inference problem. The methodology has a number of advantages. First, it is easily distributed, exploiting *local* computation and communications between nearby sensors and potentially reducing the amount of communications required. Second, it computes and makes use of estimates of the uncertainty, which may subsequently be used to determine the reliability of each sensor's location estimate. The estimates easily accommodate complex, multi-modal uncertainty. Third, it is straightforward to incorporate additional sources of information, such as a model of the probability of obtaining a distance measurement between sensor pairs. Lastly, in contrast to other methods, it is easily extensible to non-Gaussian noise models, which may be used to model and increase robustness to measurement outliers. In empirical simulations, NBP's performance is comparable to the centralized MAP estimate, while

additionally producing useful measures of the uncertainties in the resulting estimates.

We have also shown how modifications to the NBP algorithm can result in improved performance. The NBP framework easily accommodates an outlier process model, increasing the method's robustness to a few large errors in distance measurements for little to no computation and communication overhead. Also, carefully chosen proposal distributions can result in improved small–sample performance, reducing the computational costs associated with calibration. Finally, appropriate message schedules require very few message transmissions, and reduced–complexity representations may be applied to lessen the cost of each message transmission with little or no impact on the final solution.

There remain many open directions within sensor localization for future research. First, other message–passing inference algorithms, for example the max–product algorithm, might improve localization performance if adapted to high–dimensional non-Gaussian problems. Also, alternative graphical model representations may bear investigating; it may be possible to retain fewer edges, or improve the accuracy of BP by clustering nodes, i.e., grouping tightly connected variables, performing optimal inference within these groups, and passing messages between groups [112]. Given its promising performance and many possible avenues of improvement, NBP appears to provide a useful tool for estimating unknown sensor locations in large *ad-hoc* networks.

Chapter 7

Conclusions and Future Directions

THIS chapter begins with a high-level summary of the major themes and contributions of the thesis. However, there remain a plethora of open questions which have either been raised by, or simply remain unaddressed by, our work. Accordingly, we describe some of these open areas for future research and some suggestions for how progress might be made on these difficult problems.

7.1 Summary and Contributions

Graphical models and belief propagation have already received some attention for their applicability to problems arising in sensor networks. In particular, tree–structured and loopy graphical models have been applied to perform inference over systems defined using discrete and jointly Gaussian random variables [16, 77]. Additionally, the distributed particle filtering methods common in tracking are an example of applying sample–based representations to a simple chain–structured graphical model. In this thesis we worked to combine and extend these ideas, as well as consider some of the important yet unexplored aspects of distributing the inference process within a sensor network.

At a high level, the exploration of nonparametric, sample–based inference techniques for sensor networks provided by this thesis has several major contributions.

- To extend particle filtering methods to general graphical models
- To provide an improved understanding of the behavior of approximations in belief propagation, and of belief propagation in general
- To expose and explore some of the issues underlying the communication of sample–based representations
- To describe how graphical models and sample–based estimates of uncertainty are applicable to a number of tasks in sensor networks

The first of these aspects was dealt with in Chapter 3, in which we described both the general structure and theory behind the NBP algorithm as well as the tools required for an efficient implementation, in particular the process of drawing samples from the

product of several messages. The second topic formed the core of Chapter 4, in which we provided a stability analysis of belief propagation. This analysis provides both a basis for understanding the consequences of the approximations which are a necessary part of any distributed and communication constrained implementation of belief propagation. Interestingly, it also provided insight into the behavior and properties of belief propagation more generally.

Chapter 5 considered the problem of communication constraints and sample–based representations in more detail. This is a problem which has received surprisingly little attention so far, and has a number of intriguing aspects. We described a minimum representation size for a collection of samples under certain conditions and attempted to give some insight into the most important elements of encoding methods for sample sets. More generally, we considered lossy or approximate encoding, and provided one example encoding method based on the KD-tree data structure which was capable of both lossy and lossless encoding.

In each chapter, we considered some relatively small example problems to illustrate the elements developed in that chapter. In Chapter 6 we provided a more in-depth analysis of one particular example problem. Putting our results from the first three themes together, we described an NBP-based solution to the self-localization problem, which is a particularly important task in sensor networks. We showed how the problem itself can be framed as a graphical model with cycles, with nonlinear and non-Gaussian elements. By applying NBP we showed that the sensor locations can be estimated quite well, and moreover we observed little degradation of performance even under communication constraints by applying the aforementioned lossy encoding method.

■ 7.2 Suggestions and Future Research

In addition to the contributions described, this thesis also raises a number of open questions which bear further investigation. We describe several of these open problems in the subsequent sections, and provide some suggestions for how they might be addressed.

■ 7.2.1 Communication costs in distributed inference

One of the primary contributions of Chapter 5 was the introduction of the problem of finding efficient representations and minimizing communication for sample–based estimates of uncertainty. This problem opens a plethora of questions having to do with how efficiently these distributions may be represented under various circumstances.

For example, with regard to the lossless representation and encoding of a sample set, we described the optimal representation cost when the distribution from which the samples are drawn is known at both sender and receiver. It is interesting to ask whether it is possible (or whether it is provably impossible) to create adaptive methods which always approach the optimal performance without requiring prior knowledge of the source distribution. In traditional source coding such methods are known as "universal" encoders [28]. Of course, the total cost is a function of both the number of

samples N and the desired resolution β; for fixed β, the average cost per sample will always eventually decrease toward zero. Thus, it may be more appropriate to ask either how this cost behaves when β is also increased (i.e., as a function of N), or perhaps what the relationship is between the minimal and observed *total* representational cost (as opposed to the average).

Lossy encoding is equally interesting to consider. We presented one method of encoding approximations to sample–based distribution estimates, but many others are certainly possible. Further work along these lines may be able to provide novel encoders which are more efficient (produce smaller representations) than the method we described. In particular, it would be helpful to understand the fundamental cost of representing or encoding of general distribution estimates, such as those obtained via EM or other approximation methods.

Finally, we raised a number of open questions in Chapter 5 relating to the issues of message encoding and approximation in iterative message passing algorithms. When several approximations to the same message, perhaps resulting from versions of that message created using incomplete information, are sent from one sensor S_1 to another S_2 over a period of time, the previous versions of each message provide information useful for encoding the next approximation. However, it remains unclear how that information can be used effectively. Perhaps a similar analysis to the stability work in Chapter 4 can be used to analyze BP when the changes in messages from iteration to iteration are restricted in some simple form.

Another aspect which must arise in iterative algorithms is due to the potential for feedback. When information is sent back from S_2 to S_1, it may be possible to further improve the encoding process. This feedback may be motivated purely by improving the representation of S_1's message, or may stem from S_2 communicating related information, for example an inference message from S_2 to S_1.

It may also be possible to exploit more global knowledge of the joint statistical model. However, finding constructive methods of encoding in these cases can be difficult even in traditional source coding problems [20, 80, 91]; it is unclear whether the problem becomes easier or harder by taking the viewpoint of transmitting messages which directly represent likelihood functions or posterior versions of probability distributions.

Finally, although we have confined our questioning in this area to sample–based estimates of uncertainty, many of the same questions can be asked for messages which consist of distributions or likelihoods defined over a discrete state space. Lossless and lossy encoding of these messages is still of major importance for distributed inference and estimation, yet remains relatively unexplored. It may be possible to use similar ideas to those described in Chapter 5 to obtain significant reduction in the communications cost of, say, discrete belief propagation in sensor networks, particularly in the relatively common case that the discrete state space results from a quantization of some continuous–valued random variable.

7.2.2 Graphical models and belief propagation

In addition to the questions of how best to represent and communicate messages in distributed inference, we have also raised a number of questions relating to graphical models and the belief propagation algorithm itself. In particular, Chapter 4 contributes significantly to our understanding of belief propagation, and the process of approximations in BP messages.

One immediate question which arises from our viewpoint of "errors" in BP messages is how a controlled level of error in the BP messages can be exploited in order to reduce the required computational effort. Nonparametric belief propagation, for example, can be interpreted as an efficient implementation making use of approximate messages, but we currently have no way of measuring how close or far these sample–based messages are from the correct (exactly computed) message, which in even the most mild problems considered in this thesis consists of a mixture of an exponentially large number of Gaussian components. It may be easier to find an interpretation for, or principled modification of, some other form of efficient approximate BP, for example adaptive methods of reducing state dimension [13, 14].

Another open issue is the local stability of BP fixed points, as opposed to the global stability implied by our analysis. When there are multiple BP fixed points, our current analysis is unable to tell us the answers to such questions as how many fixed points there are, or how many of these are stable. Perhaps given a particular fixed point, it may be possible to use a similar analysis of error propagation to demonstrate a region of local stability. More hypothetically, understanding the local stability properties of BP might enable us to infer how many stable fixed points are possible for a given graphical model.

There is an interesting gap between our analysis of when loopy BP converges, and the more commonly asked question of how well it performs, i.e., how closely the fixed point obtained via loopy BP matches the correct marginal distributions. BP is, of course, exact on tree–structured graphs, for which convergence is trivially guaranteed, and the conventional understanding of BP is that it does well on "tree–like" graphs, with long, weak cycles. Thus, one might conjecture that the performance of BP is closely related to its convergence behavior. Perhaps the convexity bounds of Wainwright [102] or other results on the quality of BP might be combined with our error analysis, or examined in the special case of graphs with guaranteed convergence properties, to obtain a more precise statement relating the two qualities.

If there is a link between BP convergence and the quality of BP's marginal estimates, the convergence criterion and fixed point properties described in Chapter 4 may be useful in many other ways. For example, our convergence criterion could be used to assess how appropriate a given graphical model might be, given that we intend to use BP to perform approximate inference on that model. For example, we might be able to use our criterion to select from among many possible graph structures, for example to find turbo or LDPC codes [24] which are likely to have good performance. Alternatively, we may be able to use our convergence criterion to inform algorithms for finding graph

structure and parameter selection in machine learning problems, limiting a search to graphical models which are known to lead to good behavior of BP.

■ 7.2.3 Nonparametric belief propagation

Nonparametric belief propagation provides one way of performing approximate inference over complex distributions defined on general graphical models. NBP is applicable to a wide variety of otherwise difficult problems, particularly those involving relatively high–dimensional continuous–valued random variables. In order to make this inference algorithm even more useful, there are a number of questions which bear further investigation.

For example, in this thesis we have often applied the technique of "belief sampling" in order to reduce computation and use fewer messages in the inter-sensor communication process. However, there remain many open issues with belief sampling. As we discussed in Section 6.7, the selection of the proposal distribution can have a significant impact. Finding ways to select good proposal distributions, perhaps adaptively, is an important sub-problem. Furthermore, given a set of samples from the belief M_t, we used a simple reweighting technique to represent samples from the product M_{ts}. This may not be the best way to accurately represent M_{ts}; we should consider whether and how this approximation can be improved. Being able to represent the messages and message products using relatively few samples is extremely important, in part to reduce the communications required, but also to decrease the computational overhead of NBP.

This leads to another important question—whether it is possible to automatically determine an appropriate number of samples N to represent each message. It is important that N be small enough to enable computations to be performed efficiently; however, if N is too small, the approximate messages are not accurate and do not lead to good marginal estimates. If we could detect the complexity of the distribution somehow, we might be able to determine whether a particular value of N was sufficiently large.

A related question is whether it is possible to estimate the relative error in our message approximations. Even for a given value of N, each message computation involves an approximation, in which a product distribution of N^d Gaussian mixtures is approximated by only N samples. If we could estimate the level of errors introduced in each of these steps, we might be able to determine how closely the solutions obtained by NBP approximate the ideal, continuous BP estimates.

If some or all of these questions could be answered, the approximations made by NBP could be understood in a more principled manner, particularly when the number of samples N is very small. This could both assist in justifying NBP's application to, and improving its performance on, many difficult estimation problems.

■ 7.2.4 Other sensor network applications

Finally, in this thesis, we considered the sensor localization problem in some depth, as well as somewhat simplified versions of several other sensor network applications such

as tracking mobile objects (Sections 3.9.2 and 5.7.2) and estimating spatial random processes (Section 5.7.3). Many of these problems could benefit from a closer, more detailed examination using the results of the thesis.

For example, distributed tracking of multiple moving objects is one of the most common applications for sensor networks. Considering how more sophisticated graphical models may be applied to capture the temporal dynamics of and statistical relationships between multiple targets is one important open problem. Distributing the computational effort while limiting the required communications overhead is equally important. While much work has gone into thinking about certain aspects of these problems, such as distributing the representations of objects which operate independently [61] and selecting what sensors are responsible for storing and updating the representations [113, 114], these problems are not independent of the cost of communicating the representations and should be considered with these costs in mind.

More generally, object tracking is also a localization problem, and may even be considered jointly with the process of estimating sensor locations. Sensors may also wish to calibrate themselves in other ways, by estimating additional variables which have impact on the sensing process (for example, antenna or microphone gain or the level of ambient noise or interference). We should be able to frame these additional problems as graphical models, develop relatively sparse, local approximations suitable for performing distributed estimation, and perhaps solve them efficiently using NBP as well.

Bibliography

[1] Ibrahim A. Ahmad and Pi-Erh Lin. A nonparametric estimation of the entropy for absolutely continuous distributions. *IEEE Transactions on Information Theory*, 22(3):372–375, May 1976.

[2] M. Aitkin and D. B. Rubin. Estimation and hypothesis testing in finite mixture models. *Journal of Royal Statistical Society, Series B*, 47(1):67–75, 1985.

[3] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[4] J. Beirlant, E. J. Dudewicz, L. Györfi, and E. C. van der Meulen. Nonparametric entropy estimation: An overview. *International Journal of Math. Stat. Sci.*, 6(1):17–39, June 1997.

[5] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[6] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Uncertainty in Artificial Intelligence*, pages 33–42, 1998.

[7] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.

[8] M. A. Carreira-Perpinan. Mode-finding for mixtures of gaussian distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1318–1323, 2000.

[9] Hei Chan and Adnan Darwiche. A distance measure for bounding probabilistic belief change. *International Journal of Approximate Reasoning*, 38(2):149–174, Feb 2005.

[10] L. Chen, M. Wainwright, M. Cetin, and A. Willsky. Data association based on optimization in graphical models with application to sensor networks. Submitted to *Mathematical and Computer Modeling*, 2004.

[11] P. Clifford. Markov random fields in statistics. In G. R. Grimmett and D. J. A. Welsh, editors, *Disorder in Physical Systems*, pages 19–32. Oxford University Press, Oxford, 1990.

[12] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 1990.

[13] J. M. Coughlan and S. J. Ferreira. Finding deformable shapes using loopy belief propagation. In *European Conference on Computer Vision 7*, May 2002.

[14] J. M. Coughlan and H. Shen. Shape matching with belief propagation: Using dynamic quantization to accomodate occlusion and clutter. In CVPR Workshop on Generative Model Based Vision, 2004.

[15] T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[16] Christopher Crick and Avi Pfeffer. Loopy belief propagation as a basis for communication in sensor networks. In *Uncertainty in Artificial Intelligence 18*, August 2003.

[17] K. Deng and A. W. Moore. Multiresolution instance-based learning. In *International Joint Conference on Artificial Intelligence*, 1995.

[18] L. Doherty, L. El Ghaoui, and K. S. J. Pister. Convex position estimation in wireless sensor networks. In *Infocom*, Apr 2001.

[19] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, New York, 2001.

[20] Stark Draper. Universal incremental slepian–wolf coding. In *Allerton Conf. on Comm., Control, and Computing*, October 2004.

[21] T. Eren, D. Goldenberg, W. Whiteley, Y. R. Yang, A. S. Morse, B. D. O. Anderson, and P. N. Belhumeur. Rigidity, computation, and randomization in network localization. In *International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2673–2684, March 2004.

[22] M. Fazel, H. Hindi, and S. P. Boyd. Log-det heuristic for matrix rank minimization with applications to Hankel and Euclidean distance matrices. In *Proceedings, American Control Conference*, 2003.

[23] W. T. Freeman and E. C. Pasztor. Markov networks for low–level vision. Technical Report 99-08, MERL, February 1999.

[24] B. Frey, R Koetter, G. Forney, F. Kschischang, R. McEliece, and D. Spielman (Eds.). Special issue on codes and graphs and iterative algorithms. *IEEE Transactions on Information Theory*, 47(2), February 2001.

[25] M. Gastpar and M. Vetterli. Source-channel communication in sensor networks. In L. Guibas and F. Zhao, editors, *Information Processing in Sensor Networks*. Springer-Verlag, 2003.

[26] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

[27] Hans-Otto Georgii. *Gibbs measures and phase transitions*. Studies in Mathematics. de Gruyter, Berlin / New York, 1988.

[28] A. Gersho and R. M. Gray. *Vector quantization and signal compression*. Kluwer, Boston, 1991.

[29] H. Gharavi and S. Kumar. Special issue on sensor networks and applications. *Proceedings of the IEEE*, 91(8):1151–1153, August 2003.

[30] Mark Girolami and Chao He. Probability density estimation from optimally condensed data samples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1253–1264, October 2003.

[31] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian bayesian state estimation. *IEE Proceedings on Radar and Signal Processing*, 140:107–113, 1993.

[32] A. G. Gray and A. W. Moore. Very fast multivariate kernel density estimation via computational geometry. In *Joint Stat. Meeting*, 2003.

[33] L. Greengard and V. Rokhlin. The rapid evaluation of potential fields in three dimensions. In C. Greengard (Eds.) C. Anderson, editor, *Vortex Methods*, volume 1360 of *Lecture Notes in Mathematics*, pages 121–? Springer-Verlag, Berlin, 1988.

[34] L. Greengard and J. Strain. The fast Gauss transform. *SIAM J. Sci Stat Comput*, 12(1):79–94, 1991.

[35] L. Greengard and X. Sun. A new version of the fast Gauss transform. *Documenta Mathematica*, Extra Volume ICM(III):575–584, 1998.

[36] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

[37] T. Heskes. On the uniqueness of loopy belief propagation fixed points. *Neural Computation*, 16(11):2379–2413, 2004.

[38] Jason L. Hill and David E. Culler. MICA: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, Nov–Dec 2002.

[39] G. E. Hinton. Training products of experts by minimizing contrastive divergence. Technical Report 2000-004, Gatsby Computational Neuroscience Unit, 2000.

[40] A. T. Ihler, J. W. Fisher III, R. L. Moses, and A. S. Willsky. Nonparametric belief propagation for self-calibration in sensor networks. In *Information Processing in Sensor Networks*, 2004.

[41] A. T. Ihler, J. W. Fisher III, R. L. Moses, and A. S. Willsky. Nonparametric belief propagation for sensor self-calibration. In *International Conference on Acoustics, Speech, and Signal Processing*, 2004.

[42] A. T. Ihler, J. W. Fisher III, R. L. Moses, and A. S. Willsky. Nonparametric belief propagation for self-calibration in sensor networks. *Submitted to IEEE Journal on Selected Areas in Communications*, 2004.

[43] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. Communication-constrained inference. Technical Report 2601, MIT, Laboratory for Information and Decision Systems, 2004.

[44] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. Message errors in belief propagation. Technical Report 2602, MIT, Laboratory for Information and Decision Systems, 2004.

[45] A. T. Ihler, J. W. Fisher III, and A. S. Willsky. Particle filtering under communication constraints. In *Submitted toInformation Processing in Sensor Networks*, 2005.

[46] A. T. Ihler, E. B. Sudderth, W. T. Freeman, and A. S. Willsky. Efficient multiscale sampling from products of Gaussian mixtures. In *Neural Information Processing Systems 17*, 2003.

[47] Alexander Ihler. Kernel density estimation toolbox for matlab.

[48] M. Isard. PAMPAS: Real–valued graphical models for computer vision. In *IEEE Computer Vision and Pattern Recognition*, 2003.

[49] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[50] Julian Alan Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413):205–224, March 1991.

[51] Neha Jain, M. Dilip Kutty, and Dharma P. Agrawal. Energy aware multipath routing for uniform resource utilization in sensor networks. In *Information Processing in Sensor Networks*, pages 473–487, April 2003.

[52] Harry Joe. Estimation of entropy and other functionals of a multivariate density. *Annals of the Institute of Statistical Mathematics*, 41(4):683–697, 1989.

[53] V. M. Bove Jr. and J. Mallett. Collaborative knowledge building by smart sensors. *BT Technology Journal*, 22(4):45–51, 2004.

[54] S. Julier and J. Uhlmann. A general method for approximating nonlinear transformations of probability distributions. Technical report, RRG, Dept. of Eng. Science, Univ. of Oxford, 1996.

[55] John C. Kieffer. A tutorial on hierarchical lossless data compression. In *Modelling Uncertainty*, volume 46 of *Internat. Ser. Oper. Res. Management Sci.*, pages 711–733. Kluwer, Boston, MA, 2002.

[56] D. Koller, U. Lerner, and D. Angelov. A general algorithm for approximate inference and its application to hybrid Bayes nets. In *Uncertainty in Artificial Intelligence 15*, pages 324–333, 1999.

[57] S. Kumar, F. Zhao, and D. Shepherd. Collaborative signal and information processing in microsensor networks. *IEEE Signal Processing Magazine*, 19(2):13–14, March 2002.

[58] N. Kurata, B. F. Spencer Jr., M. Ruiz-Sandoval, Y. Miyamoto, and Y. Sako. A study on building risk monitoring using wireless sensor network MICA mote. In *International Conference on Structural Health Monitoring and Intelligent Infrastructure (SHMII)*, 2003.

[59] Koen Langendoen and Niels Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Computer Networks*, 43(4):499–518, November 2003.

[60] S. L. Lauritzen. *Graphical Models*. Oxford University Press, Oxford, 1996.

[61] J. J. Liu, J. Liu, M. Chu, J. E. Reich, and F. Zhao. Distributed state representation for tracking problems in sensor networks. In *Information Processing in Sensor Networks*, pages 234–242, 2004.

[62] J. S. Liu and C. Sabatti. Generalised Gibbs sampler and multigrid Monte Carlo for Bayesian computation. *Biometrika*, 87(2):353–369, 2000.

[63] Jessica D. Lundquist, Daniel R. Cayan, and Michael D. Dettinger. Meteorology and hydrology in yosemite national park: A sensor network application. In F. Zhao and L. Guibas, editors, *Information Processing in Sensor Networks*, pages 518–528. Springer-Verlag, 2003.

[64] J. MacCormick and A. Blake. Probabilistic exclusion and partitioned sampling for multiple object tracking. *International Journal of Computer Vision*, 39(1):57–71, 2000.

[65] David MacKay. Introduction to monte carlo methods. In M. I. Jordan, editor, *Learning in Graphical Models*. MIT Press, 1999.

[66] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In C. S. Raghavendra and Krishna M. Sivalingam, editors, *International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Atlanta, GA, USA, 2002. ACM.

[67] Y. Mao and A. Banihashemi. Decoding low-density parity check codes with probabilistic scheduling. *IEEE Communications Letters*, 5(10):414–416, October 2001.

[68] R. Min, M. Bhardwaj, S. Cho, N. Ickes, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan. Energy-centric enabling technologies for wireless sensor networks. *IEEE Wireless Communications Magazine*, 9(4):28–39, August 2002.

[69] T. Minka. Expecatation propagation for approximate bayesian inference. In *Uncertainty in Artificial Intelligence*, 2001.

[70] Andrew Moore. Very fast em-based mixture model clustering using multiresolution kd-trees. In *Neural Information Processing Systems 11*, pages 543–549, 1999.

[71] Andrew Moore. The anchors hierarchy: Using the triangle inequality to survive high-dimensional data. In *Uncertainty in Artificial Intelligence 12*, pages 397–405. AAAI Press, 2000.

[72] R. Moses, D. Krishnamurthy, and R. Patterson. Self-localization for wireless networks. *Eurasip Journal on Applied Signal Processing*, 2003.

[73] R. Moses and R. Patterson. Self-calibration of sensor networks. In *SPIE vol. 4743: Unattended Ground Sensor Technologies and Applications IV*, 2002.

[74] R. Moses, R. Patterson, and W. Garber. Self localization of acoustic sensor networks. In *Military Sensing Symposia (MSS) Specialty Group on Battlefield Acoustic and Seismic Sensing, Magnetic and Electric Field Sensors*, 2002.

[75] Stephen M. Omohundro. Five balltree construction algorithms. Technical Report TR-89-063, ICSI, U.C. Berkeley, 1989.

[76] E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[77] M. A. Paskin and C. E. Guestrin. Robust probabilistic inference in distributed systems. In *Uncertainty in Artificial Intelligence 20*, 2004.

[78] Neal Patwari and Alfred Hero. Relative location estimation in wireless sensor networks. *IEEE Transactions on Signal Processing*, 51(8):2137–2148, August 2003.

[79] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufman, San Mateo, 1988.

[80] S. S. Pradhan and K. Ramchandran. Distributed source coding using syndromes (discus): Design and construction. *IEEE Transactions on Information Theory*, 49:626–643, March 2003.

[81] N. Priyantha, H. Balakrishnan, E. Demaine, and S. Teller. Anchor-free distributed localization in sensor networks. Technical Report 892, MIT LCS, 2003.

[82] R. H. Randles and D. A. Wolfe. *Introduction to the Theory of Nonparametric Statistics*. Wiley, New York, 1979.

[83] Matthew Ridley, Eric Nettleton, Ali Göktogan, Graham Brooker, Salah Sukkarieh, and Hugh F. Durrant-Whyte. Decentralised ground target tracking with heterogeneous sensing nodes on multiple uavs. In F. Zhao and L. Guibas, editors, *Information Processing in Sensor Networks*, pages 545–565. Springer-Verlag, 2003.

[84] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27(3):832–837, September 1956.

[85] Andreas Savvides, Heemin Park, and Mani B. Srivastava. The bits and flops of the n-hop multilateration primitive for node localization problems. In *ACM Workshop on Wireless Sensor Networks and Applications*, pages 112 – 121. ACM, 2003.

[86] D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 1992.

[87] J. M. Shapiro. Embedded image-coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[88] L. Sigal, S. Bhatia, S. Roth, M. J. Black, and M. Isard. Tracking loose-limbed people. In *IEEE Computer Vision and Pattern Recognition*, 2004.

[89] L. Sigal, M. Isard, B. H. Sigelman, and M. J. Black. Attractive people: Assembling loose-limbed models using non-parametric belief propagation. In *Neural Information Processing Systems 16*, 2003.

[90] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, New York, 1986.

[91] D. Slepian and J. Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19:471–480, July 1973.

[92] J. Strain. The fast Gauss transform with variable scales. *SIAM Journal on Scientific and Statistical Computing*, 12(5):1131–1139, 1991.

[93] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. In *IEEE Computer Vision and Pattern Recognition*, 2003.

[94] E. B. Sudderth, M. I. Mandel, W. T. Freeman, and A. S. Willsky. Distributed occlusion reasoning for tracking with nonparametric belief propagation. In *Neural Information Processing Systems*, 2004.

[95] S. Tatikonda and M. Jordan. Loopy belief propagation and gibbs measures. In *Uncertainty in Artificial Intelligence*, 2002.

[96] S. Thrun, J. Langford, and D. Fox. Monte Carlo HMMs. In *International Conference on Machine Learning*, pages 415–424, 1999.

[97] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. In *Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

[98] M. W. Trosset. The formulation and solution of multidimensional scaling problems. Technical Report TR93-55, Rice University, 1993.

[99] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Tree–based reparameterization for approximate inference on loopy graphs. In *Neural Information Processing Systems 14*. MIT Press, 2002.

[100] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky. Tree–based reparameterization analysis of sum–product and its generalizations. *IEEE Transactions on Information Theory*, 49(5), May 2003.

[101] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. Technical Report 629, UC Berkeley Dept. of Statistics, September 2003.

[102] Martin J. Wainwright. *Stochastic processes on graphs with cycles: geometric and variational approaches*. PhD thesis, MIT, 2002.

[103] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman & Hall, 1995.

[104] Y. Weiss. Belief propagation and revision in networks with loops. Technical Report 1616, MIT AI Lab, 1997.

[105] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12(1), 2000.

[106] Y. Weiss and W. T. Freeman. On the optimality of solutions of the Max–Product Belief–Propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744, February 2001.

[107] Yair Weiss and William T. Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200, 2001.

[108] K. Whitehouse. The design of calamari: an ad-hoc localization system for sensor networks. Master's thesis, U. C. Berkeley, 2002.

[109] A. Willsky. Relationships between digital signal processing and control and estimation theory. *Proceedings of the IEEE*, 66(9):996–1017, September 1978.

[110] A. Willsky. Multiresolution markov models for signal and image processing. *Proceedings of the IEEE*, 90(8):1396–1458, August 2002.

[111] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *International Joint Conference on Artificial Intelligence*, August 2001.

[112] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report 2004-040, MERL, May 2004.

[113] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information-directed approach. *Proceedings of the IEEE*, 91(8):1199–1209, August 2003.

[114] F. Zhao, J. Shin, and J. Reich. Information-driven dynamic sensor collaboration for tracking applications. *IEEE Signal Processing Magazine*, 19(2):61–72, March 2002.