TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Netzwerktheorie und Signalverarbeitung

Approximate inference algorithms in large networked systems

Chongning Na

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: ***Univ.-Prof. Dr.-Ing. Eckehard Steinbach***
Prüfer der Dissertation:
1. ***Univ.-Prof. Dr.techn. Josef A. Nossek***
2. ***Univ.-Prof. Dr.-Ing. habil. Gerhard Rigoll***

Die Dissertation wurde am ***27.11.2009*** bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am ***26.04.2010*** angenommen.

Acknowledgements

During the course of my Ph.D., I have received assistant and support from my supervisors, colleagues, family and friends. I am grateful for this opportunity to acknowledge these people and their kindly contributions.

First and foremost I want to express my deep gratitude to my advisor Dr. Dragan Obradovic for his dedicated guidance, enlightening suggestions and enthusiastic discussions from the initial to the final level. I am very fortunate to work with him. I am also thankful for his offering of the research position at Siemens AG, Corporate Technology for me, with which I am able to finish my Ph.D. and participate several interesting projects. Using these opportunities, I have grown tremendously as a researcher. I would like to thank Professor Josef A. Nossek who also supervised my thesis, gave me many useful advises and shared his vision and experience with me. I could not wish for a better or friendlier supervisor.

I would like express my gratitude to Dr. Ruxandra Lupas Scheiterer for her generous encouragement and help. She patiently taught me many important skills, from writing good articles to making impressive presentations. I believe that I will benefit from those skills throughout my research career. I would like to thank my colleagues, Dr. Andrei Szabo, Dr. Marco Pellegrino and Dr. Philipp Wolfrum for their suggestions and support in my work. I am thankful for the help and friendship of all my officemates.

I would like to thank Siemens AG, Corporate Technology for funding my Ph.D. research. I appreciated the support from Professor Bernd Schürmann and Dr. Thomas Runkler. During my stay at Siemens, I had the opportunity to work on several industrial projects which gave me valuable experiences.

On a personal level, I am most grateful to my family, my parents Na Lvguan and Guo Lianqing, my sister Na Yichao and my brother-in-law Cai Xiaochuan. They have always loved and supported me. I know they are very happy and proud for me when I am getting this degree. I am grateful to my close friends Fu Bo, Wang Hui, Sun Yiming and Liu Junqi. They made my life in Munich an enjoyable experience and their sincere friendship was invaluable to me.

Last but not least, I thank the MSCE program and all the professors who have supported this program. It was this master program which brought me to Germany and offered me the invaluable opportunity to study at the Technical University of Munich, which will be a precious experience both for my career and life.

Contents

List of Figures

List of Tables

Abstract

Networked systems have become an essential part of our daily life over the last decades. Many applications benefit from the information exchange enabled by networking. Obtaining the desired behavior of the system usually relies on the estimation of certain critical quantities. However, estimation in a networked system is not a trivial task due to the large amount of participants, the complex uncertainties and processes. Graphical models provide a powerful tool for organizing all sources of information and describing explicitly the statistical structure of the estimation problem. The whole system is then characterized by the joint probability distribution of a large collection of random variables. Estimation is performed by inferring the posterior probability distributions of the variables that are related to the quantities of interest, given the observation variables. A large number of efficient inference algorithms are available which exploit the specific structures embedded in the graphical models. The focus of this thesis is the development of inference algorithms that solve the estimation problems in large networked systems. These new methods are based on the existing inference algorithms but extend them to adapt to the specific properties of applications in networked systems.

Many applications require a distributed implementation of inference due to constraints on power consumption, computational complexity, maximum allowable latency, etc. Some existing inference algorithms (e.g. belief propagation and the sum-product algorithm) can be formulated in a message passing style. Based on these algorithms, we present a basic framework for distributed implementation of inference in a networked system where computations needed by the inference are carried out locally and messages are passed via the communication links.

Some estimation problems involve very complicated processes which generate random variables that do not have a closed form distribution. Appropriate representation of the random variables or their distributions is needed to simplify the computation and to reduce the communication required by message passing. Two approaches are considered in the thesis. First we summarize sample-based representation, i.e., non-parametric belief propagation. Then we develop Fourier domain belief propagation which approximates the distribution via truncated Fourier series expansions.

Some applications in networked systems do not have a convenient statistical structure which forbids exact inference to be implemented in a distributed manner. In particular, in networked dynamical systems, the distribution of the hidden state loses its indepen-

dence structure with the evolution of the process. In this case, we introduce approximate distributions with simpler structures which enable distributed inference.

Finally, distributed and approximate inference methods are applied to solve two practical problems: node localization and clock synchronization. Calibrating the coordinates of networked nodes in time and space is a fundamental requirement for many applications. We demonstrate Fourier domain belief propagation and non-parametric belief propagation on the problem of self-organized sensor localization. In the clock synchronization problem, we use a probabilistic model to quantify the uncertainties of the observation and to describe the stochastic processes involved. Then synchronization is formulated as an inference problem. Based on its graphical model, we first develop a centralized inference method which produces exact results but is practically impossible to implement. Then we develop distributed inference algorithms which are not exact, but are feasible for realization in practice.

1. Introduction

1.1 Background

Nowadays, networks are playing a more and more important role in our daily life. Networked systems are widely used in many applications. The most typical and important examples include telecommunications, sensor networks, industrial automation and control networks, the power grid, transportation systems, social networks and many more. A networked system is usually composed of a large number of spatially distributed nodes which are connected via an underlying communication network so that they can interact with each other. Information is propagated through the network which enables the whole system to work in a cooperative fashion or to have an optimal performance in the global sense. Achieving desired behaviors of the entire system requires reliable estimation of the state of the system, which is not a trivial task due to inaccurate information, incomplete knowledge and the large number of network participants.

Two estimation problems are of special interest in a networked system: node localization and clock synchronization. Calibrating the coordinates of the network nodes in time and space is the fundamental requirement for many other applications to function correctly. For example, in a sensor network which measures a physical field, without a precise knowledge about each sensor's position and without calibrated clocks, it is not possible to construct the correct model that describes the distribution of the physical field in space and time. Another example is in automation networks. Imagine that several robot arms operate together on a product. All of them have to follow a predefined schedule with critical time constraints. Without clock synchronization, the robot arms can not coordinate their actions which may lead to severe consequences.

These two estimation problems, which are representative of many other problems, have some characteristics in common. First of all, the parameters of interest are usually not directly measured, but are related to some other parameters that can be observed. Secondly, there are a lot of uncertainties involved, e.g., noisy observations, random behavior of the network, stochasticity of the process at each node and so on. Third, there exist a lot of spatial and temporal correlations between the parameters. Furthermore, many networked system contains a large number of nodes. Estimation on such a large scale is not trivial.

In order to obtain an optimal estimation, we need to find a reasonable way to organize all the available information, which includes:

- the noisy measurements
- our prior knowledge on the relationship between the measurements and the quantities of interest. This knowledge often comes from the physics of the problem.
- our prior knowledge on the spatial and temporal correlations
- structure, i.e., topology of the network.

Probabilistic graphical models provide a powerful tool that assembles all the information mentioned above. They use probability functions to quantify the uncertainties. In this approach, the physical parameters of interest are characterized by random variables. Probability functions define the distribution of randomness and model the strength of the spatial and temporal correlation. In this way, the system is modeled by a joint probability or a global function of large, complex collections of variables, from which we can compute the probability of a specific configuration of the random variables. A graphical model visualizes the probabilistic relationships between those variables, which offers a framework for a deeper understanding of the nature of the problem. In some cases, the structure of the probabilistic model (dependence or independence relations) can be modified so as to adapt to the topology of the network, possibly by making approximations.

Using the graphical model, the estimation problem turns into a probabilistic inference problem. Based on the joint probability distribution, a set of quantities can be computed:

- likelihoods
- marginal distributions of sets of variables
- conditional distributions
- configuration of a set of variables that maximizes a given probability function (e.g., maximum a posteriori estimate, maximum likelihood).

Many estimation problems in networked system can be formulated as computing one or several of the above mentioned quantities. For example, in the sensor localization problem, we obtain noisy measurements of the mutual distances between sensors. Assuming knowledge of the distribution of the measurement noise, and having derived the relationship between the mutual distances and the sensor positions, we can pose sensor localization as a maximum a posteriori distribution problem, where based on the joint probability distribution of all random variables, we compute and maximize the posteriori distribution of the sensor positions, i.e., the conditional probability distribution of the sensor positions given the noisy measurements. Such an approach not only computes the estimate, but also provides a measure of the uncertainty, or the confidence of the estimation. Such a soft information could be very useful for the subsequent applications that rely on the estimation results. For example, in channel equalization, soft decision [59] methods assign a confidence value to each output bit obtained by hard decision [63]. Empirical implementation has verified that such a method improves the decoding process by reducing the probability of the decoding bit error [32].

Probabilistic inference has a rich coverage in the literature. Its successful applications can be found in bioinformatics, computer vision, image processing, speech processing, error-correct coding, etc. Many well-known algorithms can be interpreted as special cases of probabilistic inference. For example, Kalman filter is the posteriori distribution computation in a linear state space model with Gaussian distributions. The Viterbi algorithm is related to maximum likelihood estimation. Usually, modifications have to be made to the standard inference algorithms according to the requirement of the specific application. It is very common that similar inference algorithms have totally unrelated names in different communities.

1.2 Problem statement

Probabilistic inference in a networked system is not a trivial task since usually we face a large scale computation or optimization problem. It has been proved that conditional independence between the random variables plays a very important role in simplifying the inference. The relationship between the variables can be intuitively represented by a graphical model by casting our expectation of local conditional dependence or independence into the form of the graph, based on which we can easily judge the conditional dependence or independence of not directly related variables. If two random variables are independent, the distribution of each variable can be evaluated individually, while evaluating the distribution of statistically dependent variables has to consider jointly all possible configurations of the variables, resulting in an exponential complexity. Many inference algorithms exploit this property and carefully choose the sequence of computations so as to reduce the number of necessary operations. Such an approach simplifies the computation to some extent but could still be too expensive for many applications. Further simplifications are needed to make the problem tractable.

Several issues complicate the inference in large networked system. First of all, such an estimation problem usually operates on a large number of variables. Therefore, centralized inference usually has an extremely high complexity. For example, the state estimation of a linear Gaussian state space model with 100 state variables can be implemented by a centralized Kalman filter. But the computation involves multiplication and inversion of matrices with very high dimensions. In addition, to enable centralized inference, all the local information at the spatially distributed nodes has to be transmitted to an inference center. And some times the inference results are transmitted back to the nodes. Typically, this will consume a lot of transmission power and introduce extra delays, making centralized inference infeasible for systems with power constraints or time constraints. Therefore, it is always preferable to have a distributed inference method in which information is processed as locally as possible.

Another problem arises from the complexity of the probability functions in the system. Standard inference deals with discrete random variables or well known distributions, e.g., Gaussian distributions. However, complicated underlying physics introduces very complicated unknown distribution functions, which are not easily parameterized. Inappropriate representation of the distribution function results in high computational complexity and unnecessary waste of transmission bandwidth.

The third difficulty is the spatial correlation between the dynamics at different locations, which results in a probabilistic model with both spatial and temporal correlations. Although the individual temporal or spatial correlation is usually sparse, the combination of them introduces strong coupling between the random variables. As a consequence, the conditional independence structure is lost and the inference becomes very expensive. Exact inference usually leads to a centralized state-space model where at each time step, the joint distribution of all the variables has to be evaluated, which is not suitable for distributed inference and sometimes can be intractable.

This thesis studies how we can develop efficient inference algorithms that take the specific properties of estimation problems in networked systems into consideration. The abovementioned difficulties of inference in networked systems will be discussed in detail.

1.3 Contributions and overview of the thesis

Graphical Models and Probabilistic Inference. Chapter 2 and Chapter 3 review graphical models and inference algorithms. They provide the basic knowledge for the understanding of the subsequent chapters. We try to make our presentation as detailed as possible so that it would be easy to understand the concepts, even for people who are not familiar with probabilistic inference. We construct simple examples for illustration and provide proofs when necessary. Chapter 2 presents the three most commonly used graphical models, i.e., Bayesian networks, Markov random fields and factor graphs. We explain the nature of these graphical models and illustrate them with simple examples. We then discuss how to judge the conditional independence, one of the most important properties represented by graphical models. Chapter 3 starts with a simple inference example. Using that example, we explain the principle of inference algorithms: exploiting the independence properties to achieve simplifications. We present belief propagation for a graph with a tree structure. In particular, we present the sum-product algorithm for factor graphs and prove that it implements in an efficient way the computation of the marginal probability distributions. For graphs with loops, we present two approaches. The first one, i.e., loopy belief propagation applies the formulas of normal belief propagation on a graph with loops. As an iterative method, it is simple to implement but exactness or even convergence is not guaranteed. The second approach, i.e., the junction tree algorithm converts graphs with loops to trees and then runs the normal belief propagation. Junction tree algorithms produce exact results, but finding the optimal tree representation is an NP hard problem. At the end, we present the variational inference algorithms, e.g., the mean field method, which simplify inference by introducing approximations.

Probabilistic Inference in Networked Systems. Chapter 4 addresses three most important issues in the typical estimation problems in networked system: distributed inference, approximations and state estimation in dynamical systems. We first analyze how the inference and the communication influence each other. Then we propose a general procedure of distributed inference in a networked system. In the second part, we focus on the approximation of the distribution functions. Based on the Fourier density approximation method proposed in [11], we derive belief propagation based on Fourier densities, which we call Fourier domain belief propagation. Then we summarize a Monte Carlo sampling based distribution approximation approach, the non-parametric belief propagation. In the third part, we review state estimation for dynamical systems. We present dynamic Bayesian networks which visualize the relationships between the variables involved in a dynamical system. Then we briefly review the general inference algorithms applied on a dynamical Bayesian network, in particular, the interface algorithm and Boyen-Koller's approximation method. This chapter provides a general discussion of the problems encountered during inference in typical applications of networked systems. We analyze the nature of the problems and provide a comprehensive summary of the relevant techniques that solve these problems.

Self-Organized Sensor Localization. Chapter 5 applies the inference methods to the self-organized sensor localization problem in wireless sensor networks. Due to the nonlinearity of the functions involved in the model, we apply the function approximation methods introduced in Chapter 4. Distributed inference is implemented by using belief propagation. We develop simplified transmission based on Fourier series approximation which reduces transmission data using the Fourier transform. Then we derive Fourier domain belief propagation in which the functions are represented by truncated Fourier series. All the computations and transmissions of the relevant functions are based on the Fourier series. We show that using such a representation, we simplify both the computation and the transmission. For comparison, we also implement the non-parametric belief propagation for sensor localization. At the end we use simulation to verify the performance of different localization algorithms.

Clock Synchronization Of Networked Nodes. Chapter 6 shows the application of probabilistic inference to clock synchronization of cascaded network elements, where all the network participants synchronize their clocks to a master element which provides the reference time. We first briefly introduce the Precision Time Protocol (PTP) specified in the IEEE 1588 standard, which provides the basic framework for clock synchronization. The PTP protocol defines a mechanism to allow the elements to exchange timing information, based on which, different synchronization algorithms can be obtained. We propose several synchronization algorithms based on probabilistic inference. We first establish probabilistic models that assemble all the information that can help to improve the synchronization precision. Based on that model, we develop different inference algorithms. These algorithms realize centralized or distributed implementation of inference. We test and compare the performance of the algorithms through simulations under different scenarios.

Conclusions and Future Work. Chapter 7 summarizes the contribution of this thesis and discusses several directions for future research.

Some of the research results developed in this thesis have been published in conference proceedings and journals. Fourier domain belief propagation and its application to sensor localization was published in [85, 86]. Analysis of the synchronization performance of the

PTP protocol was summarized in [81, 97]. Probabilistic model, centralized and distributed inference algorithms were originally presented in [79], [84] and [80, 83] respectively.

2. Graphical Models

A graphical model represents the probabilistic model of a multivariate system on a graph. In general, a graphical model is a graph $\mathcal{G}$ in which a set of nodes $\mathcal{V}$ present random variables and the presence of edges $\mathcal{E}$ between nodes reveals the existence of a probabilistic relationship between the variables. The whole graph is a compact representation of a joint distribution or a global function, which can be expressed as the product of many local functions defined on the locally connected subsets of the variables in the graph.

Regarding the means to express the probabilistic relationships, there are two most common graphs: directed and undirected. A Bayesian network [39], or belief network is a typical directed graph in which edges carry arrows and the direction of an arrow reveals the causality. A Markov random field [58] or Markov network is an example of undirected graphical model in which edges do not carry arrows and directionality is of no importance. A suitable graph representation of the probabilistic model is usually chosen according to the nature of the system. Normally, undirected models are more popular in statistical physics or image processing where few causality relationships exist, while directed models are often used in artificial intelligence or bioinformatics to represent generative models.

Factor graphs, developed by Kschischang et al. [62] are drawing growing attention in many empirical applications. In factor graphs, factors that make up the joint distribution are explicitly represented, which provides a convenient way of defining a messaging-passing-based inference algorithm.

In this chapter, we review the above mentioned graphical models, i.e., Bayesian networks, Markov random fields and factor graphs in Section 2.1, 2.2 and 2.3. Simple examples will be given for illustration. Conversion between different types of graphical models will be shown in Section 2.4. Finally, we give some remarks on the graphical models in Section 2.5. In Appendix 2.A, we show the judgement of conditional independence for a three node Bayesian network. Appendix 2.B makes a list of the notations that appear in this chapter.

2.1 Bayesian networks

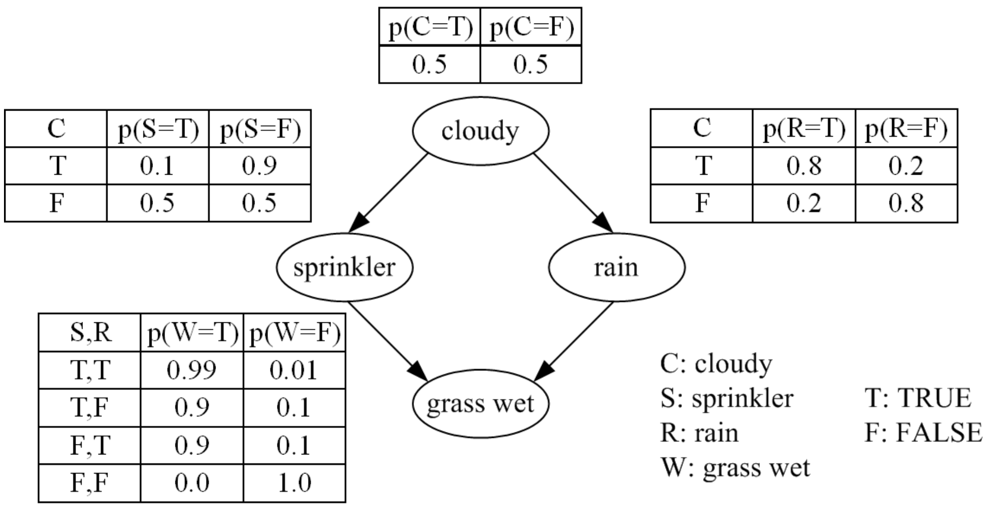
A Bayesian network is an acyclic directed graph in which nodes are connected by edges with arrows. The direction of the arrow along an edge expresses the causal relationship between random variables. A Bayesian network can be constructed based on the causality of the variables. It is usually used to represent a generative model.

Let $\mathcal{G}_{\text{BN}} = \{\mathcal{V}_{\text{BN}}, \mathcal{E}_{\text{BN}}\}$ denote a directed acyclic graph, where $\mathcal{V}_{\text{BN}}$ are nodes representing the variables $\mathbf{x}_{\mathcal{V}_{\text{BN}}} = \{x_v : v \in \mathcal{V}_{\text{BN}}\}$ and $\mathcal{E}_{\text{BN}}$ are directed edges that map the statistical dependencies between the nodes. A Bayesian network represents the factorization of the joint probability density function $p(\mathbf{x}_{\mathcal{V}_{\text{BN}}})$ into the product the conditional probability density functions, which can be formulated as follows:

$$p(\mathbf{x}_{\mathcal{V}_{\text{BN}}}) = \prod_{v \in \mathcal{V}_{\text{BN}}} p(x_v | \mathbf{x}_{PA(v)}) \tag{2.1}$$

where $PA(v)$ are the parent nodes of node v. On the graph, there is always an edge starting from a parent node and pointing to the child node. It will be seen from (2.1) that each variable x_v is associated with a conditional probability density function.

A famous Bayesian network example from [94], which we call ”wet grass” model, is shown in Figure 2.1. In this example, symbol `C` stands for ”cloudy weather”, `S` for ”sprinkler”, `R` for ”rain” and `W` for ”grass wet”.



p(C=T)	p(C=F)
0.5	0.5

C	p(S=T)	p(S=F)
T	0.1	0.9
F	0.5	0.5

C	p(R=T)	p(R=F)
T	0.8	0.2
F	0.2	0.8

S,R	p(W=T)	p(W=F)
T,T	0.99	0.01
T,F	0.9	0.1
F,T	0.9	0.1
F,F	0.0	1.0

Fig. 2.1. Bayesian Network Example
(Russell and Norvig, Artificial Intelligence: A Modern Approach, 1995)

The causality relationship between variables is that the cloudy weather may trigger sprinkler or rain with probability density functions: $p(x_{\mathtt{S}}|x_{\mathtt{C}})$ and $p(x_{\mathtt{R}}|x_{\mathtt{C}})$. Both sprinkler and rain may cause the grass to be wet and the probability density function is given by $p(x_{\mathtt{W}}|x_{\mathtt{S}}, x_{\mathtt{R}})$. Here, `C` is the parent of `S` and `R`. `S` and `R` are parents of `W`. So from the graph, we can read the following factorization of the joint probability:

$$p(x_{\mathtt{W}}, x_{\mathtt{S}}, x_{\mathtt{R}}, x_{\mathtt{C}}) = p(x_{\mathtt{C}})p(x_{\mathtt{S}}|x_{\mathtt{C}})p(x_{\mathtt{R}}|x_{\mathtt{C}})p(x_{\mathtt{W}}|x_{\mathtt{S}}, x_{\mathtt{R}}) \tag{2.2}$$

As a Bayesian network is constructed using the causality inherent in the domain, it is a directed acyclic graph, i.e. there should be no directed cycles. A cycle is present in Figure 2.1. However, it is not directed.

In many cases, we are interested in conditional independence, i.e., given a subset $\mathbf{x}_S (S \subset \mathcal{V}_{\text{BN}})$, we would like to know if variables x_a ($a \in \mathcal{V}_{\text{BN}}$ and $a \notin S$) and x_b ($b \in \mathcal{V}_{\text{BN}}$ and $b \notin S$) are independent. Borrowing the notation from [16], we use $x_a \perp\!\!\!\perp x_b \,|\, \mathbf{x}_S$ to express that x_a is conditionally independent of x_b given $\mathbf{x}_S$.

We first consider the simplest cases with three variables: x_a, x_b and x_c, which are represented by three variables nodes x_a, x_b, and x_c in a Bayesian network. There are edges connecting x_a with x_c and x_b with x_c. No edge exists between a and b. Depending on the direction of the edges and whether x_c is observed or not, there are eight different cases. Figure 2.2 depicts all these cases where hidden variables are represented by unshaded circles, observed variables are represented by shaded circles and dashed lines represent a path from x_a to x_b via x_c or vice versa. Figure 2.2 shows whether a path between x_a and x_b is blocked by x_c or not. In Figure 2.2, if the dashed line is straight, then we say that path is through. Otherwise, the path is blocked. The results depicted in Figure 2.2 are obtained by using the Bayes rule, which is shown in Appendix 2.A. If the path is blocked, then we can say that x_a is conditionally independent of x_b given x_c and vice versa. For example, the first column tells us that if edges merge at x_c and x_c is not observed (Figure 2.2 (a1)), then x_a and x_b are independent, i.e., $p(x_a, x_b) = p(x_a)p(x_b)$. If x_c is observed (Figure 2.2 (a2)), then in general $p(x_a, x_b|x_c) \neq p(x_a|x_c)p(x_b|x_c)$.

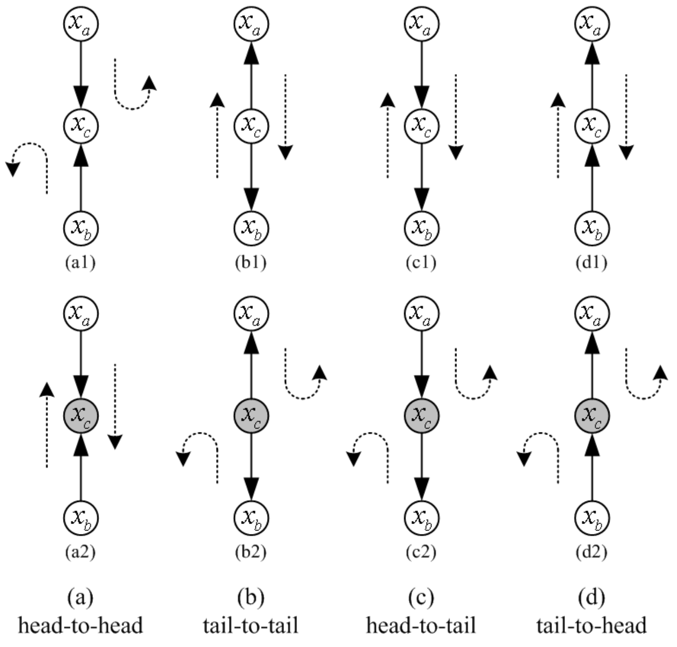


Fig. 2.2. Conditional independence property in a three node Bayesian network
Three nodes graphs showing that nodes x_a and x_b are connected through x_c in different forms: (a) head-to-head connections; (b) tail-to-tail connections; (c) head-to-tail connections; (d) tail-to-head connections, and whether a path from x_a to x_b is blocked by x_c or not.

Now let us consider the general case. Given three nonintersecting sets $\mathbf{x}_T$, $\mathbf{x}_U$ and $\mathbf{x}_S$, we can judge the validity of the independence statement $\mathbf{x}_T \perp\!\!\!\perp \mathbf{x}_U \,|\, \mathbf{x}_S$ by using the d-separation algorithm [92] which compute all the conditional independence relations entailed by the directed graphs. To do so, we first check if a path from a node in T to a node in U is blocked or not. In a general graph, a path is said to be blocked if along the path there is a single node that:

1) two edges meet at the node head-to-tail or tail-to-head(Figure 2.2 (c) and (d)) and the node is in S, or

2) two edges meet at the node tail-to-tail(Figure 2.2 (b)) and the node is in S, or

3) two edges meet at the node head-to-head(Figure 2.2 (a)) and neither the node nor any of its descendant is in S

If all possible paths from any node in T to any node in U are blocked, then we say that T and U are d-separated by S, which implies that set $\mathbf{x}_T$ is conditionally independent on the set $\mathbf{x}_U$ given the set $\mathbf{x}_S$, i.e., $\mathbf{x}_T \perp\!\!\!\perp \mathbf{x}_U \mid \mathbf{x}_S$.

Conditional independence is a very important property that should be well studied. If we want to solve the problem of inferring one variable x_a in a directed graph, usually it is not necessary to have the knowledge of all other variables and probability density functions. Using a graph's d-separation property, we can identify a node a's Markov blanket $MB(a)$, which is the smallest set that "isolates" that node from the rest of the network, i.e.:

$$p(x_a|\mathbf{x}_{MB(a)}, \mathbf{x}_{\mathcal{V}_{\text{BN}}\setminus(\{x_a\}\cup\mathbf{x}_{MB(a)})}) = p(x_a|\mathbf{x}_{MB(a)}) \tag{2.3}$$

where $S \setminus T$ means the set S excluding the subset T.

In a Bayesian network, the Markov blanket of a node a is the set composed of a's parents, its children and its children's other parents. It can be seen from (2.3) that the Markov blanket of a node contains all knowledge needed to infer the behavior of that node.

2.2 Markov random fields

A Markov random field (MRF) is an undirected graph. It is a non-causal model. The graph $\mathcal{G}_{\text{MRF}} = \{\mathcal{V}_{\text{MRF}}, \mathcal{E}_{\text{MRF}}\}$ contains variable nodes and undirected edges that connect the nodes according to their dependency. A set of local potential functions $\{\psi_{C_k}(\mathbf{x}_{C_k})\}$ are defined along the graph where each function $\psi_{C_k}(\mathbf{x}_{C_k})$ has the domain of some clique C_k in the graph. According to the definition of the functions, the whole graph can be divided into K cliques $\mathcal{C}_{\text{MRF}} = \{C_1, \dots C_K\}$, with a clique C_k being defined as a fully connected sub-graph. The potential functions map each concrete assignment of the variables $\mathbf{x}_{C_k}$ in the clique to a non-negative real number. The joint distribution of all variables, i.e., $\mathbf{x}_{\mathcal{V}_{\text{MRF}}}$, called MRF distribution is then defined as the normalized product of all potential functions:

$$p(\mathbf{x}_{\mathcal{V}_{\text{MRF}}}) = \frac{1}{Z}\prod_{k=1}^{K}\psi_{C_k}(\mathbf{x}_{C_k}) \tag{2.4}$$

where Z is the normalization factor calculated by summation or integration.

An example of a Markov random field is shown in Figure 2.3, which represents the MRF distribution

$$p(\mathbf{x}_{\mathcal{V}_{\text{MRF}}}) = \frac{1}{Z}\psi_{12}(x_1, x_2)\psi_{23}(x_2, x_3)\psi_{24}(x_2, x_4)\psi_{45}(x_4, x_5)\psi_{356}(x_3, x_5, x_6) \tag{2.5}$$

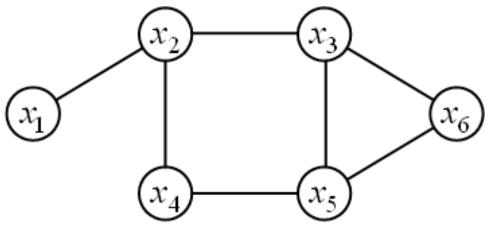


Fig. 2.3. A Markov random field example.

where the normalization factor is obtained by, assuming discrete random variables in (2.5):

$$Z = \sum_{x_1,x_2,x_3,x_4,x_5,x_6} \psi_{12}(x_1,x_2)\psi_{23}(x_2,x_3)\psi_{24}(x_2,x_4)\psi_{45}(x_4,x_5)\psi_{356}(x_3,x_5,x_6) \tag{2.6}$$

Judging conditional independence is easier in Markov random field because now no combination of directions have to be considered. In general, variables x_a and x_b are conditionally independent given set $\{\mathbf{x}_S\}$ if S separates node a and node b in the graph, i.e., every path from a to b or vice versa has to pass through at least one node in S.

In the example in Figure 2.3, x_2 and x_6 are conditionally independent given x_3 and x_5. The set $\{x_3, x_5\}$ is also the Markov blanket (MB) of x_6. In a Markov random field, a variable node a's Markov blanket is the set of variables that are directly connected to it, i.e., its neighboring nodes.

2.3 Factor graphs

If the joint distribution can be written as a product of a set of factors $\{f_u : u \in \mathcal{U}_{\text{FG}}\}$ where each factor is a function as follows:

$$p(\mathbf{x}_{\mathcal{V}_{\text{FG}}}) = \prod_{u \in \mathcal{U}_{\text{FG}}} f_u(\mathbf{x}_{C_u}) \tag{2.7}$$

where C_u contains the indices of the variables that are associated with factor f_u, then we can represent this factorization in a factor graph. Factor graph $\mathcal{G}_{\text{FG}} = \{\mathcal{V}_{\text{FG}}, \mathcal{U}_{\text{FG}}, \mathcal{E}_{\text{FG}}\}$ is a bipartite graph over the variables $\mathbf{x}_{\mathcal{V}_{\text{FG}}}$ and functions $\{f_u : u \in \mathcal{U}_{\text{FG}}\}$. A bipartite graph is defined as a graph whose vertices can be divided into two sets, such that every edge has one endpoint in each set. A factor graph uses variable nodes $\mathcal{V}_{\text{FG}}$ to present the variables and function nodes $\mathcal{U}_{\text{FG}}$ to present the local functions of random variables. Edges $\mathcal{E}_{\text{FG}}$ connect function nodes with their arguments. In a factor graph, function nodes are connected only to variable nodes and vice versa. So the set C_u is composed of the neighboring variable nodes of the function node u.

An example of a factor graph is depicted in Figure 2.4. Variable nodes are presented by circles and local function nodes are presented by squares. The whole graph represents the following function:

$$p(x_1, x_2, x_3, x_4, x_5) = f_1(x_1) f_2(x_1, x_2, x_4) f_3(x_2, x_3) f_4(x_3, x_4, x_5) \tag{2.8}$$

One advantage of using a factor graph is that it explicitly identifies the functions in a more fine-grained way which enables the direct use of inference algorithms like the sum-product algorithm.

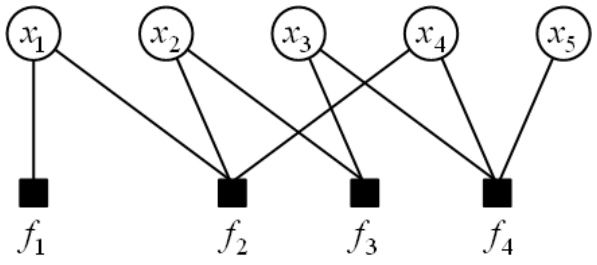


Fig. 2.4. A factor graph example

A Forney style factor graph [22] provides a more compact presentation of the factorization of functions. In the Forney style factor graph $\mathcal{G}_{\text{FFG}} = \{\mathcal{E}_{\text{FFG}}, \mathcal{U}_{\text{FFG}}\}$,

- each factor f_u is presented as a function node $u \in \mathcal{U}_{\text{FFG}}$;
- each variable x_e is presented as a full edge or a half edge $e \in \mathcal{E}_{\text{FFG}}$. There may be only one edge associated with each variable;
- an edge e that represents a variable x_e is connected with a node u that represents factor f_u if f_u is a function of x_e. A full edge connects two function nodes. A half edge is connected only to one function node.

The second rule forbids a variable to be shared by more than two functions. This can be avoided by adding nodes that correspond to "equality" in the graph. Variables connected to this node are equal to each other.

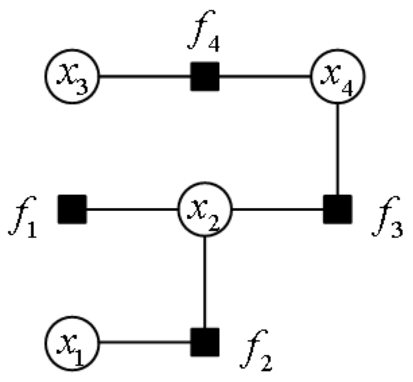
Figure 2.5 shows the conversion from a standard factor graph to a Forney style graph. Both of them represent the factorization of the following function:

$$f(x_1, x_2, x_3, x_4) = f_1(x_2)f_2(x_1, x_2)f_3(x_2, x_4)f_4(x_3, x_4) \tag{2.9}$$

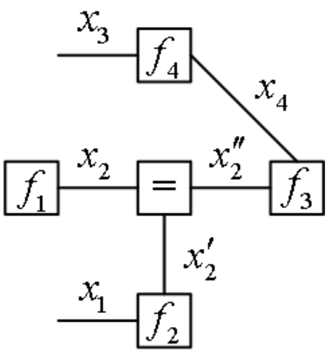
It can be observed in Figure 2.5(b) that in order to avoid x_2 being connected by three functions, an equality function node is added to replicate x_2.

2.4 Conversion between different graphs

Although Bayesian networks, Markov random fields and factor graphs have different forms, in principle, they all represent the factorization of a joint probability or a global function. This can be observed from (2.1), (2.4) and (2.7). Some conversions between different types of graph models are possible, as we outline next.



(a) Standard factor graph



(b) Forney style factor graph

Fig. 2.5. Factor graph vs Forney style factor graph

2.4.1 Conversion from Bayesian networks to Markov random fields

If we compare (2.1) and (2.4), we find that the factorization in a Bayesian network is just a special case of that in a Markov random field where the normalization constant in a Bayesian network is 1 and the potential functions are just conditional or marginal probabilities. In this case, each clique consists of a variable and its parents. So we can convert a Bayesian network to a Markov random field by marrying parents (connecting mutually all the parents of each node) and dropping the directions, which is known as moralization [42]. Such a conversion will hide some independence properties that were explicitly expressed by the Bayesian networks. However, we often transform a Bayesian network into an undirected model for the purpose of inference. A conversion example is shown in Figure 2.6. The Bayesian network on the left represents

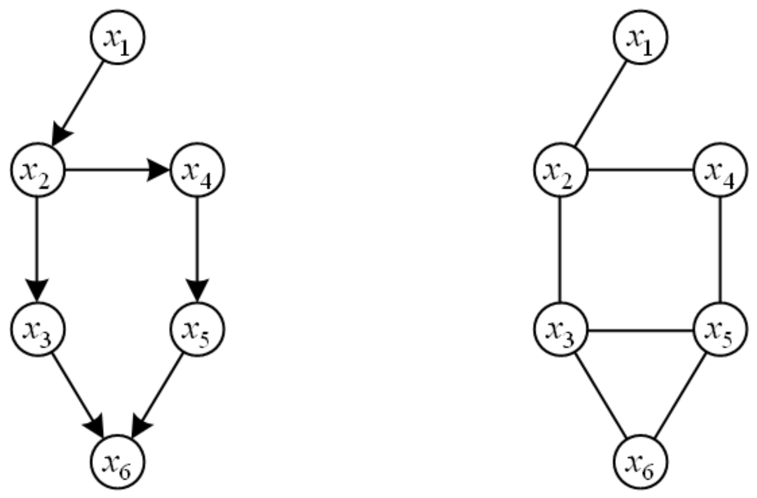
$$p_{\mathrm{BN}}(x_1,x_2,x_3,x_4,x_5,x_6) = p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_2)p(x_5|x_4)p(x_6|x_3,x_5) \tag{2.10}$$

and the Markov random field on the right represents:

$$p_{\mathrm{MRF}}(x_1,x_2,x_3,x_4,x_5,x_6) = \frac{1}{Z}\psi_1(x_1,x_2)\psi_2(x_2,x_3)\psi_3(x_2,x_4)\psi_4(x_4,x_5)\psi_5(x_3,x_5,x_6) \tag{2.11}$$

The relationships between the factors in (2.10) and (2.11) are:

$$\begin{aligned} Z &= 1 \\ \psi_1(x_1,x_2) &= p(x_1)p(x_2|x_1) \\ \psi_2(x_2,x_3) &= p(x_3|x_2) \\ \psi_3(x_2,x_4) &= p(x_4|x_2) \\ \psi_4(x_4,x_5) &= p(x_5|x_4) \\ \psi_5(x_3,x_5,x_6) &= p(x_6|x_3,x_5) \end{aligned} \tag{2.12}$$



(a) Bayesian network (b) Markov random field

Fig. 2.6. Conversion from a Bayesian network to a Markov random field

2.4.2 Conversion from Bayesian networks to factor graphs

If we look on the marginal or conditional probabilities in a Bayesian network as local functions, then it is easy to create a factor graph from a Bayesian network. We first create variable nodes. For each variable node, we create a function node and connect it to that variable node and its parents. Then we set the local function of each function node to the corresponding conditional probability. The global function is then the joint probability. A conversion example is shown in Figure 2.7. The Bayesian network on the left represents the same probability density function as in (2.10) and the factor graph on the right represents:

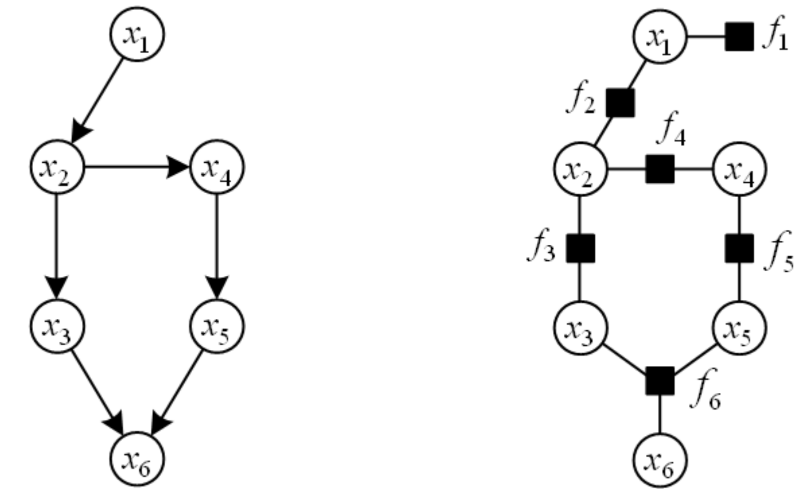
$$f_{\mathrm{FG}}(x_1, x_2, x_3, x_4, x_5, x_6) = f_1(x_1)f_2(x_1, x_2)f_3(x_2, x_3)f_4(x_2, x_4)f_5(x_4, x_5)f_6(x_3, x_5, x_6) \quad (2.13)$$

The relationships between the factors in (2.10) and (2.13) are:

$$\begin{aligned} f_1(x_1) &= p(x_1) \\ f_2(x_1, x_2) &= p(x_2|x_1) \\ f_3(x_2, x_3) &= p(x_3|x_2) \\ f_4(x_2, x_4) &= p(x_4|x_2) \\ f_5(x_4, x_5) &= p(x_5|x_4) \\ f_6(x_3, x_5, x_6) &= p(x_6|x_3, x_5) \end{aligned} \quad (2.14)$$

2.4.3 Conversion from Markov random fields to factor graphs

For a Markov random field, we can set the potential functions as local functions and then convert the graph into a factor graph. We first create the variable nodes. Then we create a function node for each clique, connect each function node with the variable nodes in that



(a) Bayesian network (b) Factor graph

Fig. 2.7. Conversion from a Bayesian network to a factor graph

clique and set the local function to be the potential function. The global function is then the MRF probability. A conversion example is shown in Figure 2.8. The Markov random field on the left represents

$$p_{\mathrm{MRF}}(x_1,x_2,x_3,x_4,x_5,x_6) = \frac{1}{Z}\psi_1(x_1,x_2)\psi_2(x_2,x_3)\psi_3(x_2,x_4)\psi_4(x_4,x_5)\psi_5(x_3,x_5,x_6) \quad (2.15)$$

and the factor graph on the right represents:

$$f_{\mathrm{FG}}(x_1,x_2,x_3,x_4,x_5,x_6) = f_1(x_1,x_2)f_2(x_2,x_3)f_3(x_2,x_4)f_4(x_4,x_5)f_5(x_3,x_5,x_6) \quad (2.16)$$

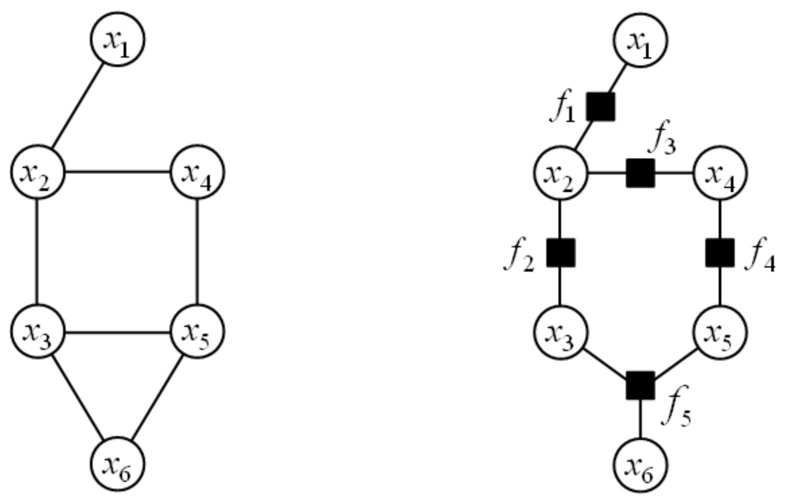
The relationships between the factors in (2.15) and (2.16) are:

$$\begin{aligned} f_1(x_1,x_2) &= \frac{1}{Z}\psi_1(x_1,x_2) \\ f_2(x_2,x_3) &= \psi_2(x_2,x_3) \\ f_3(x_2,x_4) &= \psi_3(x_2,x_4) \\ f_4(x_4,x_5) &= \psi_4(x_4,x_5) \\ f_5(x_3,x_5,x_6) &= \psi_5(x_3,x_5,x_6) \end{aligned} \quad (2.17)$$

Note that the factorization constant $\frac{1}{Z}$ can be contained in any of the five factors in (2.16).

2.5 Some remarks on graphical models

Now we make some remarks on the graphical models. Some of the remarks could be helpful for the understanding of the content in the upcoming parts of the thesis.



(a) Markov random field (b) Factor graph

Fig. 2.8. Conversion from a Markov random field to a factor graph

2.5.1 A comparison of different graphs

Three graphical models were introduced in the previous sections. Directed graphical models, e.g. Bayesian networks, contain directed edges which reveal the causal relationship between variables. The whole graph represents the decomposition of a joint probability into the product of conditional probabilities. In undirected graphs, e.g. Markov random fields, the direction of the edges has no significance. A Markov random field usually represents the soft constraints, e.g. correlations between variables. In general, such constraints or correlations are symmetric. The potential function in a Markov random field does not necessarily have an explicit probabilistic interpretation. In principle, we should notice that a directed graph and an undirected graph are different languages. It is not convenient to express non-causal relationships by using Bayesian networks. And if we convert a Bayesian network into a Markov random field, some independence properties cannot be read directly from the graph any more. For a system where both non-causal and causal relationships coexist, it is not suitable to express the probabilistic model by using either a Bayesian network or a Markov random field. A factor graph provides a more explicit representation of the relationships between variables, in which factors (local functions) that connect variables are identified on the graph. In this case, it is the best way to express the factorization. A Forney style factor graph provides a more compact form. Extra nodes, i.e., ”equality” nodes have to be introduced to avoid that a variable is shared by more than two function nodes. The advantage of using Forney style factor graphs in some of the inference problems will be demonstrated in the later chapters. Introducing directions is another extension to the standard factor graphs. Like in the Bayesian network, we use direction to express causality. In this way, we construct a directed factor graph [25], which unifies directed and undirected graphs. The benefit of doing this will be discussed when we introduce inference algorithms in the later chapters.

2.5.2 Parameter estimation based on graphical models

To carry out inference, local distribution functions (conditional probability density functions for Bayesian networks, local potential functions for Markov random fields and local factors for factor graphs) need to be defined. If we deal with discrete random variables, we usually list in a table the values of the local function that are generated from different configurations of its arguments. The values of each conditional probability in the ”wet grass” model (Section 2.1) are listed in Figure 2.1. In that example, all variables are Boolean, i.e. they have two possible values: TRUE or FALSE. If local distribution functions are defined over continuous random variables, we need an analytical expression. The Gaussian distribution is the most popular distribution function. In some applications, the local function might take a very complex form. In that case, approximations are desirable to simplify the expression of the local functions.

Sometimes, the type of the local distribution function is known but parameters of the distribution are unknown or only partially known. In that case, we can include parameters in the graph. An example is shown in Figure 2.9. Based on the structure of the graph,

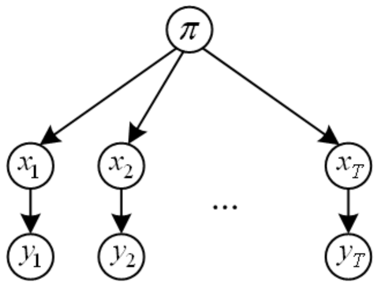


Fig. 2.9. Bayesian network with parameters.
$\{y_t : t = 1, ...T\}$ are measurements of $\{x_t : t = 1, ...T\}$ which are samples drawn from a uniform distribution. The parameter of the uniform distribution is π, i.e., $x_t \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and $p(x_t) = \frac{1}{\pi}$. Measurements are corrupted by additive Gaussian noise, i.e., $y_t = x_t + \nu_t$ and $\nu_t \sim \mathcal{N}(0, \sigma^2)$

different learning algorithms, e.g., expectation maximization (EM) algorithm, can be developed to learn the parameters from data. Examples can be found in [26].

2.5.3 Independence property

The most important information embedded in graphical models is the independence property between random variables. The conditional independence can be easily read from an undirected graph. In a directed graph, we have to use the d-separation algorithm defined in Section 2.1 to judge the conditional independence. Given a single node, we can also identify its Markov blanket, i.e., the set of nodes that matter for the inference of that single node.

In the next chapters, we will see what an important role the independence property will play in the simplification of the computational complexity of inference algorithms. This is since the graphical models encode the conditional independences in their structure. Using conditional independence in algorithms means exploiting the special structure of the graphical models. Therefore, graphical models offer a visible tool for the design of new algorithms.

2.5.4 Benefits from using graphical models

Graphical models provide a diagrammatic tool to demonstrate the relationship between the components of a complex system, which brings several benefits:

1) it visualizes the properties of the model, especially the conditional dependences or statistic correlations
2) it provides an intuitive understanding of the complexity of the system
3) it enables researchers to communicate the system model to other researchers and users in a more convenient way
4) it leads to algorithms that make use of the structure of the graph and indicates possible approximations on the model which simplify the computation
5) it helps us develop new models
6) some complicated systems can be modeled by the manipulation of simple structures

2.A Judgement of conditional independence using Bayes rule

This appendix presents the conditional independence property of the simplest structure of a directed graph, i.e., a three node Bayesian network. We study the graph representing the joint probability of density function of $p(x_a, x_b, x_c)$ in which nodes x_a and x_b are not directly connected, but are connected via x_c. Depending on the direction of the edges, there are four different cases. Now we study the independence property of these graphs case by case.

- head-to-head case (Figure 2.10): an arrow goes from x_a to x_c and another arrow goes from x_b to x_c.

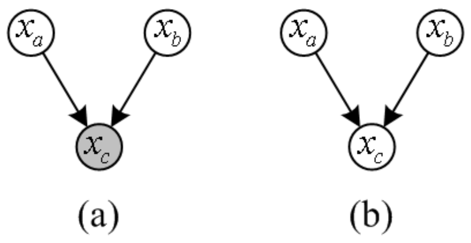


Fig. 2.10. Three node Bayesian network: head to head

If x_c is observed, let the observation be x_c^*. It is represented on the graph by a shaded circle, as shown in Figure 2.10(a). In this case, the directed graph represents:

$$p(x_a, x_b, x_c = x_c^*) = p(x_a)p(x_b)p(x_c = x_c^*|x_a, x_b) \tag{2.18}$$

Dividing on both sides of (2.18) by $p(x_c = x_c^*)$ which is equal to 1, we obtain:

$$\frac{p(x_a, x_b, x_c = x_c^*)}{p(x_c = x_c^*)} = \frac{p(x_a)p(x_b)p(x_c = x_c^*|x_a, x_b)}{p(x_c = x_c^*)} \tag{2.19}$$

In general, we cannot obtain the equality of $p(x_a, x_b|x_c = x_c^*) = p(x_a|x_c = x_c^*)p(x_b|x_c = x_c^*)$ from (2.19). Therefore, in head-to-head case, x_a is not conditionally independent on x_b given x_c. Then we say that the path from x_a to x_b via x_c is open.

If x_c is not observed, it is represented by an unshaded circle in the graph, as shown in Figure 2.10(b). In this case, the directed graph represents:

$$p(x_a, x_b, x_c) = p(x_a)p(x_b)p(x_c|x_a, x_b) \tag{2.20}$$

If we marginalize on both sides of (2.18) over x_c, we obtain:

$$\begin{aligned}
&\sum_{x_c} p(x_a, x_b, x_c) = \sum_{x_c} p(x_a)p(x_b)p(x_c|x_a, x_b) \\
&\Rightarrow p(x_a, x_b) = p(x_a)p(x_b) \sum_{x_c} p(x_c|x_a, x_b) \\
&\Rightarrow p(x_a, x_b) = p(x_a)p(x_b)
\end{aligned} \tag{2.21}$$

So given no observations, x_a and x_b are independent. That means, the path between x_a and x_b is blocked at x_c.

- tail-to-tail case (Figure 2.11): an arrow goes from x_c to x_a and another arrow goes from x_c to x_b.

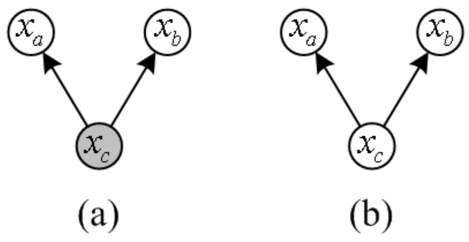


Fig. 2.11. Three node Bayesian network: tail to tail

If x_c is observed, let the observation be x_c^*. It is represented on the graph by a shaded circle, as shown in Figure 2.11(a). In this case, the directed graph represents:

$$p(x_a, x_b, x_c = x_c^*) = p(x_c = x_c^*)p(x_a|x_c = x_c^*)p(x_b|x_c = x_c^*) \tag{2.22}$$

Since $p(x_c = x_c^*) = 1$, we can divide both sides of (2.22) by $p(x_c = x_c^*)$. Using the definition of conditional probability, we obtain:

$$\begin{aligned}
&\frac{p(x_a, x_b, x_c = x_c^*)}{p(x_c = x_c^*)} = \frac{p(x_c = x_c^*)p(x_a|x_c = x_c^*)p(x_b|x_c = x_c^*)}{p(x_c = x_c^*)} \\
&\Rightarrow p(x_a, x_b|x_c = x_c^*) = p(x_a|x_c = x_c^*)p(x_b|x_c = x_c^*)
\end{aligned} \tag{2.23}$$

So given the observation of x_c, x_a and x_b are independent. The path from x_a to x_b is blocked at x_c.

If x_c is not observed, the graph is shown in Figure 2.11(b), which represents:

$$p(x_a, x_b, x_c) = p(x_c)p(x_a|x_c)p(x_b|x_c) \tag{2.24}$$

Marginalizing both sides of (2.24) over x_c, we obtain:

$$\begin{aligned} \sum_{x_c} p(x_a, x_b, x_c) &= \sum_{x_c} p(x_c)p(x_a|x_c)p(x_b|x_c) \\ \Rightarrow p(x_a, x_b) &= \sum_{x_c} p(x_c)p(x_a|x_c)p(x_b|x_c) \qquad (2.25) \end{aligned}$$

In general, (2.25) does not lead to the conclusion that $p(x_a, x_b) = p(x_a)p(x_b)$. So x_a and x_b are not marginally independent. The path between a and b via c is open.

- tail-to-head case (Figure 2.12): an arrow goes from x_c to x_a and another arrow goes from x_b to x_c.

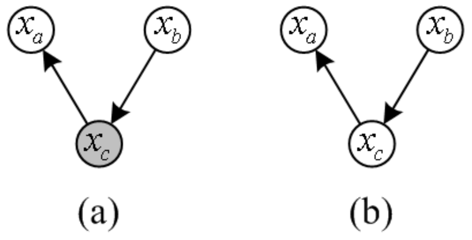


Fig. 2.12. Three node Bayesian network: tail to head

If x_c is observed, the graph is shown in Figure 2.12(a), which represents:

$$p(x_a, x_b, x_c = x_c^*) = p(x_b)p(x_c = x_c^*|x_b)p(x_a|x_c = x_c^*) \tag{2.26}$$

Since now $p(x_c = x_c^*) = 1$, we can divide both sides of (2.26). Using the definition of conditional probability, we obtain:

$$\begin{aligned} \frac{p(x_a, x_b, x_c = x_c^*)}{p(x_c = x_c^*)} &= \frac{p(x_b)p(x_c = x_c^*|x_b)p(x_a|x_c = x_c^*)}{p(x_c = x_c^*)} \\ \Rightarrow p(x_a, x_b|x_c = x_c^*) &= \frac{p(x_c = x_c^*, x_b)}{p(x_c = x_c^*)} p(x_a|x_c = x_c^*) \\ \Rightarrow p(x_a, x_b|x_c = x_c^*) &= p(x_b|x_c = x_c^*)p(x_a|x_c = x_c^*) \qquad (2.27) \end{aligned}$$

So given the observation of x_c, x_a and x_b are independent. The path from x_a to x_b is blocked at x_c.

If x_c is hidden, the graph is shown in Figure 2.12(b), which represents:

$$p(x_a, x_b, x_c) = p(x_b)p(x_c|x_b)p(x_a|x_c) \tag{2.28}$$

Marginalizing both sides of (2.28) over x_c, we obtain:

$$\begin{aligned}\sum_{x_c} p(x_a, x_b, x_c) &= \sum_{x_c} p(x_b)p(x_c|x_b)p(x_a|x_c) \\ \Rightarrow p(x_a, x_b) &= p(x_b)\sum_{x_c} p(x_c|x_b)p(x_a|x_c)\end{aligned} \tag{2.29}$$

In general, (2.29) does not lead to the conclusion that $p(x_a, x_b) = p(x_a)p(x_b)$. So x_a and x_b are not marginally independent. The path between x_a and x_b via x_c is open.

- head-to-tail case (Figure 2.13): an arrow goes from x_a to x_c and another arrow goes from x_c to x_b.

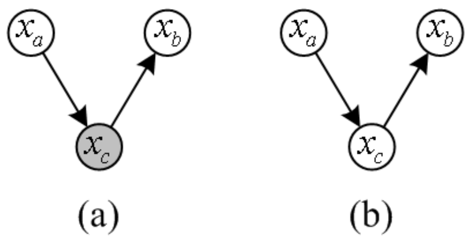


Fig. 2.13. Three node Bayesian network: head to tail

If x_c is observed, the graph is shown in Figure 2.13(a), which represents:

$$p(x_a, x_b, x_c = x_c^*) = p(x_a)p(x_c = x_c^*|x_a)p(x_b|x_c = x_c^*) \tag{2.30}$$

Since now $p(x_c = x_c^*) = 1$, we can divide both sides of (2.30). Using the definition of conditional probability, we obtain:

$$\begin{aligned}\frac{p(x_a, x_b, x_c = x_c^*)}{p(x_c = x_c^*)} &= \frac{p(x_a)p(x_c = x_c^*|x_a)p(x_b|x_c = x_c^*)}{p(x_c = x_c^*)} \\ \Rightarrow p(x_a, x_b|x_c = x_c^*) &= \frac{p(x_c = x_c^*, x_a)}{p(x_c = x_c^*)}p(x_b|x_c = x_c^*) \\ \Rightarrow p(x_a, x_b|x_c = x_c^*) &= p(x_a|x_c = x_c^*)p(x_b|x_c = x_c^*)\end{aligned} \tag{2.31}$$

So given the observation of x_c, x_a and x_b are independent. The path from x_a to x_b is blocked at x_c.

If x_c is hidden, the graph is shown in Figure 2.13(b), which represents:

$$p(x_a, x_b, x_c) = p(x_a)p(x_c|x_a)p(x_b|x_c) \tag{2.32}$$

Marginalizing both sides of (2.32) over x_c, we obtain:

$$\begin{aligned}\sum_{x_c} p(x_a, x_b, x_c) &= \sum_{x_c} p(x_a)p(x_c|x_a)p(x_b|x_c) \\ \Rightarrow p(x_a, x_b) &= p(x_b)\sum_{x_c} p(x_c|x_a)p(x_b|x_c)\end{aligned} \tag{2.33}$$

In general, (2.33) does not lead to the conclusion that $p(x_a, x_b) = p(x_a)p(x_b)$. So x_a and x_b are not marginally independent. The path between a and b via c is open.

The above derivation is summarized as the Bayes ball algorithm presented in Figure 2.2. Based on the analysis of these simple structures of Bayesian network, we can judge the conditional independence for more complicated structures, e.g., the d-separation algorithm.

2.B Summary of notations

This section contains a summary of the notation that appear in this chapter.

General graph

$\mathcal{G}$	a graph
$\mathcal{V}$	the variable node set of $\mathcal{G}$
$\mathcal{E}$	edges in a graph
a, b, c	indices of variables and variable nodes
R, S, T	subsets of $\mathcal{V}$
v	index of a variable node
u	index of a function node
e	index of an edge
$\mathbf{x}_{\mathcal{V}}$	set of variables that are represented by nodes $\mathcal{V}$ in a graph
x_v	a random variable indexed by v
$\mathbf{x}_S$	a set of variables indexed by S
$MB(a)$	Markov blanket of node a

Probability theory

$p(\cdot)$	joint density function
$p(\cdot\|\cdot)$	conditional probability density function
$\mathbf{x}_T \perp\!\!\!\perp \mathbf{x}_U \| \mathbf{x}_S$	conditional independence: $\mathbf{x}_T$ and $\mathbf{x}_U$ are independent given $\mathbf{x}_S$

Bayesian network

$\mathcal{G}_{\text{BN}}$	a Bayesian network	10
$\mathcal{V}_{\text{BN}}$	the variable node set of a Bayesian network	10
$\mathcal{E}_{\text{BN}}$	edges in a Bayesian network	10
$PA(v)$	parent nodes of node v	10

Markov random field

$\mathcal{G}_{\text{MRF}}$	a Markov random field	12
$\mathcal{V}_{\text{MRF}}$	the variable node set of a Markov random field	12
$\mathcal{E}_{\text{MRF}}$	edges in a Markov random field	12
$\mathcal{C}$	set of cliques in a graph	12

K	number of cliques	12
k	index of a clique	12
C_k	indices of variables in clique k	12
Z	normalization factor	12
ψ_{C_k}	a local potential function that is defined on the variables in C_k	12

Factor graph

$\mathcal{G}_{\text{FG}}$	a factor graph	13
$\mathcal{V}_{\text{FG}}$	the variable node set of a factor graph	13
$\mathcal{E}_{\text{FG}}$	edges in a factor graph	13
$\mathcal{U}_{\text{FG}}$	the function node set of a factor graph	13
f_u	a function indexed by u	13
C_u	indices of variables mentioned in f_u	13

Forney style factor graph

$\mathcal{G}_{\text{FFG}}$	a Forney style factor graph	14
$\mathcal{E}_{\text{FFG}}$	edges in a Forney style factor graph	14
$\mathcal{U}_{\text{FFG}}$	the function node set of a Forney style factor graph	14
x_e	a random variable associated with edge e	14

“wet grass“ example

C	symbol to denote the event “cloud”	10
R	symbol to denote the event “rain”	10
S	symbol to denote the event “sprinkler”	10
W	symbol to denote the event “wet grass”	10

3. Inference Algorithms

The task of inference is to acquire the desired information about some variables based on the observations of other variables and on the probabilistic relationship between variables. In probabilistic inference, we are usually interested in calculating the marginal probability distribution of certain variables or the posterior probability distribution of the hidden variables given the observations. Calculating marginals from a joint probability distribution function is not trivial if a large number of variables is involved. We should find a suitable way to carry out the inference so that it is efficient. Otherwise, the problem could be intractable.

This chapter studies this problem in detail. It will be shown how independence properties and function factorizations are exploited to simplify the calculation in the case that a ”brute force” approach is intractable. Different state-of-the-art inference algorithms will be introduced. If the probabilistic model of a system can be represented by a graphical model that has a tree structure, an exact inference solution is achievable by using the *junction tree algorithm* or *belief propagation*. However, in practice, the following difficulties may make exact inference impossible or intractable:

1) loops in the graph

2) number of variables in factors that are not further decomposable

3) complexity of the factors, e.g., complex probability density function of continuous random variables with non-Gaussian distribution, non-linear functions.

In such cases, approximations are needed to simplify the computation so that at least an approximate result can be found. This chapter reviews the most common inference algorithms that solve the first two problems and leaves the discussion of the third problem to the next chapter. To deal with loops in graphs, Pearl[92] suggested extending belief propagation to general graphs, i.e., graphs with loops. Such an extension is called *loopy belief propagation*. Variational methods[112][113] approximate a complex factor with a slightly varied factor with simpler structures. The *mean field method* uses variational factors that can be fully factorized, i.e., the variables in that factor are mutually independent.

This chapter is organized as follows. We first introduce the general form of the inference problem. Then we explain the idea of exploiting the independences and factorizations to simplify the inference. In the part of exact inference algorithms, junction tree algorithms and belief propagation on trees are presented. In the approximate inference part, we will introduce loopy belief propagation, and variational inference, in particular the mean field method. Appendices show some discussion on belief propagation.

3.1 Problem formulation

Suppose we have a system described by a set of random variables $\mathbf{x}_{\mathcal{V}} = \{x_1, x_2, ...x_N\}$ and a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, which represents the factorization of the joint probability distribution $p(\mathbf{x}_{\mathcal{V}})$. Some of these variables are observed or instantiated: $\mathbf{x}_{\mathcal{I}} = \{x_i\}_{i\in\mathcal{I}}$, while others are hidden or unobserved: $\mathbf{x}_{\mathcal{H}} = \{x_i\}_{i\in\mathcal{H}}$. With respect to $\mathcal{V}$, these two sets are mutually exclusive and $\mathbf{x}_{\mathcal{V}} = \{\mathbf{x}_{\mathcal{I}}, \mathbf{x}_{\mathcal{H}}\}$. Usually, two values are of interest:

- the marginal probability distribution of some of the observed variables:

$$p(\mathbf{x}_{\mathcal{I}_1}) = \sum_{\mathbf{x}_{\mathcal{H}_1}} p(\mathbf{x}_{\mathcal{I}_1}, \mathbf{x}_{\mathcal{H}_1}) \tag{3.1}$$

  where $\{\mathcal{H}_1\} \subseteq \{\mathcal{H}\}$, $\{\mathcal{I}_1\} \subseteq \{\mathcal{I}\}$ and the summation is over all possible values that the hidden variables can take. $\{\mathcal{H}_1\}$ contains the all indices of the hidden variables that are related to the observed variables whose indices are in $\{\mathcal{I}_1\}$.

- the posterior probability distribution of the hidden variables given the observations:

$$p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}}) = \frac{p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})}{p(\mathbf{x}_{\mathcal{I}})} \tag{3.2}$$

These two tasks are tightly related. Once we obtained the marginal, we can easily calculate the posterior.

In the following sections, we will show how the problems presented in (3.1) and (3.2) can be solved, either exactly or approximately. For the explanation of inference algorithms, we mainly consider the undirected graph because a directed graph is usually converted into an undirected graph for the inference [53].

3.2 Exact inference

In this section, we discuss several exact inference algorithms. We start from the ”brute force” approach, which theoretically produces the exact inference algorithm but in practice often involves intractable computations. Then we will show that by using factorization and by carefully choosing the sequence of computation, the complexity of inference can be greatly reduced. Such an approach is called *variable elimination*, which is connected to node elimination on graphs. The rest of this section explain belief propagation on trees and the junction tree algorithm, which illustrate the variable elimination method in different formalisms.

3.2.1 Brute force and variable elimination approach

Let us use a simple example to illustrate inference algorithms. Suppose we have a joint probability distribution:

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = \frac{1}{Z}\psi_1(x_1, x_2)\psi_2(x_2, x_3)\psi_3(x_2, x_4)\psi_4(x_4, x_5)\psi_5(x_3, x_5, x_6) \tag{3.3}$$

over six discrete random variables and we want to calculate the marginal probability distribution of x_4. The naive way of doing this is to calculate the summation over the other five variables, i.e.,

$$p(x_4) = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_5}\sum_{x_6} p(x_1, x_2, x_3, x_4, x_5, x_6) \tag{3.4}$$

For notational simplicity, we suppose all variables take value from a finite set $\mathcal{L} = \{1, 2, ...L\}$. Then the complexity of the calculation in (3.4) is in the order of L^6 because we have to consider all possible values that each variable can take. This complexity can be largely reduced if we exploit the factorization in (3.3) and apply the distributed law to find a suitable sequence of the calculations:

$$\begin{aligned}
p(x_4) &= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_5}\sum_{x_6} p(x_1, x_2, x_3, x_4, x_5, x_6) \\
&= \frac{1}{Z}\sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_5}\sum_{x_6} \psi_1(x_1, x_2)\psi_2(x_2, x_3)\psi_3(x_2, x_4)\psi_4(x_4, x_5)\psi_5(x_3, x_5, x_6) \\
&= \frac{1}{Z}\sum_{x_2}\left\{\psi_3(x_2, x_4)\left(\sum_{x_1}\psi_1(x_1, x_2)\right)\sum_{x_5}\left[\psi_4(x_4, x_5)\sum_{x_3}\left(\psi_2(x_2, x_3)\sum_{x_6}\psi_5(x_3, x_5, x_6)\right)\right]\right\} \\
&= \frac{1}{Z}\sum_{x_2}\left\{\psi_3(x_2, x_4)m_1(x_2)\sum_{x_5}\left[\psi_4(x_4, x_5)\sum_{x_3}\left(\psi_2(x_2, x_3)\sum_{x_6}\psi_5(x_3, x_5, x_6)\right)\right]\right\} \\
&= \frac{1}{Z}\sum_{x_2}\left\{\psi_3(x_2, x_4)m_1(x_2)\sum_{x_5}\left[\psi_4(x_4, x_5)\sum_{x_3}\left(\psi_2(x_2, x_3)m_6(x_3, x_5)\right)\right]\right\} \\
&= \frac{1}{Z}\sum_{x_2}\left\{\psi_3(x_2, x_4)m_1(x_2)\sum_{x_5}\left[\psi_4(x_4, x_5)m_3(x_2, x_5)\right]\right\} \\
&= \frac{1}{Z}\sum_{x_2}\left\{\psi_3(x_2, x_4)m_1(x_2)m_5(x_2, x_4)\right\} \\
&= \frac{1}{Z}m_2(x_4)
\end{aligned} \tag{3.5}$$

where the intermediate factor $m_i(\cdot)$ is the result of summation over variable x_i. In each step, we operate on a local function with no more than three variables. That means, each time when we calculate the summation, we need to consider at most L^3 different combinations of possible values that the variables of the local function can take. Therefore, the complexity of the total calculation in (3.5) is in the order of L^3, which is a remarkable simplification of (3.4).

In the calculation in (3.5), we eliminated certain variables in each step by summing over them, which gives the name "variable elimination algorithm". It can be easily concluded that the time complexity (for computations) and space complexity (memory for saving the intermediate results) of variable elimination depends on the largest number of the variables to be tackled in a single summation. Eliminating variables in different sequences may result in different complexities. Choosing a schedule with lowest complexity is an NP-hard problem[3].

Variable elimination reduces the computational complexity by exploiting the factorization of the joint probability distribution or global function. As introduced in the last chapter, this function or probability factorization can be represented by graphical models. Variable elimination in equations can be represented by node elimination in graphical models. An undirected graphical model for the joint probability distribution in (3.3) is depicted in Figure 3.1

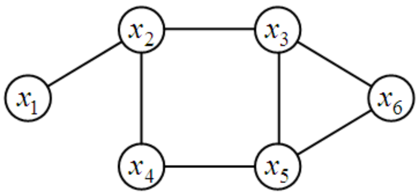


Fig. 3.1. Markov random field for the example in (3.5)

Figure 3.2 shows the sequence of eliminating nodes in the graph which resembles the variable elimination shown in (3.5).

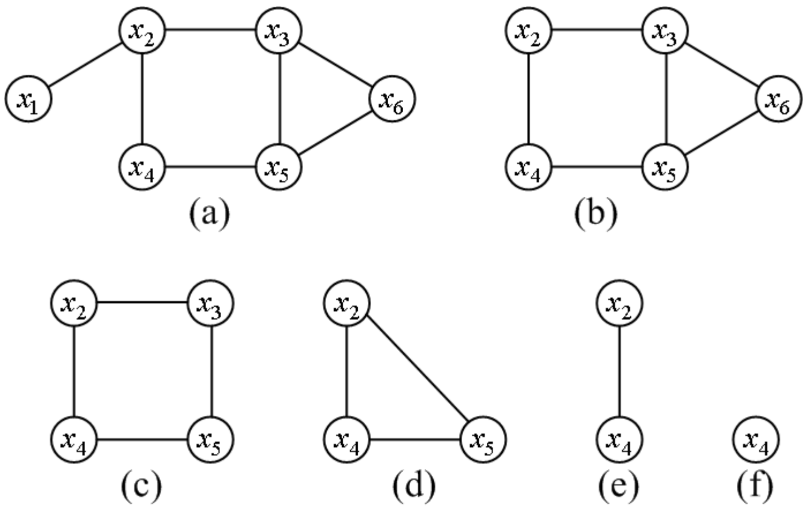


Fig. 3.2. Procedure of node elimination

Comparing the variable elimination in (3.5) and the node elimination in Figure 3.2, we can find the correspondence. In the third line of (3.5), we have an intermediate factor $m_6(x_3, x_5)$, which is a result of summing over x_6. And this intermediate factor itself can be seen as a potential function. This means, after each variable elimination step, we can construct a new Markov random field for the remaining variables. So the fourth line of (3.5) is represented by Figure 3.2(b). It can be also observed that in each step of node elimination, we have always removed a clique, i.e., a fully connected sub-graph.

The variable elimination or node elimination algorithm simplifies inference by exploiting the independence properties of the joint distribution. However, they are query sensitive, i.e., if we want to calculate the marginals of a new query variable, the whole variable elimination procedure has to be repeated. We should notice that the inference of different variables may share some intermediate results. An efficient algorithm should avoid recomputing these intermediate results each time it infers a new variable. In the following sections, we introduce *belief propagation* and the *junction tree algorithm*. Belief propagation realizes an efficient implementation of variable elimination. It produces exact results if it is applied on a graph that has a tree structure. In case that the graph has loops, we use the junction tree algorithm to convert the graph into a tree and then apply belief propagation.

3.2.2 Belief propagation on graphical models

Belief propagation is a message passing algorithm that can solve several types of inference problems, e.g., calculating marginals or solving the maximum a posteriori (MAP) problem. In the literature, belief propagation can be defined for different types of graphical models. In this section, we present belief propagation on the most common graphical models that we have introduced in the last chapter, i.e., Bayesian network, Markov random field and factor graph. We present the *sum-product algorithm* which efficiently calculates marginals of variables [41] and the *max-product algorithm* which solves the MAP problem [6]. In the literature, the term *“summary-product algorithms“* is used to unify these two algorithms. The summary-product algorithms can be presented in a particularly simple and general form if we use factor graphs. As discussed in the previous chapter, Bayesian network and Markov random field can be easily converted to a factor graph. Summary-product algorithm on factor graph can also solve the inference problem for Bayesian network and Markov random field.

In belief propagation on a factor graph, a message is defined as a function associated with an edge. It takes the variable node of the corresponding edge as its argument. For example, in Figure 3.3 the message from function node f_a to variable node x_i is a function of x_i, which will be denoted by $m_{f_a \to x_i}(x_i)$. The message from variable node x_j to function node f_a is a function of x_j and is denoted by $m_{x_j \to f_a}(x_j)$. In a factor graph with a tree structure, belief propagation starts from the leaves. If a leaf is a variable node, it sends out a message with constant value 1. If a leaf is a function node, its message to the neighboring variable nodes is the function represented by the function node. The intermediate nodes produce messages to their neighbors following the summary-product rule. Using that rule, two messages are calculated for each edge in the graph, one in each direction. The summary-product rule regulates that a function node f_a (or variable node x_i) can produce a message for its neighboring variable node x_i (or function node f_a) if f_a (or x_i) has received messages from all other neighbors. Depending on the operations needed in the summary step, we have two message passing rules, i.e., sum-product rule and max-product rule.

- **Sum-Product Rule**

  In the sum-product algorithm, messages sent between nodes are defined as follows:

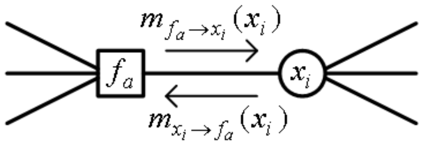


Fig. 3.3. Message passing on a factor graph

- Message from a variable node x_i to a function node f_a:

$$m_{x_i \to f_a}(x_i) := \prod_{c \in NE(i) \setminus a} m_{f_c \to x_i}(x_i) \tag{3.6}$$

where $NE(i)$ denotes the index set of the neighbors of variable node x_i.

- Message from a function node f_a to a variable node x_i:

$$m_{f_a \to x_i}(x_i) := \sum_{\mathbf{x}_a \setminus x_i} \left(f_a(\mathbf{x}_a) \prod_{j \in NE(a) \setminus i} m_{x_j \to f_a}(x_j) \right) \tag{3.7}$$

where $NE(a)$ denotes the neighbors of function node f_a.

After all messages are passed, marginals can be calculated:

- Marginal for a variable x_i:

$$\beta_i(x_i) \propto \prod_{c \in NE(i)} m_{f_c \to x_i}(x_i) \tag{3.8}$$

- Marginal for a clique $\mathbf{x}_a$ at function node a:

$$\beta_a(\mathbf{x}_a) \propto f_a(\mathbf{x}_a) \prod_{j \in NE(a)} m_{x_j \to f_a}(x_j) \tag{3.9}$$

- **Max-Product Rule**

The max-product algorithm resembles the sum-product algorithm but replaces the $\sum$ operator with max:

- Message from a variable node x_i to a function node f_a:

$$m_{x_i \to f_a}(x_i) := \prod_{c \in NE(i) \setminus a} m_{f_c \to x_i}(x_i) \tag{3.10}$$

where $NE(i)$ denotes the neighbors of variable node x_i.

- Message from a function node f_a to a variable node x_i:

$$m_{f_a \to x_i}(x_i) := \max_{\mathbf{x}_a \setminus x_i} f_a(\mathbf{x}_a) \prod_{j \in NE(a) \setminus i} m_{x_j \to f_a}(x_j) \tag{3.11}$$

where $NE(a)$ denotes the neighbors of function node f_a.

After all messages are passed, we can calculate the configuration of the values of the variables that maximize the joint probability function or the global function. For x_i:

$$x_{i\max} = \max_{x_i} \prod_{c \in NE(i)} m_{f_c \to x_i}(x_i) \tag{3.12}$$

If the factor graph is a tree, the summary-product rule guarantees that all the messages can be sequentially computed and the results are identical to those obtained from the variable elimination algorithm. (See the discussion and a simple example in Appendix 3.A.) Extension of the sum-product algorithm to a general graph with loops will be discussed in Section 3.3.

3.2.3 Junction tree algorithm

The summary-product algorithms presented in the previous section produces exact results only if the factor graph has a tree structure. The junction tree algorithm deals with loops in the graph. It converts a general graph to a junction tree and then runs the belief propagation algorithm on that tree.

Given a graph $\mathcal{G}$, the junction tree $\mathcal{T} = \{\mathcal{K}, \mathcal{E}\}$ of it is a cluster graph with each cluster K_u being composed of a subset of variables $\mathbf{x}_{K_u}$. There are edges $\mathcal{E}$ connecting clusters with common variables. To be a junction tree, the cluster graph has to have the following properties [42]:

- single connection: there is only one path between two clusters
- covering: each clique C in $\mathcal{G}$ must be in at least one cluster in $\mathcal{G}$
- running intersection: if cluster K_u and cluster K_v have a common variable x_i, then all clusters on the path between K_u and K_v must also contain variable x_i.

Now we use the example shown in (3.3) and Figure 3.2 to illustrate how to construct a junction tree. We start from the undirected graph shown in Figure 3.1. To construct a junction tree $\mathcal{T}$ from a graph $\mathcal{G}$, we can use the following steps [42]:

1) if we start with a directed graph, convert it to an undirected graph using moralization as shown in Section 2.4.1.

2) ordering the nodes, use variable/node elimination to obtain the (non-unique) sequence of elimination cliques. This can be illustrated by using the example in Figure 3.2. Figure 3.4 depicts the sequence of eliminating cliques. We start with the orginal undirected graph (Figure 3.4(a)). Then we identify a clique, i.e., $\{x_1, x_2\}$ and remove it so that we obtain a trimmed graph in Figure 3.4(b). Then another clqiue ($\{x_3, x_5, x_6\}$) will be identified and removed. The clique that is removed in each step is called elimination clique, which is highlighted in Figure 3.4. This clique elimination process will be repeated until at the end the resulting graph is a clique (Figure 3.4(d)), which becomes the last elimination clique. As illustrated in Figure 3.4, the elimination cliques

in our example are: $\{x_1, x_2\}$, $\{x_3, x_5, x_6\}$, $\{x_2, x_3, x_5\}$, $\{x_2, x_4, x_5\}$. Note that sometimes extra edges have to be added in order to form a clique, e.g., the dashed line added in Figure 3.4(c). Such an operation for finding the cliques is referred in the literature as triangulation. The relevance between the identification of elimination cliques and variable elimination in (3.5) is as follows. In (3.5), we computed step by step the summation over the following functions: $\psi_1(x_1, x_2)$, $\psi_5(x_3, x_5, x_6)$, $\psi_2(x_2, x_3)m_6(x_3, x_5)$, $\psi_4(x_4, x_5)m_3(x_2, x_5)$, $\psi_3(x_2, x_4)m_1(x_2)m_5(x_2, x_4)$. As a result, the consecutive elimination cliques, i.e., variables involved in the functions that are processed in each step, are: $\{x_1, x_2\}$, $\{x_3, x_5, x_6\}$, $\{x_2, x_3, x_5\}$, $\{x_2, x_4, x_5\}$, $\{x_2, x_4\}$. If one clique is a subset of the other clique, we delete the smaller clique. So at the end we obtian the same set of elimination cliques.

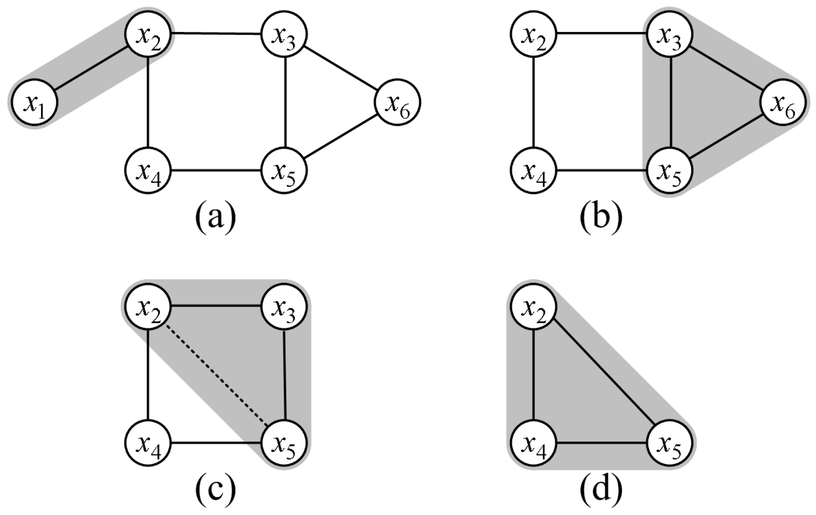


Fig. 3.4. Clique elimination in a undirected graph

3) generating a complete cluster graph, i.e. a fully connected graph, using the elimination cliques. From the cliques found in step 2, we choose the clusters to be $K_1 = \{1, 2\}$, $K_2 = \{3, 5, 6\}$, $K_3 = \{2, 3, 5\}$, $K_4 = \{2, 4, 5\}$ where each K_u is a collection of the indices of the variables involved in that cluster.

4) weighting the edge between two clusters K_u and K_v by $\|\{\mathbf{x}_{K_u} \cap \mathbf{x}_{K_v}\}\|$ where the operator $\|\cdot\|$ calculates the size of a set. The junction tree for graph $\mathcal{G}$ is the maximal weight spanning tree according to [52]. Figure 3.5(a) shows a complete cluster graph and the weights between different clusters. A spanning tree with maximal weights, i.e., the junction tree is generated and shown in Figure 3.5(b). Sometimes, the maximal weights spanning tree is not unique. Another possible junction tree for our example is shown in Figure 3.5(c).

Now we distribute the factors (potential functions) of (3.3) into the clusters in the junction tree $\mathcal{T}$. To do that, we first assign for each cluster K_u a potential function ϕ_{K_u} and initialize it to unity. Then we select one of the potential functions $\psi_C(C \in \mathcal{C})$ of p in (3.3) and assign it into the one cluster K_u which covers the clique C. Then we update the cluster potential function by multiplying it with the clique potential functions that were assigned to it. The covering property of a junction tree ensures that every clique potential in p will appear in

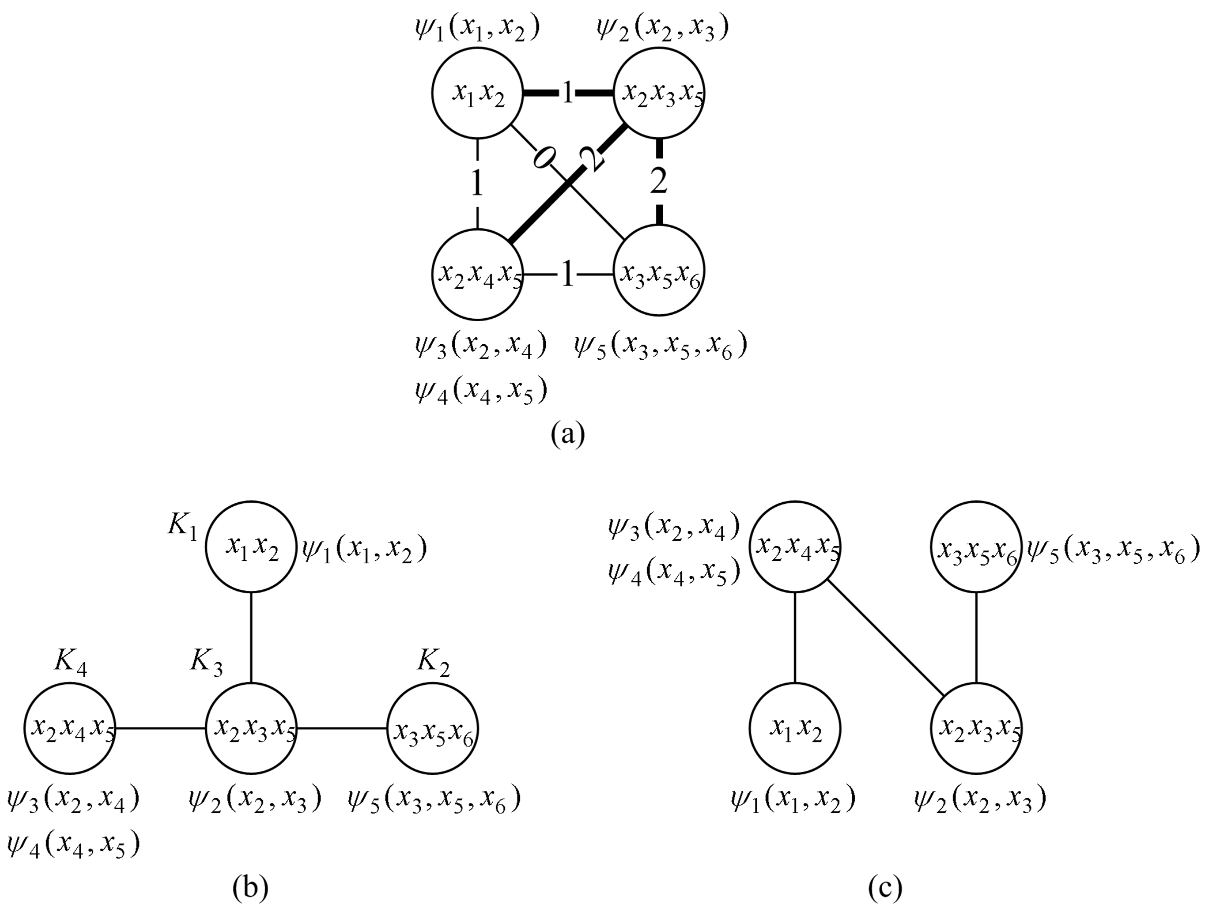


Fig. 3.5. Generating a junction tree from a complete cluster graph

the junction tree. The final assignment of the potential functions is as follows:

$$
\begin{aligned}
\phi_{K_1}(x_1,x_2) &= \psi_1(x_1,x_2) \\
\phi_{K_2}(x_3,x_5,x_6) &= \psi_5(x_3,x_5,x_6) \\
\phi_{K_3}(x_2,x_3) &= \psi_2(x_2,x_3) \\
\phi_{K_4}(x_2,x_4,x_5) &= \psi_3(x_2,x_4)\psi_4(x_4,x_5)
\end{aligned}
\tag{3.13}
$$

which is also shown in Figure 3.5(b). It can be observed that the potential function assigned to cluster K_3 only contains x_2 and x_3. x_5 is also present in K_3 because the *running intersection* property has to be fulfilled.

It can be seen that the indentification of clusters on the graph and the assignment of the original potential functions $\psi_C(C \in \mathcal{C})$ into the clusters is a very tedious process if high dimensional probability functions is considered. Therefore, finding a suitable juntion tree for a given distribution is by itself an NP hard problem.

Now we can run the junction tree algorithm on Figure 3.5(b), which is similar to the summary-product rule in factor graphs. In a junction tree, each cluster K_u knows its local potential function and its neighborhood $NE(K_u)$. Clusters exchange messages between each other. By combining its local potential with the messages from all its neighbors $NE(K_u)$, a cluster K_u can calculate the marginals of the variables it covers. Message passing obeys the rule that cluster K_u can send a message to its neighbor $K_v \in NE(K_u)$ only if K_v has received messages from all its neighbors except K_v.

Usually, we schedule the message passing procedure by choosing a root in the junction tree and starting message passing from the leaves. If we are interested in the marginal of a specific variable, we should choose the cluster that contains the variable to be the root. Figure 3.6 illustrates the message passing schedule. The arrows denote the messages and their direction, with the numbers ordering the sequence of messages.

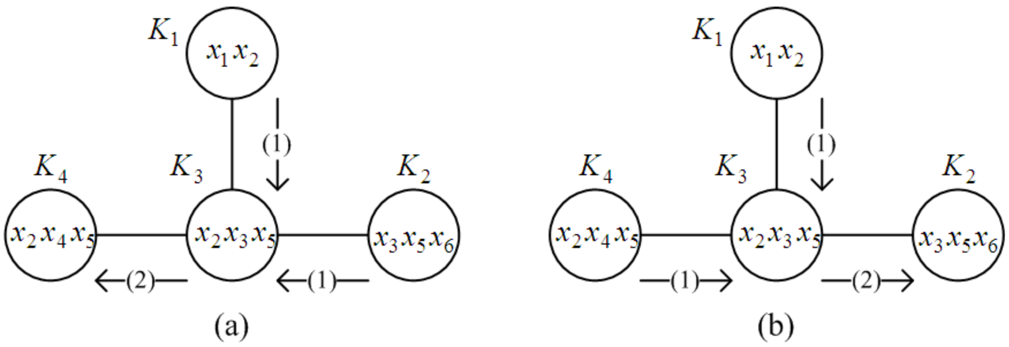


Fig. 3.6. Message passing in a junction tree

In the example in Figure 3.6(a), we choose cluster K_4 to be the root. So we first start sending messages from the leaves K_1 and K_2. Once cluster K_3 has received all messages from its neighbors other than K_4, it can combine them with its local potential function and generates the message for K_4. Another example is shown in Figure 3.6(b) where K_2 is chosen to be the root.

In general, the message sent from a cluster K_u to its neighbor K_v is defined as:

$$m_{K_u \to K_v}(\mathbf{x}_{K_u \cap K_v}) = \sum_{\mathbf{x}_{K_u \setminus K_v}} \phi_{K_u}(\mathbf{x}_{K_u}) \prod_{K_w \in NE(K_u) \setminus K_v} m_{K_w \to K_u}(\mathbf{x}_{K_w \cap K_u}) \tag{3.14}$$

Here we define $NE(K_u)$ as the set of K_u's neighbors. From (3.14) we can see, the message K_u sends to K_v is the product of its local potential function with all the messages it received from its neighbors other than K_v, summing over the variables that are not in K_v. So the messages between K_u and K_v, in both directions, are functions of the intersection of $\mathbf{x}_{K_u}$ and $\mathbf{x}_{K_v}$.

If a cluster has received messages from all its neighbors, then it can calculate the marginal, which is called cluster belief:

$$\beta_{K_u}(\mathbf{x}_{K_u}) = \phi_{K_u}(\mathbf{x}_{K_u}) \prod_{v \in NE(K_u)} m_{K_v \to K_u}(\mathbf{x}_{K_u \cap K_v}) \tag{3.15}$$

It can be shown that the cluster belief $\beta_{K_u}(\mathbf{x}_{K_u})$ is proportional to the marginal probability $p(\mathbf{x}_{K_u})$. Exact marginals can be calculated by normalizing the cluster beliefs.

If we want to calculate the marginal probabilities of multiple variables that are contained in different clusters, we need to repeat the message passing choosing different clusters as

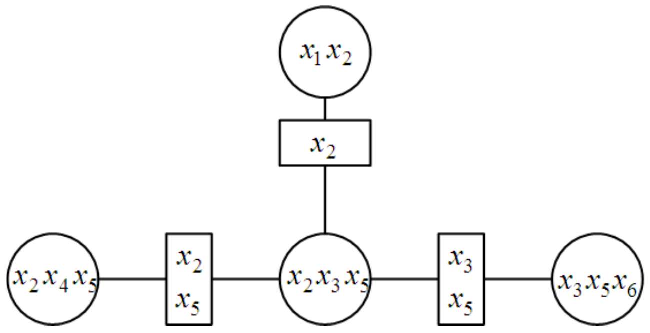


Fig. 3.7. Junction tree with separators

the roots. It should be noticed that some intermediate results can be shared by the inference of different marginals. For example in the calculations presented in Figure 3.6, the message from K_1 to K_3 is shared by both calculations. To make the junction tree algorithm more efficient, we introduce separators $S \in \mathcal{S}$ between neighboring clusters. A separator contains the common variables in the neighboring clusters. A junction tree with separators is shown in Figure 3.7. Using the separators, we modify our inference algorithm:

1) Initialization. Give each cluster K and separator S a potential function and initialize them to:

$$\varphi_K(\mathbf{x}_K) = \phi_K(\mathbf{x}_K) \tag{3.16}$$
$$\varphi_S(\mathbf{x}_S) = 1 \tag{3.17}$$

2) Update. Once cluster K_u wants to send message to cluster K_v, it sends the message first to the separator S_{uv}, so S_{uv} updates its potential function as:

$$\varphi'_{S_{uv}}(\mathbf{x}_{S_{uv}}) = \sum_{\mathbf{x}_{K_u \setminus S_{uv}}} \varphi_{K_u}(\mathbf{x}_{K_u}) \tag{3.18}$$

Then K_v updates its potential function as:

$$\varphi'_{K_v}(\mathbf{x}_{K_v}) = \varphi_{K_v}(\mathbf{x}_{K_v}) \frac{\varphi'_{S_{uv}}(\mathbf{x}_{S_{uv}})}{\varphi_{S_{uv}}(\mathbf{x}_{S_{uv}})} \tag{3.19}$$

3) Termination. When all messages are passed, the potential $\varphi_K(\mathbf{x}_K)$ is proportional to the marginal probability $p(x_K)$.

The junction tree algorithm is an extension of node elimination in Markov random fields. The message passing method that we have used in the junction tree algorithm is also an example of belief propagation. It is belief propagation applied on a cluster graph, i.e., each node on the graph represent several variables. The sum-product algorithm introduced in the Section 3.2.2 is another example of belief propagation. It is belief propagation applied on a factor graph where each node represent only a single variable or a single function. The messages passed in the junction tree algorithm can be multivariate functions whereas the messages passed in factor graph is always a function of a single variable.

Belief propagation is an iterative algorithm for computing marginals of functions or variables on graphical models. Judea Pearl formulated this algorithm on trees [91] in 1982. Kim and Pearl formulate it on polytrees [57] in in 1983. Belief propagation applied on a tree can generate exact result. In a general graph, which usually contains loops, we convert it to a junction tree before we use belief propagation. Another approach uses loopy belief propagation [92] as an approximate inference method. Loopy belief propagation will be introduced in Section 3.3.

3.3 Approximate inference

Until now, all the inference methods we introduced are exact. Applying these algorithms will yield exact solutions. They are more efficient than a "brute force" approach, because they exploit the independence properties of the stochastic system. However, what can we do if the system is so involved that few independent relationships can be found. An extreme example would be a system described by a fully connected graph. In this case, the nature of the system does not provide us anything to help designing algorithms simpler than "brute force". In this section, we will introduce several approximate inference algorithms. Another "uncomfortable" case would be a system with complicated local functions, e.g. non-Gaussian distribution, non-linear function. Both these situations make an exact inference impossible. The solution to the second problem will be left to the next chapter.

3.3.1 Loopy belief propagation

In Section 3.2, we introduced the belief propagation method. Exact marginalization can be obtained if the graph is a tree. However, in many cases, the graph contains loops. One way of calculating the exact marginals is to use the junction tree algorithm, which modifies the graph to guarantee a tree structure. This was demonstrated on the example in Section 3.2.3, where a graph with loops (Figure 3.1) was turned into a tree (Figure 3.6). However, in a complex system, finding a suitable junction tree for the graphical model is an NP-hard problem by itself. Pearl has suggested using belief propagation as an approximation for the inference in loopy network, which is known as loopy belief propagation algorithm.

Loopy belief propagation can be implemented almost in the same way as the belief propagation that is implemented in a tree. During the belief propagation in a tree, neighboring nodes exchange their messages only once and the order of the inference is always from leaves to the root of the tree where the root is chosen to be the variable whose marginal is to be calculated. However, no leaves can be found in a loop which raises the problem of finding the order of message passing. In order to solve this problem, we initialize all the messages with 1. Then the messages will be iteratively updated by the constraints presented in the function nodes. The message updating rules introduced in Section 3.2.2 are used for the messages calculation at every iteration. Typically, messages will be propagated along the loops for several rounds with each node in the loop being visited several times until the termination criterion is met.

The precise conditions of the convergence of a loopy belief propagation are still not well understood. It is shown in [107] that graphs containing a single loop will converge to a correct solution. Several sufficient conditions for the convergence of the sum-product algorithm are shown in [72]. An example where the belief propagation does not converge to the exact solution can be found in [106]. An extrinsic information transfer chart (EXIT chart) [103] can be used to illustrate visually the progress of belief propagation and to judge the convergence for some applications. Although the convergence of loopy belief propagation is still an open question, it has been used in many practical applications with great empirical success. Typical applications include joint decoding of turbo code [68] or low density parity checking (LDPC) code [23], medical diagnosis [78], image processing and computer vision problems [26][24][21].

3.3.2 Variational inference

Let us recall the inference problem in (3.1). Evaluating the marginal of observations $\mathbf{x}_{\mathcal{I}}$ in a graph $\mathcal{G}$ requires marginalizing out all the hidden variables $\mathbf{x}_{\mathcal{H}}$. The computational complexity is determined by the density of the sub-graph of $\mathcal{G}$ over the hidden variables. Let us assume the worst case that nodes in this sub-graph $\mathcal{G}_{\mathcal{H}}$ are so densely connected that it is impossible to run variable elimination algorithms for exact solutions. In this case, we can use variational methods to reduce the complexity. The basic idea of variational inference is to pose the probability distribution under query as the solution of an optimization problem. Then a perturbation is introduced and the solution for the pertubed problem is found. This section reviews one of the variational inference algorithms based on the minimization of Kullback-Leibler distance.

We pose an optimization problem where the cost function J is defined as follows:

$$J(q) = \log p(\mathbf{x}_{\mathcal{I}}) - KL(q(\mathbf{x}_{\mathcal{H}}) \| p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})) \tag{3.20}$$

Here $q(\mathbf{x}_{\mathcal{H}})$ is called variational probability. It is a probability distribution over the hidden variables. And the Kullback-Leibler distance between $q(\mathbf{x}_{\mathcal{H}})$ and $p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})$ is given by:

$$KL(q(\mathbf{x}_{\mathcal{H}}) \| p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})) = \sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log \frac{q(\mathbf{x}_{\mathcal{H}})}{p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})} \tag{3.21}$$

The KL distance is always non-negative. It is zero if and only if the two distributions $p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})$ and $q(\mathbf{x}_{\mathcal{H}})$ are identical. This means the cost function reaches its maximum $\log p(\mathbf{x}_{\mathcal{I}})$ only if q is identical to the posterior probability. So the optimization problem is to find q which maximizes the cost function. The definition of the cost function is intuitive. By optimizing the cost function, we simultaneously obtain the marginal probability ($J_{\text{opt}} = \log p(\mathbf{x}_{\mathcal{I}})$) and the posterior probability ($q_{\text{opt}} = p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})$).

Now, let us rewrite (3.20) as follows,

$$\begin{aligned}
J(q) &= \log p(\mathbf{x}_{\mathcal{I}}) - KL(q(\mathbf{x}_{\mathcal{H}}) \| p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})) \\
&= \log p(\mathbf{x}_{\mathcal{I}}) - \sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log \frac{q(\mathbf{x}_{\mathcal{H}})}{p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})} \\
&= \sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log p(\mathbf{x}_{\mathcal{I}}) - \sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log \frac{q(\mathbf{x}_{\mathcal{H}})}{p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})} \\
&= -\sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log \frac{q(\mathbf{x}_{\mathcal{H}})}{p(\mathbf{x}_{\mathcal{I}})p(\mathbf{x}_{\mathcal{H}}|\mathbf{x}_{\mathcal{I}})} \\
&= -\sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log q(\mathbf{x}_{\mathcal{H}}) + \sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}}) \\
&= H(q) + E_q\{\log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})\}
\end{aligned} \tag{3.22}$$

In (3.22), we express the cost function J as the sum of two terms. The first term $H(q)$, is the entropy of the variational probability. And the second term is the expectation of the joint probability with respect to q.

Now, the computation of the maximum depends on two factors. One is the structure of the original joint probability $p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})$. The other is the structure of the variational probability. Assume we can factorize the joint probability as a product of local potentials:

$$p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}}) = \prod_i \phi_{c_i}(\mathbf{x}_{c_i}) \tag{3.23}$$

Inserting this factorization into the cost function, we turn the expectation into a sum of simpler terms:

$$\begin{aligned}
J(q) &= H(q) + \sum_{\mathbf{x}_{\mathcal{H}}} (q(\mathbf{x}_{\mathcal{H}}) \log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})) \\
&= H(q) + \sum_{\mathbf{x}_{\mathcal{H}}} \left(q(\mathbf{x}_{\mathcal{H}}) \log \prod_i \phi_{c_i}(\mathbf{x}_{c_i}) \right) \\
&= H(q) + \sum_{\mathbf{x}_{\mathcal{H}}} \left(q(\mathbf{x}_{\mathcal{H}}) \sum_i \log \phi_{c_i}(\mathbf{x}_{c_i}) \right) \\
&= H(q) + \sum_i \left(\sum_{\mathbf{x}_{\mathcal{H}}} q(\mathbf{x}_{\mathcal{H}}) \log \phi_{c_i}(\mathbf{x}_{c_i}) \right) \\
&= H(q) + \sum_i \left(\sum_{\mathbf{x}_{c_i \cap \mathcal{H}}} q(\mathbf{x}_{c_i \cap \mathcal{H}}) \log \phi_{c_i}(\mathbf{x}_{c_i}) \right)
\end{aligned} \tag{3.24}$$

Up to this step, we have only simplified the problem by exploiting the factorization property of the joint probability. As we did not put any constraint on the structure of q, an exact solution is still recoverable. To further simplify the problem, we will play with the structure of q.

As we can learn from the variable elimination algorithm, simplification in probabilistic inference algorithms comes from the independence property of the variables. Here, for the variational probability, the simplest choice is the one that all variables are independent of each other, i.e.:

$$q(\mathbf{x}_{\mathcal{H}}) = \prod_{j\in\mathcal{H}} q_j(x_j) \tag{3.25}$$

Let us insert (3.25) into (3.24). Based on the fact that entropy of independent variables equals the sum of the individual entropies, we obtain:

$$J(q) = \sum_{j\in\mathcal{H}} H(q_j) + \sum_i \sum_{\mathbf{x}_{c_i\cap\mathcal{H}}} q(\mathbf{x}_{c_i\cap\mathcal{H}}) \log \phi_{c_i}(\mathbf{x}_{c_i}) \tag{3.26}$$

The optimization in (3.26) can be approached by an iterative method. As we have fully factorized q into the product of marginals q_j in (3.25), each of them can be adjusted independently. Based on this, we can design an updating rule where each time, we fix the other single marginals, while adjusting one single marginal q_k to maximize the cost function. This can be done by taking the derivative of J with respect to q_k, setting to zero and then finding the solution for q_k. Before we express this mathematically, we first make a definition to make the expression more explicit. We define:

$$E_q\{\log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})|x_k\} = \sum_{\mathbf{x}_{\mathcal{H}\setminus k}} \prod_{j\in\mathcal{H}\setminus k} q_j(x_j) \log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}}) \tag{3.27}$$

Now, we can write the cost function for each individual q_k assuming that the other marginals are fixed. Using the first line of (3.24), we obtain:

$$\begin{aligned} J(q_k) &= J(q) \\ &= \sum_{j\in\mathcal{H}} H_j(q_j) + \sum_{\mathbf{x}_{\mathcal{H}}} \left(q(\mathbf{x}_{\mathcal{H}}) \log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})\right) \\ &= \sum_{j\in\mathcal{H}} H_j(q_j) + \sum_{\mathbf{x}_{\mathcal{H}}} \left(\prod_{j\in\mathcal{H}} q_j(x_j) \log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})\right) \\ &= \sum_{j\in\mathcal{H}} H_j(q_j) + \sum_{x_k} \left[q_k(x_k) \sum_{\mathbf{x}_{\mathcal{H}\setminus k}} \left(\prod_{j\in\mathcal{H}\setminus k} q_j(x_j) \log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})\right)\right] \\ &= \sum_{j\in\mathcal{H}\setminus k} H_j(q_j) + H_k(q_k) + \sum_{x_k} \left(q_k(x_k) E_q\{\log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})|x_k\}\right) \end{aligned} \tag{3.28}$$

Taking the derivative of J with respect to q_k and setting to zero, we get:

$$-1 - \log q_k(x_k) + E_q\{\log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})|x_k\} = 0 \tag{3.29}$$

Solving (3.29), we obtain the updating rule:

$$q_k(x_k) \leftarrow \frac{1}{Z_k} e^{E_q\{\log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})|x_k\}} \tag{3.30}$$

Here Z_k is the normalization factor which is given by:

$$Z_k = \sum_{x_k} e^{E_q\{\log p(\mathbf{x}_{\mathcal{H}}, \mathbf{x}_{\mathcal{I}})|x_k\}} \tag{3.31}$$

Applying (3.30) to update each factor $q_k(x_k)$, the value of the cost function $J(q)$ will be monotonically increased. In general, the choice of the initial value and the order of the updating affect the final result that the algorithm is going to converge to.

In the variational inference presented above, a fully factorizable variational probability distribution is chosen. Such an approximation is called *mean-field method*. There exist other choices of the variational probability distribution q. For example, [54] presents the so-called *structured mean-field* approach. The *cluster variation method*[56, 40, 73, 74], developed by Kikuchi et al, introduces different approximations to the cost function $J(q)$. In mean field approach, the variational probability distribution is restricted to those that are fully-factorized, i.e., each factor in q contains one variable only, whereas in Kikuchi's cluster variation method, variational probability distribution contains bigger factors, i.e., there are factors that contains multiple variables. With cluster variation method, one can find an approximation that is more accurate than the mean field method. Yedidia et al generalized the approximations used in cluster variation method and introduced *region-based approximations*. [113] shows that one of these approximation method-*Bethe approximation*-generates results that are equivalent to the belief propagation algorithm. Based on the study of region-based approximations, Yedidia et al proposed generalized belief propagation algorithm in [112]. Variational inference has a close relationship with free energies in statistical mechanics [113]. Many approximation methods are inspired by the study done by physicists and it is not surprising that many of these methods borrow the names of the counterparts in physics.

3.4 Extensions and discussions

This chapter reviewed several inference algorithms. We started from the “brute force” approach which is infeasible for many applications due to the high computational complexity. Variable elimination exploits the decomposability property of the joint probability distribution and carefully arranges the sequence of computation to reduce the computational effort. Such an idea can be visually represented as node elimination on graphical models. Belief propagation on factor graphs implements variable elimination in an efficient way in the sense that it avoids unnecessary repetition of computing the intermediate results when computing simultaneously several marginals. If the graph is a tree, belief propagation calculates the exact marginals. If the graph contains loops, node elimination has a problem of finding the starting point since every node on the loop is the ”reason” and the ”consequence” of others. One solution is to use the junction tree algorithm to convert the original graph to a tree before running belief propagation. The junction tree algorithm still discovers the exact marginals. But its complexity is determined by the size of the clusters and the separators, which are usually big for the sake of breaking the loops. Finding a junction tree that optimizes the belief propagation running on it is in itself an

NP-hard problem. So either we make a lot of effort to search for a junction tree that minimizes the computational complexity of the belief propagation or we spend less effort on searching but come up with a junction tree that still needs complicated computations. An alternative solution to the problem of loops is to run loopy belief propagation. Convergence conditions for loopy belief propagation has not been well studied. However, empirical results showed successful application of this approximation method.

In Section 3.2.2, we presented belief propagation, in particular, the sum-product algorithm and the max-product algorithm on a standard factor graph. Some researchers prefer a more compact representation on Forney style factor graphs. Forney style factor graph only contains function nodes. An edge between two function nodes is labelled by the variables that are shared by both functions. A message on an edge is defined as a function of the variable that is associated with that edge. Figure 3.8 illustrates a part of a Forney style factor graph. In this example, functions f_a and f_b have a common argument x_i. The message from f_a to f_b, denoted by: $m_{f_a \to f_b}(x_i)$ is calculated by:

$$m_{f_a \to f_b}(x_i) = \sum_{\mathbf{x}_a \setminus x_i} \left(f_a(\mathbf{x}_a) \prod_{c \in NE(a) \setminus b} m_{f_c \to f_a} \right) \tag{3.32}$$

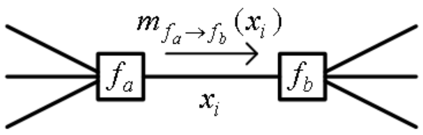


Fig. 3.8. Message passing in Forney style factor graph

This chapter showed how to calculate the marginal probability distributions on an undirected graph. If we have a directed graph, we first convert it into an undirected graph then follow the inference algorithms introduced in this chapter. Let us consider a directed graph example in Figure 3.9, which represents:

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = p(x_4)p(x_2|x_4)p(x_5|x_4)p(x_1|x_2)p(x_3|x_2)p(x_6|x_3, x_5) \tag{3.33}$$

We can convert it into an undirected graph, which is exactly identical to the graph in Figure 3.1. If we want to infer the marginal probability of x_4, we can follow the variable elimination shown in (3.5). However, the marginal probability distribution of x_4 is already given on the right hand side of (3.33). That means we do not need any calculation at all. By simply looking at the undirected model in Figure 3.1, we cannot detect this convenience. Such an example shows that some times there is a good reason to keep the arrows when we do the conversion. And it is preferable to introduce directions in a factor graph to explicitly express the causality relationship on the graph.

3.A Sum-product algorithm on a tree

If the factor graph is a tree, the sum-product algorithm on it can calculate the exact marginals. We first present how to use a factor graph to arrange the sequence of calculations in order to evaluate a particular marginal, which turns out to be a message passing

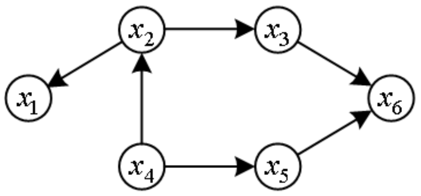


Fig. 3.9. Directed graph representing (3.33)

method, i.e., the sum-product algorithm. Then we use a simple example to illustrate the sum-product algorithm.

3.A.1 Evaluation of marginals in factor graphs

Suppose we have a factor graph $\mathcal{G} = \{\mathcal{U}, \mathcal{V}, \mathcal{E}\}$ which represents a global function:

$$f_{\mathcal{G}}(\mathbf{x}_{\mathcal{V}}) = \prod_{u \in \mathcal{U}} f_u(\mathbf{x}_u) \tag{3.34}$$

Let us consider the evaluation of the marginal of a particular variable $x \in \{\mathbf{x}_{\mathcal{V}}\}$. Figure 3.10 shows the subgraph that contains variable node x and its neighbors. We use $NE(x) = \{r_1, \ldots r_N\}$ to denote the index of N neighbors of x. Since a factor graph is a bipartite graph, a variable node is connected only to function nodes. So the neighbors of x are factors $\{f_{r_1}, \ldots f_{r_N}\}$ where each factor f_{r_i} contains x and $\mathbf{x}_{r_i}$. Since the factor graph under study is a tree, if we remove variable x, then the factor graph is divided into N subgraphs, i.e.: $\{\mathcal{G}_1, \ldots \mathcal{G}_N\}$ with $\mathcal{G}_i = \{\mathcal{U}_i, \mathcal{V}_i, \mathcal{E}_i\}$ where $\{\mathbf{x}_{\mathcal{V}_i}\} = \cup_{u \in \mathcal{U}_i}\{\mathbf{x}_u\}$. We call x the root of the subgraphs of $\{\mathcal{G}_i\}_{i=1,\cdots N}$. Each subgraph $\mathcal{G}_i$ represents:

$$\begin{aligned} f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) &= \prod_{u \in \mathcal{U}_i} f_u(\mathbf{x}_u) \\ &= f_{r_i}(x, \mathbf{x}_{r_i}) \prod_{u \in \mathcal{U}_i \backslash r_i} f_u(\mathbf{x}_u) \end{aligned} \tag{3.35}$$

and now the overall global function can be written as:

$$f_{\mathcal{G}}(\mathbf{x}_{\mathcal{V}}) = \prod_{i=1}^{N} f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) \tag{3.36}$$

The marginal of x can be calculated as follows, using the result from (3.35):

$$\begin{aligned} \beta(x) &= \sum_{\mathbf{x}_{\mathcal{V}} \backslash x} f_{\mathcal{G}}(\mathbf{x}_{\mathcal{V}}) \\ &= \sum_{\mathbf{x}_{\mathcal{V}} \backslash x} \prod_{i=1}^{N} f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) \\ &= \prod_{i=1}^{N} \sum_{\mathbf{x}_{\mathcal{V}_i} \backslash x} f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) \end{aligned} \tag{3.37}$$

Let us define:

$$m_{f_{r_i}\to x}(x) = \sum_{\mathbf{x}_{\mathcal{V}_i}\setminus x} f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) \tag{3.38}$$

It can be regarded as a message that f_{r_i} sends to x. This message contains the summary of the calculation in the subgraph $\mathcal{G}_i$. Then we can rewrite (3.37) as:

$$\beta(x) = \prod_{i=1}^{N} m_{f_{r_i}\to x}(x) \tag{3.39}$$

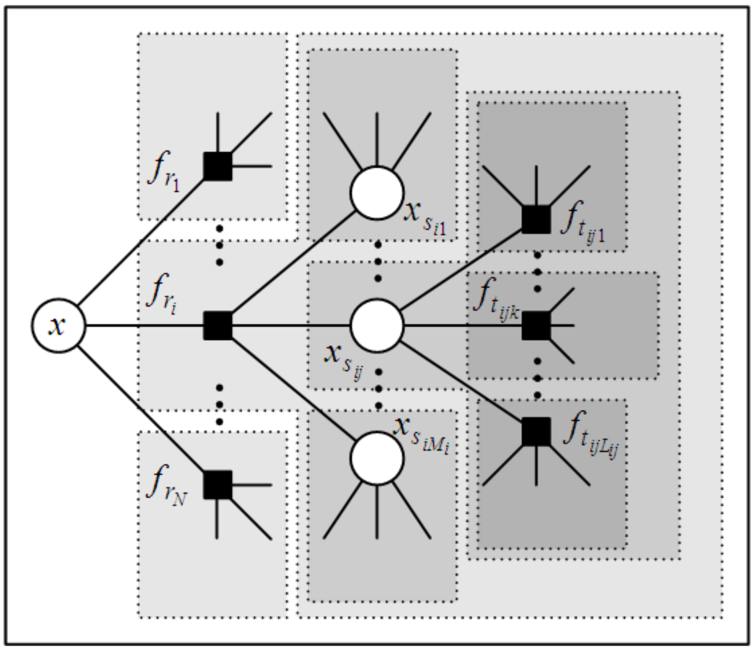


Fig. 3.10. Illustration of marginal calculation
(by dividing the factor graphs into small subgraphs)

We further divide each $\mathcal{G}_i$ into M_i subgraphs $\{\mathcal{G}_{ij} = \{\mathcal{U}_{ij}, \mathcal{V}_{ij}, \mathcal{E}_{ij}\}\}_{j=1,\ldots M_i}$ as shown in Figure 3.10, where M_i is the number of f_{r_i}'s neighbors besides of x. So $\mathbf{x}_{r_i} = \{x_{s_{i1}}, \ldots x_{s_{iM_i}}\}$. In this case, f_{r_i} is the root of the subgraphs $\{\mathcal{G}_{ij}\}_{j=1,\ldots M_i}$. Let us use $NE(f_{r_i}) = \{s_1, \ldots s_{M_i}\}$ to denote the neighbors of f_{r_i}. Then the factor $f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i})$ can be written as:

$$f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) = f_{r_i}(x, \mathbf{x}_{r_i}) \prod_{j=1}^{M_i} f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}}) \tag{3.40}$$

It should be noticed that x is not contained in any $\mathbf{x}_{\mathcal{V}_{ij}}$. And only $x_{s_{ij}}$ in $\mathbf{x}_{r_i}$ is contained in $\mathbf{x}_{\mathcal{V}_{ij}}$. Then the message $m_{f_{r_i}\to x}(x)$ can be written as:

$$\begin{aligned} m_{f_{r_i}\to x}(x) &= \sum_{\mathbf{x}_{\mathcal{V}_i}\backslash x} f_{\mathcal{G}_i}(\mathbf{x}_{\mathcal{V}_i}) \\ &= \sum_{\mathbf{x}_{\mathcal{V}_i}\backslash x} f_{r_i}(x,\mathbf{x}_{r_i}) \prod_{j=1}^{M_i} f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}}) \\ &= \sum_{\mathbf{x}_{r_i}} f_{r_i}(x,\mathbf{x}_{r_i}) \prod_{j=1}^{M_i} \sum_{\mathbf{x}_{\mathcal{V}_{ij}}\backslash x_{s_{ij}}} f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}}) \end{aligned} \tag{3.41}$$

Let us define:

$$m_{x_{s_{ij}}\to f_{r_i}}(x_{s_{ij}}) = \sum_{\mathbf{x}_{\mathcal{V}_{ij}}\backslash x_{s_{ij}}} f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}}) \tag{3.42}$$

So it is the message that $x_{s_{ij}}$ sends to f_{r_i}. It is the summary of the calculation in subgraph $\mathcal{G}_{ij}$. Inserting (3.42) into (3.41), we obtain:

$$m_{f_{r_i}\to x}(x) = \sum_{\mathbf{x}_{r_i}} f_{r_i}(x,\mathbf{x}_{r_i}) \prod_{j=1}^{M_i} m_{x_{s_{ij}}\to f_{r_i}}(x_{s_{ij}}) \tag{3.43}$$

which gives us the message updating equation at a function node.

Now let us divide each subgraph $\mathcal{G}_{ij}$ again into subgraphs as illustrated in Figure 3.10 so that $\mathcal{G}_{ij}$ is composed of $\{\mathcal{G}_{ijk} = \{\mathcal{U}_{ijk},\mathcal{V}_{ijk},\mathcal{E}_{ijk}\}\}_{k=1,\ldots L_{ij}}$, where L_{ij} is the number of neighbors of $x_{s_{ij}}$ besides f_{r_i}. Let us use $NE(x_{s_{ij}}) = \{t_1,\ldots t_{L_{ij}}\}$. Then the factor $f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}})$ can be factorized as follows:

$$f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}}) = \prod_{k=1}^{L_{ij}} f_{\mathcal{G}_{ijk}}(\mathbf{x}_{\mathcal{V}_{ijk}}) \tag{3.44}$$

Then the message calculation in (3.42) can be rewritten as:

$$\begin{aligned} m_{x_{s_{ij}}\to f_{r_i}}(x_{s_{ij}}) &= \sum_{\mathbf{x}_{\mathcal{V}_{ij}}\backslash x_{s_{ij}}} f_{\mathcal{G}_{ij}}(\mathbf{x}_{\mathcal{V}_{ij}}) \\ &= \sum_{\mathbf{x}_{\mathcal{V}_{ij}}\backslash x_{s_{ij}}} \prod_{k=1}^{L_{ij}} f_{\mathcal{G}_{ijk}}(\mathbf{x}_{\mathcal{V}_{ijk}}) \\ &= \prod_{k=1}^{L_{ij}} \sum_{\mathbf{x}_{\mathcal{V}_{ijk}}\backslash x_{s_{ij}}} f_{\mathcal{G}_{ijk}}(\mathbf{x}_{\mathcal{V}_{ijk}}) \end{aligned} \tag{3.45}$$

Let us define:

$$m_{f_{t_{ijk}}\to x_{s_{ij}}} = \sum_{\mathbf{x}_{\mathcal{V}_{ijk}}\backslash x_{s_{ij}}} f_{\mathcal{G}_{ijk}}(\mathbf{x}_{\mathcal{V}_{ijk}}) \tag{3.46}$$

Then it is the message from $f_{t_{ijk}}$ to $x_{s_{ij}}$. It summarizes the calculation in the subgraph $\mathcal{G}_{ijk}$. And now, we can rewrite (3.45) as:

$$m_{x_{s_{ij}} \to f_{r_i}}(x_{s_{ij}}) = \prod_{k=1}^{L_{ij}} m_{f_{t_{ijk}} \to x_{s_{ij}}} \tag{3.47}$$

which gives the message update equation at a variable node.

We can keep dividing subgraphs into smaller graphs until all subgraphs are not dividable anymore. A non-dividable graph may contain a single variable node or a single function node. If it contains a single variable node x_{u*}, then the message it sends to its root f_{v*} is given by:

$$m_{x_{u*} \to f_{v*}}(x_{u*}) = 1 \tag{3.48}$$

If a non-dividable subgraph contains a single function node $f_{v'}$, then the message it sends to its root $x_{u'}$ is, note in this case $f_{v'}$ must be a function of $x_{u'}$ only:

$$m_{f_{v'} \to x_{u'}}(x_{u'}) = f_{v'}(x_{u'}) \tag{3.49}$$

We can use (3.48) and (3.49) to initialize the marginal calculation at the non-dividable nodes, then use (3.43) and (3.47) to recursively calculate the message from subgraphs to their roots and expand the graph. At the end, when the expansion reaches our target variable x, we can use (3.39) to summarize the marginal of x.

3.A.2 A sum-product algorithm example

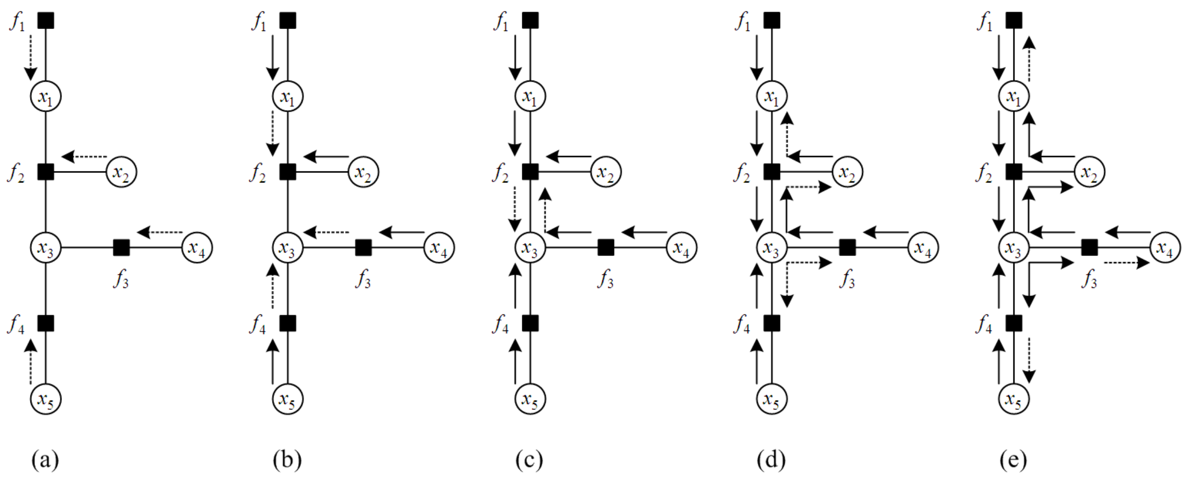


Fig. 3.11. An example of the sum-product algorithm in factor graph
Dotted arrows represent messages that are available in current step. Solid arrows represent the messages that were calculated in previous steps.

An example that illustrates the sum-product algorithm is shown in Figure 3.11. The factor graph in Figure 3.11 represents:

$$f(x_1, x_2, x_3, x_4, x_5) = f_1(x_1) f_2(x_1, x_2, x_3) f_3(x_3, x_4) f_4(x_3, x_5) \tag{3.50}$$

Message passing starts from the leaves. So in the first step (Figure 3.11(a)), messages are sent out from function node f_1 and variable nodes x_2, x_4 and x_5. The content of the messages are given by:

$$\begin{aligned} m_{f_1\to x_1}(x_1) &= f_1(x_1) && (3.51)\\ m_{x_2\to f_2}(x_2) &= 1 && (3.52)\\ m_{x_4\to f_3}(x_4) &= 1 && (3.53)\\ m_{x_5\to f_4}(x_5) &= 1 && (3.54) \end{aligned}$$

In the next step (Figure 3.11(b)), node x_1 can calculate its message to f_2 using (3.6) and f_3 and f_4 can calculate their messages to x_3 using (3.7). So the following messages are available:

$$\begin{aligned} m_{x_1\to f_2}(x_1) &= m_{f_1\to x_1}(x_1) && (3.55)\\ m_{f_3\to x_3}(x_3) &= \sum_{x_4} f_3(x_3,x_4)m_{x_4\to f_3}(x_4) && (3.56)\\ m_{f_4\to x_3}(x_3) &= \sum_{x_5} f_4(x_3,x_5)m_{x_5\to f_4}(x_5) && (3.57) \end{aligned}$$

In the third step (Figure 3.11(c)), node f_2 can calculate its message to x_3 by using (3.7) and node x_3 calculates its messages to f_2 using (3.6):

$$\begin{aligned} m_{f_2\to x_3}(x_3) &= \sum_{x_1,x_2} f_2(x_1,x_2,x_3)m_{x_1\to f_2}(x_1)m_{x_2\to f_2}(x_2) && (3.58)\\ m_{x_3\to f_2}(x_3) &= m_{f_3\to x_3}(x_3)m_{f_4\to x_3}(x_3) && (3.59) \end{aligned}$$

In the fourth step (Figure 3.11(d)), node f_2 calculates it messages to x_1 and x_2, node x_3 calculates its messages to f_3 and f_4. The calculations are as follows:

$$\begin{aligned} m_{f_2\to x_1}(x_1) &= \sum_{x_2,x_3} f_2(x_1,x_2,x_3)m_{x_2\to f_2}(x_2)m_{x_3\to f_2}(x_3) && (3.60)\\ m_{f_2\to x_2}(x_2) &= \sum_{x_1,x_3} f_2(x_1,x_2,x_3)m_{x_1\to f_2}(x_1)m_{x_3\to f_2}(x_3) && (3.61)\\ m_{x_3\to f_3}(x_3) &= m_{f_2\to x_3}(x_3)m_{f_4\to x_3}(x_3) && (3.62)\\ m_{x_3\to f_4}(x_3) &= m_{f_2\to x_3}(x_3)m_{f_3\to x_3}(x_3) && (3.63) \end{aligned}$$

In the final step (Figure 3.11(e)), the following messages are calculated:

$$\begin{aligned} m_{x_1\to f_1}(x_1) &= m_{f_2\to x_1}(x_1) && (3.64)\\ m_{f_3\to x_4}(x_4) &= \sum_{x_3} f_3(x_3,x_4)m_{x_3\to f_3}(x_3) && (3.65)\\ m_{f_3\to x_5}(x_5) &= \sum_{x_3} f_4(x_3,x_5)m_{x_3\to f_4}(x_3) && (3.66) \end{aligned}$$

After this step, all the messages are available. Marginal probabilities can be calculated. For example, the marginal of x_3, i.e., $\beta(x_3)$ is calculated by multiplying all the incoming messages:

$$\beta(x_3) = m_{f_2\to x_3}(x_3)m_{f_3\to x_3}(x_3)m_{f_4\to x_3}(x_3) \quad (3.67)$$

Inserting the results from (3.58), (3.56) and (3.56) into (3.67), we obtain:

$$\begin{aligned}\beta(x_3) &= \sum_{x_1,x_2} f_2(x_1,x_2,x_3) m_{x_1\to f_2}(x_1) m_{x_2\to f_2}(x_2) \sum_{x_4} f_3(x_3,x_4) m_{x_4\to f_3}(x_4) \\ &\quad \sum_{x_5} f_4(x_3,x_5) m_{x_5\to f_4}(x_5) \end{aligned} \tag{3.68}$$

Using the results from (3.55), (3.52), (3.53) and (3.54), we can obtain:

$$\beta(x_3) = \sum_{x_1,x_2} f_2(x_1,x_2,x_3) m_{f_1\to x_1}(x_1) \sum_{x_4} f_3(x_3,x_4) \sum_{x_5} f_4(x_3,x_5) \tag{3.69}$$

Inserting the result from (3.51) into (3.69) we obtain:

$$\begin{aligned}\beta(x_3) &= \sum_{x_1,x_2} f_2(x_1,x_2,x_3) f_1(x_1) \sum_{x_4} f_3(x_3,x_4) \sum_{x_5} f_4(x_3,x_5) \\ &= \sum_{x_1,x_2,x_4,x_5} f_1(x_1) f_2(x_1,x_2,x_3) f_3(x_3,x_4) f_4(x_3,x_5) \\ &= \sum_{x_1,x_2,x_4,x_5} f(x_1,x_2,x_3,x_4,x_5) \end{aligned} \tag{3.70}$$

From this result we can see that the sum-product algorithm can calculate the exact marginal of x_3. The first line in (3.70) also indicates that variable elimination is implemented by the sum-product algorithm.

4. Probabilistic Inference in Networked Systems

A networked system is composed of a large number of simple systems which are connected via communication links so that they can interact with each other. Modern networked systems (e.g., industrial Ethernet) enable flexible system operation and reduce the cost of installation and maintenance. There are many potential applications of networked systems in many areas of engineering and science. Typical examples include sensor networks, industrial automation networks, power grids, transportation systems and so on. Networked systems are usually complex dynamical systems with uncertainties. Hence, achieving desired behavior of the whole system requires reliable estimation of the state of the system under uncertainties.

A lot of uncertainties are typically involved in networked systems, such as the measurement noise and random effects introduced by the communication system, e.g., varying delay, packet loss, bit errors. Probability theory provides an appropriate framework to quantify these uncertainties. On the other hand, sampled measurements are usually correlated in time and / or space. Soft constraints, e.g., local potential functions can be used to describe the strength of correlation. Therefore, it is reasonable to use probabilistic models as a suitable mathematical expression of the estimation problem in networked systems.

In a probabilistic model, uncertainties and soft constraints are modeled by random variables with given distributions, system states are modeled by hidden variables and observations are known variables. The relationships between the variables are given by the underlying physics. In this way, we can formulate state estimation as a probabilistic inference problem where the posterior probability distribution of the hidden variables, given the observations, can be computed. The result not only calculates the MAP estimate of the variables of interest, but also provides the uncertainty of the estimation via the probability distribution.

In practice, a typical networked system has a very high complexity due to the large number of participants in the system, different sources of randomness and very complicated physics or underlying dynamics of the sub-systems. As a consequence, probabilistic inference in a typical networked system involves a large scale of computations, which is a non-trivial task.

In this chapter, we will discuss tractable inference methods for the estimation problems in a typical networked system. In particular, the following issues will be addressed:

1) distributed inference in a networked system

2) function and message approximation

3) state estimation for a dynamical system

State estimation can be done in a centralized way which requires local measurements to be transmitted to a fusion center where a global model of the whole system will be established. However, we prefer distributed inference in practice in order to save the power consumed by transmitting the measurements, to parallelize the computations and to avoid intensive processing at a single network element. We will derive in this chapter the basic methodology of implementing probabilistic inference in a distributed manner, i.e., distributed inference in a networked system.

A practical networked system may involve uncertainties that have very complicated, non-standard density functions. The model of the underlying physics of the system may contain non-linear continuous functions. The high complexity of these functions may cause the inference algorithms introduced in Chapter 3 to be intractable. On the other hand, to transmit a very complicated function efficiently over communication links, we need to find out an appropriate representation. Usually, we have to introduce approximations in order to simplify the computation in inference and to reduce the size of the messages that should be communicated. In this chapter, we discuss two function and message approximation methods. We derive *Fourier domain belief propagation* which is based on the *Fourier density approximation* technique [11, 12]. Then we present *non-parametric belief propagation* [102] as another approximation method.

A networked system usually contains many stochastic processes that are coupled with each other. As a result, the whole system is a very large-scale dynamical system. Modeling and state estimation for such a complicated system is not trivial. We introduce dynamic Bayesian network as an appropriate tool to model the system. Several state estimation methods, exact or approximate, will be presented. Examples are given to illustrate these methods.

This chapter will solve the three classes of problems mentioned above in very general terms. They encompass the most critical issues in the probabilistic inference for a complicated networked system. In the next two chapters, we will elaborate and apply the methods introduced in this chapter to solve the sensor localization and clock synchronization problems. From these applications, we will see that several of the techniques mentioned in this chapter can be combined to solve a complicated problem. For example, sensor localization uses distributed inference and function approximation; clock synchronization uses distributed inference in a linear dynamical system.

The rest of this chapter is organized as follows. Section 4.1 introduces appropriate distributed inference methods for state estimation in networked systems. Section 4.2 presents function approximation methods targeting on simplifying the computation for the inference and reducing communications. Section 4.3 discusses probabilistic inference for dynamical systems. We conclude this chapter with remarks and discussions in Section 4.4. In Appendix 4.A.1 and 4.A.2, we present the derivation of the product and integration of Gaussian density functions. The results are used in the preceding sections and subsequent chapters.

4.1 Distributed inference

The message passing based inference algorithm introduced in the previous chapter already indicates the possibility of implementing probabilistic inference in a distributed way. Based on the topology of the communication system underlying a networked system, we can develop an inference graph (probabilistic graphical model). Network elements can exchange information required by the inference algorithm via the communication links so that marginal probability distributions or posterior probability distributions can be computed locally. However, the underlying communication system also influences the inference. For example, two elements are supposed by the inference algorithm to talk to each other, however there may be no direct communication link between them. Due to this constraint, the communication and the inference should be designed in a way that they fit each other, i.e.,

- inference has to be designed to adapt to the topology of the communication network
- communication links should be chosen properly to simplify the inference and to reduce the total amount of traffic

In this section, we will formulate the estimation problem mathematically. Then we present the probabilistic model for the system and introduce the most important procedures of distributed inference. An example will be shown at the end to illustrate distributed inference method.

4.1.1 Problem formulation

Let us assume that a system with N networked nodes is deployed to measure some physical environment. For each sensor node i, we define a hidden state variable x_i to denote the value of the environment or process state at that sensor position. Let y_i denote the sensor measurement at sensor node i, which should be relevant to the hidden states. The most common task of probabilistic inference is to compute at each sensor the posterior probability distribution $p(x_i|\mathbf{y})$ where $\mathbf{y} = \{y_1, y_2, ...y_N\}$.

4.1.2 General assumptions

Based on the property of many inference problems in networked systems, we observe that the following assumptions are valid for most applications:

- Given all state variables, observations at different sensor nodes are conditionally independent, i.e., $p(y_i, y_j|\mathbf{x}) = p(y_i|\mathbf{x})p(y_j|\mathbf{x})$;
- Observations made at one sensor node depend only on a subset of state variables, i.e. $p(y_i|\mathbf{x}) = p(y_i|\mathbf{x}_{PA(y_i)})$ where $\mathbf{x}_{PA(y_i)} \subset \mathbf{x}$ is a subset of the states variables that affect the sensor measurement at node i;

- Correlation exists between state variables. Usually, the correlation is local between neighboring nodes. This indicates that the joint probability distribution of state variables can be factorized into a product of local functions which represent the correlation among the nodes in neighborhoods, i.e., $p(\mathbf{x}) = \prod_C p(\mathbf{x}_C)$ where each C denotes a clique.

The first assumption comes from the fact that the observation can usually be expressed as a deterministic function of hidden variables plus additive measurement noise, e.g.,

$$y_i = g_i(\mathbf{x}) + \xi_i \tag{4.1}$$

where g_i is a function which maps the configuration of hidden state variables to the quantity that can be measured by the sensor. ξ_i denotes the random error made at the sensing process. The additive noises at different sensors are independent from each other, i.e.,

$$p(\xi_i, \xi_j) = p(\xi_i)p(\xi_j) \tag{4.2}$$

As a consequence:

$$p_{\xi_i,\xi_j}(y_i, y_j|\mathbf{x}) = p_{\xi_i}(y_i|\mathbf{x})p_{\xi_j}(y_j|\mathbf{x}) \tag{4.3}$$

The second assumption tells us that the deterministic function g_i in (4.1) is usually local. It depends only on the hidden state variables at the locations that are close to sensor i's location.

4.1.3 General inference procedure

Based on the assumptions in Section 4.1.2, and using the Bayes rule, the joint probability distribution of the state variables and the observations can be factorized as follows:

$$\begin{aligned} p(\mathbf{x}, \mathbf{y}) &= p(\mathbf{y} \mid \mathbf{x})p(\mathbf{x}) \\ &= \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_{PA(y_i)}) \cdot p(\mathbf{x}) \\ &= \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_{PA(y_i)}) \cdot \prod_{C_j} p(\mathbf{x}_{C_j}) \end{aligned} \tag{4.4}$$

Now we reorganize the factorization of the joint distribution to obtain the following structure:

$$p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} \varphi_i(y_i, \mathbf{x}_{C_i}) \tag{4.5}$$

so that each local function $\varphi_i(y_i, \mathbf{x}_{C_i})$ in (4.5) is associated with one single sensor node i. Such a factorization automatically provides the possibility to distribute the computation for the state estimation. Each node executes some local computation and the results are eventually disseminated over the whole network through the communications links between nodes.

To obtain the factorization, each factor in (4.4) should be assigned to one of the local functions in (4.5). The assignment is typically not unique. Different criteria should be considered, for example:

- if local functions share the same variables
- availability of a communication link
- link quality
- computational complexity
- power consumption of communication

Then an inference network i.e., a graphical model can be constructed. The inference network is a subnet of the communication network where only part of the communication links are used, depending on the information required by the inference algorithm. A robust architecture is presented in [90] where a tree-structured inference network is constructed taking into consideration the communication cost and the computational complexity. Thereby, it is convenient to use junction tree algorithm. This algorithm, over such an architecture provides a tractable solution for distributed inference [89], regression [34] and optimal control [33] in sensor network. The next section uses a simple example to illustrate this distributed inference approach. Other work, e.g., [15], [49] defines inference networks that contain loops. In this case, loopy belief propagation is used to solve the estimation problem. In the next chapter, we present loopy belief propagation for self-organized sensor localization.

4.1.4 Example of distributed inference

Let us assume a networked system with the topology shown in Figure 4.1. There are 6 elements in the network. Each of them measures the physical parameters at its own location and wants to estimate these parameters from the noisy measurements. We use x_i to denote the physical parameters at element i and use y_i to denote the noisy measurement of x_i. We assume that:

$$y_i = x_i + \xi_i \tag{4.6}$$

where the additive random noise variables ξ_i and ξ_j are independent if $i \neq j$. We further assume that the underlying physical parameters have space correlations, given by:

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = p(x_1, x_2)p(x_2, x_3)p(x_3, x_6)p(x_4, x_5)p(x_5, x_6) \tag{4.7}$$

In Figure 4.1, we use dashed ellipses to illustrate the spatial correlations.

Based on (4.6) and (4.7), we can write the joint probability distribution as follows:

$$\begin{aligned} p(y_1, \dots y_6, x_1, \dots x_6) &= p(x_1, \dots x_6)p(y_1, \dots y_6|x_1, \dots x_6) \\ &= p(x_1, \dots x_6) \prod_{i=1}^{6} p(y_i|x_1, \dots x_6) \\ &= p(x_1, x_2)p(x_2, x_3)p(x_3, x_6)p(x_4, x_5)p(x_5, x_6) \prod_{i=1}^{6} p(y_i|x_i) \end{aligned} \tag{4.8}$$

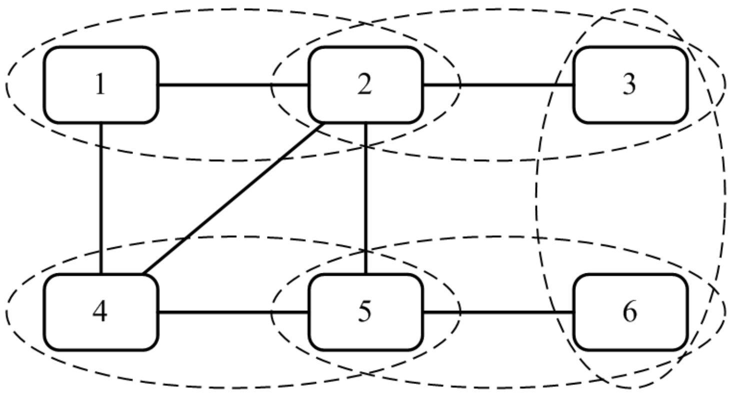


Fig. 4.1. Topology of the network and the spatial correlation

As discussed in Section 4.1.3, we want to write the joint probability distribution in the following form:

$$p(y_1, \ldots y_6, x_1, \ldots x_6) = \prod_{i=1}^{6} \varphi_i(y_i, \mathbf{x}_{C_i}) \tag{4.9}$$

so that each local function $\varphi_i(y_i, \mathbf{x}_{C_i})$ is associated with element i. Now we should assign the factors in (4.8) into one of these local functions. Obviously, the conditional probability $p(y_i|x_i)$ should be assigned to $\varphi_i(y_i, \mathbf{x}_{C_i})$. But there exist many possible ways to assign the rest of the factors.

4.1.4.1 Distributed inference-factor assignment scheme 1

One possible assignment is shown as follows:

$$\begin{aligned}
\varphi_1(y_1, x_1) &= p(y_1|x_1) \\
\varphi_2(y_2, x_1, x_2) &= p(y_2|x_2)p(x_1, x_2) \\
\varphi_3(y_3, x_2, x_3) &= p(y_3|x_3)p(x_2, x_3) \\
\varphi_4(y_4, x_4, x_5) &= p(y_4|x_4)p(x_4, x_5) \\
\varphi_5(y_5, x_5, x_6) &= p(y_5|x_5)p(x_5, x_6) \\
\varphi_6(y_6, x_3, x_6) &= p(y_6|x_6)p(x_3, x_6)
\end{aligned} \tag{4.10}$$

Figure 4.2 depicts the assignment of the factors. From that, we construct a junction tree based on the availability of communications links. The junction tree is shown in Figure 4.3.

In the junction tree in Figure 4.3, x_3 is added to local functions φ_2 and φ_5 to fulfill the running intersection property required by a junction tree (see Section 3.2.3). We choose the link between elements 2 and 5 instead of the link between elements 1 and 4 or the link between elements 2 and 4 because such a choice minimizes the complexity of the junction tree. For example, if we choose the link between elements 2 and 4, then x_3 should also be included in φ_4 to satisfy the running intersection requirement.

Figure 4.4 demonstrates the belief propagation for inferring marginals. The messages along the edges are computed using (3.14). At the end, the marginal probability distribution can be calculated by applying (3.15). The size of each message is associated with

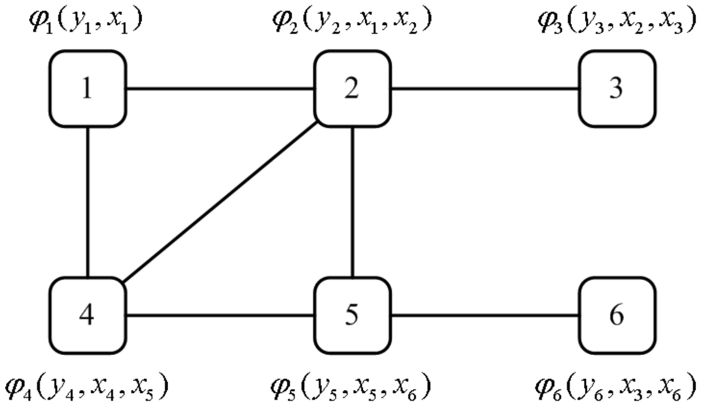

$\varphi_1(y_1, x_1)$
$\varphi_2(y_2, x_1, x_2)$
$\varphi_3(y_3, x_2, x_3)$
1
2
3
4
5
6
$\varphi_4(y_4, x_4, x_5)$
$\varphi_5(y_5, x_5, x_6)$
$\varphi_6(y_6, x_3, x_6)$

Fig. 4.2. Assignment of factors, scheme 1

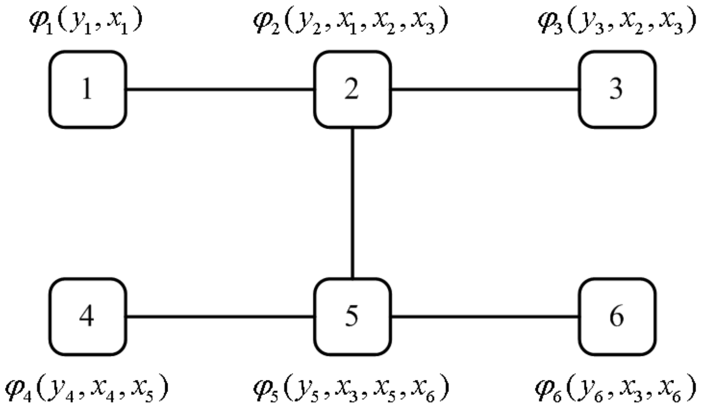

$\varphi_1(y_1, x_1)$
$\varphi_2(y_2, x_1, x_2, x_3)$
$\varphi_3(y_3, x_2, x_3)$
1
2
3
4
5
6
$\varphi_4(y_4, x_4, x_5)$
$\varphi_5(y_5, x_3, x_5, x_6)$
$\varphi_6(y_6, x_3, x_6)$

Fig. 4.3. Junction tree for assignment scheme 1

the number of variables involved. For example, two variables are inside the message from element 5 to element 6.

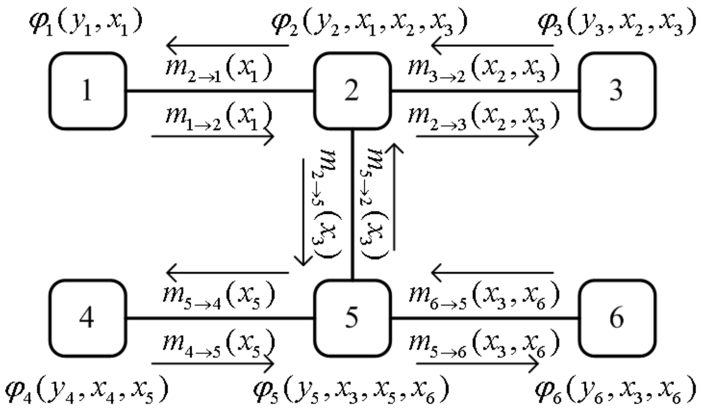


Fig. 4.4. Message passing for assignment scheme 1

Figure 4.5 illustrates the complexity of the message computation by showing how many variables should be summed out in the calculation with $\sum_{x_i, x_j}$. With no $\sum$ means no summation is needed for that message. For example, for the calculation of the message from element 2 to element 5, we should sum over all possible values that x_1 and x_2 can take. In the figure, this is expressed by labeling the corresponding message by $\sum_{x_1,x_2}$

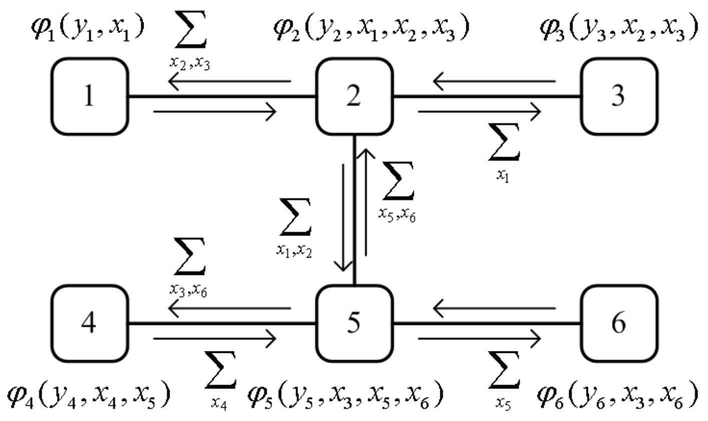


Fig. 4.5. Complexity of inference based on assignment scheme 1

4.1.4.2 Distributed inference-factor assignment scheme 2

Another possible assignment is shown as follows:

$$\begin{aligned}
\varphi_1(y_1,x_1,x_2) &= p(y_1|x_1)p(x_1,x_2)\\
\varphi_2(y_2,x_2,x_3) &= p(y_2|x_2)p(x_2,x_3)\\
\varphi_3(y_3,x_3) &= p(y_3|x_3)\\
\varphi_4(y_4,x_4,x_5) &= p(y_4|x_4)p(x_4,x_5)\\
\varphi_5(y_5,x_5,x_6) &= p(y_5|x_5)p(x_5,x_6)\\
\varphi_6(y_6,x_3,x_6) &= p(y_6|x_6)p(x_3,x_6) \qquad (4.11)
\end{aligned}$$

Such an assignment scheme is shown in Figure 4.6.

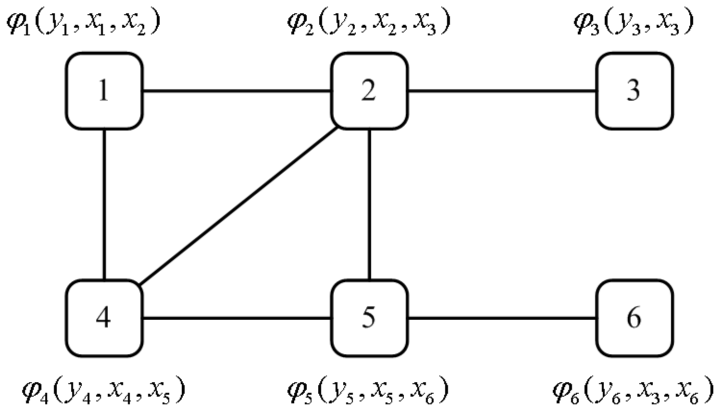
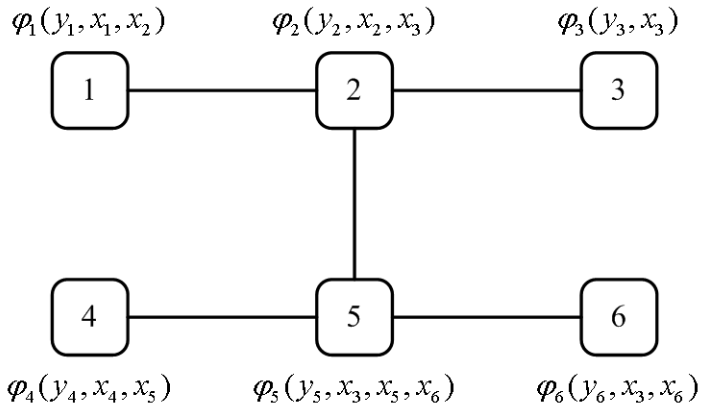


Fig. 4.6. Assignment of factors, scheme 2

The junction tree generated from this assignment scheme is shown in Figure 4.7. x_3 is added to local function φ_5 to satisfy the running intersection requirement.



$\varphi_1(y_1, x_1, x_2)$
$\varphi_2(y_2, x_2, x_3)$
$\varphi_3(y_3, x_3)$
1
2
3
4
5
6
$\varphi_4(y_4, x_4, x_5)$
$\varphi_5(y_5, x_3, x_5, x_6)$
$\varphi_6(y_6, x_3, x_6)$

Fig. 4.7. Junction tree for assignment scheme 2

The junction trees shown in Figure 4.3 and Figure 4.7 have the same topology but the clusters are different. As a consequence, the size of the messages and the complexity of message computation are different. Figure 4.8 demonstrates the message passing for the assignment scheme 2. It can be seen that Figure 4.8 has fewer large messages than Figure 4.4, i.e., only 2 with 2 arguments instead of 4. Therefore, assignment scheme 2 consumes less transmission power for the inference.

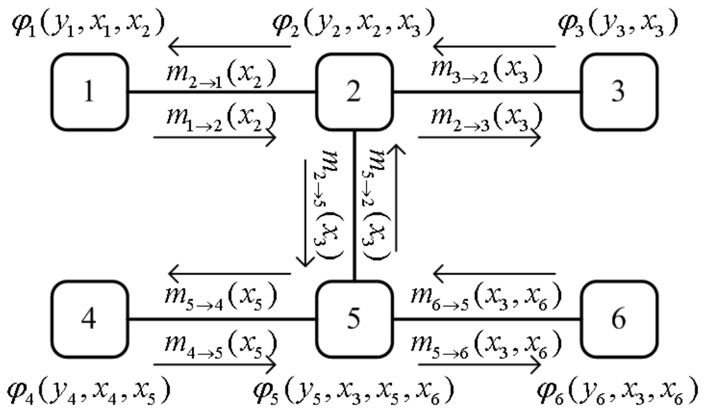


Fig. 4.8. Message passing for assignment scheme 2

Figure 4.9 illustrates the complexity of the message computation. Let us assume the each variable x_i is L-ary, i.e., it can take L possible values. From Figure 4.5, we see that the total number of summations needed for the message computation is approximately $4L^2 + 3L$ (4 messages are calculated by summing over 2 variables and 3 messages are calculated by summing over 1 variable). The total number of summations needed for the message computation in assignment scheme 2 is approximately $2L^2 + 6L$. So assignment scheme 2 also saves power for the message computation.

4.1.4.3 Centralized inference

In centralized inference, we assume that all the observations are transmitted to a fusion center. Let us suppose that node 2 is chosen to be the fusion center. Then all the other nodes transmit their local observations to node 2. Figure 4.10 illustrates the data flow.

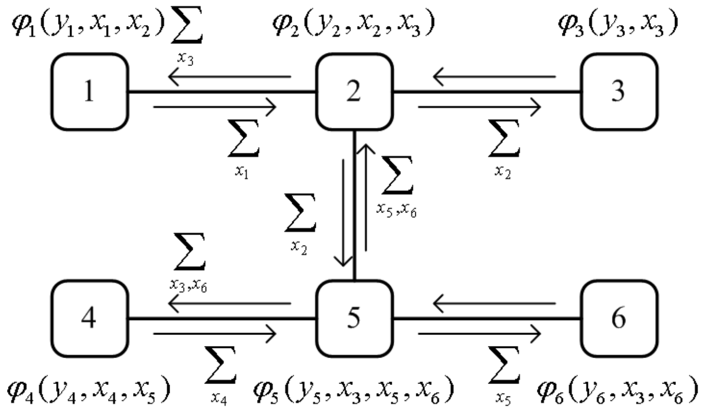


Fig. 4.9. Complexity of inference based on assignment scheme 2

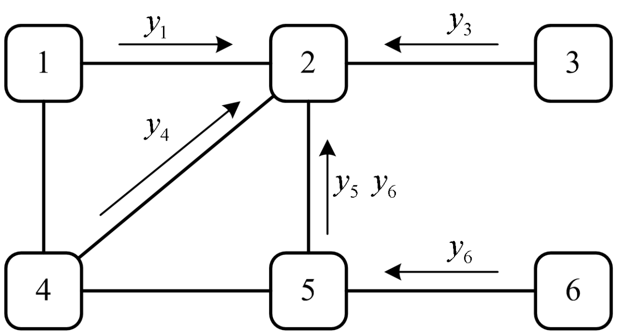


Fig. 4.10. Data transmission to the fusion center

Then in node 2 we can construct a factor graph based on the factorization in (4.7). The resulting factor graph is depicted in Figure 4.11. We can also solve the problem with a junction tree. However, factor graph and junction tree make no difference for the belief propagation that runs on them. We use factor graph for the sake of a better representation. Note that the whole graph of Figure 4.11 resides in a single node, i.e., node 2. The communication links do not influence the choice of the graphical model anymore. Therefore, although there is no direct communication between nodes 3 and 6, it does not prevent us from generating a probabilistic model where x_3 and x_6 are directly connected, which represents the local correlation between them.

Marginal probability distributions can be computed based on the sum-product algorithm. Figure 4.12 illustrates the message passing for the sum-product algorithm. Messages are computed by using (3.7) and (3.6) and the marginal probability distributions are computed by using (3.8).

Figure 4.13 illustrates the complexity of the message computation as before.

Comparing Figure 4.13 with Figure 4.5 and Figure 4.9, we can see that centralized inference requires only approximately $10L$ sum operations. It has the lowest computational complexity. However, such a model can be established only if all the data are available at a single node, which requires nodes to send the local measurements.

At the end of a centralized inference, the estimation results should be sent to the corresponding nodes. Figure 4.14 illustrates the traffic for transmitting the posterior probabil-

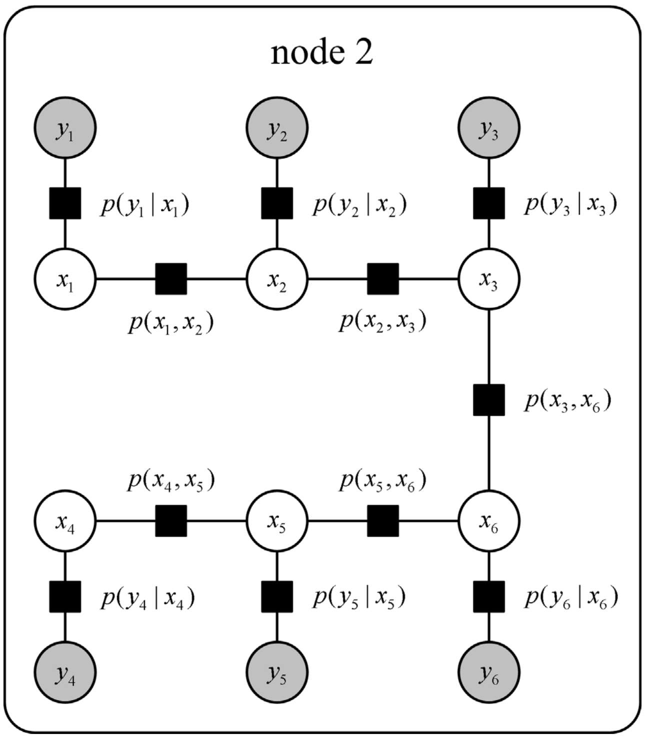


Fig. 4.11. Factor graph for centralized inference

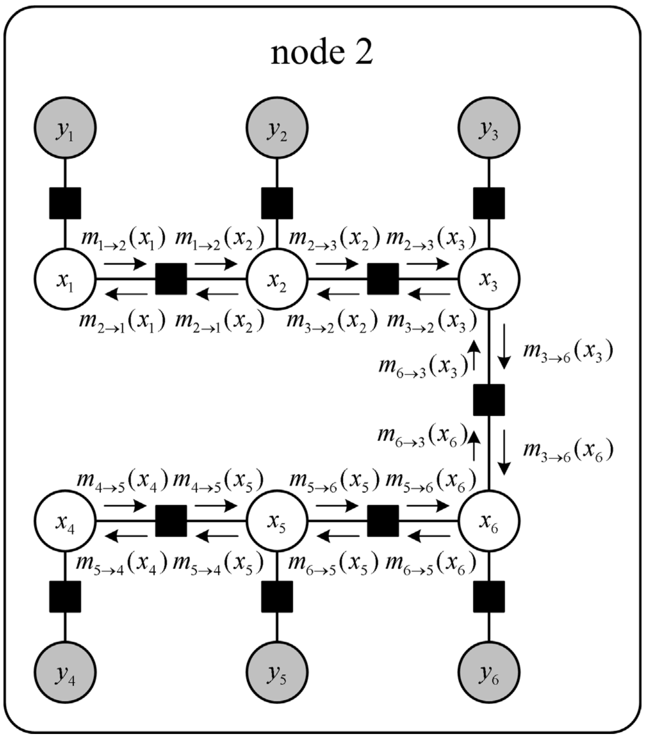


Fig. 4.12. Message passing for centralized inference

ity distributions back to each node, where each of them is represented by a vector with L entries. As a result, node 2 has to transmit $5L$ values in total.

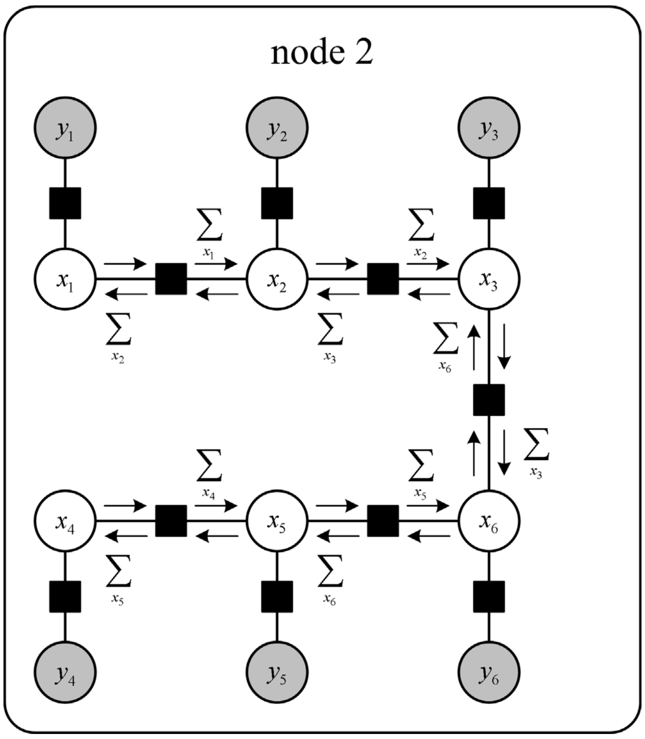


Fig. 4.13. Complexity of centralized inference

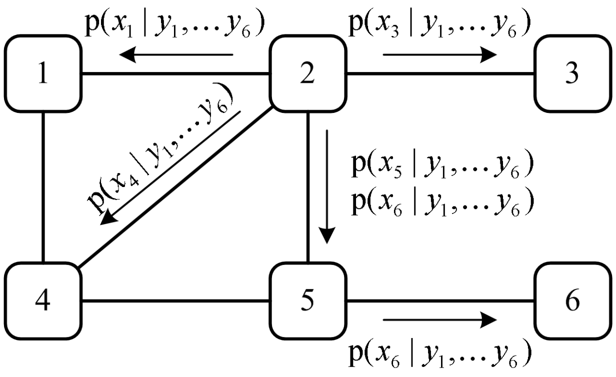


Fig. 4.14. Transmission of the final results

4.1.4.4 Comparison and discussion

We use Table 4.1 to compare the transmission cost and the computational complexity required by distributed inference and centralized inference. In this table, we use M_o to denote the number of bits that are needed in average to transmit the observations made at one single node, i.e., y_i and use M_p^n to represents the number of bits that are needed in average to transmit probability distributions or messages of dimension n (i.e., with n arguments) in message passing.

Table 4.1. Comparison of distributed and centralized inference

	Transmission			Computation		
	DI 1[1]	DI 2[2]	CI[3]	DI 1	DI 2	CI
node 1	M_p	M_p	M_o		L	
node 2	$2M_\mathsf{p} + M_\mathsf{p}^2$	$3M_\mathsf{p}$	$5M_\mathsf{p}$	$L + 2L^2$	$3L$	$10L$
node 3	M_p^2	M_p	M_o			
node 4	M_p	M_p	M_o	L	L	
node 5	$2M_\mathsf{p} + M_\mathsf{p}^2$	$2M_\mathsf{p} + M_\mathsf{p}^2$	$2M_\mathsf{o} + M_\mathsf{p}$	$L + 2L^2$	$L + 2L^2$	
node 6	M_p^2	M_p^2	M_o			
in total	$6M_\mathsf{p} + 4M_\mathsf{p}^2$	$8M_\mathsf{p} + 2M_\mathsf{p}^2$	$6M_\mathsf{o} + 6M_\mathsf{p}$	$3L + 4L^2$	$6L + 2L^2$	$10L$

[1] distributed inference assignment 1

[2] distributed inference assignment 2

[3] centralized inference

From this example, we can see that the topology of the communication network may put extra constraint on the inference. As a result, the entire computational complexity of a distributed inference method is higher than the complexity of a centralized inference method. However, if $M_\mathsf{o} \gg M_\mathsf{p}$, which is a common situation for many applications, transmission of measurement data to the fusion center would be too expensive. On the other hand, transmitting measurements to a fusion center usually involves multihop transmission. Multihop transmission consumes a lot of power because a lot of data have to be received and forwarded by the intermediate nodes. For example in Figure 4.10, node 5 has to help node 6 transmit its data y_6 to the fusion center node 2. Using distributed inference, we can avoid transmission of large amount of data to the fusion center and the total computation is distributed so that each node only need to execute a small amount of computations. By doing that, we also avoid long distance communications, which further reduces power consumption. To reduce the complexity, we can also introduce approximations to reduce L. Approximate inference will be discussed in the next section.

4.2 Function and message approximation in inference

When we introduced inference algorithms in the previous chapters, we have restricted our discussion to graphical models with discrete random variables. Another challenge arises when variables are specified by continuous, non-Gaussian distributions. For discrete random variables, marginal of a certain variable is obtained by summing out other variables. For continuous variable, calculation of marginals is done by integration. Therefore, the inference equations presented in the introduction of inference algorithms in the previous sections should be modified so that they are adapted to continuous random variables. This is done by replacing the summations with integrations. For example, the updating equations for the sum-product algorithm with continuous variables look as follows:

1) Message from a variable node i to a function node a:

$$m_{x_i \to f_a}(x_i) := \prod_{c \in NE(i) \backslash a} m_{f_c \to x_i}(x_i) \tag{4.12}$$

2) Message from a function node a to a variable node i:

$$m_{f_a \to x_i}(x_i) := \int_{\mathbf{x}_a \backslash x_i} f_a(\mathbf{x}_a) \prod_{j \in NE(a) \backslash i} m_{x_j \to f_a}(x_j) \tag{4.13}$$

Although we have used the same notations of functions and messages in (3.7) and (4.13), they have different forms. Variables in (3.7) are discrete, so the messages and functions are vectors or matrices containing the probabilities of the discrete values, whereas in (4.13), variables and functions are continuous.

If all the functions and messages in (4.12) and (4.13) are Gaussian, the calculation is simple since either the multiplication of several Gaussian functions or the integration of multivariate Gaussian distribution results in a new Gaussian function. The calculation of the mean and the variance of the new Gaussian distribution is straightforward. Some important results are shown in Appendix 4.A.1 and 4.A.2. However, in many applications with complicated continuous non-Gaussian functions and distributions, no closed form solution exists for the calculation in (4.12) and (4.13), which makes exact inference intractable. Thus, it is necessary to develop appropriate approximations to the functions and messages in a graphical model which not only provide a convenient means of representation but also make the computation of message updating in (4.12) and (4.13) simple. Particularly, it is always convenient to express a complicated function as a linear combination of basis functions. I.e., For a given function $f(x)$, we would like to approximate it by:

$$f(x) \approx f_{\mathtt{A}}(x) = \sum_i \alpha_i h_i(x) \tag{4.14}$$

where $h_i(x)$ is the i^{th} basis function, α_i is the associated weight. If an appropriate set of basis functions are chosen, we obtain a more compact representation of the original function. Such a technique has been used in information source coding or image processing [2, 13] to obtain a high compression rate. Based on this idea, we developed probability density approximation methods where the density functions are approximated with carefully chosen basis functions which have the following properties:

- original density functions are well approximated by a limited number of basis functions and their weights
- necessary operations on the basis functions (e.g., multiplication and integration of basis functions) are well defined and simple
- the specific structure of the original function is exploited

Discretization is the simplest method for the approximation of intractable continuous-valued inference problems. For example, we can uniformly sample a continuous function $f(x)$ and approximate it by:

$$f_{\mathtt{D}}(x) = \sum_{l=1}^{L} f(x_l^{\mathtt{D}})\delta(x - x_l^{\mathtt{D}}) \tag{4.15}$$

where the sub- or superscript $\mathtt{D}$ indicates that this is an approximation of a continuous function with a ”discrete” function, L is the total number of discrete samples, $\{x_l^{\mathtt{D}}\}_{l=1}^{L}$ are the sampling points, $f(x_l^{\mathtt{D}})$ calculates the value of the original function at the sample point $x_l^{\mathtt{D}}$ and the unit impulse function $\delta(x)$ is given by:

$$\delta(x) = \begin{cases} 1 & \mathtt{if} \quad x = 0 \\ 0 & \mathtt{if} \quad x \neq 0 \end{cases} \tag{4.16}$$

Some discretization-based approximate inference algorithms have produced satisfactory results in practical applications [92]. The computational complexity of a discretization-based inference algorithm is determined by the number of discrete values involved in each operation (e.g., multiplication, summation), e.g., L in (4.15). For high dimensional variables, exhaustive discretization of the entire state space results in a large number of discrete values which makes the computation in the standard inference algorithms intractable. Sophisticated approximation methods need to be developed in order to reduce the computational complexity. In this section, we will introduce two function approximation approaches that are suitable for the belief propagation algorithm. We first introduce the classical Fourier density approximation method and derive the Fourier domain belief propagation algorithm. Then we introduce the well known non-parametric belief propagation algorithm, which is based on the Monte Carlo sampling methods.

4.2.1 Fourier domain belief propagation

In this section, we introduce function approximation based on the Fourier transform. The Fourier transform was chosen to be the first candidate because it is already well studied and easy to implement.

Using the Fourier transform for function approximation has been developed decades ago [61]. Recent research refined the previous work so that it is suitable for the approximation of density functions and used it in practical applications. Some new concepts of Fourier density approximation for Bayesian inference have been developed in [11] and [12]. Here we summarize their work and derive Fourier approximation for the message representation in belief propagation or the sum-product algorithm.

4.2.1.1 Fourier series representation of functions

The basic idea of the Fourier function approximation method is to approximate a given function by a truncated Fourier series. Such an approximation is not suitable for density function approximation as a truncated Fourier series does not guarantee the non-negativity required by a density function. To overcome this problem, authors of [11] and

[12] approximate the square root of the original density function by its Fourier expansion. The approximation of the original function is then obtained by taking the square of the truncated Fourier series, which is guaranteed to be non-negative by construction.

Without loss of generality, we assume a density function $p(x)$ where x takes value in a finite interval $[-\pi, \pi]$ (For an arbitrary random variable, we first restrict its possible value in a finite interval, then project it to $x \in [-\pi, \pi]$). We use $p_{\texttt{FDA}}(x)$ to denote the Fourier density approximation (FDA) of $p(x)$. The authors of [11] proposed an optimal choice of $p_{\texttt{FDA}}(x)$ in the sense of maximizing the similarity of $p(x)$ and $p_{\texttt{FDA}}(x)$, measured by the Hellinger metric:

$$C(p(x), p_{\texttt{FDA}}(x)) = \int_{-\pi}^{\pi} \left(\sqrt{p(x)} - \sqrt{p_{\texttt{FDA}}(x)}\right)^2 dx \tag{4.17}$$

To guarantee the non-negativity of $p_{\texttt{FDA}}(x)$, we restrict our choice of $p_{\texttt{FDA}}(x)$ to the ones that can be expressed as the square of a function, which can itself be approximated by a Fourier series, i.e.,:

$$p_{\texttt{FDA}}(x) = \psi(x)\psi^*(x) = |\psi(x)|^2 \tag{4.18}$$

where the operator $*$ calculates complex conjugate and the N^{th} order Fourier series $\psi(x)$ is given by:

$$\psi(x) = \sum_{k=-N}^{N} c_k e^{jkx} \tag{4.19}$$

where j is the imaginary unit, i.e., $j^2 = -1$ and c_k is the Fourier coefficient for the k^{th} order component.

Inserting (4.19) and (4.18) into (4.17) and solving for the coefficients $\{c_k\}_{k=-N}^{N}$ which minimize the Hellinger metric, as shown in [11], we obtain:

$$c_k = \frac{1}{2\pi} \int_{-\pi}^{\pi} \sqrt{p(x)} e^{-jkx} dx \tag{4.20}$$

It can be seen from (4.20) that the coefficients can be determined independently. Calculation of coefficients can be implemented in a very efficient way, e.g., using fast Fourier transform (FFT).

Inserting (4.19) into (4.18), $p_{\texttt{FDA}}(x)$ can be expressed by:

$$\begin{aligned} p_{\texttt{FDA}}(x) &= \left(\sum_{k_1=-N}^{N} c_{k_1} e^{jk_1x}\right)\left(\sum_{k_2=-N}^{N} c_{k_2} e^{jk_2x}\right)^* \\ &= \left(\sum_{k_1=-N}^{N} c_{k_1} e^{jk_1x}\right)\left(\sum_{k_2=-N}^{N} c_{k_2}^* e^{-jk_2x}\right) \\ &= \sum_{k_1=-N}^{N} \sum_{k_2=-N}^{N} c_{k_1} c_{k_2}^* e^{j(k_1-k_2)x} \\ &= \sum_{l=-2N}^{2N} \sum_{k_2=-N}^{N} \bar{c}_{l+k_2} c_{k_2}^* e^{jlx} \end{aligned} \tag{4.21}$$

where $\bar{c}_{l+k_2}$ is given by:

$$\bar{c}_{l+k_2} = \begin{cases} c_{l+k_2} & \texttt{if} -N \leq l + k_2 \leq N \\ 0 & \texttt{otherwise} \end{cases} \tag{4.22}$$

Let us define:

$$\gamma_l = \sum_{k_2=-N}^{N} \bar{c}_{l+k_2} c^*_{k_2} \tag{4.23}$$

Then we can rewrite (4.21) as follows:

$$p_{\text{FDA}}(x) = \sum_{l=-2N}^{2N} \gamma_l e^{jlx} \tag{4.24}$$

Until now, we have found a Fourier series approximation $p_{\text{FDA}}(x)$ for a given function $p(x)$. If we use Fourier series to approximate the functions and messages in (4.12) and (4.13), we need to calculate the product of Fourier series approximations and the integration of Fourier series approximation. In the following sections, we will discuss these operations in detail.

4.2.1.2 Product of Fourier series

Let us assume we have two probability density function $p^a(x)$ and $p^b(x)$. The Fourier series approximation of these two density functions are given by:

$$p^a_{\text{FDA}}(x) = \sum_{l^a_x=-N^a_x}^{N^a_x} \gamma^a_{l^a_x} e^{jl^a_x x} \tag{4.25}$$

and

$$p^b_{\text{FDA}}(x) = \sum_{l^b_x=-N^b_x}^{N^b_x} \gamma^b_{l^b_x} e^{jl^b_x x} \tag{4.26}$$

The product of these two density functions can be expressed as:

$$\begin{aligned} p^c_{\text{FDA}}(x) &= p^a_{\text{FDA}}(x) p^b_{\text{FDA}}(x) \\ &= \sum_{l^a_x=-N^a_x}^{N^a_x} \gamma^a_{l^a_x} e^{jl^a_x x} \sum_{l^b_x=-N^b_x}^{N^b_x} \gamma^b_{l^b_x} e^{jl^b_x x} \\ &= \sum_{l^a_x=-N^a_x}^{N^a_x} \sum_{l^b_x=-N^b_x}^{N^b_x} \gamma^a_{l^a_x} \gamma^b_{l^b_x} e^{j(l^a_x+l^b_x)x} \end{aligned} \tag{4.27}$$

Let us define $N^c_x = N^a_x + N^b_x$ and rewrite (4.27) as follows:

$$p^c_{\text{FDA}}(x) = \sum_{l^c_x=-N^c_x}^{N^c_x} \sum_{l^b_x=-N^b_x}^{N^b_x} \bar{\gamma}^a_{l^c_x - l^b_x} \gamma^b_{l^b_x} e^{jl^c_x x} \tag{4.28}$$

where:

$$\bar{\gamma}^a_{l^c_x - l^b_x} = \begin{cases} \gamma^a_{l^c_x - l^b_x} & \texttt{if} - N^a_x \le l^c_x - l^b_x \le N^a_x \\ 0 & \texttt{otherwise} \end{cases} \tag{4.29}$$

Defining:

$$\gamma^c_{l^c_x} = \sum_{l^b_x = -N^b_x}^{N^b_x} \bar{\gamma}^a_{l^c_x - l^b_x} \gamma^b_{l^b_x} \tag{4.30}$$

we can express $p^c_{\mathrm{FDA}}(x)$ as:

$$p^c_{\mathrm{FDA}}(x) = \sum_{l^c_x = -N^c_x}^{N^c_x} \gamma^c_{l^c_x} e^{j l^c_x x} \tag{4.31}$$

After the multiplication, the new Fourier series approximation has an order of $N^c_x = N^a_x + N^b_x$ which indicates that the number of Fourier components grows very fast in the result of the product. Measures have to be taken to avoid the explosion of Fourier series after multiplications.

4.2.1.3 Integration of Fourier series

It can be observed from (4.13) that in the inference methods like junction tree or sum-product algorithm, we need to calculate the integral that takes the following form:

$$p^c(y) = \int_{-\pi}^{\pi} p^b(x, y) p^a(x) dx \tag{4.32}$$

Let us approximate $p^a(x)$ by:

$$p^a_{\mathrm{FDA}}(x) = \sum_{l^a_x = -N^a_x}^{N^a_x} \gamma^a_{l^a_x} e^{j l^a_x x} \tag{4.33}$$

and $p^b(x, y)$ by:

$$p^b_{\mathrm{FDA}}(x, y) = \sum_{l^b_x = -N^b_x}^{N^b_x} \sum_{l_y = -N_y}^{N_y} \gamma^b_{l^b_x, l_y} e^{j(l^b_x x + l_y y)} \tag{4.34}$$

Then the integration is calculated by:

$$\begin{aligned} p^c_{\mathrm{FDA}}(y) &= \int_{-\pi}^{\pi} \sum_{l^b_x = -N^b_x}^{N^b_x} \sum_{l_y = -N_y}^{N_y} \gamma^b_{l^b_x, l_y} e^{j(l^b_x x + l_y y)} \sum_{l^a_x = -N^a_x}^{N^a_x} \gamma^a_{l^a_x} e^{j l^a_x x} dx \\ &= \sum_{l_y = -N_y}^{N_y} \left(\int_{-\pi}^{\pi} \sum_{l^b_x = -N^b_x}^{N^b_x} \gamma^b_{l^b_x, l_y} e^{j l^b_x x} \sum_{l^a_x = -N^a_x}^{N^a_x} \gamma^a_{l^a_x} e^{j l^a_x x} dx \right) e^{j l_y y} \end{aligned} \tag{4.35}$$

Let us define:

$$\begin{aligned}\gamma^c_{l_y} &= \int_{-\pi}^{\pi} \sum_{l_y=-N^b_x}^{N^b_x} \gamma^b_{l^b_x,l_y} e^{jl^b_x x} \sum_{l^a_x=-N^a_x}^{N^a_x} \gamma^a_{l^a_x} e^{jl^a_x x} dx \\ &= \sum_{l^b_x=-N^b_x}^{N^b_x} \sum_{l^a_x=-N^a_x}^{N^a_x} \int_{-\pi}^{\pi} \gamma^b_{l^b_x,l_y} \gamma^a_{l^a_x} e^{j(l^b_x+l^a_x)x} dx \end{aligned} \tag{4.36}$$

Since $\int_{-\pi}^{\pi} e^{jlx} dx = 0$ for $l \neq 0$, (4.36) can be simplified to:

$$\begin{aligned}\gamma^c_{l_y} &= \sum_{l^c_x=-N^c_x}^{N^c_x} \int_{-\pi}^{\pi} \gamma^b_{l^c_x,l_y} \gamma^a_{-l^c_x} dx \\ &= 2\pi \sum_{l^c_x=-N^c_x}^{N^c_x} \gamma^b_{l^c_x,l_y} \gamma^a_{-l^c_x} \end{aligned} \tag{4.37}$$

where

$$N^c_x = \mathsf{min}(N^a_x, N^b_x) \tag{4.38}$$

Then, (4.35) can be rewritten as:

$$p^c_{\texttt{FDA}}(y) = \sum_{l_y=-N_y}^{N_y} \gamma^c_{l_y} e^{jl_y y} \tag{4.39}$$

which gives the Fourier series expression for $p^c_{\texttt{FDA}}(y)$.

4.2.1.4 Components reduction

For many density mixture approximation approaches like Fourier series, Gaussian mixture, Dirac mixture or Monte Carlo methods, the number of coefficients increases exponentially after the product operation. Keeping all coefficients is practically impossible. Determining how many coefficients and which ones are needed is challenging. [37] provides a progressive way to calculate the parameters of mixture densities optimally. But the computational requirement is relatively high.

Components reduction for a Fourier series approximation is done as follows. Let us assume a Fourier density:

$$p_{\texttt{FDA}}(x) = \sum_{l=-2N}^{2N} \gamma_l e^{jlx} \tag{4.40}$$

Suppose we want to reduce the number of components in the Fourier series approximation. We can do this by eliminating the components with smallest coefficients. However, the resulting density function may have negative values. As discussed before, we should operate on the square root of $p_{\texttt{FDA}}(x)$ to guarantee the non-negativity of the new density function. Let us use $\psi(x)$ to denote the square root of $p_{\texttt{FDA}}(x)$, which can be expressed as:

$$\psi(x) = \sum_{k=-N}^{N} c_k e^{jkx} \tag{4.41}$$

The relationship between $\{c_k\}_{k=-N}^{N}$ and $\{\gamma_l\}_{l=-2N}^{2N}$ is presented in (4.23). Having the values of $\{\gamma_l\}_{l=-2N}^{2N}$, we can use (4.23) (for all $l = -2N, \ldots 2N$) to solve for $\{c_k\}_{k=-N}^{N}$. Now let us eliminate components with small coefficients in the Fourier series of $\psi(x)$ and express the obtained function by:

$$\psi'(x) = \sum_{k \in S} c_k e^{jkx} \tag{4.42}$$

where $S \subset \{-N, \ldots N\}$ stores the indices of the components that are kept. Taking the square of $\psi'(x)$, we obtain a density function with reduced components:

$$p'_{\texttt{FDA}}(x) = |\psi'(x)|^2 = \sum_{l \in U} \gamma'_l e^{jlx} \tag{4.43}$$

where $U = \{k_1 - k_2 | k_1 \in S, k_2 \in S\}$ and γ'_l is given by:

$$\gamma'_l = \sum_{k \in S} \bar{c}_{l+k} c_k^* \tag{4.44}$$

where $\bar{c}_{l+k}$ is given by:

$$\bar{c}_{l+k} = \begin{cases} c_{l+k} & \texttt{if } l+k \in S \\ 0 & \texttt{otherwise} \end{cases} \tag{4.45}$$

Since S is a subset of $\{-n, \ldots n\}$, the size of U is smaller than the number of Fourier components in the original density function, i.e., $p_{\texttt{FDA}}(x)$. For density functions with specific structure, the final number of Fourier components that are used for density approximation can be greatly reduced.

4.2.1.5 Fourier domain belief propagation

Previous sections discussed basic operations on Fourier densities that are needed for the computations in belief propagation. Based on those results, we can develop a Fourier domain belief propagation algorithm as a result of this thesis where all complicated functions and messages are represented by their Fourier series approximation. Let us use the subgraph in Figure 4.15 as an example to illustrate the algorithm. In this example, variable node x_0 is connected to N_u function nodes, i.e., $f_0, \ldots f_{N_u-1}$. Function node f_0 is connected to N_v variable nodes, i.e., $x_0, \ldots x_{N_v-1}$.

We first look at the message calculation at variable node x_0. Suppose it received messages $\{m_{f_i \to x_0}(x_0)\}_{i=1}^{N_u-1}$ from all its neighbors except for f_0. Each message $m_{f_i \to x_0}(x_0)$ is represented by an M_i components Fourier series, which can be expressed as:

$$m_{f_i \to x_0}(x_0) = \sum_{l_i \in U_i} \gamma_{l_i} e^{jl_i x_0} \tag{4.46}$$

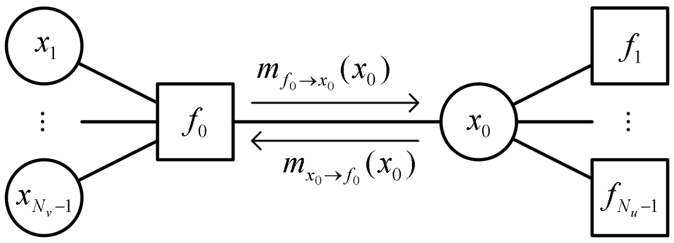


Fig. 4.15. A FDBP example

where U_i contains the orders of the M_i valid Fourier components and γ_{l_i} is the Fourier coefficient.

According to (4.12), message $m_{x_0\to f_0}(x_0)$ is calculated by:

$$m_{x_0\to f_0}(x_0) = \prod_{i=1}^{N_u-1} m_{f_i\to x_0}(x_0) \tag{4.47}$$

Algorithm 1 uses pseudo code to present the computation in (4.47).

Algorithm 1 FDBP: message propagation at variable node

```
Require: input message $\{m_{f_i\to x_0}(x_0) = \sum_{l_i\in U_i}\gamma_{l_i}e^{jl_ix_0}\}_{i=1}^{N_u-1}$
Ensure: output message $m_{x_0\to f_0}(x_0)$ from $x_0$ to $f_0$
1: initialize $g^{(0)}(x_0) = \sum_{l_0^{(0)}\in U_0^{(0)}}\gamma_{l_0^{(0)}}e^{jl_0^{(0)}x_0}$ where $U_0^{(0)} = \{0\}$ and $\gamma_{l_0}^{(0)} = 1$.
2: for $i = 1$ to $N_u - 1$ do
3:     update $g^{(i)}(x_0) = g^{(i-1)}(x_0)m_{f_i\to x_0}(x_0)$ which can be calculated by using the results in Section 4.2.1.2.
4:     reduce the number of components in $g^{(i)}(x_0)$ using the method introduced in Section 4.2.1.4
5: end for
6: assign $m_{x_0\to f_0}(x_0) \leftarrow g^{(N_u-1)}(x_0)$
```

Now let us take a look at the message calculation at a function node f_0. Suppose it received messages $\{m_{x_i\to f_0}(x_i)\}_{i=1}^{N_v-1}$ from all its neighbors except for x_0. Each message is given by:

$$m_{x_i\to f_0}(x_i) = \sum_{l_i\in V_i}\gamma_{l_i}e^{jl_ix_i} \tag{4.48}$$

According to (4.13), message $m_{f_0\to x_0}$ is calculated by:

$$m_{f_0\to x_0}(x_0) = \int_{x_1,\dots x_{N_v-1}} f_0(x_0, x_1, \dots x_{N_v-1}) \prod_{i=1}^{N_v-1} m_{x_j\to f_0}(x_i) \tag{4.49}$$

Pseudo code in Algorithm 2 presents the computation in (4.49).

Algorithm 2 FDBP: message propagation at function node

```
Require: input message $\{m_{x_i \to f_0}(x_i) = \sum_{l_i \in V_i} \gamma_{l_i} e^{j l_i x_i}\}_{i=1}^{N_v - 1}$ and local factor $f_0(x_0, x_1, \ldots x_{N_v - 1})$
Ensure: output message $m_{f_0 \to x_0}(x_0)$ from $f_0$ to $x_0$
1: initialize $g^{(N_v-1)}(x_0, x_1, \ldots x_{N_v-1}) = f_0(x_0, x_1, \ldots x_{N_v-1})$.
2: for $i = N_v - 1$ to 1 do
3:     compute $g^{(i-1)}(x_0, \ldots x_{i-1}) = \int_{x_i} g^{(i)}(x_0, \ldots x_i) m_{x_i \to f_0}(x_i)$ which can be calculated by using the results in Section 4.2.1.3.
4:     reduce the number of components in $g^{(i-1)}(x_0, \ldots x_{i-1})$ using the method introduced in Section 4.2.1.4
5: end for
6: assign $m_{f_0 \to x_0}(x_0) \leftarrow g^{(0)}(x_0)$
```

4.2.2 Non-parametric belief propagation

Monte Carlo sampling methods provide an alternative of approximate inference. Sampling methods are widely used for characterizing complex probability distributions. Lots of Monte Carlo sampling techniques, like importance sampling, Gibbs sampling, Markov chain Monte Carlo methods [66] are successfully used in statistics. The general idea is to draw samples, i.e., to generate realizations from given distributions. By carefully designing the sampling methods, the finite number of samples can represent the underlying distribution very well. Using sampling methods in belief propagation is a promising approach as samples could be packaged in the messages to deliver the information about the complex distributions or local functions where parameterization is not possible, which gives the name non-parametric belief propagation.

Using sampling in inference is not a new idea. Particle filters [60] are already widely used in signal processing where particles (samples with corresponding weights) propagate along the Markov chain which approximate the filtering distribution. The particle filter applies to arbitrary distributions but in Markov chains only. Standard belief propagation applies to general graphs, but the potential functions are either discrete or Gaussian. Non-parametric belief propagation or NBP [102] can be regarded as a combination of these two so that sampling method is applied to a general graph.

In non-parametric belief propagation, continuous functions are represented by random samples $\{\mu^{(1)}, \ldots \mu^{(M)}\}$ with associated weights $\{\omega^{(1)}, \ldots \omega^{(M)}\}$. The original function can be reconstructed by using a Gaussian mixture model:

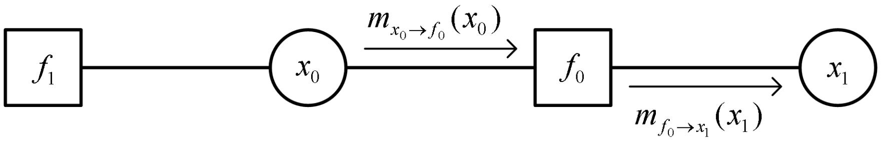
$$p_{\text{GM}}(x) = \sum_{i=1}^{M} \omega^{(i)} \mathcal{N}(x; \mu^{(i)}, \Lambda^{(i)}) \tag{4.50}$$

where the function $\mathcal{N}(x; \mu, \Lambda)$ denotes a Gaussian distribution of variable x with mean μ and variance Λ. In (4.50), the weights are normalized so that $\sum_{i=1}^{M} \omega^{(i)} = 1$. It can be seen that the Gaussian components are centered at the values of the random samples. $\Lambda^{(i)}$ is the width which controls smoothness of the reconstructed distribution function. Discussion of Gaussian mixture representation of data set or distribution function can be found in [100, 7].

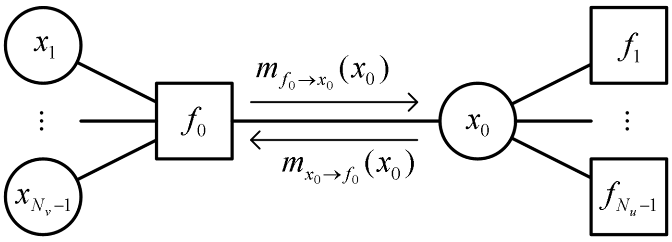
Using Gaussian mixture models, a message in belief propagation can be defined by a parameter vector $\boldsymbol{\theta} = \{\omega^{(i)}, \mu^{(i)}, \Lambda^{(i)}\}_{i=1}^{M}$, i.e., each message can be expressed by:

$$m(x; \boldsymbol{\theta}) = \sum_{i=1}^{M} \omega^{(i)} \mathcal{N}(x; \mu^{(i)}, \Lambda^{(i)}) \tag{4.51}$$

Message updating is based on (4.12) and (4.13). However, new messages are generated from stochastic process, i.e., sampling method. With different graph topologies, message computations are different. Figure 4.16 shows two typical sub-graph structures: line structure and star structure. The following sections discuss message update for each case.



(a) Line topology



(b) Star topology

Fig. 4.16. An NBP example

4.2.2.1 Message update for line structure

From (4.12), message updating at each variable node within a line structure (e.g., x_0 in Figure 4.16(a)) is simply copying the incoming message. For example, the message from node x_0 to node f_0, given the message $m_{f_1\to x_0}$, is calculated by:

$$m_{x_0\to f_0}(x_0) = m_{f_1\to x_0}(x_0) \tag{4.52}$$

At function node f_0, having received message $m_{x_0\to f_0}(x_0)$, the message from f_0 to x_1 is generated as described in Algorithm 3.

Sometimes, new message contains some samples with dominant weights whereas other samples have negligible weights. In this case, resampling techniques [17] are used to make all samples have comparable weights.

A Markov chain is composed of many line structures. The message update method presented above can be repeated to approximate the distribution of the hidden state variables

Algorithm 3 Message propagation at function node

```
Require: input message $m_{x_0 \to f_0}(x_0; \boldsymbol{\theta}_{x_0,f_0})$ where $\boldsymbol{\theta}_{x_0,f_0} = \{\omega^{(i)}_{x_0,f_0}, \mu^{(i)}_{x_0,f_0}, \Lambda^{(i)}_{x_0,f_0}\}_{i=1}^{M}$ and local
    potential function $f_0(x_0, x_1)$
Ensure: output message $m_{f_0 \to x_1}(x_1; \boldsymbol{\theta}_{f_0,x_1})$ from $f_0$ to $x_1$
 1: for $i = 1$ to $M$ do
 2:     draw a sample $x_0^{(i)}$ from $\mathcal{N}(x_0; \mu^{(i)}_{x_0,f_0}, \Lambda^{(i)}_{x_0,f_0})$
 3:     draw a sample $x_1^{(i)}$ from $f_0(x_0^{(i)}, x_1)$, i.e.: $x_1^{(i)} \sim f_0(x_0^{(i)}, x_1)$
 4:     assign a weight $\hat{\omega}_1^{(i)}$ to sample $x_1^{(i)}$, where the weight is calculated by: $\hat{\omega}_1^{(i)} = \omega^{(i)}_{x_0,f_0} \cdot$
        $\mathcal{N}(x_0^{(i)}; \mu^{(i)}_{x_0,f_0}, \Lambda^{(i)}_{x_0,f_0}) \cdot f_0(x_0^{(i)}, x_1^{(i)})$
 5: end for
 6: construct the message $m_{f_0 \to x_1}(x_1; \boldsymbol{\theta}_{f_0,x_1})$ where the parameter vector $\boldsymbol{\theta}_{f_0,x_1} =$
    $\{\omega^{(i)}_{f_0,x_1}, \mu^{(i)}_{f_0,x_1}, \Lambda^{(i)}_{f_0,x_1}\}_{i=1}^{M}$
 7: for $i = 1$ to $M$ do
 8:     normalize weights: $\omega_1^{(i)} = \frac{\hat{\omega}_1^{(i)}}{\sum_{i=1}^{M} \hat{\omega}_1^{(i)}}$
 9:     assign $\mu^{(i)}_{f_0,x_1} \leftarrow x_1^{(i)}$
10:     assign $\omega^{(i)}_{f_0,x_1} \leftarrow \omega_1^{(i)}$
11:     choose $\Lambda^{(i)}_{f_0,x_1}$ with any appropriate kernel bandwidth selection method [100]
12: end for
```

in a Markov chain. Such a method is known as particle filter. Many successful applications of particle filter can be found in the field of positioning [36], tracking [4], computer vision [71] and so on.

4.2.2.2 Message update for star structure

Message update in a star structure is more complicated because the central node has to combine messages from several neighboring nodes. Since each message is represented by a sum several values with corresponding weights, the product of the messages involves elementwise multiplications. The number of necessary multiplications and the size of the resulting message can be very large.

The following example can illustrate this. Assume in Figure 4.16(b), node x_0 received messages from nodes $f_1, \dots f_{N_u-1}$ and each message $m_{f_j \to x_0}(x_0; \boldsymbol{\theta}_{f_j,x_0})$ is composed of M random samples, i.e., $\boldsymbol{\theta}_{f_j,x_0} = \{\omega^{(i)}_{f_j,x_0}, \mu^{(i)}_{f_j,x_0}, \Lambda^{(i)}_{f_j,x_0}\}_{i=1}^{M}$. Each message $m_{f_j \to x_0}(x_0; \boldsymbol{\theta}_{f_j,x_0})$ can be expressed as follows:

$$m_{f_j \to x_0}(x_0; \boldsymbol{\theta}_{f_j,x_0}) = \sum_{i=1}^{M} \omega^{(i)}_{f_j,x_0} \cdot \mathcal{N}(x_0; \mu^{(i)}_{f_j,x_0}, \Lambda^{(i)}_{f_j,x_0}) \tag{4.53}$$

According to (4.12), the message that node x_0 sends to f_0 is produced from:

$$\begin{aligned} m_{x_0 \to f_0}(x_0; \boldsymbol{\theta}_{x_0,f_0}) &= \prod_{j=1}^{N_u-1} m_{f_j \to x_0}(x_0; \boldsymbol{\theta}_{f_j,x_0}) \\ &= \prod_{j=1}^{N_u-1} \left(\sum_{i=1}^{M} \omega_{f_j,x_0}^{(i)} \cdot \mathcal{N}(x_0; \mu_{f_j,x_0}^{(i)}, \Lambda_{f_j,x_0}^{(i)}) \right) \end{aligned} \tag{4.54}$$

Equation (4.54) produces M^{N_u-1} components, where each component is the product of $N_u - 1$ Gaussian functions. The calculation of each component is associated with $N_u - 1$ labels $\{\iota_j\}_{j=1}^{N_u-1}$ where each label $\iota_j \in \{1, \ldots M\}$ identifies a Gaussian component in the j^{th} message. The product of N_u-1 Gaussian distributions $\{\mathcal{N}(x; \mu_j^{\iota_j}, \Lambda_j^{\iota_j})\}_{j=1}^{N_u-1}$ with associated weights $\{\omega_j^{\iota_j}\}_{j=1}^{N_u-1}$ is still Gaussian, with mean and variance given by:

$$\begin{aligned} \prod_{j=1}^{N_u-1} \mathcal{N}(x; \mu_j^{\iota_j}, \Lambda_j^{\iota_j}) &\propto \mathcal{N}(x; \bar{\mu}, \bar{\Lambda}) \\ \bar{\Lambda}^{-1} &= \sum_{j=1}^{N_u-1} (\Lambda_j^{\iota_j})^{-1} \\ \bar{\Lambda}^{-1}\bar{\mu} &= \sum_{j=1}^{N_u-1} (\Lambda_j^{\iota_j})^{-1} \mu_j^{\iota_j} \end{aligned} \tag{4.55}$$

and the corresponding weight given by:

$$\bar{\omega} \propto \frac{\prod_{j=1}^{N_u-1} \omega_j^{\iota_j} \mathcal{N}(x; \mu_j^{\iota_j}, \Lambda_j^{\iota_j})}{\mathcal{N}(x; \bar{\mu}, \bar{\Lambda})} \tag{4.56}$$

The total calculation in (4.54) requires $\mathcal{O}(M^{N_u-1})$ operations and generates a Gaussian mixture with M^{N_u-1} components. An appropriate approximation method should be found to simplify the message calculation as well as control the size of the resulting message.

The non-parametric belief propagation method introduced in [102] uses Gibbs sampler [30] to draw M samples from the M^{N_u-1} component product of (4.54). In (4.54), direct calculation of the Gaussian products for all possible combination of labels $\{\iota_j\}_{j=1}^{N_u-1}$ is very complicated for large number M and $N_u - 1$. However, the conditional distribution of any single label ι_j is simple. Using Gibbs sampler, the sampling of a multivariate distribution is obtained by iteratively sampling individual variables while fixing the values of the others. At each iteration, the label of $\{\iota_k\}_{k=1\ (k \neq j)}^{N_u-1}$ are fixed and the label ι_j is sampled from its conditional distribution. At the next iteration, this label ι_j will be fixed and another input message will be picked and its label will be sampled and so on until all incoming Gaussian mixtures are sampled once. This whole procedure will be repeated for κ times to improve the sampling accuracy. The final result provides a Gaussian product identified by the final labels. A sample will then be drawn from this Gaussian product. To draw M samples like that, it takes $\mathcal{O}\{(N_u - 1)\kappa M^2\}$ operations.

Algorithm 4 Gibbs sampling for product of Gaussian mixtures

```
Require: input $N_u - 1$ Gaussian mixtures of $M$ components, denoted by
    $\{\omega_j^{(i)}, \mu_j^{(i)}, \Lambda_j^{(i)}\}_{i=1}^M, j \in \{1, \dots N\}$
Ensure: output $M$ samples $\{x^{(l)}\}_{l=1}^M$ and associated weights $\{\omega^{(l)}\}_{l=1}^M$ that represent the
    product of the input Gaussian mixtures
 1: for $l = 1$ to $M$ do
 2:   for $j = 1$ to $N_u - 1$ do
 3:     choose an initial label $\iota_j \in \{1, \dots M\}$
 4:   end for
 5:   for $t = 1$ to $\kappa$ do
 6:     for $j = 1$ to $N_u - 1$ do
 7:       calculate mean $\mu^*$ and variance $\Lambda^*$ for $\mathcal{N}(x; \mu^*, \Lambda^*) \propto \prod_{n \neq j} \mathcal{N}(x; \mu_n^{\iota_n}, \Lambda_n^{\iota_n})$
 8:       for $i = 1$ to $M$ do
 9:         calculate mean $\tilde{\mu}$ and variance $\tilde{\Lambda}$ for $\mathcal{N}(x; \tilde{\mu}, \tilde{\Lambda}) \propto \mathcal{N}(x; \mu^*, \Lambda^*) \cdot \mathcal{N}(x; \mu_j^{(i)}, \Lambda_j^{(i)})$
10:         calculate weight $\tilde{\omega}^{(i)}$ by $\tilde{\omega}^{(i)} = \omega_j^{(i)} \frac{\mathcal{N}(x^*; \mu^*, \Lambda^*) \cdot \mathcal{N}(x^*; , \mu_j^{(i)}, \Lambda_j^{(i)})}{\mathcal{N}(x^*; \tilde{\mu}, \tilde{\Lambda})}$ where $x^*$ can be cho-
            sen arbitarily
11:       end for
12:       sample a new label $\iota_j$ according to $p(\iota_j = i) = \tilde{\omega}^{(i)}$
13:     end for
14:   end for
15:   calculate mean $\hat{\mu}$ and variance $\hat{\Lambda}$ for $\mathcal{N}(x; \hat{\mu}, \hat{\Lambda}) \propto \prod_{j=1}^{N_u-1} \mathcal{N}(x; \mu_j^{\iota_j}, \Lambda_j^{\iota_j})$
16:   draw a sample $x^{(l)}$ from $\mathcal{N}(x; \hat{\mu}, \hat{\Lambda})$ and calculate its weight by: $\hat{\omega}^{(l)} = \mathcal{N}(x^{(l)}; \hat{\mu}, \hat{\Lambda})$
17: end for
18: normalize weights by: $\omega^{(l)} = \frac{\hat{\omega}^{(l)}}{\sum_{l=1}^M \hat{\omega}^{(l)}}$
```

Detailed explanation of the sampling procedure for the Gaussian mixtures multiplication was presented in [102]. Algorithm 4 uses pseudo code to illustrates this procedure.

Based on Algorithm 4, the message from x_0 to f_0 can be calculated. The calculation steps are summarized in Algorithm 5.

From (4.13), given messages $\mathbf{m}_{x_j \to f_0}(x_{v_j}; \boldsymbol{\theta}_{x_j, f_0})$ from nodes $x_1, \dots x_{N_v - 1}$ to node f_0, message from function node f_0 to variable node x_0 is produced from:

$$m_{f_0 \to x_0}(x_0; \boldsymbol{\theta}_{f_0, x_0})) = \int_{x_1, \dots x_{N_v-1}} f_0(x_0, x_1, \dots x_{N_v - 1}) \prod_{j=1}^{N_v - 1} m_{x_j \to f_0}(x_j; \boldsymbol{\theta}_{x_j, f_0}) \tag{4.57}$$

where $m_{x_j \to f_0}(x_j; \boldsymbol{\theta}_{x_j, f_0})$ is given by:

$$m_{x_j \to f_0}(x_j; \boldsymbol{\theta}_{x_j, f_0}) = \sum_{i=1}^{M} \omega_{x_j, f_0}^{(i)} \cdot \mathcal{N}(x_j; \mu_{x_j, f_0}^{(i)}, \Lambda_{x_j, f_0}^{(i)}) \tag{4.58}$$

It can be seen from (4.57) that in order to calculate a message from a function node to a variable node, the product of incoming messages should be combined with local function f_0. Local function will influence the weights of the samples.

Algorithm 5 NBP message update from variable node x_0 to function node f_0

```
Require: input messages $m_{f_j \to x_0}(x_0; \boldsymbol{\theta}_{f_j,x_0})$ with $\boldsymbol{\theta}_{f_j,x_0} = \{\omega^{(i)}_{f_j,x_0}, \mu^{(i)}_{f_j,x_0}, \Lambda^{(i)}_{f_j,x_0}\}_{i=1}^{M}$, $j = 1, \dots N_u - 1$ from $N_u - 1$ neighboring nodes
Ensure: output message $m_{x_0 \to f_0}(x_0; \boldsymbol{\theta}_{x_0,f_0})$ with $\boldsymbol{\theta}_{x_0,f_0} = \{\mu^{(i)}_{x_0,f_0}, \Lambda^{(i)}_{x_0,f_0}, \omega^{(i)}_{x_0,f_0}\}_{i=1}^{M}$ from variable node $x_0$ to function node $f_0$
1: generate $M$ samples $\{x_0^{(l)}\}_{l=1}^{M}$ from $g(x_0) = \prod_{j=1}^{N_u-1} m_{f_j \to x_0}(x_0; \boldsymbol{\theta}_{f_j,x_0})$ and calculate the associated weights $\{\omega_0^{(l)}\}_{l=1}^{M}$ using Algorithm 4
2: construct the message $m_{x_0 \to f_0}(x_0; \boldsymbol{\theta}_{x_0,f_0})$ with $\boldsymbol{\theta}_{x_0,f_0} = \{\mu^{(i)}_{x_0,f_0}, \Lambda^{(i)}_{x_0,f_0}, \omega^{(i)}_{x_0,f_0}\}_{i=1}^{M}$
3: for $i = 1$ to $M$ do
4:     assign $\mu^{(i)}_{x_0,f_0} \leftarrow x_0^{(i)}$
5:     assign $\omega^{(i)}_{x_0,f_0} \leftarrow \omega_0^{(i)}$
6:     choose $\Lambda^{(i)}_{x_0,f_0}$ with any appropriate kernel bandwidth selection method [100]
7: end for
```

A simple case was considered in [102] where each function node is connected to at most two variable nodes, as shown in Figure 4.17. In this case, only pairwise potential functions are considered. In this case, there is alway a line structure at the function node. Message update at a function node is given by Algorithm 3.

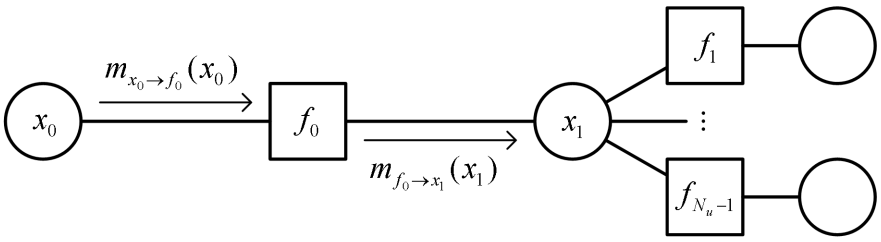


Fig. 4.17. Subgraph with simpler structure

4.3 State estimation for dynamical systems

Dynamical systems are mathematical representations of processes in which values of the state variables change over time. In addition usually, different state variables are correlated in space. Many practical problems can be modeled by a dynamical system, e.g., tracking, process monitoring etc. State estimation in simple dynamical systems can be achieved by Kalman filter or particle filter. However, in some networked systems, processes at different locations are coupled with each other, resulting in a very complicated dynamical system. And the state estimation in this case is a non-trivial task. Probabilistic models can be used to model the time dependence and the space correlation. This not only provides a clear representation of the entire system, but also offers a framework for efficient state estimation in dynamical systems. This section formulates a general probabilistic model for typical dynamical systems and presents general inference methods for the state estimation.

4.3.1 Problem formulation

We describe the dynamics involved in a system by state processes and observation processes that are discretized in time. The state of the system at time instance t is represented by state vector $\mathbf{x}^{(t)}$ which contains M state variables, i.e., $\mathbf{x}^{(t)} = \left[x_1^{(t)}, \ldots x_M^{(t)}\right]^{\mathrm{T}}$. The initial state is given by $\mathbf{x}^{(0)} = \left[x_1^{(0)}, \ldots x_M^{(0)}\right]^{\mathrm{T}}$. At each time instance, N measurements are made. We use vector $\mathbf{y}^{(t)} = \left[y_1^{(t)}, \ldots y_N^{(t)}\right]^{\mathrm{T}}$ to denote the observations. The first observation is made at time instance $t = 1$. And we use $\mathbf{Y}^{(1:T)} = \left[\mathbf{y}^{(1)}, \ldots \mathbf{y}^{(T)}\right]$ to denote all the observations made until time instance T. The most interesting inference problem in dynamical system is the optimal estimation of state variables $\mathbf{x}^{(t)}$ based on the observations made till now, i.e., $\mathbf{Y}^{(1:T)}$, which can be posed as a maximum a posteriori problem, i.e., $\max p(\mathbf{x}^{(t)} \mid \mathbf{Y}^{(1:T)})$. If $t = T$, this is a filtering problem; if $t < T$, this is a smoothing problem and if $t > T$, this is a prediction problem.

4.3.2 Graphical model for dynamical systems

A dynamic Bayesian network [31] (DBN) is a powerful tool for representing the probabilistic model of dynamical systems. A dynamic Bayesian network represents the whole system in interconnected time slices. Each time slice t represents the relationship between hidden variables $\mathbf{x}^{(t)}$ and observations $\mathbf{y}^{(t)}$. Edges from the variable nodes in one time slice to the variable nodes in another time slice reveal the time dependence of the system. Typically, a dynamic Bayesian network represents a process that is stationary and Markovian. In a stationary process, the relationships of the variables in time slice t and the transition function between time slice t and slice $t+1$ do not depend on the choice of t. With Markovian property, the transition function depends only on the immediately-preceding time slice, i.e., there is no edge between slice t to slice $t+n$ for any $t \geq 1$ and $n > 1$. We further assume that observations at time slice t only depend on the state variables in the same time slice. As a consequence, edges between neighboring slices only connect hidden state variables. Therefore, it is sufficient to represent the whole process using a simplified representation which contains the subgraph from time slice $t-1$ to t and the initial time slice 0 only. Such a representation is called two-slice temporal Bayes net (2TBN). Dynamic Bayesian network can be used as a general tool to describe a wide range of dynamical systems. For specific problems, we can use other models to represent the dynamics. For example for linear dynamical systems, we typically use state-space models to describe the systems. Although the state-space model is the most well known representation of the dynamical systems. It can represent only a subset of all possible dynamical systems. The property and the limitation of the state-space model will be discussed later.

With the Markovian assumption, the entire dynamic Bayesian network represents the following probability distribution:

$$p(\mathbf{X}^{(1:T)}, \mathbf{Y}^{(1:T)}) = p(\mathbf{x}^{(0)}) \cdot \prod_{t=1}^{T} p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}) \cdot \prod_{t=1}^{T} p(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)}) \tag{4.59}$$

The conditional probability function $p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)})$ represents the state transition model. Since it reveals the dependence between the variables in consecutive time slices, we call it inter-slice dependence. In the graph, inter-slice dependence is represented by the edges from slice $t-1$ to t. The conditional probability function $p(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)})$ represents the observation model. Since it involves variables in the same time slice, it is called intra-slice dependence. Intra-slice dependence is represented by each sub-graph $\mathcal{G}^{(t)}$. A complete dynamic Bayesian network presents the whole joint probability of (4.59) while a 2TBN represents:

$$\begin{aligned} p(\mathbf{y}^{(t)}, \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)} | \mathbf{Y}^{(1:t-1)}) &= p(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}) \cdot p(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \\ &= p(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}) \cdot p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) \cdot p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \end{aligned} \quad (4.60)$$

It can be seen from (4.60) that a 2TBN is composed of three major parts:

- observation model: $p(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})$
- state transition model: $p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$
- a conclusion of the history: $p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)})$

Usually, the observation and state transition models are factorizable, i.e.,:

$$p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) = \prod_{i=1}^{M} p(x_i^{(t)}|\mathbf{x}_{PA(x_i^{(t)})}) \quad (4.61)$$

and

$$p(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}) = \prod_{j=1}^{N} p(y_j^{(t)}|\mathbf{x}_{PA(y_j^{(t)})}) \quad (4.62)$$

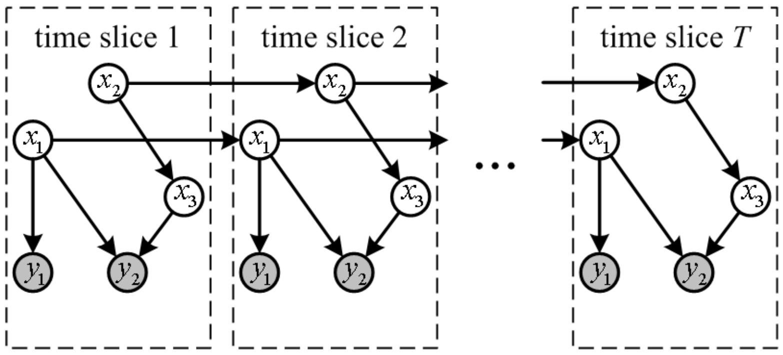
where $\mathbf{x}_{PA(x_i^{(t)})} \subset \{\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}\}$ are parents of $x_i^{(t)}$ and $\mathbf{x}_{PA(y_j^{(t)})} \subset \{\mathbf{x}^{(t)}\}$ are parents of $y_j^{(t)}$. So a 2TBN represents:

$$\begin{aligned} & p(\mathbf{y}^{(t)}, \mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \\ = & \prod_{i=1}^{M} p(x_i^{(t)}|\mathbf{x}_{PA(x_i^{(t)})}) \cdot \prod_{j=1}^{N} p(y_j^{(t)}|\mathbf{x}_{PA(y_j^{(t)})}) \cdot p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \end{aligned} \quad (4.63)$$

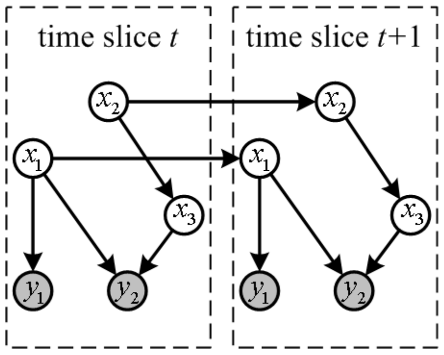
Figure 4.18 illustrates a complete dynamic Bayesian network and its 2TBN representation where white circles represent the hidden state variables and the shaded circles represent observations.

4.3.3 Inference in dynamical systems

A naive inference method in dynamical systems is to infer in a subgraph, i.e., keep ”relevant” slices and turn the dynamic Bayesian network to a static Bayesian network, then apply standard method, e.g., junction tree algorithm to infer the state variables of interest. Such a method is simple to implement. However, it is always difficult to decide which



(a) complete Bayesian network



(b) 2-slice temporal Bayes nets

Fig. 4.18. An example of 2TBN

slices are relevant to the slice of interest and by omitting some slices in the past, the history of the process is not completely kept. Usually, we want to do the filtering based on the entire time history.

Exploiting the special structure of 2TBN, recursive methods were proposed where the posterior probability density function $p(\mathbf{x}^{(t)} \mid \mathbf{Y}^{(1:t)})$, also called belief state, is estimated based on the estimate of $p(\mathbf{x}^{(t-1)} \mid \mathbf{Y}^{(1:t-1)})$, the transition model $p(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)})$ and the observation model $p(\mathbf{y}^{(t)}|\mathbf{x}^{(t)})$. Authors of [27] formulated the recursive estimation in three steps. Here we formulate the procedure in two steps for clarity.

- Prediction. In this step, we first combine the estimate from the previous time slice, i.e., $p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)})$ with the state transition model $p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$:

$$p(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) = p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) \cdot p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \tag{4.64}$$

  Then we integrate out $\mathbf{x}^{(t-1)}$ in the result in (4.64) to obtain the conditional probability density function of $\mathbf{x}^{(t)}$ given $\mathbf{Y}^{(1:t-1)}$:

$$p(\mathbf{x}^{(t)}|\mathbf{Y}^{(1:t-1)}) = \int_{\mathbf{x}^{(t-1)}} p(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \tag{4.65}$$

  The result of (4.65) is the prediction of $\mathbf{x}^{(t)}$ based on the time history.

- Estimation. In this step, the predicted probability density function will be modified by the observation in the current time slice:

$$p(\mathbf{x}^{(t)}|\mathbf{Y}^{(1:t)}) = p(\mathbf{x}^{(t)}|\mathbf{Y}^{(1:t-1)}) \cdot p(\mathbf{y}^{(t)} \mid \mathbf{x}^{(t)}) \tag{4.66}$$

  so that the final estimation in the current time slice is obtained.

Depending on the property of the dynamical systems, the recursive inference method can be implemented in different ways for specific models. The most famous implementation is the Kalman filter used for state estimation in linear-Gaussian state space model. In a linear-Gaussian state space model, the relationships between variables are linear and inter-slice and intra-slice dependences are given by Gaussian distribution functions. Such a linear dynamical system can be represented by the state-space model, which is described by a pair of linear equations, i.e.,:

- state transition equation:

$$\mathbf{x}^{(t)} = \mathbf{A}^{(t)}\mathbf{x}^{(t-1)} + \boldsymbol{\omega}^{(t)} \tag{4.67}$$

- observation equation:

$$\mathbf{y}^{(t)} = \mathbf{C}^{(t)}\mathbf{x}^{(t)} + \boldsymbol{\upsilon}^{(t)} \tag{4.68}$$

where $\mathbf{A}^{(t)} \in \mathbb{R}^{M\times M}$ defines the state transition, $\mathbf{C}^{(t)} \in \mathbb{R}^{N\times M}$ defines the observation model. $\boldsymbol{\omega}^{(t)} \in \mathbb{R}^{M\times 1}$ is process noise which is assumed to be drawn from a zero mean multivariate Gaussian distribution with covariance $\mathbf{Q}^{(t)}$. $\boldsymbol{\upsilon}^{(t)} \in \mathbb{R}^{N\times 1}$ is measurement noise which is assumed to be drawn from a zero mean multivariate Gaussian distribution with covariance $\mathbf{R}^{(t)}$.

The state-space model takes a specific form. There is no intra-slice dependence between hidden variables. Otherwise, it is not possible to use the two linear equations to fully characterize the dependence between the variables. As a consequence, the state-space model can be used to represent dynamical systems with specific structures while the dynamic Bayesian network provides a more general representation. For example, the dynamic Bayesian network shown in Figure 4.18 cannot be represented by a state-space model since the hidden state variable x_3 depends on x_1 and x_2 in the same time slice.

With a linear-Gaussian state space model, the inter-slice dependence is given by:

$$\begin{aligned} &p(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) \\ &= \frac{1}{(2\pi)^{M/2}|\mathbf{Q}^{(t)}|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{x}^{(t)} - \mathbf{A}^{(t)}\mathbf{x}^{(t-1)}\right)^{\mathrm{T}}\left(\mathbf{Q}^{(t)}\right)^{-1}\left(\mathbf{x}^{(t)} - \mathbf{A}^{(t)}\mathbf{x}^{(t-1)}\right)\right] \end{aligned} \tag{4.69}$$

and the intra-slice dependence is given by:

$$\begin{aligned} &p(\mathbf{y}^{(t)}|\mathbf{x}^{(t)}) \\ &= \frac{1}{(2\pi)^{M/2}|\mathbf{R}^{(t)}|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{y}^{(t)} - \mathbf{C}^{(t)}\mathbf{x}^{(t)}\right)^{\mathrm{T}}\left(\mathbf{R}^{(t)}\right)^{-1}\left(\mathbf{y}^{(t)} - \mathbf{C}^{(t)}\mathbf{x}^{(t)}\right)\right] \end{aligned} \tag{4.70}$$

Assume the initial state is given by a Gaussian distribution with mean $\hat{\mathbf{x}}^{(0,0)}$ and covariance $\mathbf{P}^{(0,0)}$. Since all random variables in equation (4.67) and (4.68) are Gaussian and the equations are linear, all the hidden state variables are Gaussian, i.e., $p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)})$ is a Gaussian distribution. Let us use $\hat{\mathbf{x}}^{(t-1,t-1)}$ and $\mathbf{P}^{(t-1,t-1)}$ to denote the mean and the covariance matrix of $p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)})$. Then the result of (4.65) is also a Gaussian distribution. Let us denote the mean of $p(\mathbf{x}^{(t)}|\mathbf{Y}^{(1:t-1)})$ with $\hat{\mathbf{x}}^{(t,t-1)}$ and covariance matrix with $\mathbf{P}^{(t,t-1)}$. Using the derivation in Appendix 4.A.1, we obtain:

$$\hat{\mathbf{x}}^{(t,t-1)} = \mathbf{A}^{(t)} \cdot \hat{\mathbf{x}}^{(t-1,t-1)} \tag{4.71}$$

and

$$\mathbf{P}^{(t,t-1)} = \mathbf{A}^{(t)} \cdot \mathbf{P}^{(t-1,t-1)}(\mathbf{A}^{(t)})^{\mathrm{T}} + \mathbf{Q}^{(t)} \tag{4.72}$$

Based on (4.65), the result of (4.66) is also Gaussian. Let us use $\hat{\mathbf{x}}^{(t,t)}$ and $\mathbf{P}^{(t,t)}$ to denote its mean and covariance matrix. Then using the derivation in Appendix 4.A.2, we obtain:

$$\mathbf{P}^{(t,t)} = \mathbf{P}^{(t,t-1)} - \mathbf{P}^{(t,t-1)}(\mathbf{C}^{(t)})^{\mathrm{T}} \left(\mathbf{R}^{(t)} + \mathbf{C}^{(t)}\mathbf{P}^{(t,t-1)}(\mathbf{C}^{(t)})^{\mathrm{T}}\right)^{-1} \mathbf{C}^{(t)}\mathbf{P}^{(t,t-1)} \tag{4.73}$$

and

$$\hat{\mathbf{x}}^{(t,t)} = \mathbf{P}^{(t,t)} \left((\mathbf{P}^{(t,t-1)})^{-1}\hat{\mathbf{x}}^{(t,t-1)} + (\mathbf{C}^{(t)})^{\mathrm{T}}(\mathbf{R}^{(t)})^{-1}\mathbf{y}^{(t)}\right) \tag{4.74}$$

Standard Kalman filter posed state estimation as a minimum mean square error (MMSE) problem. The derivation above gives a probabilistic interpretation of Kalman filter. Although Kalman filter is the estimation method for a very specific kind of dynamical system, it has a wide variety of applications [114].

Basic Kalman filter works under the linear assumption. For dynamical systems with nonlinear transition and / or observation functions, different versions of extension have been made to deal with non-linearity. *Extended Kalman filter* (EKF) [88] linearizes the system before the state estimation using the first order Taylor expansion. It is simple to implement. However, it may result in non-stable estimate [55]. A better solution is *unscented Kalman filter* (UKF) [55] which approximates the probability density by nonlinear transformation of a random variable. An alternative to EKF and UKF is the particle filter introduced in the Section 4.2.2. Particle filter is a sample based method. With sufficient random samples, particle filter approaches the Bayesian optimal estimate [18]. So particle filter can provide more accurate estimate than either EKF or UKF.

For high-dimensional non-linear data with non-Gaussian distributions, joint calculation in (4.64), (4.65) and (4.66) without exploiting the structure of the graph results in intractable solution. It is usually impossible to find a closed form solution like in the Kalman filter. In this case, the structure of 2TBN, especially the d-separation property should be exploited to simplify the computation. 2TBN can be treated as a static Bayesian network and the inference algorithms, e.g., junction tree algorithm, belief propagation, can be used for querying the marginals of any hidden variable.

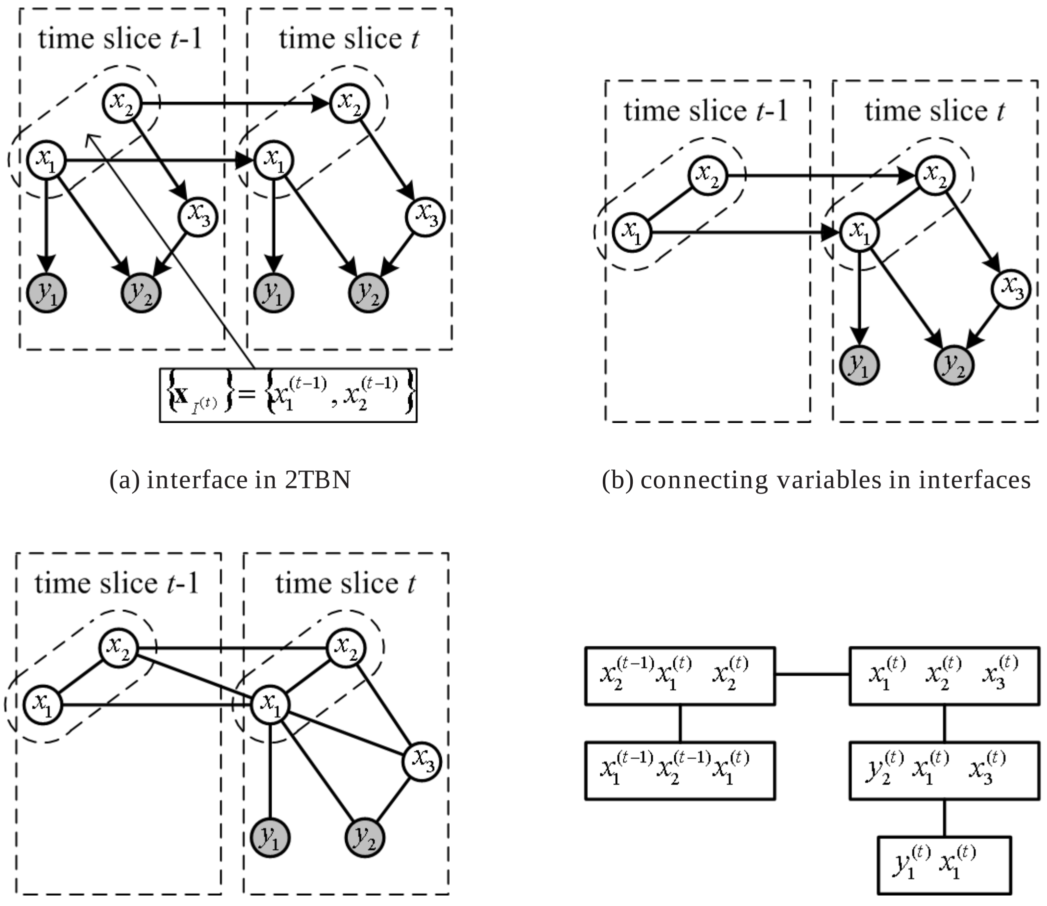
The most important issue of inference in 2TBN is the delivery of the past from one time slice to its successor. It can be observed from (4.63) and (4.64) that the joint distribution, or the belief state $p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)})$ has to be provided by the previous slice in order to enable the current time slice to retrieve all the necessary information for the exact inference.

Kevin Murphy's interface algorithm [77] defines an interface $\mathcal{I}^{(t)}$ that d-separates the past from the future. The interface encapsulates all necessary information about the history. In a DBN, the interface from time slice $t-1$ to slice t contains hidden variables in time slice $t-1$ which have children in time slice t, i.e.,

$$\{\mathbf{x}_{\mathcal{I}^{(t-1)}}\} = \{\mathbf{x}_{PA(\mathbf{x}^{(t)})}, \mathbf{x}_{PA(\mathbf{y}^{(t)})}\} \cap \{\mathbf{x}^{(t-1)}\} \tag{4.75}$$

where $\mathcal{I}^{(t-1)}$ denotes the indices of the variables in the interface from time slice $t-1$ to time slice t.

Let us use the 2TBN in Figure 4.18 as an example to illustrate the interface algorithm. According to (4.75), $\{\mathbf{x}_{\mathcal{I}^{(t-1)}}\} = \{x_1^{(t-1)}, x_2^{(t-1)}\}$ is the interface at time slice $t-1$, as shown in Figure 4.19(a).


time slice t-1
time slice t
x_1
x_2
x_3
y_1
y_2
$\{\mathbf{x}_{\mathcal{I}^{(t)}}\} = \{x_1^{(t-1)}, x_2^{(t-1)}\}$
$x_2^{(t-1)} x_1^{(t)} x_2^{(t)}$
$x_1^{(t)} x_2^{(t)} x_3^{(t)}$
$x_1^{(t-1)} x_2^{(t-1)} x_1^{(t)}$
$y_2^{(t)} x_1^{(t)} x_3^{(t)}$
$y_1^{(t)} x_1^{(t)}$

(a) interface in 2TBN

(b) connecting variables in interfaces

(c) triangulation

(d) junction tree

Fig. 4.19. Interface algorithm

Having identified hidden variables that influence the "future", we modify the graph so that variables in the interface are mutually connected and nodes that are not in the interface in the first slice in the 2TBN are removed. The resulting graph is shown in Figure 4.19(b). Based on the new graph, we can construct a junction tree and use belief propagation to compute the marginals of the hidden variables. We first triangulate Figure 4.19(b) which leads to a graph in Figure 4.19(c). Then we construct the junction tree which is shown in Figure 4.19(d). As explained in the previous chapter, the junction tree algorithm implements variable elimination which exploits the independence property of a distribution. As a consequence, junction tree algorithm reduces the complexity of the inference in dynamic Bayesian network. The interface algorithm guarantees the exactness of the result by introducing new constraints into the original graph. It requires that variables in the interface are mutually connected, which complicates the graphical model.

In practice, many constraints forbid a centralized implementation of the Kalman filter or the interface algorithm due to the constraints on communications, computational complexity or allowable latencies. For example, self contained power supply limits the power consumption of transmitting all necessary information to an inference center; time critical applications cannot tolerate the delays incurred by transmitting information to the inference center and waiting for the results. Distributed inference exploits the decomposability of the problem and process the information as locally as possible. However, according to the interface algorithm, state variables in the interface have to be mutually connected in the inference to produce exact results, which indicate that the belief state, i.e., $p(\mathbf{x}^{(t)} \mid \mathbf{Y}^{(1:t)})$ should not be factorized. Such a property make it impossible to have a distributed implementation of the inference.

Boyen and Koller proposed an approximate algorithm in [10] that assumes additional independence among variables in the same time slice so that the distribution can be factorized, i.e.,:

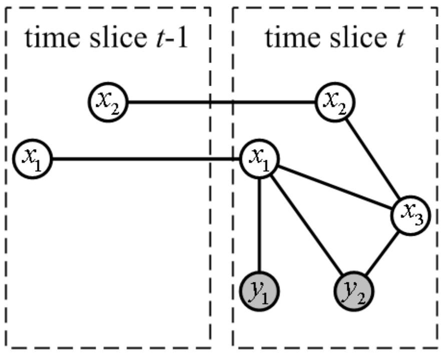
$$p(\mathbf{x}^{(t-1)}|\mathbf{Y}^{(1:t-1)}) = \prod_{i=1}^{L} p(\mathbf{x}_{\mathcal{K}_i^{(t-1)}}|\mathbf{Y}^{(1:t-1)}) \tag{4.76}$$

where $\mathbf{x}_{\mathcal{K}_i^{(t-1)}} \subset \mathbf{x}^{(t-1)}$ is called BK cluster. Obviously, factorization in (4.76) can further simplify the inference since now we can construct a junction tree with smaller clusters. On the other hand, it allows distributed implementation of the inference. Figure 4.20 shows a junction tree that is constructed based on the assumption that the belief state is factorized as:

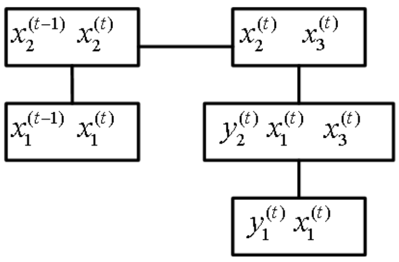
$$p(x_1^{(t-1)}, x_2^{(t-1)}|\mathbf{Y}^{(1:t-1)}) = p(x_1^{(t-1)}|\mathbf{Y}^{(1:t-1)})p(x_2^{(t-1)}|\mathbf{Y}^{(1:t-1)}) \tag{4.77}$$

With the introduction of extra independence, the results are not exact. However, it is proved that the error made by the approximation, measured by Kullback-Leibler distance from the true distribution, remains bounded indefinitely [10]. In practice, the factorization of the belief state is based on the topology of the network. We can use the procedure of distributed inference introduced in Section 4.1.3 to construct such a factorization.

The central idea of Boyen-Koller's approximation is to use a factorizable distribution to approximate a non-factorizable distribution, as shown in (4.76). Variational methods introduced in Chapter 3 search for the optimal factorization in the sense that the Kullback-Leibler distance between the exact belief state and the approximate belief state is minimized. Using variational method in Boyen-Koller's approximation, the inference algorithm should converge faster. However, implementing variational method is not trivial since iterative optimization is involved. Usually we construct the approximate belief state in a simpler way. For example, given a belief state $p(\mathbf{x}^{(t)} \mid \mathbf{Y}^{(1:t)})$, we may approximate



(a) triangulation



(b) junction tree

Fig. 4.20. Boyen and Koller algorithm

it by the product of its marginal probability distributions. We should not forget that the graphical model of dynamical systems also provide a tool for selecting the approximate belief state. In Chapter 6, we will modify the structure of the graphs to discover feasible inference algorithms for the state estimation. Other examples of using Boyen-Koller's approximation for distributed inference can be found in [27, 28, 76].

4.4 Remark and discussions

In this chapter, we presented the probabilistic inference techniques that are most relevant to the estimation problems encountered in networked systems. Three most important issues are discussed: distributed inference, approximations and probabilistic models for dynamical systems. All these techniques aim at reducing the communication cost and the computational complexity required by the probabilistic inference to make it operationally feasible.

Distributed inference makes computations as local as possible. Each network element does some processing before it sends the data to the network. Only the most necessary information should to be exchanged between elements. In this way, the communication cost is reduced to the lowest level. However, distributed inference is not trivial. The underlying communication network puts extra constraint on the inference. When designing the inference algorithm, we should take into consideration these constraints. We used a simple example to show how the communications influence the inference and how different designs of inference result in different resource requirements. Distributed inference is important for networked systems with power constraints, e.g., wireless sensor network. In the next chapter, we will see a practical example.

Approximations are needed because exact inference could be too expensive. In this chapter, we have focused on function and message approximation. We have shown that by carefully choosing the proper representations, very complicated functions can be well approximated by functions with simple structure. Using approximations, both the computational complexity and the transmission cost can be reduced. We have presented two function approximation methods which simplify belief propagation. Fourier domain belief propagation represents a function by its truncated Fourier series expansion. Similar representation can be established by using other basis functions, e.g., wavelets, Gaussian. Non-parametric belief propagation uses random samples to represent functions. It is a Monte Carlo approach. Like the loopy belief propagation introduced in the last chapter, it is very difficult to derive analytically the total error introduced by these approximation methods. However, we can use numerical simulations to examine their performance.

State estimation in simple dynamical system can be achieved by Kalman filter or its extensions. However, in networked systems, hidden state variables are typically correlated in space, which results in a very complicated dynamical system. We can use dynamic Bayesian networks to represent the relationships between the variables. Based on such a graphical model, we can develop different inference algorithms, exact or approximate, for the state estimation. A practical application will be shown in Chapter 6.

4.A Property of Gaussian density functions

Although the property of Gaussian density functions (e.g., the product of is still Gaussian and the integration of multivariate Gaussian density function with respect to a subset of its arguments result in another Gaussian function) is well known in the literature, we did not find a good reference that explicitly derives the resulting mean and covariance. Therefore, we make the derivation in this section.

4.A.1 Derivation of Gaussian integral

Suppose we want to calculate:

$$p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x})p(\mathbf{x}) \tag{4.78}$$

where $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{y} \in \mathbb{R}^M$, $\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Lambda}_{\mathbf{x}}\right)$ and $\mathbf{y}|\mathbf{x} \sim \mathcal{N}\left(\mathbf{A}\mathbf{x}, \boldsymbol{\Lambda}_{\mathbf{y}|\mathbf{x}}\right)$ (i.e., $\mathbf{y} = \mathbf{A}\mathbf{x} + \boldsymbol{\nu}$ and $\nu \sim \mathcal{N}(0, \boldsymbol{\Lambda}_{\mathbf{y}|\mathbf{x}})$). We already know that the result of the product of two Gaussian probability density functions results in new Gaussian pdf. The mean can be calculatd by:

$$\boldsymbol{\mu}_{\mathbf{y}} = E\left\{\mathbf{y}\right\} = E\left\{E\left\{\mathbf{y}|\mathbf{x}\right\}\right\} = E\left\{\mathbf{A}\mathbf{x}\right\} = \mathbf{A}\boldsymbol{\mu}_{\mathbf{x}} \tag{4.79}$$

and the covariance can be calculated by:

$$\begin{aligned}
\boldsymbol{\Lambda}_{\mathbf{y}} &= E\left\{(\mathbf{y}-\boldsymbol{\mu}_{\mathbf{y}})(\mathbf{y}-\boldsymbol{\mu}_{\mathbf{y}})^{\mathrm{T}}\right\} \\
&= E\left\{\mathbf{y}\mathbf{y}^{\mathrm{T}}\right\} - \boldsymbol{\mu}_{\mathbf{y}}\boldsymbol{\mu}_{\mathbf{y}}{}^{\mathrm{T}} \\
&= E\left\{E\left\{\mathbf{y}\mathbf{y}^{\mathrm{T}}|\mathbf{x}\right\}\right\} - \boldsymbol{\mu}_{\mathbf{y}}\boldsymbol{\mu}_{\mathbf{y}}{}^{\mathrm{T}} \\
&= E\left\{\boldsymbol{\Lambda}_{\mathbf{y}|\mathbf{x}} + \boldsymbol{\mu}_{\mathbf{y}|\mathbf{x}}\boldsymbol{\mu}_{\mathbf{y}|\mathbf{x}}{}^{\mathrm{T}}\right\} - \boldsymbol{\mu}_{\mathbf{y}}\boldsymbol{\mu}_{\mathbf{y}}{}^{\mathrm{T}} \\
&= \boldsymbol{\Lambda}_{\mathbf{y}|\mathbf{x}} + E\left\{\mathbf{A}\mathbf{x}\mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\right\} - \mathbf{A}\boldsymbol{\mu}_{\mathbf{x}}\boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}} \\
&= \boldsymbol{\Lambda}_{\mathbf{y}|\mathbf{x}} + \mathbf{A}\left(E\left\{\mathbf{x}\mathbf{x}^{\mathrm{T}}\right\} - \boldsymbol{\mu}_{\mathbf{x}}\boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\right)\mathbf{A}^{\mathrm{T}} \\
&= \boldsymbol{\Lambda}_{\mathbf{y}|\mathbf{x}} + \mathbf{A}\boldsymbol{\Lambda}_{\mathbf{x}}\mathbf{A}^{\mathrm{T}}
\end{aligned} \tag{4.80}$$

4.A.2 Derivation of Gaussian product

Suppose we want to calculate:

$$p_2(\mathbf{x}) = p(\mathbf{y}|\mathbf{x})\, p_1(\mathbf{x}) \tag{4.81}$$

where $\mathbf{x} \in \mathbb{R}^N$, $\mathbf{y} \in \mathbb{R}^M$, the likelihood function $p(\mathbf{y}|\mathbf{x}) = \frac{1}{(2\pi)^{M/2}|\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}|^{1/2}} e^{-\frac{1}{2}(\mathbf{y}-\mathbf{C}\mathbf{x})^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}(\mathbf{y}-\mathbf{C}\mathbf{x})}$ and the pdf $p_1(\mathbf{x}) = N(\boldsymbol{\mu}_{\mathbf{x}}, \mathbf{\Lambda}_{\mathbf{x}})$. The product is calculated as follows:

$$\begin{aligned}
p_2(\mathbf{x}) &= p(\mathbf{y}|\mathbf{x})\, p_1(\mathbf{x}) \\
&= \frac{1}{(2\pi)^{M/2}|\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y}-\mathbf{C}\mathbf{x})^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}(\mathbf{y}-\mathbf{C}\mathbf{x})\right) \\
&\quad \cdot \frac{1}{(2\pi)^{N/2}|\mathbf{\Lambda}_{\mathbf{x}}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{x}})^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}(\mathbf{x}-\boldsymbol{\mu}_{\mathbf{x}})\right) \\
&\propto \exp\left(-\frac{1}{2}\left\{\begin{array}{c}\left(\mathbf{x}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{C}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{C}\mathbf{x} + \mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y}\right) \\ +\left(\mathbf{x}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}} - \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\mathbf{x} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\right)\end{array}\right\}\right) \\
&= \exp\left(-\frac{1}{2}\left\{\begin{array}{c}\mathbf{x}^{\mathrm{T}}\left(\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{C} + \mathbf{\Lambda}_{\mathbf{x}}^{-1}\right)\mathbf{x} - \mathbf{x}^{\mathrm{T}}\left(\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\right) \\ -\left(\mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{C} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\right)\mathbf{x} + \mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\end{array}\right\}\right) \qquad (4.82)
\end{aligned}$$

Let us denote $\left(\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{C} + \mathbf{\Lambda}_{\mathbf{x}}^{-1}\right)^{-1}$ with $\mathbf{D}$ and $\left(\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\right)$ with $\mathbf{z}$. Then we continue with (4.82):

$$\begin{aligned}
p_2(\mathbf{x}) &\propto \exp\left(-\frac{1}{2}\left\{\mathbf{x}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{z} - \mathbf{z}^{\mathrm{T}}\mathbf{x} + \mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\right\}\right) \\
&= \exp\left(-\frac{1}{2}\left\{\begin{array}{r}\mathbf{x}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{D}\mathbf{z} - \mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{D}^{-1}\mathbf{x} \\ +\left(\mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{D}^{-1}\mathbf{D}\mathbf{z} - \mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{D}^{-1}\mathbf{D}\mathbf{z}\right) + \mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\end{array}\right\}\right) \\
&= \exp\left(-\frac{1}{2}\left\{\begin{array}{r}\mathbf{x}^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{x}-\mathbf{D}\mathbf{z}) - \mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{D}^{-1}(\mathbf{x}-\mathbf{D}\mathbf{z}) - \mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{D}^{-1}\mathbf{D}\mathbf{z} \\ +\mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\end{array}\right\}\right) \\
&= \exp\left(-\frac{1}{2}\left\{(\mathbf{x}^{\mathrm{T}} - \mathbf{z}^{\mathrm{T}}\mathbf{D})\mathbf{D}^{-1}(\mathbf{x}-\mathbf{D}\mathbf{z}) - \mathbf{z}^{\mathrm{T}}\mathbf{D}\mathbf{z} + \mathbf{y}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \boldsymbol{\mu}_{\mathbf{x}}{}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\right\}\right) \\
&\propto \exp\left(-\frac{1}{2}\left\{(\mathbf{x}-\mathbf{D}\mathbf{z})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{x}-\mathbf{D}\mathbf{z})\right\}\right) \qquad (4.83)
\end{aligned}$$

So the resulting $p_2(\mathbf{x})$ has a Gaussian distribution. Its mean is:

$$\bar{\mathbf{x}} = E_{p_2(\mathbf{x})}\{\mathbf{x}\} = \mathbf{D}\mathbf{z} = \mathbf{D}\left(\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{y} + \mathbf{\Lambda}_{\mathbf{x}}^{-1}\boldsymbol{\mu}_{\mathbf{x}}\right) \tag{4.84}$$

Its covariance matrix is:

$$\mathrm{cov}(\mathbf{x},\mathbf{x}) = E_{p_2(\mathbf{x})}\{(\mathbf{x}-\bar{\mathbf{x}})(\mathbf{x}-\bar{\mathbf{x}})^{\mathrm{T}}\} = \mathbf{D} = \left(\mathbf{C}^{\mathrm{T}}\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}}^{-1}\mathbf{C} + \mathbf{\Lambda}_{\mathbf{x}}^{-1}\right)^{-1} \tag{4.85}$$

Using the *matrix inversion lemma*, the covariance matrix can also be written as:

$$\mathrm{cov}(\mathbf{x},\mathbf{x}) = \mathbf{\Lambda}_{\mathbf{x}} - \mathbf{\Lambda}_{\mathbf{x}}\mathbf{C}^{\mathrm{T}}\left(\mathbf{\Lambda}_{\mathbf{y}|\mathbf{x}} + \mathbf{C}\mathbf{\Lambda}_{\mathbf{x}}\mathbf{C}^{\mathrm{T}}\right)^{-1}\mathbf{C}\mathbf{\Lambda}_{\mathbf{x}} \tag{4.86}$$

5. Belief Propagation For Sensor Localization

Advances in sensing technology and telecommunications make wireless sensor networks an appropriate solution for a wide variety of applications. In a wireless sensor network, sensor nodes are spatially distributed to monitor the physical or environmental parameters, e.g., temperature, humidity, pressure, sound, visible and infrared light, magnetic fields, acceleration and so on. Each sensor is also equipped with a computing unit so that initial data processing can be carried out locally within the sensor node. In order to reduce the cost, the capacity of the processor and memory in a sensor node is limited. The processed information can be exchanged through the wireless channel so that the whole network works in a cooperative fashion. To save transmission power, each node only communicates with the nodes in its neighborhood. In order to increase the flexibility of the whole system, each sensor is equipped with a self-contained power supply, typically a battery. For the reason of low infrastructure and maintenance cost, these power supplies are usually neither replaced nor recharged. Therefore the life of the power supply determines the life of the sensor node.

In the scenario of pervasive sensing, a large number of sensor nodes are scattered in a region which report everything they measured. Based on the noisy measurements and the underlying physical model, we can learn the distribution over time and space of the parameters of interest. Simple tasks require only basic operations, e.g., mean and sum. Sophisticated tasks require the computation of posterior probabilities or likelihood functions. In this case, probabilistic inference algorithms are needed. For a system with such a large scale and due to the constraint of the power supply and the computing ability, we have to be very careful about the complexity of the probabilistic inference algorithms to make them operationally feasible.

Inference can be implemented in a centralized way in which all the local measurements are transmitted to a fusion center and a big inference engine is established there to compute the quantities of interest. However, such a scheme requires the transmission of a large amount of raw data. The main bottleneck in a wireless sensor network is the power supply as each sensor is equipped with a non-replaceable and non-rechargeable battery. It has been observed in empirical applications that communication typically consumes many times the amount of energy required for computation or sensing. Therefore, sending raw data without any processing should be avoided to reduce the unnecessary power consumption.

The distributed inference method introduced in Section 4.1 processes measurements locally. Only brief summaries, i.e., functions of the measurement data, usually represented by probability functions, have to be exchanged between sensor nodes. In such a way, the power consumed by the transmission can be greatly reduced. However, for complicated probability distributions, the computations required by the distributed inference are still complicated. It is usually impossible to implement them in sensor nodes with limited computing resources. To solve this problem, we use approximation methods introduced in Section 4.2, which not only simplify the inference, but also further reduce the amount of transmitted data.

Having introduced distributed and approximate inference algorithms in Chapter 4, we use them to solve the self organized sensor localization problem in this chapter. In particular, we propose message passing based inference methods as suitable solutions to this problem and use Fourier density approximation and non-parametric belief propagation to simplify the computation and reduce the transmission power. Non-parametric belief propagation for self-organized sensor localization was discussed in detail in [49]. In this chapter, we present our implementation of this inference algorithm. We also want to compare the performance of these two function approximation methods, one based on function analysis and the other one based on Monte Carlo sampling.

The rest of the chapter is organized as follows. Section 5.1 briefly introduced the background of sensor localization. The system model is presented in Section 5.2. The probabilistic model is developed in Section 5.3 and belief propagation for distributed inference is formulated in Section 5.4. Two approximate inference algorithms, i.e., Fourier domain belief propagation and non-parametric belief propagation are applied to this problem in Section 5.5 and Section 5.6. Simulation results are provided in Section 5.7 to illustrate the performance of the localization algorithms. Finally, Section 5.8 concludes this chapter.

5.1 Background

Sensor localization is an important task in pervasive sensing in which sensor nodes are typically arbitrarily scattered in a region. At initialization, sensors have to calibrate their relative and absolute positions in order to carry out subsequent tasks. Sensor positioning can be achieved by using the GPS service. However, due to the high cost, it is not possible to equip every sensor with a GPS receiver. Therefore, a self-organized sensor positioning scheme is desirable. Relative positions can be estimated from associated measurements, e.g., strength of received signal power, transmission delay and so on. A subset of the sensor nodes can be equipped with GPS receivers to obtain absolute positions. Then all the information, i.e., associated measurements and the information provided by GPS will be combined to enable all sensor nodes to estimate their absolute positions. Usually, it would be beneficial to compute not only the estimate of the sensor position, but also the uncertainty of the estimation. Such information could be very useful for subsequent tasks. In that case, sensor localization becomes a probabilistic inference problem in which we want to obtain the posterior probability distribution of the sensor locations given the measurements. An efficient algorithm is preferable to fulfill the computational and power constraint of a wireless sensor network.

5.2 System model

In this chapter, we present sensor localization based on noisy distance measurements.

Let us assume that we have a wireless sensor network with N sensors distributed in a planar space. The two dimensional location of sensor i is denoted by $\mathbf{x}_i$. The measurement taken at sensor i about its distance to sensor j takes the form:

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\| + \nu_{ij} \tag{5.1}$$

where d_{ij} denotes the distance measurement. The operator $\|\cdot\|$ calculates the Euclidean norm. Additive random noise ν_{ij} is drawn from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

Distance measurement d_{ij} is not always available since sensor i does not always detect its neighbor j. We use a binary random variable o_{ij} to indicate whether a distance measurement is available, i.e. $o_{ij} = 1$ when observation is made, and $o_{ij} = 0$ otherwise. A model is suggested in [75] in which the availability of the distance measurement depends on the distance of two sensor nodes. In particular, the probability that sensor i detects sensor j is given by an exponential function of their distance:

$$p(o_{ij} = 1|\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2R^\rho}\right) \tag{5.2}$$

where R is a parameter that determines the detection range. Parameter ρ can take different values according to the environment. Typical values are between 2 and 4.

Furthermore, each sensor has some prior knowledge about its position, which is given by a prior distribution $p(\mathbf{x}_i)$. The prior distribution is usually uninformative unless the sensor has obtained its position information from other resources, e.g. GPS signal. In this case, the prior distribution takes the form of a Dirac function.

5.3 Probabilistic model for sensor localization

Based on the system model described in the previous section, the joint probability distribution of the whole system is given by:

$$p(\{\mathbf{x}_i\}, \{d_{ij}\}, \{o_{ij}\}) = \left(\prod_{\{i,j\}:i\neq j} p(o_{ij}|\mathbf{x}_i, \mathbf{x}_j)\right)\left(\prod_{\{i,j\}:i\neq j, o_{ij}=1} p_{v_{ij}}(d_{ij}|\mathbf{x}_i, \mathbf{x}_j)\right)\left(\prod_i p(\mathbf{x}_i)\right) \tag{5.3}$$

The goal of sensor localization is to estimate the positions of sensors, which can be formulated as a maximum a posteriori (MAP) problem. In that way, position estimation becomes a problem of estimating the posterior probability distribution of each individual sensor position given observations, i.e., calculating $p(\mathbf{x}_i|\{d_{ij}\}, \{o_{ij}\})$. As discussed at the beginning of the chapter, the estimation should be implemented in a distributed manner and the computation should be efficient for the sake of minimizing power consumption.

A distributed inference procedure has been discussed in Section 4.1. According to that, we need to identify for each sensor i a local function φ_i and obtain a factorization of the joint probability distribution with the following structure:

$$p(\{\mathbf{x}_i\}, \{d_{ij}\}, \{o_{ij}\}) = \prod_i \varphi_i \tag{5.4}$$

In order to achieve such a factorization, we need to assign each factor in (5.3) into one of the local functions. At the end, each local function φ_i should take the following form:

$$\varphi_i(\mathbf{x}_1, \ldots \mathbf{x}_N) = p(\mathbf{x}_i) \prod_{\{j\}:j\neq i} p(o_{ij}|\mathbf{x}_i, \mathbf{x}_j) \prod_{\{j\}:j\neq i, o_{ij}=1} p_{v_{ij}}(d_{ij}|\mathbf{x}_i, \mathbf{x}_j) \tag{5.5}$$

since o_{ij}, d_{ij} and $p(\mathbf{x}_i)$ are only available at sensor node i without communications. Other assignments require the measurements or the local prior to be exchanged between sensor nodes.

To simplify the notation, we rewrite (5.5) in the following form:

$$\varphi_i(\mathbf{x}_1, \ldots \mathbf{x}_N) = \phi_i(\mathbf{x}_i) \prod_{\{j\}:j\neq i} \phi_{ij}(\mathbf{x}_i, \mathbf{x}_j) \tag{5.6}$$

where

$$\phi_i(\mathbf{x}_i) = p(\mathbf{x}_i) \tag{5.7}$$

is called single-node potential function and

$$\phi_{ij}(\mathbf{x}_i, \mathbf{x}_i) = \begin{cases} p(o_{ij}|\mathbf{x}_i, \mathbf{x}_j)p(d_{ij}|\mathbf{x}_i, \mathbf{x}_j) & \texttt{if} \quad o_{ij} = 1 \\ p(o_{ij}|\mathbf{x}_i, \mathbf{x}_j) & \texttt{if} \quad o_{ij} = 0 \end{cases} \tag{5.8}$$

is called pairwise potential function.

From (5.1) and (5.2), the pairwise potential function is given by:

$$\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2R^2}\right) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(d_{ij}-\|\mathbf{x}_i-\mathbf{x}_j\|)^2}{2\sigma^2}\right) & \texttt{if} \quad o_{ij} = 1 \\ 1 - \exp\left(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2R^2}\right) & \texttt{if} \quad o_{ij} = 0 \end{cases} \tag{5.9}$$

Based on (5.5) we can construct a graphical model from the sensor network, in which each sensor is a node and each detectable communication link is an edge that connects two nodes. Each node i in the graph is associated with a local function $\varphi_i(\mathbf{x}_1, \ldots \mathbf{x}_N)$ that is given by (5.5). Possibly, there will be a lot of loops in the graph since sensors located inside a small region can probably communicate mutually with each other.

5.4 Belief propagation for sensor localization

Now, we use belief propagation as a distributed inference algorithm to solve the MAP estimation problem in sensor localization. We will apply the loopy belief propagation method to deal with loops in the graph. As introduced in Section 3.3.1, in loopy belief propagation initialize all messages by uniform distributions and then use the sum-product algorithm to iteratively update the message until the termination criterion, either convergence or reaching the defined maximum iteration number, has been fulfilled.

Let us use $m_{ij}^{(t)}$ to denote message from node i to node j at the t^{th} iteration. According to equation (3.7), message updating is given by:

$$m_{ij}^{(t)}(\mathbf{x}_1, \ldots \mathbf{x}_N) = \alpha \varphi_i(\mathbf{x}_1, \ldots \mathbf{x}_N) \prod_{l \in NE(i), l \neq j} m_{li}^{(t-1)}(\mathbf{x}_1, \ldots \mathbf{x}_N) \tag{5.10}$$

where α is a normalization factor and $NE(i)$ denotes the neighbors of node i, i.e., sensors that sensor i can detect.

It can be seen that each message in (5.10) is a function of N variables. It is impossible to transmit such complex functions in an efficient way between nodes. Furthermore, the multiplication of messages will be so complicated that it makes the inference intractable. To reduce the complexity, we modify the definition of messages so that a message from node i to node j is a function that only involves $\mathbf{x}_j$. In other words, message from node i to node j only contains a summary of sensor i's belief on the position of j, position informations about other sensor nodes are summed out. Based on this simplification, (5.10) will be revised to:

$$m_{ij}^{(t)}(\mathbf{x}_j) = \alpha \int_{\mathbf{x}_i} \phi_i(\mathbf{x}_i)\phi_{ij}(\mathbf{x}_i, \mathbf{x}_i) \prod_{l \in NE(i), l \neq j} m_{li}^{(t-1)}(\mathbf{x}_i) d\mathbf{x}_i \tag{5.11}$$

The intuition of such a simplifications is that sensor node i sends to node j only the information that is directed connected to the estimation of sensor j's position. The convergence of the belief propagation using this form of messages was discussed in [49], which shows that with this simple form, the belief propagation still converges after iterations provided that each sensor can observe several other sensors in its neighborhood. There exist other forms of simplification, e.g., messages that contains more hidden variables. However, such methods involve more complicated message computations. Discussions on other forms of simplification can be found in [49].

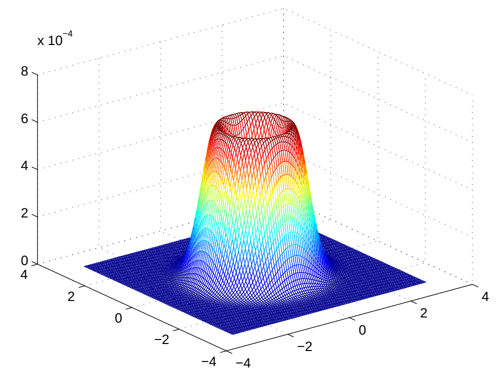
Using the proposed simplification, the posterior probability distribution of $\mathbf{x}_i$ is calculated according to (3.8) by:

$$p(\mathbf{x}_i | \{d_{st}\}, \{o_{st}\}) \propto \prod_{l \in NE(i)} m_{li}^{(t)}(\mathbf{x}_i) \tag{5.12}$$

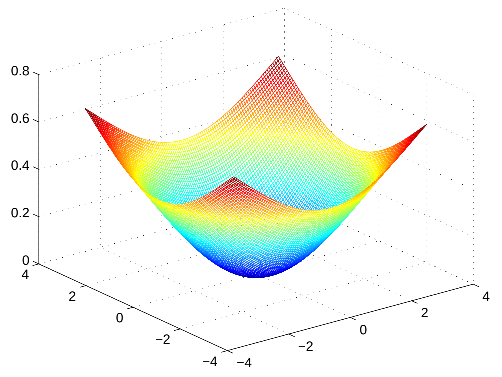
where $\{d_{st}\}$ and $\{o_{st}\}$ contain the observations made in all sensor nodes.

Although the complexity of messages has been greatly simplified in (5.11), computation in (5.11) and (5.12) is still complicated because the calculation involves very complex functions defined in (5.9). Figure 5.1 illustrates the pairwise potential functions according to (5.8) involved in the message computation. Figure 5.2 demonstrates the typical shapes of the messages generated in belief propagation for sensor localization according to (5.11). It can be seen that messages are complicated functions.

As discussed at the beginning of this chapter, the inference algorithm has to be efficient so that it consumes little power. For that purpose, we need to find a suitable representation

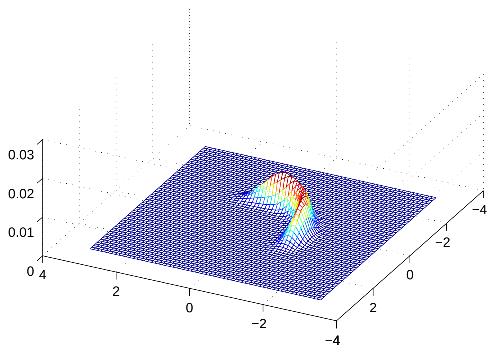


(a) $\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j = [0, 0])$ when $o_{ij} = 1$

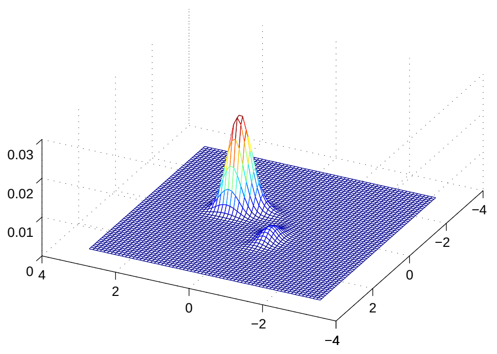


(b) $\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j = [0, 0])$ when $o_{ij} = 0$

Fig. 5.1. Pairwise potential function



(a)



(b)

Fig. 5.2. Messages in BP in sensor localization

of functions and messages so that fewer bits have to be transmitted and the local computation can also be simplified. Two message representation methods have been introduced in Section 4.2, i.e., non-parametric belief propagation and Fourier domain belief propagation. Now we apply these two methods to the MAP estimation of sensor positions.

5.5 Fourier domain belief propagation for sensor localization

Using Fourier series expansion and the coefficient reduction method introduced in Section 4.2.1, the sizes of the messages are significantly reduced. This has brought two-fold benefits. On one side, it reduces the transmission power. On the other hand, it reduces the complexity of the computation in sum-product algorithm with a penalty of computing the Fourier transform.

5.5.1 Discretization of the space domain

In order to use the efficient fast Fourier transform, we first discretize all the continuous functions. For the convenience of fast Fourier transform, the approximation points are chosen to be an equidistant two dimensional grid. According to the *Sampling Theorem*, if the distance between neighboring samples is close enough, the original function can be recovered from its discrete approximation.

So the single potential function $\phi_i(\mathbf{x}_i)$ is approximated by:

$$\phi_i^*(\mathbf{x}_i) = \sum_{n_i=1}^{N_i} \phi_i(\mathbf{x}_{n_i}^*)\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*) \tag{5.13}$$

where N_i is the number of discrete samples, $\{\mathbf{x}_{n_i}^*\}_{n1=1}^{N_i}$ are approximation points and $\phi_i(\mathbf{x}_{n_i}^*)$ is the value of the function at an approximation point $\mathbf{x}_{n_i}^*$. The unit impulse function is given by:

$$\delta(x) = \begin{cases} 1 & \texttt{if} \quad x = 0 \\ 0 & \texttt{if} \quad x \neq 0 \end{cases} \tag{5.14}$$

The pairwise potential function $\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ is approximated by:

$$\phi_{ij}^*(\mathbf{x}_i, \mathbf{x}_j) = \sum_{n_i=1}^{N_i} \sum_{n_j=1}^{N_j} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*)\delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \tag{5.15}$$

We initialize all messages by a discrete uniform distribution. So message $m_{ij}^{(0)}$ is given by:

$$m_{ij}^{(0)}(\mathbf{x}_j) = \sum_{n_j=1}^{N_j} \frac{1}{N_j}\delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \tag{5.16}$$

Sensor node i calculates its message $m_{ij}^{(t)}$ to node j based on the messages from its neighbors except j using (5.11). Let us suppose that the message that node i received from its neighbor $l(l \neq j)$ is given by the following discrete function:

$$m_{li}^{(t-1)}(\mathbf{x}_i) = \sum_{n_i=1}^{N_i} \omega_{li,n_i}^{(t-1)}\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*) \tag{5.17}$$

Inserting (5.17), (5.15) and (5.13) into (5.11), we can calculate the message from i to j. Since now all the inputs for the calculation are discretized, the integration of (5.11) will be replaced by summation. The calculation is shown as follows:

$$\begin{aligned}
m_{ij}^{(t)}(\mathbf{x}_j) &\propto \sum_{\mathbf{x}_i} \begin{pmatrix} \sum_{n_i=1}^{N_i} \phi_i(\mathbf{x}_{n_i}^*)\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*)\cdot \\ \sum_{n_i=1}^{N_i} \sum_{n_j=1}^{N_j} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*)\delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*)\cdot \\ \prod_{l\in NE(i), l\neq j} \sum_{n_i=1}^{N_i} \omega_{li,n_i}^{(t-1)}\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*) \end{pmatrix} \\
&= \sum_{\mathbf{x}_i} \begin{pmatrix} \sum_{n_i=1}^{N_i} \sum_{n_j=1}^{N_j} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*)\delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*)\cdot \\ \sum_{n_i=1}^{N_i} \phi_i(\mathbf{x}_{n_i}^*) \prod_{l\in NE(i), l\neq j} \omega_{li,n_i}^{(t-1)}\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*) \end{pmatrix} \\
&= \sum_{\mathbf{x}_i} \left(\sum_{n_i=1}^{N_i} \sum_{n_j=1}^{N_j} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\phi_i(\mathbf{x}_{n_i}^*) \prod_{l\in NE(i), l\neq j} \omega_{li,n_i}^{(t-1)}\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*)\delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \right) \\
&= \sum_{n_j=1}^{N_j} \left(\sum_{\mathbf{x}_i} \sum_{n_i=1}^{N_i} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\phi_i(\mathbf{x}_{n_i}^*) \prod_{l\in NE(i), l\neq j} \omega_{li,n_i}^{(t-1)}\delta(\mathbf{x}_i - \mathbf{x}_{n_i}^*) \right) \delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \\
&= \sum_{n_j=1}^{N_j} \left(\sum_{n_i=1}^{N_i} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\phi_i(\mathbf{x}_{n_i}^*) \prod_{l\in NE(i), l\neq j} \omega_{li,n_i}^{(t-1)} \right) \delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*)
\end{aligned} \tag{5.18}$$

So the message $m_{ij}^{(t)}(\mathbf{x}_j)$ can be written in the following form:

$$m_{ij}^{(t)}(\mathbf{x}_j) = \sum_{n_j=1}^{N_j} \omega_{ij,n_j}^{(t)}\delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \tag{5.19}$$

where $\omega_{ij,n_j}^{(t)}$ is given by:

$$\omega_{ij,n_j}^{(t)} = \sum_{n_i=1}^{N_i} \phi_{ij}(\mathbf{x}_{n_i}^*, \mathbf{x}_{n_j}^*)\phi_i(\mathbf{x}_{n_i}^*) \prod_{l\in NE(i), l\neq j} \omega_{li,n_i}^{(t-1)} \tag{5.20}$$

The new message $m_{ij}^{(t)}(\mathbf{x}_j)$ will be transmitted over the wireless channel. $2N_j$ values have to be sent. They are the position of the samples, i.e., $\{\mathbf{x}_{n_j}^*\}_{n_j=1}^{N}$ and the associated weights $\{\omega_{ij,n_j}^{(t)}\}_{n_j=1}^{N}$. As discussed before, a suitable approximation requires dense sampling, i.e., n_j is a big number. Transmission of such a large number of values consumes a lot of power, which should be avoided in order to elongate the life of the sensor node.

We propose two algorithms. The simplified transmission based on Fourier series approximation (ST-FSA) algorithm uses Fourier series expansion only to reduce the transmission power. It can be regarded as a data compression method. The simplified transmission and computation based on Fourier domain belief propagation (SCT-FDBP) algorithm does all the calculation in the frequency domain thus reducing both the transmission power and the computational complexity.

5.5.2 Simplified transmission based on Fourier series approximation

In the ST-FSA algorithm, all the message computation is carried out in the original space domain, i.e., using (5.18). We observed in numerical simulations that in the Fourier series expansion of messages, many components have considerably small zeroth order coefficients. Based on this observation, we can use a Fourier series expansion to compress the message. Given a message $m_{ij}^{(t)}(\mathbf{x}_j)$ that is obtained from (5.18), we write its Fourier series approximation $m_{ij}^{(t),\mathtt{F}}(\mathbf{x}_j)$ as follows:

$$m_{ij}^{(t),\mathtt{F}}(\mathbf{x}_j) = \sum_{n_j=1}^{N_j} \left(\sum_{n_l=1}^{N_j} \gamma_{m_{ij}}(\mathbf{f}_{n_l}^*) \exp(2j\pi {\mathbf{x}_{n_j}^\star}^{\mathtt{T}} \mathbf{f}_{n_l}^*) \right) \delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \tag{5.21}$$

where the Fourier coefficients $\{\gamma_{m_{ij}}(\mathbf{f}_{n_l}^*)\}_{n_l=1}^{N_j}$ can be computed efficiently using N_j points fast Fourier transform on (5.19).

In the expression in (5.21), there are $2N_j$ parameters. We use the component reduction method introduced in Section 4.2.1.4 to truncate the Fourier series. Suppose there are R_j components left, then the new Fourier series approximation is given by:

$$m_{ij}^{(t),\mathtt{rF}}(\mathbf{x}_j) = \sum_{n_j=1}^{N_j} \left(\sum_{r_j=1}^{R_j} \gamma_{m_{ij}}(\mathbf{f}_{r_j}^{'}) \exp(2j\pi {\mathbf{x}_{n_j}^\star}^{\mathtt{T}} \mathbf{f}_{r_j}^{'}) \right) \delta(\mathbf{x}_j - \mathbf{x}_{n_j}^*) \tag{5.22}$$

where $m_{ij}^{(t),\mathtt{rF}}(\mathbf{x}_j)$ denotes the Fourier series approximation after component reduction, $\{\mathbf{f}_{r_j}^{'}\}_{r_j=1}^{R_j} \subset \{\mathbf{f}_{n_j}^*\}_{n_j=1}^{N_j}$ are the remaining components and the associated coefficients $\{\gamma_{m_{ij}}(\mathbf{f}_{r_j}^{'})\}_{r_j=1}^{R_j}$ are obtained by using the formulas in Section 4.2.1.4. If $R_j \ll N_j$, transmission of $m_{ij}^{(t),\mathtt{rF}}(\mathbf{x}_j)$ consumes much less power than that of $m_{ij}^{(t)}(\mathbf{x}_j)$. Based on this idea, we developed the ST-FSA algorithm, i.e., simplified transmission based on Fourier series approximation. The algorithm is summarized in Table 5.1

Table 5.1. Description of ST-FSA Algorithm

ST-FSA
1) Discretize the local potential functions. 2) Initialize messages with uniform distributions. 3) Calculate the outgoing message using (5.18). Use FFT to transform the outgoing message into the frequency domain and use the coefficient reduction method introduced in Section 4.2.1 to reduce the size of the messages. 4) Once a new message (represented by Fourier coefficients) is received, an IFFT will be used to change the message to the 2D space domain for the SPA. 5) Run SPA for a defined number of iterations. 6) The posterior probability distribution can be calculated by using (3.8).

5.5.3 Simplified computation and transmission based on FDBP

In the ST-FSA algorithm, Fourier series approximation is only used for compressing the message. It does not simplify the computation in (5.18). All the message calculations are carried out in the original space domain. FFT and IFFT are required before the message transmission and after the message reception, which increases the computational complexity. As shown in Section 4.2, message computation can also be done in the Fourier domain. In order to do that, we first find the Fourier series expansion for the single potential function and the pairwise potential function. From the discretized single potential function in (5.13), we can calculate the Fourier coefficients $\{\mathbf{f}^*_{n_i}, \gamma_i(\mathbf{f}^*_{n_i})\}_{n_1=1}^{N_i}$ using FFT. Removing the components that have small coefficients, we keep only R_i components $\{\mathbf{f}^\star_{r_i}\}_{r_i=1}^{R_i} \subset \{\mathbf{f}^*_{n_i}\}_{n_1=1}^{N_i}$. Note that in order to guarantee the non-negativity of the resulting density function, component reduction should be conducted by the means introduced in Section 4.2.1.4. Then the shortened Fourier series expression of $\phi_i(\mathbf{x}^*_{n_i})$ is given by:

$$\phi_i^{\mathtt{sF}}(\mathbf{x}_i) = \sum_{r_i=1}^{R_i} \gamma_i(\mathbf{f}^\star_{r_i}) \exp(2j\pi \mathbf{x}_i^{\mathrm{T}} \mathbf{f}^\star_{r_i}) \tag{5.23}$$

Using FFT to calculate the Fourier coefficients $\{\mathbf{f}^*_{n_i}, \mathbf{f}^*_{n_j}, \gamma_{ij}(\mathbf{f}^*_{n_i}, \mathbf{f}^*_{n_j})\}$ for $n_i \in \{1, \ldots N_i\}$ and $n_j \in \{1, \ldots N_j\}$ and doing the component reduction, we can obtain a Fourier series expression of $\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ with $r_i \times r_j$ components. The shortened Fourier series expression is given by:

$$\phi_{ij}^{\mathtt{sF}}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{r_i=1}^{R_i} \sum_{r_j=1}^{R_j} \gamma_{ij}(\mathbf{f}^\star_{r_i}, \mathbf{f}^\star_{r_j}) \exp(2j\pi \mathbf{x}_i^{\mathrm{T}} \mathbf{f}^\star_{r_i} + 2j\pi \mathbf{x}_j^{\mathrm{T}} \mathbf{f}^\star_{r_j}) \tag{5.24}$$

Sensor node i calculates its message $m_{ij}^{(t)}$ to node j based on the messages from its neighbors except j using (5.11). Let us suppose that the Fourier series expression of the message that node i received from its neighbor $l (l \neq j)$ is given by:

$$m_{li}^{(t-1),\mathtt{sF}}(\mathbf{x}_i) = \sum_{r_i=1}^{R_i} \gamma_{m_{li}}(\mathbf{f}^\star_{r_i}) \exp(2j\pi \mathbf{x}_i^{\mathrm{T}} \mathbf{f}^\star_{r_i}) \tag{5.25}$$

Then the message from i to j will be calculated by, according to (5.11):

$$m_{ij}^{(t),\mathtt{sF}}(\mathbf{x}_j) \propto \int_{\mathbf{x}_i} \left(\phi_i^{\mathtt{sF}}(\mathbf{x}_i) \phi_{ij}^{\mathtt{sF}}(\mathbf{x}_i, \mathbf{x}_j) \prod_{l \in NE(i), l \neq j} m_{li}^{(t-1),\mathtt{sF}}(\mathbf{x}_i) \right) \tag{5.26}$$

Let us define:

$$\psi_i^{(t-1),\mathtt{sF}}(\mathbf{x}_i) = \phi_i^{\mathtt{sF}}(\mathbf{x}_i) \prod_{l \in NE(i), l \neq j} m_{li}^{(t-1),\mathtt{sF}}(\mathbf{x}_i) \tag{5.27}$$

Then the Fourier series expression of $\psi_i^{(t-1),\mathtt{sF}}(\mathbf{x}_i)$ can be produced by the multiplication operation of Fourier series introduced in Section 4.2.1.2.

Using (5.27), message calculation in (5.26) becomes:

$$m_{ij}^{(t),\mathtt{sF}}(\mathbf{x}_j) \propto \int_{\mathbf{x}_i} \left(\phi_{ij}^{\mathtt{sF}}(\mathbf{x}_i, \mathbf{x}_j)\psi_i^{(t-1),\mathtt{sF}}(\mathbf{x}_i)\right) \tag{5.28}$$

Then we can use the integration operation of Fourier series introduced in Section 4.2.1.3 to calculate the Fourier series expression of $m_{ij}^{(t),\mathtt{sF}}(\mathbf{x}_j)$. After normalization and component reduction, $m_{ij}^{(t),\mathtt{sF}}(\mathbf{x}_j)$ can be finally expressed as:

$$m_{ij}^{(t),\mathtt{sF}}(\mathbf{x}_j) = \sum_{r_j=1}^{R'_j} \gamma_{m_{ij}}(\mathbf{f}_{r_j}^{\star}) \,\mathsf{exp}(2j\pi \mathbf{x}_j^{\mathrm{T}} \mathbf{f}_{r_j}^{\star}) \tag{5.29}$$

where R'_j is the final number of Fourier components in message $m_{ij}^{(t),\mathtt{sF}}(\mathbf{x}_j)$.

In this way, when we calculate a new message, we just calculate the coefficients of its Fourier series. We call it belief propagation in the Fourier domain. The reduced number of Fourier components together with their coefficients then will be transmitted to the destination node. By doing this, we not only reduce the number of values that should be transmitted between nodes, but also reduce the values involved in the message computation. We give this method the name SCT-FDBP algorithm, i.e., simplified computation and transmission based on Fourier domain belief propagation. Table 5.2 summarizes SCT-FDBP Algorithm for sensor localization.

Table 5.2. Description of SCT-FDBP Algorithm

SCT-FDBP
1) Discretize the local potential functions. 2) Initialize messages with uniform distributions. 3) Use FFT to transform all messages and potential functions to frequency domain. Use coefficient reduction method (Sect. 4.2.1.4) to reduce the number of Fourier components. All messages stay in frequency domain until the end of the algorithm. 4) Implement the sum-product algorithm by using Algorithm 1 and Algorithm 2 in Section 4.2.1. Coefficient reduction is done in each step. 5) Run SPA for a defined number of iterations. 6) Finally, use IFFT to convert the posterior probability distribution from frequency domain into space domain.

The ST-FDA and the SCT-FDBP algorithms implement belief propagation using truncated Fourier series approximations. The compact Fourier series will be transmitted in both cases. The ST-FDA algorithm implements the message computation in the original space domain and the SCT-FDBP algorithm implements the message computation in the Fourier domain. Later on, we will see that both algorithms achieve comparable approximation qualities.

5.6 Non-parametric belief propagation for sensor localization

As explained in Section 4.2.2, messages are represented by Gaussian mixture models in non-parametric belief propagation. Assume now sensor node i wants to calculate its message to sensor node j based on the messages it received from all the other neighbors $\{l|l \in NE(i), l \neq j\}$. Let us use $\mathbf{m}_{ij}^{(t),\text{NBP}}(\mathbf{x}_j)$ to denote the belief message that sensor node i sends to sensor node j at the t^{th} iteration. According to (5.11), message $\mathbf{m}_{ij}^{(t),\text{NBP}}(\mathbf{x}_j)$ is calculated by:

$$\mathbf{m}_{ij}^{(t),\text{NBP}}(\mathbf{x}_j) \xleftarrow{\text{K-GMM}} \alpha \int_{\mathbf{x}_i} \phi_i(\mathbf{x}_i)\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j) \prod_{l \in NE(i), l \neq j} m_{li}^{(t-1),\text{NBP}}(\mathbf{x}_i) d\mathbf{x}_i \tag{5.30}$$

In (5.30), $\phi_i(\mathbf{x}_i)$ and $\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ are analytical functions. Each input message is represented by a K-component Gaussian mixture model, which takes the following form:

$$\mathbf{m}_{li}^{(t-1),\text{NBP}}(\mathbf{x}_i) = \sum_{k=1}^{K} \omega_{li}^{(t-1),(k)} \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_{li}^{(t-1),(k)}, \boldsymbol{\Lambda}_{li}^{(t-1),(k)}) \tag{5.31}$$

where $\boldsymbol{\mu}_{li}^{(t-1),(k)}$ and $\boldsymbol{\Lambda}_{li}^{(t-1),(k)}$ are the mean and covariance of each Gaussian component. $\omega_{li}^{(t-1),(k)}$ is the weight associated with the k^{th} Gaussian component. The operation $h(x) \xleftarrow{\text{K-GMM}} g(x)$ generates a K-components Gaussian mixture model $h(x)$ from the analytical function $g(x)$.

Let us define

$$\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i) \xleftarrow{\text{K-GMM}} \prod_{l \in NE(i), l \neq j} m_{li}^{(t-1),\text{NBP}}(\mathbf{x}_i) \tag{5.32}$$

so that $\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i)$ is a K-component Gaussian mixture model that is generated from the product of input messages.

In Algorithm 4 in Section 4.2.2, it has been explained how to generate a new Gaussian mixture with K components from the product of several K-components Gaussian mixtures.

Let us write $\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i)$ as:

$$\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i) = \sum_{k=1}^{K} \omega_{\psi_{ij}^{(t-1),\text{BNP}}}^{(k)} \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_{\psi_{ij}^{(t-1),\text{BNP}}}^{(k)}, \boldsymbol{\Lambda}_{\psi_{ij}^{(t-1),\text{BNP}}}^{(k)}) \tag{5.33}$$

Using Algorithm 4, $\omega_{\psi_{ij}^{(t-1),\text{BNP}}}^{(k)}$ $\boldsymbol{\mu}_{\psi_{ij}^{(t-1),\text{BNP}}}^{(k)}$ and $\boldsymbol{\Lambda}_{\psi_{ij}^{(t-1),\text{BNP}}}^{(k)}$ can be calculated. Now we use $\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i)$ to approximate $\prod_{l \in NE(i), l \neq j} m_{li}^{(t-1),\text{NBP}}(\mathbf{x}_i)$ and change the formula of message calculation in (5.30) as follows:

$$\mathbf{m}_{ij}^{(t),\text{NBP}} \xleftarrow{\text{K-GMM}} \alpha \int_{\mathbf{x}_i} \phi_i(\mathbf{x}_i)\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j)\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i) d\mathbf{x}_i \tag{5.34}$$

According to (5.8), the pairwise potential function $\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ takes two different forms depending on the value of o_{ij}. Therefore, the message calculation in (5.34) falls into the following two cases.

- If sensor node i detects the signal from sensor node j, i.e., $o_{ij} = 1$ and

$$\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = p(o_{ij}|\mathbf{x}_i, \mathbf{x}_j)p(d_{ij}|\mathbf{x}_i, \mathbf{x}_j) \tag{5.35}$$

$\mathbf{m}_{ij}^{(t),\text{NBP}}$ can be generated based on Algorithm 3 described in Section 4.2.2. We first generate K random samples $\{\mathbf{x}_i^{(k)}\}_{k=1}^K$ from $\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i)$ and calculate the associated weights:

$$\omega_i^{(k)} = \psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i^{(k)}) \tag{5.36}$$

For each $\mathbf{x}_i^{(k)}$, generate a sample $\mathbf{x}_j^{(k)}$ from the distribution $\phi_{ij}(\mathbf{x}_i^{(k)}, \mathbf{x}_j)$. The factor $p(d_{ij}|\mathbf{x}_i, \mathbf{x}_j)$ in (5.35) is generated from the measurement model in (5.1). So the sample $\mathbf{x}_j^{(k)}$ can be generated by:

$$\mathbf{x}_j^{(k)} = \mathbf{x}_i^{(k)} + (d_{ij} + \nu_j^{(k)})[\cos(\theta_j^{(k)}), \sin(\theta_j^{(k)})]^{\mathrm{T}} \tag{5.37}$$

where $\nu_j^{(k)}$ is a random variable generated from the distribution of the measurement noise, i.e., $\mathcal{N}(0, \sigma^2)$ and $\theta_j^{(k)}$ is a random variable generated from a uniform distribution $\mathcal{U}(0, \frac{1}{2\pi})$. So the distance between $\mathbf{x}_j^{(k)}$ and $\mathbf{x}_i^{(k)}$ is controlled by $(d_{ij} + \nu_j^{(k)})$ and the orientation of $\mathbf{x}_j^{(k)}$ with respect to $\mathbf{x}_i^{(k)}$ is controlled by the vector $[\cos(\theta_j^{(k)}), \sin(\theta_j^{(k)})]^{\mathrm{T}}$. The weight associated with each sample $\mathbf{x}_j^{(k)}$ is calculated by:

$$\omega_j^{(k)} = \omega_i^{(k)}\phi_i(\mathbf{x}_i^{(k)})\phi_{ij}(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}) \tag{5.38}$$

and normalized by:

$$\omega_j^{(k)} = \frac{\omega_j^{(k)}}{\sum_{k=1}^K \omega_j^{(k)}} \tag{5.39}$$

Suppose the K-component Gaussian mixture $\mathbf{m}_{ij}^{(t),\text{NBP}}$ takes the following form:

$$\mathbf{m}_{ij}^{(t),\text{NBP}} = \sum_{k=1}^K \omega_{ij}^{(t),(k)}\mathcal{N}(\mathbf{x}_j; \boldsymbol{\mu}_{ij}^{(t),(k)}, \boldsymbol{\Lambda}_{ij}^{(t),(k)}) \tag{5.40}$$

We use $\mathbf{x}_j^{(k)}$ obtained in (5.37) as the mean $\boldsymbol{\mu}_{ij}^{(t),(k)}$ in each Gaussian components and use $\omega_j^{(k)}$ obtained in (5.39) as the weight of each Gaussian component, i.e., we set:

$$\boldsymbol{\mu}_{ij}^{(t),(k)} = \mathbf{x}_j^{(k)} \tag{5.41}$$

and

$$\omega_{ij}^{(t),(k)} = \omega_j^{(k)} \tag{5.42}$$

At the end, we choose an appropriate covariance matrix $\boldsymbol{\Lambda}_{ij}^{(t),(k)}$ for each Gaussian component.

- If sensor node i does not detect the signal from sensor node j, i.e., $o_{ij} = 0$, then the pairwise potential function is given by:

$$\phi_{ij}(\mathbf{x}_i, \mathbf{x}_j) = 1 - p(o_{ij}|\mathbf{x}_i, \mathbf{x}_j) \tag{5.43}$$

In this case, $\mathbf{m}_{ij}^{(t),\text{NBP}}(\mathbf{x}_j)$ can be generated based on Algorithm 3 described in Section 4.2.2. We first generate K random samples $\{\mathbf{x}_i^{(k)}\}_{k=1}^K$ from $\psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i)$ and calculate the associated weights:

$$\omega_i^{(k)} = \psi_{ij}^{(t-1),\text{BNP}}(\mathbf{x}_i^{(k)}) \tag{5.44}$$

For each $\mathbf{x}_i^{(k)}$, generate a sample $\mathbf{x}_j^{(k)}$ from the distribution $\phi_{ij}(\mathbf{x}_i^{(k)}, \mathbf{x}_j)$,i.e.,

$$\mathbf{x}_j^{(k)} \sim \phi_{ij}(\mathbf{x}_i^{(k)}, \mathbf{x}_j) \tag{5.45}$$

Then we calculate the weights for each sample by:

$$\omega_j^{(k)} = \omega_i^{(k)} \phi_i(\mathbf{x}_i^{(k)}) \phi_{ij}(\mathbf{x}_i^{(k)}, \mathbf{x}_j^{(k)}) \tag{5.46}$$

and normalize weights:

$$\omega_j^{(k)} = \frac{\omega_j^{(k)}}{\sum_{k=1}^K \omega_j^{(k)}} \tag{5.47}$$

Suppose the K-component Gaussian mixture $\mathbf{m}_{ij}^{(t),\text{NBP}}$ takes the following form:

$$\mathbf{m}_{ij}^{(t),\text{NBP}} = \sum_{k=1}^{K} \omega_{ij}^{(t),(k)} \mathcal{N}(\mathbf{x}_j; \boldsymbol{\mu}_{ij}^{(t),(k)}, \boldsymbol{\Lambda}_{ij}^{(t),(k)}) \tag{5.48}$$

We use $\mathbf{x}_j^{(k)}$ obtained in (5.45) as the mean $\boldsymbol{\mu}_{ij}^{(t),(k)}$ in each Gaussian components and use $\omega_j^{(k)}$ obtained in (5.47) as the weight of each Gaussian component, i.e., we set:

$$\boldsymbol{\mu}_{ij}^{(t),(k)} = \mathbf{x}_j^{(k)} \tag{5.49}$$

and

$$\omega_{ij}^{(t),(k)} = \omega_j^{(k)} \tag{5.50}$$

At the end, we choose an appropriate covariance matrix $\boldsymbol{\Lambda}_{ij}^{(t),(k)}$ for each Gaussian component.

5.7 Simulation results

To verify the performance of the belief propagation methods introduced in the previous sections, we simulate belief propagation algorithms for self-localization problem.

We first use a simple scenario to illustrate the function approximation. Then we use a large scale sensor network to evaluate and compare the localization performance obtained by different algorithms.

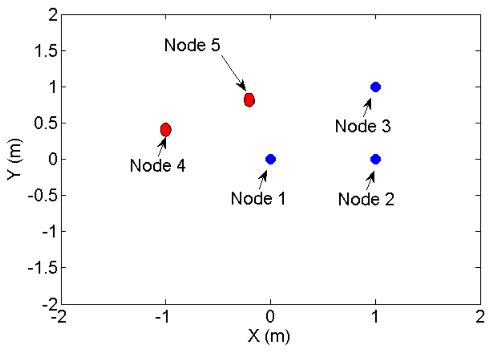


Fig. 5.3. Sensor distribution

5.7.1 A simple scenario

We first run the simulation in a wireless sensor network that is illustrated in Figure 5.3.

The positions of sensor nodes 1, 2 and 3 are known as (0, 0), (1, 0) and (1, 1) respectively. Unknown sensor nodes 4 and 5 are located at (-1, 0.4) and (-0.2, 0.8). In this simulation setting, we limit the area to $[-\pi, \pi]^2$ for simplicity.

The parameters ρ and R in (5.2) are set to 2 and 3m respectively. The standard deviation of distance measurements σ in (5.1) is set to 0.4m. The belief propagation algorithm is forced to stop after 7 iterations.

Figure 5.4 illustrates the message approximation by Fourier series approximation and random sample based approximation (Monte Carlo method) using different number of components to represent a single potential function or a message. We use the contour lines and the darkness of the color to represent the values of the density functions, i.e., the darker the color, the larger the value of the density function at the corresponding point. The results are compared with the true result generated by uniform sampling based method. The sampling resolution is 65x65 for all experiments. For the Fourier series approximations, we implemented both ST-FDA algorithm and the SCT-FDBP algorithm. The results depicted in Figure 5.4 demonstrate equivalent performance of these two implementations of Fourier series approximation. In the subsequent simulations, we will only show the results of the SCT-FDBP algorithm.

From the results in Figure 5.4 we can see that with 49 Fourier components, the approximation is already very close to the true distribution, whereas too few (e.g., 16) components cannot fully characterize the very non-linear, non-Gaussian distribution. It is also the case for random sample based method. The more the components, the better the approximation. In general, with the same number of components, Fourier series based density approximation outperforms the Monte Carlo method. This is because the Fourier series approximation analyses the structure of the function whereas the other method is based on randomly generated samples.

Figure 5.5 shows the approximation results from Fourier series approximation using different sampling resolutions. Sampling resolution of 15x15, 30x30, 45x45, 65x65 are applied to Figure 5.5 (a) to (d). The sampling resolution determines the precision of the estimate. According to *Nyquist Theorem*, the original function can be recovered from its samples only if the sampling rate is greater than twice the maximum frequency of that function. Bad results can be observed from Figure 5.5(a) because the sampling rate is too low.

Figure 5.6 depicts the density function, recovered by interpolating the random samples with Gaussian basis functions with different choices of the bandwidth, i.e., variance of each Gaussian component. In this simulation, for a given set of random samples with value $\{\boldsymbol{\mu}(k)\}_{k=1}^{M}$ and associated weights $\{\omega^{(k)}\}_{k=1}^{M}$, the variance is computed by:

$$\boldsymbol{\Lambda}^{(k)} = \frac{\sqrt{\sum_{k=1}^{M} \omega^{(k)} \|\boldsymbol{\mu}^{(k)} - \bar{\boldsymbol{\mu}}\|^2}}{M^{\frac{1}{\alpha}}} \tag{5.51}$$

where

$$\bar{\boldsymbol{\mu}} = \frac{1}{M} \sum_{k=1}^{M} \boldsymbol{\mu}^{(k)} \tag{5.52}$$

is the mean value of the centers of the Gaussian components and α is a parameter which controls the smoothness of the resulting Gaussian mixture model. In Figure 5.6, we present the result produced when α is chosen to be 2, 3, 4 and 6. It can be seen that if the variance of each Gaussian component is too small, i.e., α is small, the resulting Gaussian mixture function is discontinuous (Figure 5.6(a)). If the variance is too big, the details of the functions cannot be characterized (Figure 5.6(d)).

5.7.2 Simulation in a large scale sensor network

Having illustrated the function approximations through the simple example, now we apply belief propagation methods to a sensor network with a larger scale. We randomly spread 30 sensor nodes in a given region. 9 of them are supposed to be pre-calibrated, i.e., they already have a precise prior knowledge on their locations. In the simulation, we use maximally 64 components for Fourier density approximation and for Gaussian mixtures. Sampling resolution is 65×65. Other parameters are the same as those used in the simple example.

Figure 5.7 to Figure 5.10 depicts the localization performance achieved by each method after different number of iterations. Plus signs represent the pre-calibrated nodes. Circles represent the true position of the nodes that should be localized, which are connected with their corresponding estimates (solid dots) via straight lines. The length of the line reveals the magnitude of the error made in the estimation.

From the results we can see that the belief propagation stabilizes after 4 iterations, which shows that the self-organized localization converges very fast. Fourier domain belief propagation achieves a very similar result as the uniform sampling based belief propagation while non-parametric belief propagation is not as good as the other two. The result also shows that nodes that are close to the pre-calibrated nodes can be precisely localized.

The averaged communication costs, approximated by the number of parameters that is required by different methods to represent the messages throughout the entire simulation are compared in Table 5.3. The results are obtained from 10 simulation runs of a sensor network with 30 nodes. From the results we can see that Fourier series approximations and non-parametric belief propagations greatly reduce the transmission costs.

method	total number of samples transmitted
uniform sampling	1022307
SCT-FDBP	17445
NBP	35200

Table 5.3. Transmission cost of different methods

5.8 Discussions

In this chapter, we studied a self-organized sensor localization problem. Using belief propagation, we implemented message passing algorithms for the computation of the posterior probability distribution of the sensor positions conditioned on the measurements of the mutual distances. Due to the non-linearity of the problem and due to the very complicated distribution functions, we introduce approximations to simplify the algorithms. Most important, the communication cost for the message passing should be minimized to make it possible to implement the algorithm on sensors with self-contained power supplies. We introduced uniform sampling, Fourier series and Monte Carlo methods for the message and function approximation. Uniform sampling does not exploit the special structure of the data. To achieve a good performance, the density functions have to be densely sampled. Fourier series approximation uses truncated Fourier series to represent the density functions. Non-parametric belief propagation use random samples to represent the density function. The simulation results have shown that Fourier and random sample based method can greatly reduce the transmission cost while keeping good localization performance.

In this chapter, we focus on the suitable message representation for the belief propagation and the results are obtained from software simulation. In practice, there are many other issues to be considered, for example, the message compression. We presented compacted representation of the density functions or belief messages. Such representations should be compressed before the transmission. The compression process involves quantization of the continuous values and source coding. The compression performance also greatly influence the transmission cost and the computational complexity. The value listed in Table 5.3 illustrates the transmission cost without considering the compression. Therefore, further research can be conducted to study the number of bits for the transmission required by different algorithms.

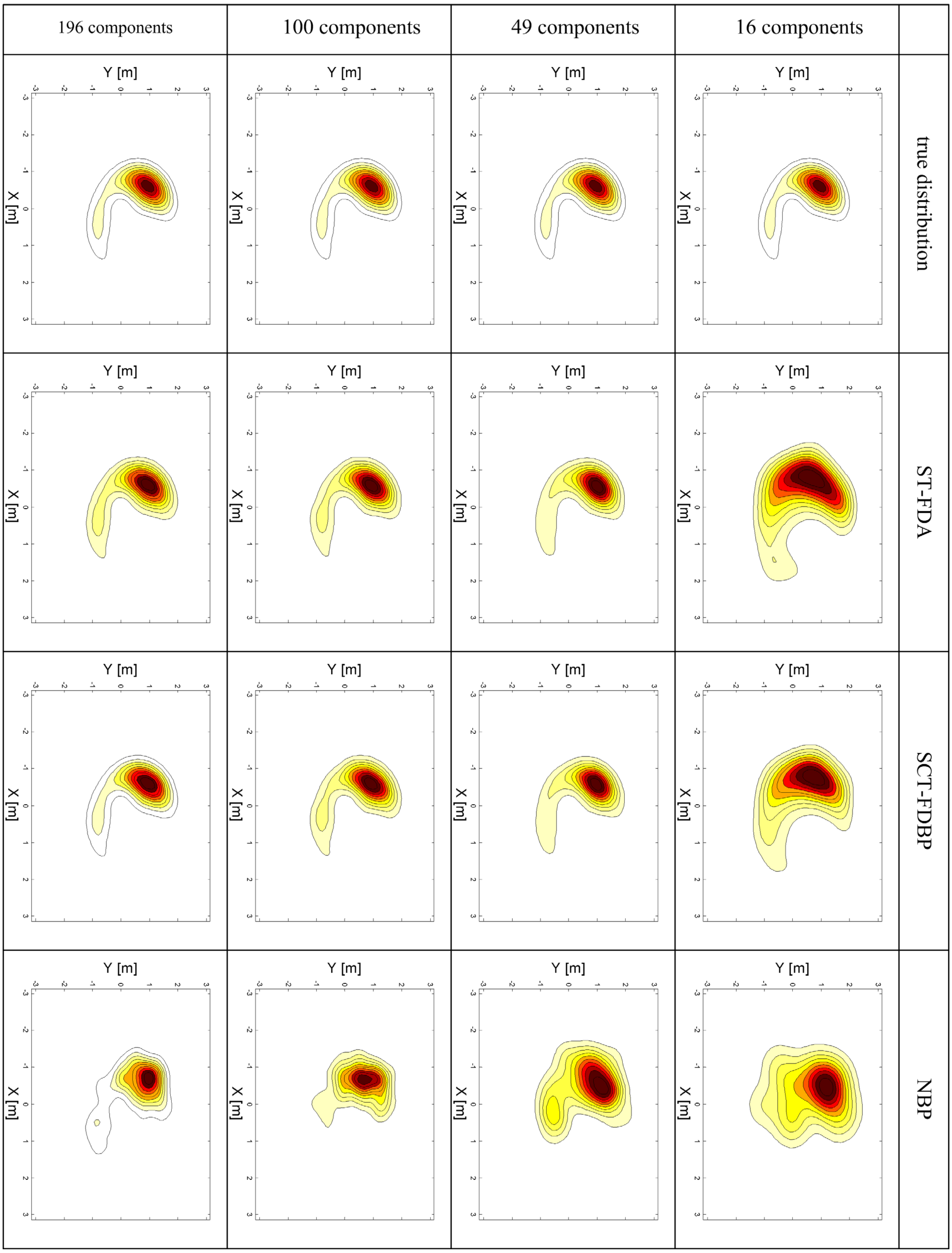


Fig. 5.4. Comparison of different approximation methods

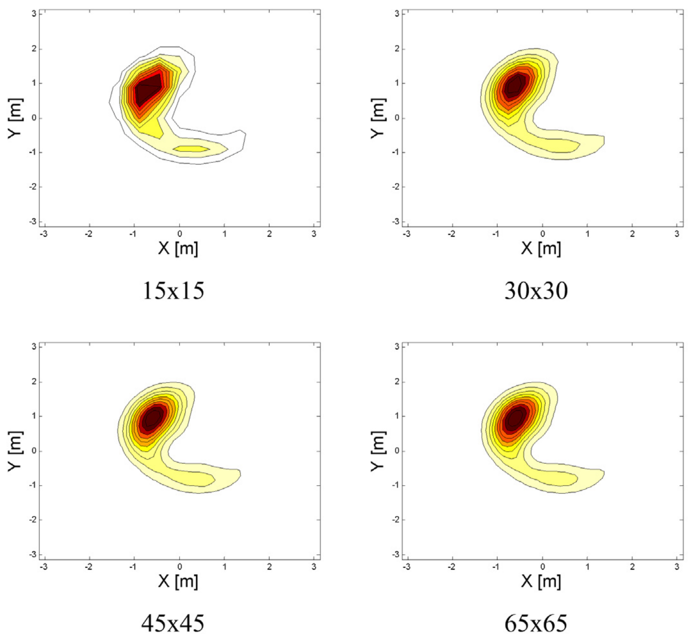


Fig. 5.5. Results of Fourier series approximation with different sampling rate

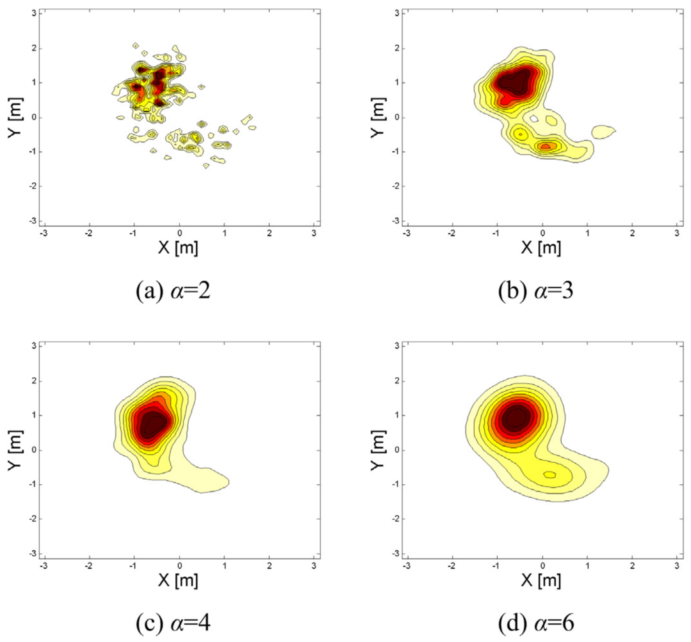
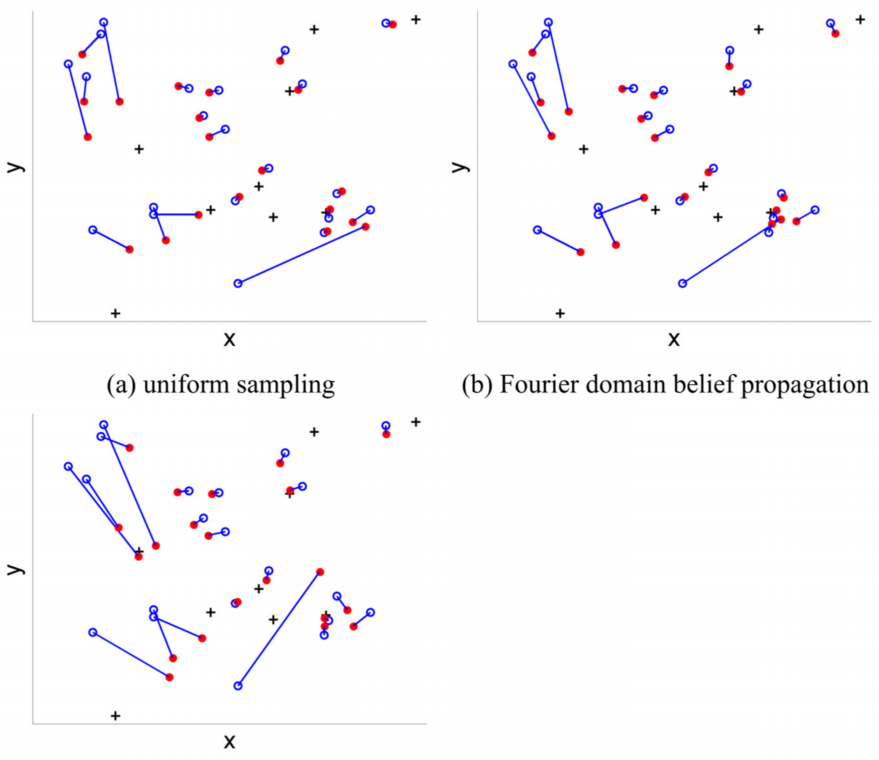


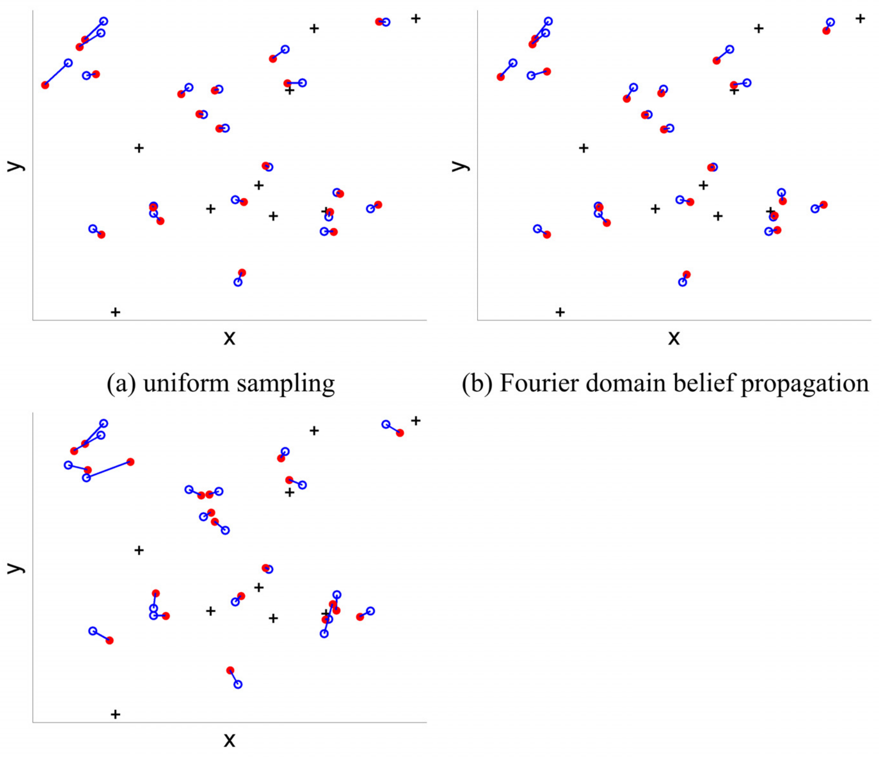
Fig. 5.6. Results of Gaussian interpolation with different variances



(a) uniform sampling (b) Fourier domain belief propagation

(c) Non-parametric belief propagation

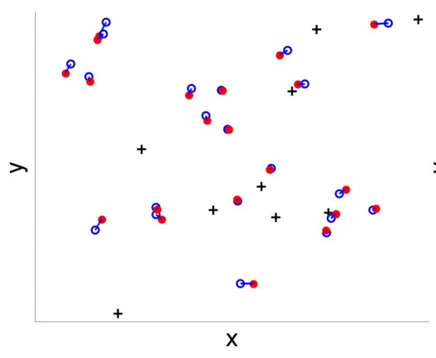
Fig. 5.7. The localization results after 1 iteration



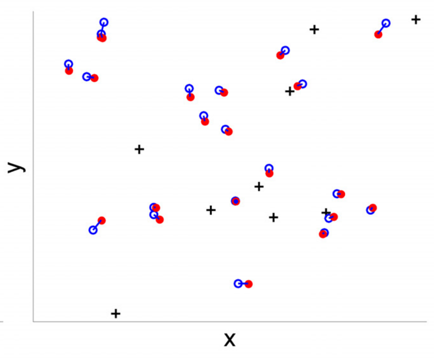
(a) uniform sampling (b) Fourier domain belief propagation

(c) Non-parametric belief propagation

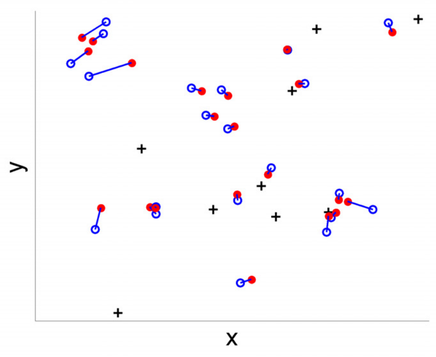
Fig. 5.8. The localization results after 2 iterations



(a) uniform sampling

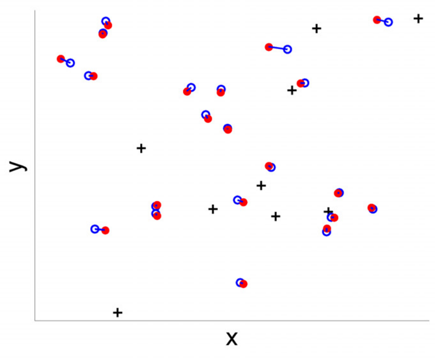


(b) Fourier domain belief propagation

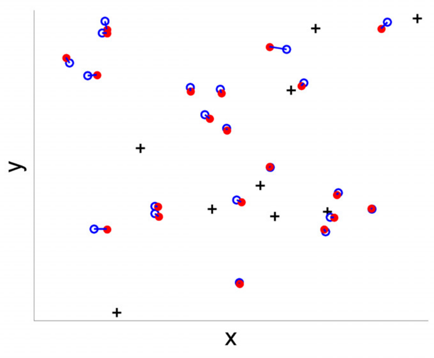


(c) Non-parametric belief propagation

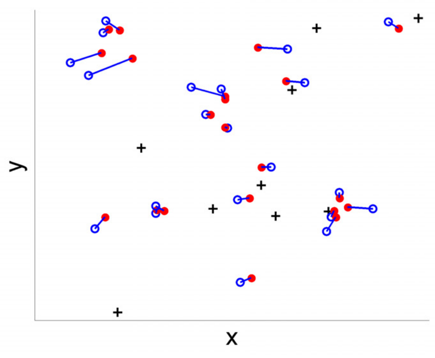
Fig. 5.9. The localization results after 4 iterations



(a) uniform sampling



(b) Fourier domain belief propagation



(c) Non-parametric belief propagation

Fig. 5.10. The localization results after 7 iterations

6. Clock Synchronization Of Networked Nodes

Ethernet [43], due to its cheap cabling and infrastructure costs, high bandwidth, efficient switching technology and good interoperability, has been adopted in various areas to provide the basic networking solution. Many Ethernet-based applications require the networked clocks to be precisely synchronized. Typical examples include synchronization of base stations for hand-over or interference cancellation [87] in telecommunication networks, distribution of audio / video streams over Ethernet based networks [48], and motion control in industrial Ethernet [14]. Standard Network Time Protocol (NTP) [69, 70] synchronization over Ethernet provides a time accuracy at the millisecond level, which is enough for processes that are not time critical. However, in many applications, for example base station synchronization or motion control where only sub-microsecond level synchronization errors are allowed (the so-called isochronous mode), a more accurate solution is needed. The Precision Time Protocol (PTP), specified by the IEEE 1588 standard [45] published in 2002, constitutes a promising Ethernet synchronization protocol, in which messages carrying precise timing information, obtained via hardware time stamping in the physical layer, are propagated in the network to synchronize the slave clocks to a master clock. Boundary clocks [45] adjust their own clock to the master clock and then serve as masters for the next network segment. Authors of [51] introduced the transparent clock (TC) concept, in which intermediate bridges are treated as network components with known delay. By doing this, no control loop in the intermediate element is needed for providing timing information to the next local clock and hence the synchronization at the time client is not dependent on the control loop design in the intermediate bridges. The transparent clock concept has been adopted in the new version of IEEE 1588 published in 2007 [47] and the IEC 61588 standard [50]. Exciting applications of IEEE 1588 can be found in [19].

In industrial automation and manufacturing systems, the line topology is very important. Cabling often leads to networks with line topology. Typically, an industrial automation network may have tens or even hundreds of cascaded elements. It is always desirable to synchronize the clocks of all these elements to a single master clock. However, due to long line length, latency and jitters in the intermediate bridges will greatly influence the achievable synchronization performance. In this chapter, we will focus on the topic of synchronization of cascaded clocks, i.e., synchronizing the clocks in a network with line topology. We assume that all the element in the network are IEEE 1588 compatible and apply the transparent clock concept. Based on the timing information provided by the

PTP protocol, we develop synchronization algorithms that support as many as possible elements that can be synchronized to the master clock.

Factors that affect the synchronization quality achievable by PTP include the stability of oscillators, the resolution and precision of time stamping the message, the frequency of sending synchronization messages, and the propagation delay variation caused by the jitter in the intermediate elements. A comprehensive analytical work has been presented in [96] to show the influence of these factors on the synchronization accuracy. It can be seen from the analytical results that stamping errors, including quantization error, stamping jitters, have very adverse effect because the errors introduced by different elements accumulate along the network. On the other hand, each clock in the network is a dynamic system. Discrete observation of the state of the clocks are obtained by generating time stamps. Clock synchronization can be formulated as a state estimation of dynamic systems based on discrete noisy measurement, i.e., the time stamps. This chapter discusses the probabilistic modeling of clock synchronization and the inference algorithm for state estimation. Through this application, we also demonstrate how important it is to explore the rich context of graphical models for solving practical signal processing problems.

The rest of this chapter is organized as follows. Section 6.1 presents the system model and introduces briefly the synchronization mechanism of PTP protocol. Section 6.2 lists the notations of the variables. Section 6.3 shows the basic parameters for the simulations. We first introduce the standard synchronization, i.e., the one that is already implemented in the industrial Ethernet products, in Section 6.4. Probabilistic modeling and inference algorithms for clock synchronization will be derived in Section 6.5. The performance of different synchronization algorithms are verified and compared through numerical simulations. Results are presented in Section 6.6.

6.1 Background

6.1.1 Characteristics of clocks

A clock consists of an oscillator and a counter. The oscillator should generate repeatable time intervals. By counting this intervals, the clock can establish a time scale that is required by certain applications.

Characterizing oscillators is a very complex topic. We present here a simplified model that will be used for the rest part of this chapter. Comprehensive understanding of the oscillators can be found in [44, 104, 105].

Based on the modeling of the oscillators introduced in [104], the time scale produced by an oscillator can be described by:

$$c(t') = \lfloor c_0 + \int_0^{t'} f(t)dt \rfloor \tag{6.1}$$

where:

- $c(t)$ is the counter value at a given absolute reference time t;

- c_0 is the origin of the time scale;
- $f(t)$ is the evolution of frequency over time;
- $\lfloor \cdot \rfloor$ calculates the floor of a number.

In the ideal case, an oscillator runs at a constant frequency, which is defined as the nominal frequency. However, the true frequency of the oscillator may be affected by different environmental and inherent factors. Environmental effects include the changes in temperature, pressure or supply voltage and inherent effect comes mainly from the aging of the oscillator. So in practice, $f(t)$ can be a very complex function over time. Fortunately, linear approximation of $f(t)$ already provides satisfactory results. In the following parts of this chapter, we will use two types of models:

- constant frequency model

  In this model, the frequency $f(t)$ of the oscillator has a constant value:

  $$f(t) = f \tag{6.2}$$

  Inserting (6.2) into (6.1), we obtain the formula for calculating the counter state at a given time t':

  $$c(t') = \lfloor c_0 + ft' \rfloor \tag{6.3}$$

  Note that although the frequency is a constant value, this value is not necessarily equal to the nominal frequency. That means, two clocks might run with different frequencies although they have the same nominal frequency.

- linear frequency drift model

  In this model, the frequency $f(t)$ of the oscillator drifts with a constant speed Δ. The frequency is given by the following affine equation:

  $$f(t) = f + \Delta t \tag{6.4}$$

  where f represent the initial frequency now. Inserting (6.4) into (6.1), we obtain the formula for calculating the counter state at a given time t':

  $$c(t') = \lfloor c_0 + ft' + \frac{\Delta}{2} t^2 \rfloor \tag{6.5}$$

6.1.2 Network topology

In an IEEE 1588 compatible system, all clocks are organized into a master-slave hierarchy. To enable that, the PTP protocol has to operate on a communication topology that is loop free. In fact, many underlying communication systems run a minimum spanning tree or similar protocol which offers the PTP protocol a tree-structured communication topology. The spanning tree protocol (STP) used in Ethernet systems is defined in IEEE Standard 802.1D [46]. A detailed description of the spanning tree algorithm can be found in [93].

A typical topology of an industrial automation network is shown in Figure 6.1 where dashed and solid lines represent the physical cabling between network elements. Out of all the physical links, the spanning tree algorithm picks the communication links (solid lines in Figure 6.1) to form a tree. Communication over any link that is not in the tree (dashed line in Figure 6.1) will be forbidden by the communication protocol or only used for robustness to the link failure, e.g., Media Redundancy Protocol (MRP).

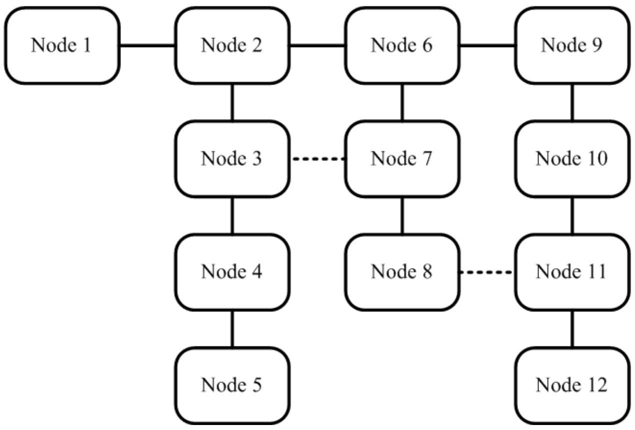


Fig. 6.1. Network topology

6.1.3 PTP messages

In the initial phase, the PTP protocol runs a best master algorithm [47] to elect a master clock which provides the reference time to the other clocks, i.e., slave clocks. In order to synchronize its clock to the reference clock provided by the master clock, a slave has to acquire information about the state of the master clock. In the PTP protocol, clocks exchange timing information by sending PTP messages which contain discrete time stamps. Based on the timing information delivered by the PTP messages, each slave estimates the state of the master clock and thus adjusts its clock to follow the master clock.

Figure 6.2 depicts the PTP messages and the timing information associated with all the messages on a simplified diagram with two elements: a master and a slave. Four types of messages are defined in PTP, they are:

- **Sync message**: the master element sends periodically Sync messages to the directly connected slaves. A time stamp $\overrightarrow{s}_{\mathtt{M}}$ is generated according to the master's local clock when the message leaves. When the slave receives the Sync message, it will generate a time stamp $\overrightarrow{s}_{\mathtt{S}}$ of the receiving time based on the slave's local clock.

- **Follow_Up message**: in order to let the slave know the precise time stamp of the Sync message's sending time, the master element sends a Follow_Up message to the slave which carries the time stamp $\overrightarrow{s}_{\mathtt{M}}$.

- **Delay_Req message**: the slave sends a delay request (Delay_Req) message to the master. The slave clock generates a time stamp $\overleftarrow{s}^{\mathtt{D}}_{\mathtt{S}}$ based on its local clock when the Delay_Req message leaves the slave. The master clock generates a time stamp $\overleftarrow{s}^{\mathtt{D}}_{\mathtt{M}}$ based on its local clock when the Delay_Req message arrives at the master.

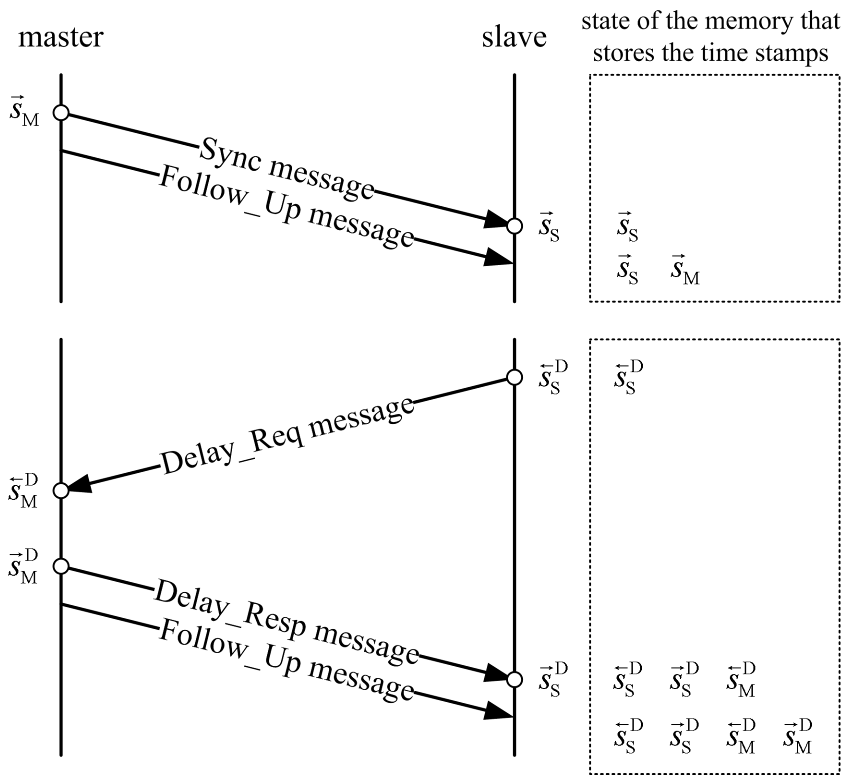


Fig. 6.2. Synchronization messages in PTP protocol

- **Delay_Resp message**: the master sends a delay response (Delay_Resp) message to the slave which carries the receipt time stamp of $\overleftarrow{s}^D_M$. Delay_Resp message is followed by another Follow_Up message which carries the time stamps generated at the transmission of the Delay_Resp, i.e., $\vec{s}^D_M$ to the slave. At the slave side, it stamps the local time when it receives the Delay_Resp message which is $\vec{s}^D_S$.

All the messages will be sent periodically so that at the slave side, it may maintain the updated values of $\vec{s}_M$, $\vec{s}_S$, $\overleftarrow{s}^D_S$, $\overleftarrow{s}^D_M$, $\vec{s}^D_S$ and $\vec{s}^D_M$. The time stamps associated with the Sync message is used by the slave to estimate the master time. The time stamps that are associated with the Delay_Req and Delay_Resp messages are used by the slave to estimate the transmission delay of a message between the master and the slave. The following sections will show how to use these time stamps to achieve the synchronization.

6.1.4 Peer to peer transparent clock

The previous section introduced the basic PTP messaging in a master-slave hierarchy. Such a messaging scheme will be applied to the entire network. It has been discussed in Section 6.1.2 that the PTP works on top of a communication protocol which provides a tree-structured topology. So it is enough to study just one branch of the tree, i.e., from the root (master) to the leaf (the most remote slave). For example in the network shown in Figure 6.1, if node 1 is elected to be the master, then we will study how to synchronize all the elements in the longest line, i.e., $\{1, 2, 6, 9, 10, 11, 12\}$.

Figure 6.3 shows a system with $N+1$ cascaded elements connected in a line topology. The first element is the time source, also called (grand)master, which provides the reference time to the rest N elements, called slave elements.

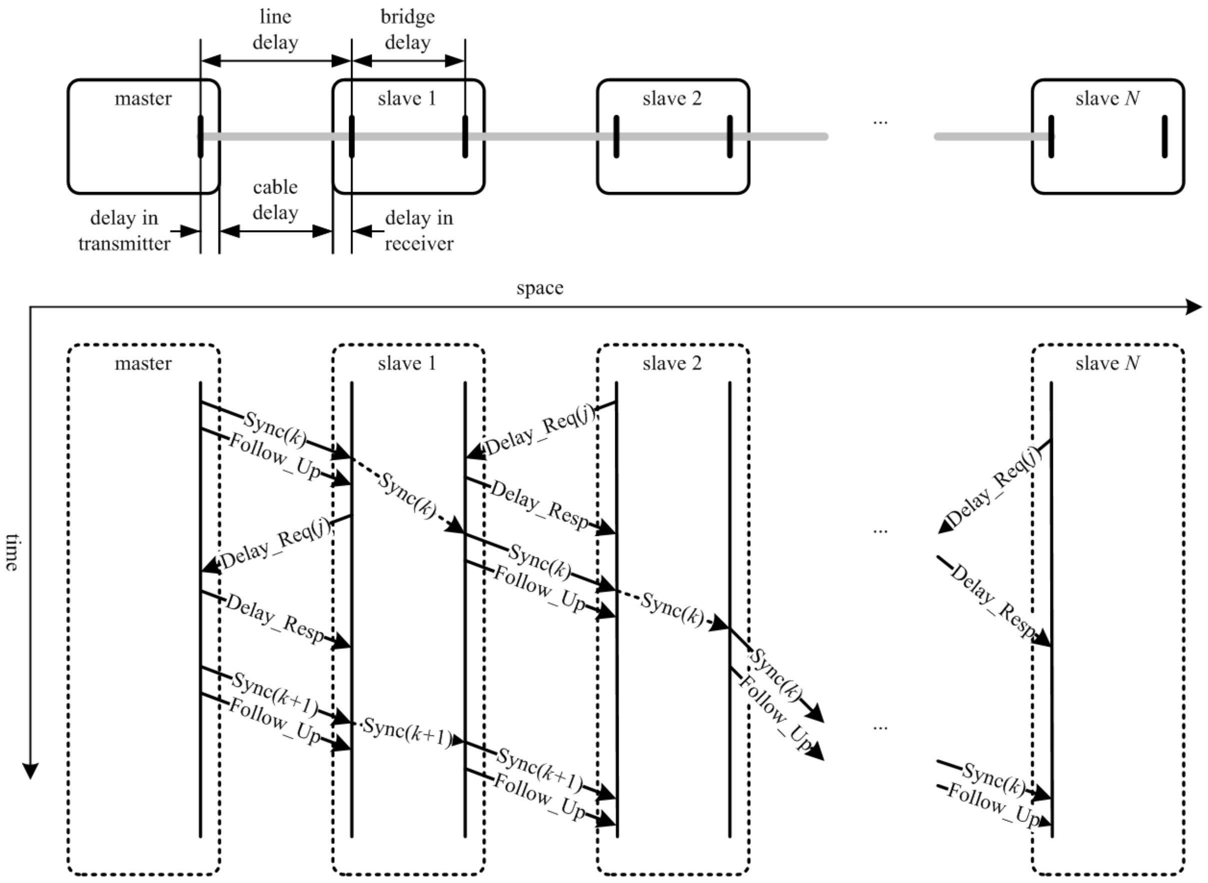


Fig. 6.3. System model and the propagation of PTP messages

Synchronization of the entire network can be implemented in different ways. The following discussion is based on the peer to peer transparent clock concept.

Figure 6.3 also depicts the propagation of the PTP messages along the line. The Sync message, which carries the timing information of the master clock, will propagate along the line until it arrives at the last slave. The line delay $\overrightarrow{d_n^{\mathrm{LD}}}$ is the total propagation between slave n and its upstream (with respect to the propagation of Sync messages) element. A Sync message will be processed at each slave before it can be forwarded to the next slave element. The residence time of a Sync message at slave n is called bridge delay and is denoted by $\overrightarrow{d_n^{\mathrm{BD}}}$. In the peer to peer transparent clock implementation, each slave sends to its neighboring upstream node a Delay_Req message and receives a Delay_Resp message from the upstream node. The four time stamps generated at the transmission and the reception of these two messages, are collected by the requester. Since the delay messages are only exchanged between neighboring nodes and used for estimating the line delay between those nodes, such an implementation is called peer to peer delay estimation. In such a system, the propagation of Sync messages and the line delay estimation can be regarded as two independent processes.

6.1.5 Peer to peer line delay estimation

In order to synchronize the clocks in the entire network, it is important to estimate all the delays, i.e., line delays and bridge delays that a Sync messages experiences. In comparison with the bridge delay estimation, the line delay estimation is more complicated. Here we introduce the estimation of line delay between two neighboring nodes (i.e., an upstream node and it neighboring downstream node). The results will be used in the subsequent sections. Before we introduce the calculation, we first summarize the notations that will appear in the formulas.

- general notations

  $s_{\mathtt{E}}$: a time stamp of the clock of element $\mathtt{E} \in \{\mathtt{U}, \mathtt{D}\}$ where U stands for the upstream element and D stands for the downstream element. If U is master, then D is slave 1. If U is slave n, then D is slave $n + 1$.

  $c_{\mathtt{E}}$: the true local time according to the clock of element E that corresponds to $s_{\mathtt{E}}$

  $\xi_{\mathtt{E}}$: stamping error, i.e.,

  $$s_{\mathtt{E}} = c_{\mathtt{E}} + \xi_{\mathtt{E}} \tag{6.6}$$

  In the following derivations, we assume that $c_{\mathtt{E}}$ is a real number without quantization. The quantization error is included in $\xi_{\mathtt{E}}$ and $s_{\mathtt{E}}$ is always an integer.

- parameters of the clocks

  $c^{\circ}_{\mathtt{U}}$: initial value of the upstream clock

  $c^{\circ}_{\mathtt{D}}$: initial value of the downstream clock

  $f_{\mathtt{U}}$: initial frequency of the upstream clock

  $f_{\mathtt{D}}$: initial frequency of the downstream clock

  $\Delta_{\mathtt{U}}$: speed of frequency drift of the upstream clock

  $\Delta_{\mathtt{D}}$: speed of frequency drift of the downstream clock

- time, counter values and time-stamps that are associated with the peer to peer delay messages (See Figure 6.4).

  $\overleftarrow{t}_{\mathtt{D}}(j)$: the absolute reference time when the j^{th} Delay_Req message is sent by the downstream element. (The true downstream element's local time at this point is $\overleftarrow{c}_{\mathtt{D}}(j)$. A time-stamp of the local clock is generated at this time, which is $\overleftarrow{s}_{\mathtt{D}}(j)$)

  $\overleftarrow{t}_{\mathtt{U}}(j)$: the absolute reference time when the j^{th} Delay_Req message is received by the upstream element. (The true upstream element's local time at this point is $\overleftarrow{c}_{\mathtt{U}}(j)$. A time-stamp of the local clock is generated at this time, which is $\overleftarrow{s}_{\mathtt{U}}(j)$)

  $\overrightarrow{t}_{\mathtt{U}}(j)$: the absolute reference time when the j^{th} Delay_Resp message is sent from the upstream element. (The true upstream element's local time at this point is $\overrightarrow{c}_{\mathtt{U}}(j)$. A time-stamp of the local clock is generated at this time, which is denoted by $\overrightarrow{s}_{\mathtt{U}}(j)$)

  $\overrightarrow{t}_{\mathtt{D}}(j)$: the absolute reference time when the j^{th} Delay_Resp message is received by the downstream element. (The true downstream element's local time at this point is $\overrightarrow{c}_{\mathrm{D}}(j)$. A time-stamp of the local clock is generated at this time, which is denoted by $\overrightarrow{s}_{\mathrm{D}}(j)$)

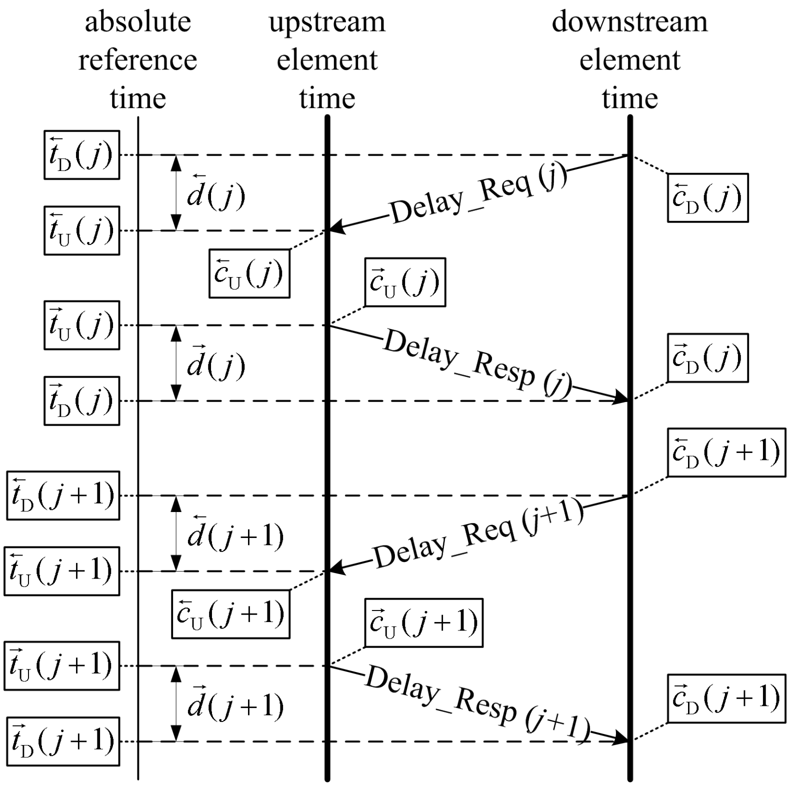

absolute reference time
upstream element time
downstream element time
$\overleftarrow{t}_{\mathrm{D}}(j)$
$\overleftarrow{d}(j)$
$\overleftarrow{t}_{\mathrm{U}}(j)$
$\overleftarrow{c}_{\mathrm{U}}(j)$
$\overleftarrow{c}_{\mathrm{D}}(j)$
Delay_Req (j)
$\overrightarrow{c}_{\mathrm{U}}(j)$
$\overrightarrow{t}_{\mathrm{U}}(j)$
$\overrightarrow{d}(j)$
$\overrightarrow{t}_{\mathrm{D}}(j)$
$\overrightarrow{c}_{\mathrm{D}}(j)$
Delay_Resp (j)
$\overleftarrow{c}_{\mathrm{D}}(j+1)$
$\overleftarrow{t}_{\mathrm{D}}(j+1)$
$\overleftarrow{d}(j+1)$
$\overleftarrow{t}_{\mathrm{U}}(j+1)$
Delay_Req (j+1)
$\overleftarrow{c}_{\mathrm{U}}(j+1)$
$\overrightarrow{c}_{\mathrm{U}}(j+1)$
$\overrightarrow{t}_{\mathrm{U}}(j+1)$
$\overrightarrow{d}(j+1)$
$\overrightarrow{t}_{\mathrm{D}}(j+1)$
Delay_Resp (j+1)
$\overrightarrow{c}_{\mathrm{D}}(j+1)$

Fig. 6.4. Relationships between time variables

Using the peer to peer delay messages, the slave element can estimate the line delay to its upstream element. Based on different clock models, the line delay estimation has different complexity.

6.1.5.1 Clocks with identical constant frequency values

Now we first study the simplest case when two clocks are running with the same frequency, i.e., $f_{\mathrm{U}} = f_{\mathrm{D}} = f_0$ and there is no frequency drift, i.e., $\Delta_{\mathrm{U}} = \Delta_{\mathrm{D}} = 0$. In this case, the evolution of the upstream element's local time is given by, using (6.3):

$$c_{\mathrm{U}}(t) = c_{\mathrm{U}}^{\circ} + f_{\mathrm{U}} \cdot t = c_{\mathrm{U}}^{\circ} + f_0 \cdot t \tag{6.7}$$

and the downstream element's local time is given by:

$$c_{\mathrm{D}}(t) = c_{\mathrm{D}}^{\circ} + f_{\mathrm{D}} \cdot t = c_{\mathrm{D}}^{\circ} + f_0 \cdot t \tag{6.8}$$

Comparing (6.8) with (6.7), the difference between these two local time values is a constant value:

$$o_{\mathrm{UD}} = c_{\mathrm{U}}^{\circ} - c_{\mathrm{D}}^{\circ} \tag{6.9}$$

According to the clock model in (6.7) and (6.8), we have the following relationships between local time and the absolute reference time:

$$\begin{aligned}
\overrightarrow{c}_{\text{U}}(j) = c_{\text{U}}(\overrightarrow{t}_{\text{U}}(j)) = c^{\circ}_{\text{U}} + f_0 \cdot \overrightarrow{t}_{\text{U}}(j) \\
\overrightarrow{c}_{\text{D}}(j) = c_{\text{D}}(\overrightarrow{t}_{\text{D}}(j)) = c^{\circ}_{\text{D}} + f_0 \cdot \overrightarrow{t}_{\text{D}}(j) \\
\overleftarrow{c}_{\text{D}}(j) = c_{\text{D}}(\overleftarrow{t}_{\text{D}}(j)) = c^{\circ}_{\text{D}} + f_0 \cdot \overleftarrow{t}_{\text{D}}(j) \\
\overleftarrow{c}_{\text{U}}(j) = c_{\text{U}}(\overleftarrow{t}_{\text{U}}(j)) = c^{\circ}_{\text{U}} + f_0 \cdot \overleftarrow{t}_{\text{U}}(j)
\end{aligned} \tag{6.10}$$

Using (6.6), the corresponding time stamps can be expressed as follows:

$$\overrightarrow{s}_{\text{U}}(j) = \overrightarrow{c}_{\text{U}}(j) + \overrightarrow{\xi}_{\text{U}}(j) = c^{\circ}_{\text{U}} + f_0 \cdot \overrightarrow{t}_{\text{U}}(j) + \overrightarrow{\xi}_{\text{U}}(j) \tag{6.11}$$

$$\overrightarrow{s}_{\text{D}}(j) = \overrightarrow{c}_{\text{D}}(j) + \overrightarrow{\xi}_{\text{D}}(j) = c^{\circ}_{\text{D}} + f_0 \cdot \overrightarrow{t}_{\text{D}}(j) + \overrightarrow{\xi}_{\text{D}}(j) \tag{6.12}$$

$$\overleftarrow{s}_{\text{D}}(j) = \overleftarrow{c}_{\text{D}}(j) + \overleftarrow{\xi}_{\text{D}}(j) = c^{\circ}_{\text{D}} + f_0 \cdot \overleftarrow{t}_{\text{D}}(j) + \overleftarrow{\xi}_{\text{D}}(j) \tag{6.13}$$

$$\overleftarrow{s}_{\text{U}}(j) = \overleftarrow{c}_{\text{U}}(j) + \overleftarrow{\xi}_{\text{U}}(j) = c^{\circ}_{\text{U}} + f_0 \cdot \overleftarrow{t}_{\text{U}}(j) + \overleftarrow{\xi}_{\text{U}}(j) \tag{6.14}$$

Let us use

$$\overrightarrow{d}(j) = \overrightarrow{t}_{\text{D}}(j) - \overrightarrow{t}_{\text{U}}(j) \tag{6.15}$$

to denote the transmission delay, also called line delay of the j^{th} Delay_Resp message in terms of the absolute reference time and use

$$\overleftarrow{d}(j) = \overleftarrow{t}_{\text{U}}(j) - \overleftarrow{t}_{\text{D}}(j) \tag{6.16}$$

to denote the line delay of the j^{th} Delay_Req message in terms of the absolute reference time. We assume that the line delay is symmetric and constant, i.e., we always have:

$$\overrightarrow{d}(j) = \overleftarrow{d}(j) = d \tag{6.17}$$

Combining (6.11) and (6.12), we obtain:

$$\begin{aligned}
\overrightarrow{s}_{\text{U}}(j) - \overrightarrow{s}_{\text{D}}(j) &= \left(c^{\circ}_{\text{U}} + f_0 \cdot \overrightarrow{t}_{\text{U}}(j) + \overrightarrow{\xi}_{\text{U}}(j)\right) - \left(c^{\circ}_{\text{D}} + f_0 \cdot \overrightarrow{t}_{\text{D}}(j) + \overrightarrow{\xi}_{\text{D}}(j)\right) \\
&= (c^{\circ}_{\text{U}} - c^{\circ}_{\text{D}}) + f_0 \cdot \left(\overrightarrow{t}_{\text{U}}(j) - \overrightarrow{t}_{\text{D}}(j)\right) + \left(\overrightarrow{\xi}_{\text{U}}(j) - \overrightarrow{\xi}_{\text{D}}(j)\right) \\
&= o_{\text{UD}} - f_0 \cdot \overrightarrow{d}(j) + \left(\overrightarrow{\xi}_{\text{U}}(j) - \overrightarrow{\xi}_{\text{D}}(j)\right) \\
&= o_{\text{UD}} - f_0 \cdot d + \left(\overrightarrow{\xi}_{\text{U}}(j) - \overrightarrow{\xi}_{\text{D}}(j)\right)
\end{aligned} \tag{6.18}$$

where we applied (6.11) and (6.12) in the first line; (6.9) and (6.15) for the third line; (6.17) for the last line.

Subtracting (6.13) from (6.14) on both sides, we obtain:

$$\begin{aligned}
\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j) &= \left(c_{\mathrm{U}}^{\circ} + f_0 \cdot \overleftarrow{t}_{\mathrm{U}}(j) + \overleftarrow{\xi}_{\mathrm{U}}(j)\right) - \left(c_{\mathrm{D}}^{\circ} + f_0 \cdot \overleftarrow{t}_{\mathrm{D}}(j) + \overleftarrow{\xi}_{\mathrm{D}}(j)\right) \\
&= (c_{\mathrm{U}}^{\circ} - c_{\mathrm{D}}^{\circ}) + f_0 \cdot \left(\overleftarrow{t}_{\mathrm{U}}(j) - \overleftarrow{t}_{\mathrm{D}}(j)\right) + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j)\right) \\
&= o_{\mathrm{UD}} + f_0 \cdot \overleftarrow{d}(j) + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j)\right) \\
&= o_{\mathrm{UD}} + f_0 \cdot d + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j)\right) \qquad (6.19)
\end{aligned}$$

where we applied (6.13) and (6.14) in the first line; (6.9) and (6.16) for the third line; (6.17) for the last line.

Subtracting (6.18) from (6.19) on both sides, we obtain:

$$\begin{aligned}
&(\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j)) - (\overrightarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{D}}(j)) \\
= \; & 2f_0 \cdot d + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j)\right) - \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j)\right) \qquad (6.20)
\end{aligned}$$

Then the true line delay with respect to the clock frequency f_0 is given by:

$$f_0 \cdot d = \frac{[(\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j)) - (\overrightarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{D}}(j))] - \left[\left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j)\right) - \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j)\right)\right]}{2} \qquad (6.21)$$

Based on (6.21), the downstream element estimates the line delay with respect to its clock frequency f_0 by:

$$\hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) = \frac{[(\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j)) - (\overrightarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{D}}(j))]}{2} \qquad (6.22)$$

where $c_{\mathrm{E}}(x)$ represent the element E's local measurement of the absolute reference time interval x. A hat on c represents the estimate of this quantity. The true line delay $c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j))$ is equal to $f_0 \cdot d$, given in (6.21). Comparing (6.21) with (6.22), we can see that the estimation error $\tilde{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j))$ is given by:

$$\begin{aligned}
\tilde{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) &= c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) - \hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) \\
&= \frac{\left[\left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j)\right) - \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j)\right)\right]}{2} \qquad (6.23)
\end{aligned}$$

6.1.5.2 Clocks with different constant frequency values

Now we consider a more complicated situation, where the upstream element's clock and the downstream element's clock are running with different frequencies, i.e., $f_{\mathrm{U}} \neq f_{\mathrm{D}}$. But we still assume that both frequencies are constant, i.e., $\Delta_{\mathrm{U}} = \Delta_{\mathrm{D}} = 0$. In this case, the master time at t is given by:

$$c_{\mathrm{U}}(t) = c_{\mathrm{U}}^{\circ} + f_{\mathrm{U}} \cdot t \qquad (6.24)$$

and the slave time is given by:

$$c_{\mathrm{D}}(t) = c^{\circ}_{\mathrm{D}} + f_{\mathrm{D}} \cdot t \tag{6.25}$$

According to the clock models in (6.24) and (6.25):

$$\begin{aligned}
\overrightarrow{c}_{\mathrm{U}}(j) &= c_{\mathrm{U}}(\overrightarrow{t}_{\mathrm{U}}(j)) = c^{\circ}_{\mathrm{U}} + f_{\mathrm{U}} \cdot \overrightarrow{t}_{\mathrm{U}}(j)\\
\overrightarrow{c}_{\mathrm{D}}(j) &= c_{\mathrm{D}}(\overrightarrow{t}_{\mathrm{D}}(j)) = c^{\circ}_{\mathrm{D}} + f_{\mathrm{D}} \cdot \overrightarrow{t}_{\mathrm{D}}(j)\\
\overleftarrow{c}_{\mathrm{D}}(j) &= c_{\mathrm{D}}(\overleftarrow{t}_{\mathrm{D}}(j)) = c^{\circ}_{\mathrm{D}} + f_{\mathrm{D}} \cdot \overleftarrow{t}_{\mathrm{D}}(j)\\
\overleftarrow{c}_{\mathrm{U}}(j) &= c_{\mathrm{U}}(\overleftarrow{t}_{\mathrm{U}}(j)) = c^{\circ}_{\mathrm{U}} + f_{\mathrm{U}} \cdot \overleftarrow{t}_{\mathrm{U}}(j)
\end{aligned} \tag{6.26}$$

Using (6.6), the time stamps can be expressed by:

$$\overrightarrow{s}_{\mathrm{U}}(j) \;=\; \overrightarrow{c}_{\mathrm{U}}(j) + \overrightarrow{\xi}_{\mathrm{U}}(j) = c^{\circ}_{\mathrm{U}} + f_{\mathrm{U}} \cdot \overrightarrow{t}_{\mathrm{U}}(j) + \overrightarrow{\xi}_{\mathrm{U}}(j) \tag{6.27}$$

$$\overrightarrow{s}_{\mathrm{D}}(j) \;=\; \overrightarrow{c}_{\mathrm{D}}(j) + \overrightarrow{\xi}_{\mathrm{D}}(j) = c^{\circ}_{\mathrm{D}} + f_{\mathrm{D}} \cdot \overrightarrow{t}_{\mathrm{D}}(j) + \overrightarrow{\xi}_{\mathrm{D}}(j) \tag{6.28}$$

$$\overleftarrow{s}_{\mathrm{D}}(j) \;=\; \overleftarrow{c}_{\mathrm{D}}(j) + \overleftarrow{\xi}_{\mathrm{D}}(j) = c^{\circ}_{\mathrm{D}} + f_{\mathrm{D}} \cdot \overleftarrow{t}_{\mathrm{D}}(j) + \overleftarrow{\xi}_{\mathrm{D}}(j) \tag{6.29}$$

$$\overleftarrow{s}_{\mathrm{U}}(j) \;=\; \overleftarrow{c}_{\mathrm{U}}(j) + \overleftarrow{\xi}_{\mathrm{U}}(j) = c^{\circ}_{\mathrm{U}} + f_{\mathrm{U}} \cdot \overleftarrow{t}_{\mathrm{U}}(j) + \overleftarrow{\xi}_{\mathrm{U}}(j) \tag{6.30}$$

Let us define:

$$c_{\mathrm{D}}(\overrightarrow{d^{\mathrm{LD}}_{\mathrm{D}}}(j)) = f_{\mathrm{D}} \cdot d \tag{6.31}$$

to be the line delay measured by the downstream element's clock where we assume that the line delay is symmetric and constant. Then, we have:

$$\overrightarrow{d}(j) = \overrightarrow{d}(j+1) = \overleftarrow{d}(j) = \overleftarrow{d}(j+1) = d \tag{6.32}$$

From (6.32), we obtain:

$$\overrightarrow{t}_{\mathrm{D}}(j) - \overrightarrow{t}_{\mathrm{U}}(j) = \overrightarrow{t}_{\mathrm{D}}(j+1) - \overrightarrow{t}_{\mathrm{U}}(j+1) \tag{6.33}$$

Moving $\overrightarrow{t}_{\mathrm{U}}(j+1)$ to the LHS and $\overrightarrow{t}_{\mathrm{D}}(j)$ to the RHS, we obtain:

$$\overrightarrow{t}_{\mathrm{U}}(j+1) - \overrightarrow{t}_{\mathrm{U}}(j) = \overrightarrow{t}_{\mathrm{D}}(j+1) - \overrightarrow{t}_{\mathrm{D}}(j) \tag{6.34}$$

Using (6.26) in (6.34):

$$\begin{aligned}
\frac{\overrightarrow{c}_{\mathrm{U}}(j+1) - c^{\circ}_{\mathrm{U}}}{f_{\mathrm{U}}} - \frac{\overrightarrow{c}_{\mathrm{U}}(j) - c^{\circ}_{\mathrm{U}}}{f_{\mathrm{U}}} &= \frac{\overrightarrow{c}_{\mathrm{D}}(j+1) - c^{\circ}_{\mathrm{D}}}{f_{\mathrm{D}}} - \frac{\overrightarrow{c}_{\mathrm{D}}(j) - c^{\circ}_{\mathrm{D}}}{f_{\mathrm{D}}}\\
&\Downarrow\\
\frac{\overrightarrow{c}_{\mathrm{U}}(j+1) - \overrightarrow{c}_{\mathrm{U}}(j)}{f_{\mathrm{U}}} &= \frac{\overrightarrow{c}_{\mathrm{D}}(j+1) - \overrightarrow{c}_{\mathrm{D}}(j)}{f_{\mathrm{D}}}\\
&\Downarrow\\
\frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} &= \frac{\overrightarrow{c}_{\mathrm{U}}(j+1) - \overrightarrow{c}_{\mathrm{U}}(j)}{\overrightarrow{c}_{\mathrm{D}}(j+1) - \overrightarrow{c}_{\mathrm{D}}(j)}
\end{aligned} \tag{6.35}$$

Let us define

$$r_{\mathrm{UD}} = \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} \tag{6.36}$$

to be the frequency ratio between the upstream and downstream elements' local clocks. Based on (6.35), we can estimate r_{UD} by:

$$\hat{r}_{\mathrm{UD}} = \frac{\overrightarrow{s}_{\mathrm{U}}(j+1) - \overrightarrow{s}_{\mathrm{U}}(j)}{\overrightarrow{s}_{\mathrm{D}}(j+1) - \overrightarrow{s}_{\mathrm{D}}(j)} \tag{6.37}$$

Now we derive the formula to estimate the delay $c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j))$. We first define:

$$u_{\mathrm{UD}} = c_{\mathrm{U}}^{\circ} - c_{\mathrm{D}}^{\circ} \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} \tag{6.38}$$

to be the skewed offset between the clocks.

Now we compute (6.27) minus $\frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\times$(6.28) and use (6.15), (6.38) to obtain:

$$\begin{aligned}
\overrightarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} &= \left(c_{\mathrm{U}}^{\circ} - c_{\mathrm{D}}^{\circ} \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) + f_{\mathrm{U}} \cdot \left(\overrightarrow{t}_{\mathrm{U}}(j) - \overrightarrow{t}_{\mathrm{D}}(j)\right) + \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
&= u_{\mathrm{UD}} - f_{\mathrm{U}} \cdot \overrightarrow{d}(j) + \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
&= u_{\mathrm{UD}} - f_{\mathrm{U}} \cdot d + \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \qquad (6.39)
\end{aligned}$$

Computing (6.30) minus $\frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\times$(6.29) and using (6.16), (6.38), we obtain:

$$\begin{aligned}
\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} &= \left(c_{\mathrm{U}}^{\circ} - c_{\mathrm{D}}^{\circ} \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) + f_{\mathrm{U}} \cdot \left(\overleftarrow{t}_{\mathrm{U}}(j) - \overleftarrow{t}_{\mathrm{D}}(j)\right) + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
&= u_{\mathrm{UD}} + f_{\mathrm{U}} \cdot \overleftarrow{d}(j) + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
&= u_{\mathrm{UD}} + f_{\mathrm{U}} \cdot d + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \qquad (6.40)
\end{aligned}$$

Subtracting (6.40) by (6.39) on both sides, we obtain:

$$\begin{aligned}
& \left(\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) - \left(\overrightarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
= {} & 2 f_{\mathrm{U}} \cdot d + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) - \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
= {} & 2 \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} \cdot (f_{\mathrm{D}} \cdot d) + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) - \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \\
= {} & 2 \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}} \cdot c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) + \left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overleftarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) - \left(\overrightarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j) \cdot \frac{f_{\mathrm{U}}}{f_{\mathrm{D}}}\right) \qquad (6.41)
\end{aligned}$$

Based on (6.41), we can estimate $c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j))$ by:

$$\begin{aligned}
\hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) &= \frac{(\overleftarrow{s}_{\mathrm{U}}(j) - \overleftarrow{s}_{\mathrm{D}}(j)\cdot\hat{r}_{\mathrm{UD}}) - (\overrightarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{D}}(j)\cdot\hat{r}_{\mathrm{UD}})}{2\hat{r}_{\mathrm{UD}}} \\
&= \frac{(\overleftarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{U}}(j)) - (\overleftarrow{s}_{\mathrm{D}}(j) - \overrightarrow{s}_{\mathrm{D}}(j))\cdot\hat{r}_{\mathrm{UD}}}{2\hat{r}_{\mathrm{UD}}} \\
&= \frac{(\overleftarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{U}}(j))\cdot\hat{r}_{\mathrm{DU}} - (\overleftarrow{s}_{\mathrm{D}}(j) - \overrightarrow{s}_{\mathrm{D}}(j))}{2}
\end{aligned} \tag{6.42}$$

where:

$$\hat{r}_{\mathrm{DU}} = \frac{1}{\hat{r}_{\mathrm{UD}}} = \frac{\overrightarrow{s}_{\mathrm{D}}(j+1) - \overrightarrow{s}_{\mathrm{D}}(j)}{\overrightarrow{s}_{\mathrm{U}}(j+1) - \overrightarrow{s}_{\mathrm{U}}(j)} \tag{6.43}$$

The estimation error is given by, according to (6.41) and (6.42):

$$\begin{aligned}
\tilde{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) &= c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) - \hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) \qquad (6.44) \\
&= \frac{(\overleftarrow{s}_{\mathrm{U}}(j) - \overrightarrow{s}_{\mathrm{U}}(j))\cdot\tilde{r}_{\mathrm{DU}}}{2} - \frac{\left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{U}}(j)\right)\cdot\hat{r}_{\mathrm{DU}} - \left(\overleftarrow{\xi}_{\mathrm{D}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j)\right)}{2}
\end{aligned}$$

According to the realistic values of the variables in (6.44) (see Table 6.1), the estimation error can be closely approximated by:

$$\tilde{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j)) \approx -\frac{\left(\overleftarrow{\xi}_{\mathrm{U}}(j) - \overrightarrow{\xi}_{\mathrm{U}}(j)\right) - \left(\overleftarrow{\xi}_{\mathrm{D}}(j) - \overrightarrow{\xi}_{\mathrm{D}}(j)\right)}{2} \tag{6.45}$$

6.1.5.3 General case

In general, the frequency of the master and slave can be given by very complicated functions thus the constant frequency assumption used in the above derivation is not valid anymore. However, the frequency drift is usually very slow. If we do the delay estimation frequently enough, the error is negligible. In the derivation in the previous sections, we always assume that the line delay is constant. However, from one transmission to the other, the true line delay may vary. In order to minimize the influence of delay variation as well as the stamping error, usually several delay estimates are averaged. The averaged line delay will be eventually used for knowing the line delay experienced by the Sync messages. Let us define $\hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(k))$ to be the line delay value that is used to approximate the line delay in the transmission of the k's Sync message from element U to element D (U is the direct upstream element of D). Then $\hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(k))$ can be expressed as:

$$\hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(k)) = \frac{1}{L}\sum_{i=0}^{L-1}\hat{c}_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(j-i)) \tag{6.46}$$

where j is the last delay estimation cycle that has been finished at element D before the k^{th} Sync message arrives at element D. Let $c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(k))$ denote the true line delay of the k^{th} Sync message. It can be expressed as:

$$c_{\mathrm{D}}(\overrightarrow{d_{\mathrm{D}}^{\mathrm{LD}}}(k)) = f_{\mathrm{D}}\cdot d + \chi_{\mathrm{D}}(k) \tag{6.47}$$

where $\chi_{\mathtt{D}}(k)$ is a Gaussian random variable that models the variation of the line delay. Using (6.46) and (6.44), we can obtain that:

$$\hat{c}_{\mathtt{D}}(\overrightarrow{d_{\mathtt{D}}^{\mathtt{LD}}}(k)) - c_{\mathtt{D}}(\overrightarrow{d_{\mathtt{D}}^{\mathtt{LD}}}(k)) = \frac{1}{L}\sum_{i=0}^{L-1}\tilde{c}_{\mathtt{D}}(\overrightarrow{d_{\mathtt{D}}^{\mathtt{LD}}}(j-i)) + \chi_{\mathtt{D}}(k) \tag{6.48}$$

Let us define:

$$\tilde{c}_{\mathtt{D}}(\overrightarrow{d_{\mathtt{D}}^{\mathtt{LD}}}(k)) = -\left(\frac{1}{L}\sum_{i=0}^{L-1}\tilde{c}_{\mathtt{D}}(\overrightarrow{d_{\mathtt{D}}^{\mathtt{LD}}}(j-i)) + \chi_{\mathtt{D}}(k)\right) \tag{6.49}$$

to be the estimation error, then we can write:

$$\hat{c}_{\mathtt{D}}(\overrightarrow{d}_{\mathtt{D}}^{\mathtt{LD}}(k)) = c_{\mathtt{D}}(\overrightarrow{d}_{\mathtt{D}}^{\mathtt{LD}}(k)) - \tilde{c}_{\mathtt{D}}(\overrightarrow{d}_{\mathtt{D}}^{\mathtt{LD}}(k)) \tag{6.50}$$

From (6.49), we can see that the line delay estimation error is composed of the sum of L i.i.d. random variables $\{\tilde{c}_{\mathtt{D}}(\overrightarrow{d_{\mathtt{D}}^{\mathtt{LD}}}(j-i))\}_{i=0}^{L-1}$ plus a Gaussian random variable $\chi_{\mathtt{D}}(k)$. As a result, the line delay estimation error itself is a Gaussian random variable, i.e., $\tilde{c}_{\mathtt{D}}(\overrightarrow{d}_{\mathtt{D}}^{\mathtt{LD}}(k)) \sim \mathcal{N}(0, \sigma^2_{d_{\mathtt{D}}^{\mathtt{LD}}})$.

All the synchronization algorithms developed later will assume a prior estimation of the line delay, which is obtained by using (6.42).

6.2 Notations

Before introducing the synchronization algorithms, we list again the notations that will appear in the following sections for the sake of clarity.

- $\overrightarrow{s}_{\mathtt{TB}}^{\mathtt{TT}}(k)$: time-stamps generated at the transmission or the reception of the k^{th} Sync message. Subscript $\mathtt{TB} \in \{\mathtt{M}, \mathtt{S}_1, \ldots \mathtt{S}_N\}$ indicates at which element this is generated. Superscript $\mathtt{TT} \in \{\mathtt{in}, \mathtt{out}\}$ indicates whether the time-stamp is generated-at the reception ($\mathtt{in}$) or at the transmission ($\mathtt{out}$).

- $\overrightarrow{c}_{\mathtt{TB}}^{\mathtt{TT}}(k)$: the true local time at the transmission or the reception of the k^{th} Sync message.

- $\overrightarrow{\xi}_{\mathtt{TB}}^{\mathtt{TT}}(k)$: stamping error, i.e.,

$$\overrightarrow{s}_{\mathtt{TB}}^{\mathtt{TT}}(k) = \overrightarrow{c}_{\mathtt{TB}}^{\mathtt{TT}}(k) + \overrightarrow{\xi}_{\mathtt{TB}}^{\mathtt{TT}}(k) \tag{6.51}$$

- $r_{\mathtt{MS}_n}(k)$ the k^{th} frequency ratio of the master frequency and slave n's frequency

- $\overrightarrow{y}_{n}^{\mathtt{TT}}(k)$: master time when slave n's local time is $\overrightarrow{c}_{\mathtt{S}_n}^{\mathtt{TT}}(k)$.

- $\overrightarrow{x}_{n}^{\mathtt{TT}}(k)$: master time when slave n's counter state is really $\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{TT}}(k)$.

- $c_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k))$: the true line delay between slave $n-1$ and slave n, measured by slave n's clock

- $\hat{c}_{\mathrm{S}_n}(\overrightarrow{d_n^{\mathrm{LD}}}(k))$: estimate of the line delay between slave $n-1$ and slave n, measured by slave n's clock. According to (6.50),

$$\hat{c}_{\mathrm{S}_n}(\overrightarrow{d}_n^{\mathrm{LD}}(k)) = c_{\mathrm{S}_n}(\overrightarrow{d}_n^{\mathrm{LD}}(k)) - \tilde{c}_{\mathrm{S}_n}(\overrightarrow{d}_n^{\mathrm{LD}}(k)) \tag{6.52}$$

  where $\tilde{c}_{\mathrm{S}_n}(\overrightarrow{d}_n^{\mathrm{LD}}(k)) \sim \mathcal{N}(0, \sigma^2_{d_n^{\mathrm{LD}}})$ is the estimation error.

Figure 6.5 depicts the relationships between the variables we defined above.

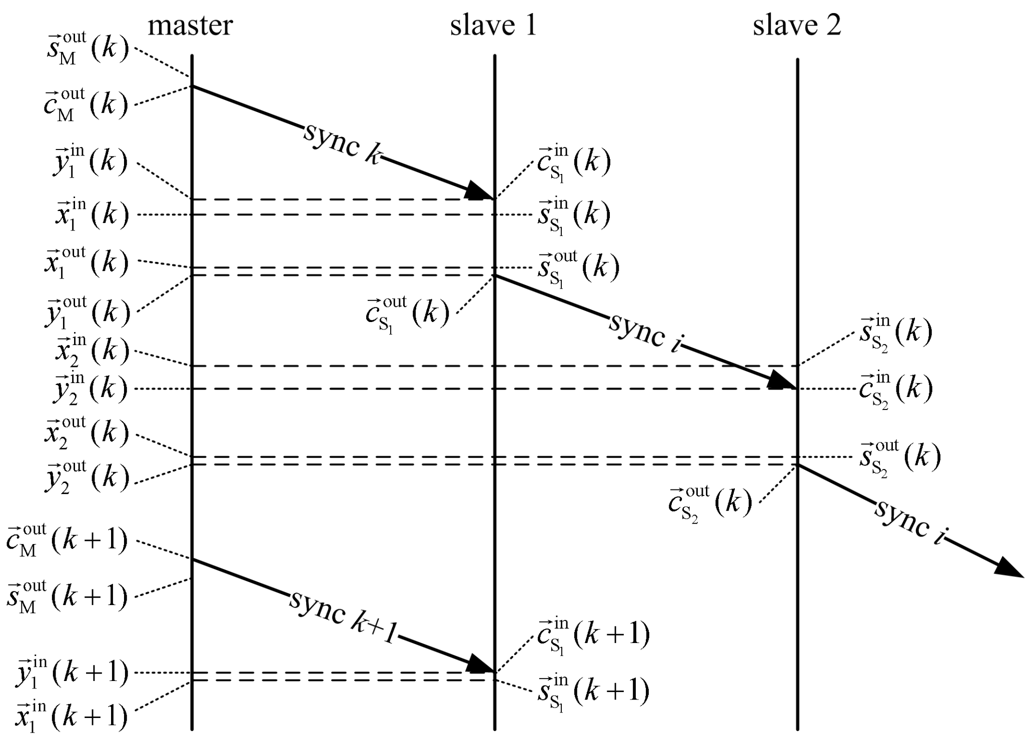

master
slave 1
slave 2
$\vec{s}_{\mathrm{M}}^{\mathrm{out}}(k)$
$\vec{c}_{\mathrm{M}}^{\mathrm{out}}(k)$
$\vec{y}_1^{\mathrm{in}}(k)$
$\vec{x}_1^{\mathrm{in}}(k)$
$\vec{x}_1^{\mathrm{out}}(k)$
$\vec{y}_1^{\mathrm{out}}(k)$
$\vec{x}_2^{\mathrm{in}}(k)$
$\vec{y}_2^{\mathrm{in}}(k)$
$\vec{x}_2^{\mathrm{out}}(k)$
$\vec{y}_2^{\mathrm{out}}(k)$
$\vec{c}_{\mathrm{M}}^{\mathrm{out}}(k+1)$
$\vec{s}_{\mathrm{M}}^{\mathrm{out}}(k+1)$
$\vec{y}_1^{\mathrm{in}}(k+1)$
$\vec{x}_1^{\mathrm{in}}(k+1)$
sync k
$\vec{c}_{\mathrm{S}_1}^{\mathrm{in}}(k)$
$\vec{s}_{\mathrm{S}_1}^{\mathrm{in}}(k)$
$\vec{s}_{\mathrm{S}_1}^{\mathrm{out}}(k)$
$\vec{c}_{\mathrm{S}_1}^{\mathrm{out}}(k)$
sync i
$\vec{s}_{\mathrm{S}_2}^{\mathrm{in}}(k)$
$\vec{c}_{\mathrm{S}_2}^{\mathrm{in}}(k)$
$\vec{s}_{\mathrm{S}_2}^{\mathrm{out}}(k)$
$\vec{c}_{\mathrm{S}_2}^{\mathrm{out}}(k)$
sync i
sync k+1
$\vec{c}_{\mathrm{S}_1}^{\mathrm{in}}(k+1)$
$\vec{s}_{\mathrm{S}_1}^{\mathrm{in}}(k+1)$

Fig. 6.5. Notation of variables

6.3 Simulation settings

In the following sections, we will introduce different synchronization algorithms. To illustrate the synchronization performance of each algorithm, we will present simulation results after the description of the algorithm. Comprehensive simulation results and performance comparisons will be shown at the end of the whole chapter.

Table 6.1 summarizes the parameters for the simulation. Where the quartz precision indicates the maximum frequency deviation from the nominal frequency omitting the environmental effects. The line delay is equal to the constant cable delay plus varying jitters, which are approximated by Gaussian random variables. The master sends out Sync messages every $30ms$. Each slave initializes a delay estimation process every 8s. Errors made in the time stamping is modeled by uniformly distributed random variables.

Parameter	Value
Quartz precision	$100ppm$
Cable delay	$100ns$
Bridge delay	uniform $[125, 250]\mu s$
Interval of Sync message	$30ms$
Interval of Delay_request	$8s$
Stamping jitter	uniform $[-40, 40]ns$
Jitter in line delay	zero mean Gaussian, standard deviation: $40ns$

Table 6.1. Simulation parameters

6.4 Standard synchronization algorithm

Using the ”transparent clock” concept [51], the synchronization procedure is as follows:

- master element generates periodically Sync messages with an interval of $\Delta c_{\mathtt{M}}^{\mathtt{SYNC}}$ and generates a time-stamp $\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k)$ at the transmission.

- a Follow_Up message will be sent right after Sync message which delivers $\overrightarrow{s}_{\mathtt{M}}(k)$ to slave 1

- upon the reception of the Sync message, slave n generates a time-stamp $\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k)$

- after some bridge delay $\overrightarrow{d}_n^{\mathtt{BD}}(k)$, slave n forwards the Sync message to slave $n+1$, a time stamp $\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{out}}(k)$ will be generated at the transmission. Then the bridge delay measured by slave n is given by:

$$\hat{c}_{\mathtt{S}_n}(\overrightarrow{d}_n^{\mathtt{BD}}(k)) = \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{out}}(k) - \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k) \tag{6.53}$$

- upon the reception of Follow_Up message, slave 1 estimates the frequency ratio $r_{\mathtt{MS}_1}$ by:

$$\hat{r}_{\mathtt{MS}_1}(k) = \frac{\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) - \overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k-1)}{\overrightarrow{s}_{\mathtt{S}_1}^{\mathtt{in}}(k) - \overrightarrow{s}_{\mathtt{S}_1}^{\mathtt{in}}(k-1)} \tag{6.54}$$

  and estimates the master time by:

$$\hat{x}_1^{\mathtt{in}}(k) \;=\; \overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) + \hat{r}_{\mathtt{MS}_1}(k) \cdot \hat{c}_{\mathtt{S}_1}(\overrightarrow{d}_1^{\mathtt{LD}}(k)) \tag{6.55}$$

  where the line delay estimate $\hat{c}_{\mathtt{S}_1}(\overrightarrow{d}_1^{\mathtt{LD}}(k))$ is given by (6.46).

- a Follow_Up message will be sent by slave 1 right after the Sync message, which carries its master time estimate at the transmission of the Sync message, i.e., $\hat{x}_1^{\mathtt{out}}(k)$, which is calculated as follows:

$$\hat{x}_1^{\mathtt{out}}(k) \;=\; \hat{x}_1^{\mathtt{in}}(k) + \hat{r}_{\mathtt{MS}_1}(k) \cdot \hat{c}_{\mathtt{S}_1}(\overrightarrow{d}_1^{\mathtt{BD}}(k)) \tag{6.56}$$

  where $\hat{c}_{\mathtt{S}_1}(\overrightarrow{d}_1^{\mathtt{BD}}(k))$ is given by (6.53) and $\hat{r}_{\mathtt{MS}_1}(k)$ is given by (6.54)

- upon reception of Follow_Up message from slave $n-1$, slave n estimates the frequency ratio between master clock and its local clock, i.e., $r_{\mathtt{MS}_n}$ by:

$$\hat{r}_{\mathtt{MS}_n}(k) = \frac{\hat{x}_{n-1}^{\mathtt{out}}(k) - \hat{x}_{n-1}^{\mathtt{out}}(k-1)}{\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k) - \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k-1)} \tag{6.57}$$

  and estimates the master time by:

$$\hat{x}_n^{\mathtt{in}}(k) \quad = \quad \hat{x}_{n-1}^{\mathtt{out}}(k) + \hat{r}_{\mathtt{MS}_n}(k) \cdot \hat{c}_{\mathtt{S}_n}(\overrightarrow{d}_n^{\mathtt{LD}}(k)) \tag{6.58}$$

  where the line delay estimate $\hat{c}_{\mathtt{S}_n}(\overrightarrow{d}_n^{\mathtt{LD}}(k))$ is given by (6.46).

- slave n estimates the master time at the forwarding time of the Sync message by:

$$\hat{x}_n^{\mathtt{out}}(k) \quad = \quad \hat{x}_n^{\mathtt{in}}(k) + \hat{r}_{\mathtt{MS}_n}(k) \cdot \hat{c}_{\mathtt{S}_n}(\overrightarrow{d}_n^{\mathtt{BD}}(k)) \tag{6.59}$$

To illustrate the synchronization performance, we simulated the standard synchronization algorithm using the simulation parameters summarized in Section 6.3. The performance is evaluated through the synchronization error at the reception of the Sync message, i.e. the difference between the estimated master time $\hat{x}_n^{\mathtt{in}}(k)$ and the true master time $\overrightarrow{x}_n^{\mathtt{in}}(k)$. Simulation results are shown in Figure 6.6.

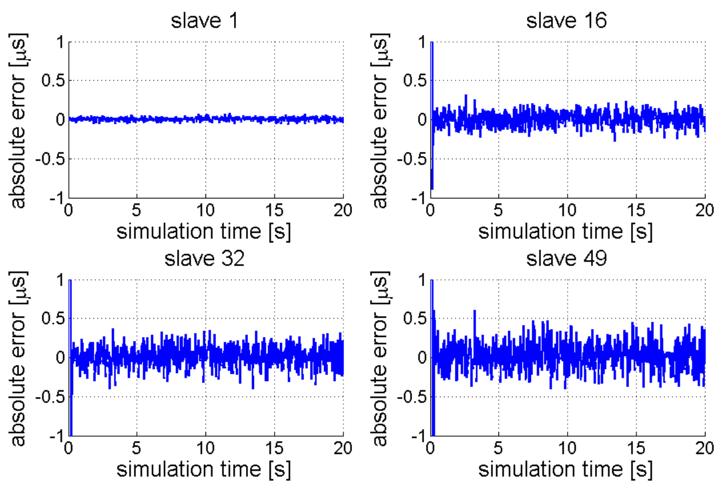


Fig. 6.6. Synchronization error of the standard algorithm. $(\overrightarrow{x}_n^{\mathtt{in}}(k) - \hat{x}_n^{\mathtt{in}}(k))$

The synchronization procedure introduced above estimates master time based on the noisy time-stamps. The achievable precision relies on the precision of the time-stamps. From the simulation results in Figure 6.6 we can see the effect of error propagation. For remote slave elements, the synchronization performance is greatly influenced by the stamping noise accumulated along the line.

Improvements can be done from different aspects. Authors of [64] improve the accuracy of the time-stamping by careful (but costly) design of the hardware implementation. The authors of [65] summarize software based methods that improve the synchronization precision for PTP without transparent clocks. A Kalman filtering algorithm is presented in [1] to achieve end-to-end synchronization based on the NTP protocol. Another NTP based Kalman filtering method can be found in [9]. Both these works can be extended to PTP protocol, but not to PTP with transparent clocks, since they did not study the case where time stamps are also available at the intermediate bridges. The authors of [5] presented a Kalman filter approach which tracks what they call the ”skew”, i.e., the rate of change of the offset between a pair of clocks. The resulting skew is used as the input to the PI controller to regulate the clocks. As to our knowledge, most related works are only applied to a small number of devices and not in a cascaded networked system where the estimation error propagates and greatly degrades the synchronization accuracy at a remote slave element. Some of these works exploited the distribution of the uncertainties. But the relationships between the time-stamps generated at different elements are not modeled.

In [97], we analytically studied the effect of error propagation. It has been shown that the estimation error is not simply the sum of all the uncertainties. As we can see in (6.57), the frequency ratio estimated at slave n depends on the estimates of the master time at slave $n-1$. At the end, the synchronization error can be expressed as a very complicated function of the uncertainties. A big drawback of the standard synchronization algorithm is that it treats each Sync message independently. But actually the relationships between the time stamps associated with consecutive Sync messages can be used to average the noise. In the following sections, we exploit this property and formulate clock synchronization as a state estimation problem. We use probabilistic graphical models to explicitly express the statistical relationships involved in the system and discuss different probabilistic approaches to this estimation problem.

6.5 Probabilistic model for clock synchronization

In this section, we establish probabilistic models for the clock synchronization, which results in a linear dynamical system [8], i.e., temporal evolutions and the spatial correlations in the system are linear and all the random variables are Gaussian.

We first study centralized inference, where the hidden state associated with all slaves will be estimated jointly. This method has to be based on an ideal assumption that all the time-stamps are transmitted to a fusion center, where a probabilistic representation is established to model the relationships between variables. Using this model, we translate the synchronization into the problem of estimating hidden state variables (the master time corresponding to a given slave time) given the observations (time stamping associated with IEEE 1588 messages). Then we use Kalman filtering technique to solve this estimation problem. The solution is optimal in the sense of minimizing the mean square error. The results will then be sent to corresponding slave elements so that they can build the relationship between their own clock and the master clock.

Then we study the case where the state estimation is done in a distributed way. Each slave maintains its estimation of the master time. Only a limited amount of information has to be exchanged between network elements. Distributed inference is only possible when the problem can be factorized. However, as discussed in Section 4.3, due to the coupling of the processes, the belief state, i.e., the joint distribution of the state variables, is usually not decomposable. We have to introduce an approximate belief state which has a convenient structure for distributed inference. This can be done by assuming extra structure on the graphical model, i.e., modifying the graph to make distributed implementation possible. Based on the modified graph, we derive efficient synchronization algorithms.

6.5.1 Probabilistic model

Since we assume constant frequencies, the state transition model of frequency ratio is given by:

$$r_{\mathtt{MS}_n}(k) = r_{\mathtt{MS}_n}(k-1) + \omega_n(k) \tag{6.60}$$

where $\omega_n(k)$ is the process noise of frequency ratio. If the frequency drifts, the performance is sensitive to the choice of $\omega_n(k)$. We will discuss the proper choice of the value of $\omega_n(k)$ later.

The state transition of $\overrightarrow{x}_n^{\mathtt{in}}(k)$ is given by:

$$\overrightarrow{x}_n^{\mathtt{in}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k-1) + \left(\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k) - \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k-1)\right) \cdot r_{\mathtt{MS}_n}(k) \tag{6.61}$$

Let us define:

$$a_n(k) = \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k) - \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k-1) \tag{6.62}$$

then (6.61) can be rewritten as:

$$\overrightarrow{x}_n^{\mathtt{in}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k-1) + a_n(k) \cdot r_{\mathtt{MS}_n}(k) \tag{6.63}$$

$\overrightarrow{x}_n^{\mathtt{in}}(k)$ and $\overrightarrow{x}_n^{\mathtt{out}}(k)$ are related by:

$$\overrightarrow{x}_n^{\mathtt{out}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k) + \left(\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{out}}(k) - \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k)\right) \cdot r_{\mathtt{MS}_n}(k) \tag{6.64}$$

Let us define

$$\begin{aligned} b_n(k) &= \hat{c}_{\mathtt{S}_n}(\overrightarrow{d}_n^{\mathtt{BD}}(k)) \\ &= \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{out}}(k) - \overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k) \end{aligned} \tag{6.65}$$

and rewrite (6.64):

$$\overrightarrow{x}_n^{\mathtt{out}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k) + b_n(k) \cdot r_{\mathtt{MS}_n}(k) \tag{6.66}$$

$\overrightarrow{c}_{\mathtt{M}}^{\mathtt{out}}(k)$ and $\overrightarrow{y}_1^{\mathtt{in}}(k)$ are related by the line delay between master and slave 1:

$$\begin{aligned} \overrightarrow{c}_{\mathtt{M}}^{\mathtt{out}}(k) &= \overrightarrow{y}_1^{\mathtt{in}}(k) - c_{\mathtt{M}}(\overrightarrow{d_1^{\mathtt{LD}}}(j)) \\ &= \overrightarrow{y}_1^{\mathtt{in}}(k) - c_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(j)) \cdot r_{\mathtt{MS}_1}(k) \\ &= \overrightarrow{y}_1^{\mathtt{in}}(k) - \left(\hat{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(j)) + \tilde{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(j))\right) \cdot r_{\mathtt{MS}_1}(k) \end{aligned} \tag{6.67}$$

For $n > 1$, $\overrightarrow{y}_{n-1}^{\mathtt{out}}(k)$ and $\overrightarrow{y}_n^{\mathtt{in}}(k)$ are related by:

$$\begin{aligned} \overrightarrow{y}_{n-1}^{\mathtt{out}}(k) &= \overrightarrow{y}_n^{\mathtt{in}}(k) - c_{\mathtt{M}}(\overrightarrow{d_n^{\mathtt{LD}}}(j)) \\ &= \overrightarrow{y}_n^{\mathtt{in}}(k) - c_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(j)) \cdot r_{\mathtt{MS}_n}(k) \\ &= \overrightarrow{y}_n^{\mathtt{in}}(k) - \left(\hat{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(j)) + \tilde{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(j))\right) \cdot r_{\mathtt{MS}_n}(k) \end{aligned} \tag{6.68}$$

For all $\mathtt{TT} \in \{\mathtt{in}, \mathtt{out}\}$, the relation holds by definition of $r_{\mathtt{MS}_n}(k)$:

$$\begin{aligned}\overrightarrow{x}_n^{\mathtt{TT}}(k) - \overrightarrow{y}_n^{\mathtt{TT}}(k) &= r_{\mathtt{MS}_n}(k) \cdot \left(\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{TT}}(k) - \overrightarrow{c}_{\mathtt{S}_n}^{\mathtt{TT}}(k)\right) \\ &= r_{\mathtt{MS}_n}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_n}^{\mathtt{TT}}(k) \end{aligned} \tag{6.69}$$

where we have applied (6.51) for the second equality.

Applying (6.51) with $\overrightarrow{c}_{\mathtt{M}}^{\mathtt{out}}(k)$ on the left hand side and (6.69) on the 1st term on the right hand side, (6.67) can be rewritten as:

$$\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) - \overrightarrow{\xi}_{\mathtt{M}}^{\mathtt{out}}(k) = \overrightarrow{x}_1^{\mathtt{in}}(k) - r_{\mathtt{MS}_1}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_1}^{\mathtt{in}}(k) - \left(\hat{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(k)) + \tilde{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(k))\right) \cdot r_{\mathtt{MS}_1}(k) \tag{6.70}$$

Moving $\overrightarrow{\xi}_{\mathtt{M}}^{\mathtt{out}}(k)$ to the right hand side, we obtain:

$$\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) = \overrightarrow{x}_1^{\mathtt{in}}(k) - r_{\mathtt{MS}_1}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_1}^{\mathtt{in}}(k) - \left(\hat{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(k)) + \tilde{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(k))\right) \cdot r_{\mathtt{MS}_1}(k) + \overrightarrow{\xi}_{\mathtt{M}}^{\mathtt{out}}(k) \tag{6.71}$$

Using (6.69) on both left hand side and 1st term on the right hand side, (6.68) can be rewritten as:

$$\begin{aligned}& \overrightarrow{x}_{n-1}^{\mathtt{out}}(k) - r_{\mathtt{MS}_{n-1}}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_{n-1}}^{\mathtt{out}}(k) \\ = \; & \overrightarrow{x}_n^{\mathtt{in}}(k) - r_{\mathtt{MS}_n}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_n}^{\mathtt{in}}(k) - \left(\hat{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k)) + \tilde{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k))\right) \cdot r_{\mathtt{MS}_n}(k) \end{aligned} \tag{6.72}$$

Using (6.66) and moving $r_{\mathtt{MS}_{n-1}}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_{n-1}}^{\mathtt{out}}(k)$ to the right hand side, (6.72) can be rewritten as:

$$\begin{aligned}& \overrightarrow{x}_{n-1}^{\mathtt{in}}(k) + b_{n-1}(k) \cdot r_{\mathtt{MS}_{n-1}}(k) \\ = \; & \overrightarrow{x}_n^{\mathtt{in}}(k) - r_{\mathtt{MS}_n}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_n}^{\mathtt{in}}(k) - \left(\hat{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k)) + \tilde{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k))\right) \cdot r_{\mathtt{MS}_n}(k) + \\ & + r_{\mathtt{MS}_{n-1}}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_{n-1}}^{\mathtt{out}}(k) \end{aligned} \tag{6.73}$$

We define:

$$\nu_n(k) = \begin{cases} -\tilde{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(k)) \cdot r_{\mathtt{MS}_1}(k) + \overrightarrow{\xi}_{\mathtt{M}}^{\mathtt{out}}(k) - r_{\mathtt{MS}_1}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_1}^{\mathtt{in}}(k) & \mathtt{for} \quad n = 1 \\ -\tilde{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k)) \cdot r_{\mathtt{MS}_n}(k) + r_{\mathtt{MS}_{n-1}}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_{n-1}}^{\mathtt{out}}(k) - r_{\mathtt{MS}_n}(k) \cdot \overrightarrow{\xi}_{\mathtt{S}_n}^{\mathtt{in}}(k) & \mathtt{for} \quad n > 1 \end{cases} \tag{6.74}$$

and

$$d_n(k) = \hat{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k)) \tag{6.75}$$

and then rewrite (6.71) and (6.73) as follows:

$$\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) = \overrightarrow{x}_1^{\mathtt{in}}(k) - d_1(k) \cdot r_{\mathtt{MS}_1}(k) + \nu_1(k) \tag{6.76}$$

$$\overrightarrow{x}_{n-1}^{\mathtt{in}}(k) + b_{n-1}(k) \cdot r_{\mathtt{MS}_{n-1}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k) - d_n(k) \cdot r_{\mathtt{MS}_n}(k) + \nu_n(k) \tag{6.77}$$

Let us use $r_{\mathtt{MS}_n}^{\mathsf{nom}}$ to be the ratio of the nominal frequencies of master and slave n, which is a constant. Now we replace $r_{\mathtt{MS}_n}(k)$ with $r_{\mathtt{MS}_n}^{\mathsf{nom}}$ in (6.74), i.e., we define:

$$\nu_n(k) = \begin{cases} -\tilde{c}_{\mathtt{S}_1}(\overrightarrow{d_1^{\mathtt{LD}}}(k)) \cdot r_{\mathtt{MS}_1}^{\mathsf{nom}} + \overrightarrow{\xi}_{\mathtt{M}}^{\mathtt{out}}(k) - r_{\mathtt{MS}_1}^{\mathsf{nom}} \cdot \overrightarrow{\xi}_{\mathtt{S}_1}^{\mathtt{in}}(k) & \mathtt{for} \quad n = 1 \\ -\tilde{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k)) \cdot r_{\mathtt{MS}_n}^{\mathsf{nom}} + r_{\mathtt{MS}_{n-1}}^{\mathsf{nom}} \cdot \overrightarrow{\xi}_{\mathtt{S}_{n-1}}^{\mathtt{out}}(k) - r_{\mathtt{MS}_n}^{\mathsf{nom}} \cdot \overrightarrow{\xi}_{\mathtt{S}_n}^{\mathtt{in}}(k) & \mathtt{for} \quad n > 1 \end{cases} \tag{6.78}$$

As shown in Table 6.1, the maximum frequency deviation is very small, the error introduced by this replacement is negligible. But it makes $\nu_n(k)$ independent on the hidden variables. Once the hardware specification of the network element is given, we can determine the distribution of the noise variable $\nu_n(k)$.

6.5.2 Model analysis

Now, let us summarize the most important formulas that have been derived:

- (6.60): $r_{\mathtt{MS}_n}(k) = r_{\mathtt{MS}_n}(k-1) + \omega_n(k)$
- (6.63): $\overrightarrow{x}_n^{\mathtt{in}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k-1) + a_n(k) \cdot r_{\mathtt{MS}_n}(k)$
- (6.76): $\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) = \overrightarrow{x}_1^{\mathtt{in}}(k) - d_1(k) \cdot r_{\mathtt{MS}_1}(k) + \nu_1(k)$
- (6.77): $\overrightarrow{x}_{n-1}^{\mathtt{in}}(k) + b_{n-1}(k) \cdot r_{\mathtt{MS}_{n-1}}(k) = \overrightarrow{x}_n^{\mathtt{in}}(k) - d_n(k) \cdot r_{\mathtt{MS}_n}(k) + \nu_n(k)$

The variables above fall into the following categories:

- known variables: $\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k)$, $\{a_n(k)\}_{n=1}^N$, $\{d_n(k)\}_{n=1}^N$ and $\{b_{n-1}(k)\}_{n=2}^N$.
- hidden state variables: $\{\overrightarrow{x}_n^{\mathtt{in}}(k)\}_{n=1}^N$ and $\{r_{\mathtt{MS}_n}(k)\}_{n=1}^N$.
- random variables with known distribution: $\{\nu_n(k)\}_{n=1}^N$ and $\{\omega_n(k)\}_{n=1}^N$

Let us assume all noises are Gaussian noise, i.e., $\omega_n(k) \sim \mathcal{N}(0, \sigma_\omega^2)$ and $\nu_n(k) \sim \mathcal{N}(0, \sigma_\nu^2)$ for all $n = 1, \ldots N$. Assuming same hardware for each network element (master or slave), then the distributions of $\omega_n(k)$ and $\nu_n(k)$ are independent of time step k and element number n. Therefore, we dropped k and n in σ_ω and σ_ν.

As we can see from (6.78), $\nu_n(k)$ is a linear combination of many zero mean stamping noises where each noise $\overrightarrow{\xi}_{\mathtt{TB}}^{\mathtt{TT}}(k)$ is a random variable generated from a distribution $p_{\overrightarrow{\xi}_{\mathtt{TB}}^{\mathtt{TT}}(k)}(\overrightarrow{\xi}_{\mathtt{TB}}^{\mathtt{TT}}(k))$. Usually, we obtain the distribution function of the stamping errors from the description of the hardware or from experiment. Given all the distributions, we can use Monte Carlo method to generate many samples $v^{(i)} \sim p_{\overrightarrow{\xi}_{\mathtt{TB}}^{\mathtt{TT}}(k)}(v)$. Using (6.78), we can produce samples $\{\nu^{(i)}\}$ of $\nu_n(k)$ from $\{v^{(i)}\}$. Then we can find out the best Gaussian distribution that fits the histogram of the data $\{\nu^{(i)}\}$ and thus obtain the value of σ_ν.

The state transition model of the frequency ratio in (6.60) indicates that the underlying model assumes constant frequency and uses the parameter σ_ω to control the strength of this assumption. If the true frequency varies, we will face a model mismatch problem. We can determine how much such a model should be trusted by adjusting the value of σ_ω. It balances the importance between the state transition model and the measurements. If σ_ω is big, then we can somehow decrease the error introduced by model mismatch by trusting the measurements. But we will tolerate the noise made in the observation. If σ_ω is small, we force the estimate of frequency to be constant. By doing that, we will take the risk of model mismatch. However, if the true frequency is constant, small σ_{ω_n} can minimize the influence of the observation noise. In practice, the choice of the value of

σ_{ω_n} should be based on the stability of the oscillator and the magnitude of the stamping errors.

For more compact notation we lump the two hidden variables into one hidden vector-valued variable $\mathbf{x}_n(k) = [\overrightarrow{x}^{\mathtt{in}}_n(k) \quad r_{\mathtt{MS}_n}(k)]^{\mathrm{T}}$, and define accordingly $\mathbf{A}_n(k) = \begin{bmatrix} 1 & a_n(k) \\ 0 & 1 \end{bmatrix}$, $\mathbf{e} = [0,1]^{\mathrm{T}}$, $\mathbf{c}_n(k) = [1, -d_n(k)]^{\mathrm{T}}$, $\mathbf{b}_n(k) = [1, b_n(k)]^{\mathrm{T}}$. Then the state-space equations become:

$$\mathbf{x}_n(k) = \mathbf{A}_n(k)\mathbf{x}_n(k-1) + \omega_n(k)\mathbf{e} \tag{6.79}$$

$$\overrightarrow{s}^{\,\mathsf{out}}_{\mathtt{M}}(k) = \mathbf{c}_1^{\mathrm{T}}(k)\mathbf{x}_1(k) + \nu_1(k) \tag{6.80}$$

$$\mathbf{b}_{n-1}^{\mathrm{T}}(k)\mathbf{x}_{n-1}(k) = \mathbf{c}_n^{\mathrm{T}}(k)\mathbf{x}_n(k) + \nu_n(k) \tag{6.81}$$

The random vectors $\mathbf{x}_n(k)$ and $\mathbf{x}_n(k-1)$ are related via the probability density function $p_{\omega_n(k)}(\omega_n(k))$ of the noise $\omega_n(k)$; the distribution $\mathbf{x}_1(k)$ is given by the probability density function $p_{\nu_1(k)}(\nu_1(k))$ of the noise $\nu_1(k)$; and the random vectors $\mathbf{x}_n(k)$ and $\mathbf{x}_{n-1}(k)$ are related via the probability density function $p_{\nu_n(k)}(\nu_n(k))$ of the noise $\nu_n(k)$.

Since we assume all the noise are Gaussian, i.e., for all $n \in \{1, \dots N\}$, $\omega_n(k) \sim \mathcal{N}(0, \sigma^2_\omega)$ and $\nu_n(k) \sim \mathcal{N}(0, \sigma^2_\nu)$. Then

$$p_{\omega_n(k)}(\omega_n(k)) = \frac{1}{\sqrt{2\pi}\sigma_\omega} \exp\left(-\frac{(\omega_n(k))^2}{2\sigma^2_\omega}\right) \tag{6.82}$$

and since from (6.79)

$$\omega_n(k) = \mathbf{e}^{\mathrm{T}}\left(\mathbf{x}_n(k) - \mathbf{A}_n(k)\mathbf{x}_n(k-1)\right) \tag{6.83}$$

we define:

$$\begin{aligned} f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1)) &= p_{\omega_n(k)}(\omega_n(k)) = p_{\omega_n(k)}(\mathbf{e}^{\mathrm{T}}\left(\mathbf{x}_n(k) - \mathbf{A}_n(k)\mathbf{x}_n(k-1)\right)) \\ = \quad & \frac{1}{\sqrt{2\pi}\sigma_\omega} \exp\left(-\frac{\left(\mathbf{x}_n(k) - \mathbf{A}_n(k)\mathbf{x}_n(k-1)\right)^{\mathrm{T}} \mathbf{e}\mathbf{e}^{\mathrm{T}} \left(\mathbf{x}_n(k) - \mathbf{A}_n(k)\mathbf{x}_n(k-1)\right)}{2\sigma^2_\omega}\right) \end{aligned} \tag{6.84}$$

Here we write f_k^n in a form of conditional probability to show that this function represents the evlolution of the hidden state. Later, when we represent this function on a graphical model, we use directed edges to represent this relationship.

Likewise:

$$p_{\nu_n(k)}(\nu_n(k)) = \frac{1}{\sqrt{2\pi}\sigma_\nu} \exp\left(-\frac{(\nu_n(k))^2}{2\sigma^2_\nu}\right) \tag{6.85}$$

and from (6.80), (6.81)

$$\nu_1(k) = \overrightarrow{s}^{\,\mathsf{out}}_{\mathtt{M}}(k) - \mathbf{c}_1^{\mathrm{T}}(k)\mathbf{x}_1(k) \tag{6.86}$$

$$\nu_n(k) = \mathbf{b}_{n-1}^{\mathrm{T}}(k)\mathbf{x}_{n-1}(k) - \mathbf{c}_n^{\mathrm{T}}(k)\mathbf{x}_n(k) \tag{6.87}$$

hence we define:

$$\begin{aligned} g_k^1(\mathbf{x}_1(k); \overrightarrow{s}^{\,\mathsf{out}}_{\mathtt{M}}(k)) &= p_{\nu_1(k)}(\nu_1(k)) = p_{\nu_1(k)}(\overrightarrow{s}^{\,\mathsf{out}}_{\mathtt{M}}(k) - \mathbf{c}_1^{\mathrm{T}}(k)\mathbf{x}_1(k)) \\ = \quad & \frac{1}{\sqrt{2\pi}\sigma_\nu} \exp\left(-\frac{\left(\overrightarrow{s}^{\,\mathsf{out}}_{\mathtt{M}}(k) - \mathbf{c}_1^{\mathrm{T}}(k)\mathbf{x}_1(k)\right)^2}{2\sigma^2_\nu}\right) \end{aligned} \tag{6.88}$$

and

$$g_k^n(\mathbf{x}_n(k), \mathbf{x}_{n-1}(k)) = p_{\nu_n(k)}(\nu_n(k)) = p_{\nu_n(k)}(\mathbf{b}_{n-1}^{\mathrm{T}}(k)\mathbf{x}_{n-1}(k) - \mathbf{c}_n^{\mathrm{T}}(k)\mathbf{x}_n(k))$$
$$= \frac{1}{\sqrt{2\pi}\sigma_\nu} \exp\left(-\frac{\left(\mathbf{b}_{n-1}^{\mathrm{T}}(k)\mathbf{x}_{n-1}(k) - \mathbf{c}_n^{\mathrm{T}}(k)\mathbf{x}_n(k)\right)^2}{2\sigma_\nu^2}\right) \tag{6.89}$$

The functions f_k^n and g_k^n ($n = 1, \ldots N$ and $k = 1, \ldots K$) reveal all possible relationships between hidden and observed variables. Figure 6.7 uses a factor graph to visualize those relationships where the function nodes represents the functions defined in (6.84), (6.88) and (6.89).

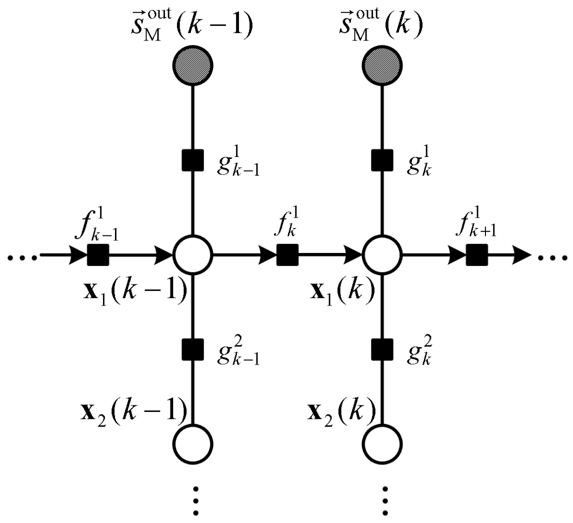


Fig. 6.7. Factor graph for clock synchronization

The joint probability represented by Figure 6.7 can be expressed as follows:

$$p(\{\mathbf{X}(k)\}_{k=1}^K, \{\vec{s}_{\mathrm{M}}^{\,\mathrm{out}}(k)\}_{k=1}^K) = \prod_{n=1}^N p(\mathbf{x}_n(0)) \cdot \tag{6.90}$$
$$\cdot \prod_{k=1}^K \left(\prod_{n=1}^N f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1)) \cdot g_k^1(\mathbf{x}_1(k); \vec{s}_{\mathrm{M}}^{\,\mathrm{out}}(k)) \cdot \prod_{n=2}^N g_k^n(\mathbf{x}_n(k), \mathbf{x}_{n-1}(k)) \right)$$

where $\mathbf{X}(k) = [\mathbf{x}_1(k), \ldots \mathbf{x}_N(k)]$ ensembles all the hidden state variables at the same time step k. The distribution $p(\mathbf{x}_n(0))$ represents the prior knowledge on the distribution of the initial state.

Our goal is to use probabilistic inference algorithms to estimate the a posteriori probability distribution of $\mathbf{x}_n(k)$, i.e., $p(\mathbf{x}_n(K)|\{\vec{s}_{\mathrm{M}}^{\,\mathrm{out}}(k)\}_{k=1}^K)$ for each slave n at time step K.

The factor graph shown in Figure 6.7 resembles a dynamic Bayesian network. However, there are function nodes and undirected edges in the graph. Therefore, we call it dynamic factor graph. The arrows reveal the transition of the hidden states. Like 2TBN, we can also develop a compact graphical representation of the dynamical system, which we call

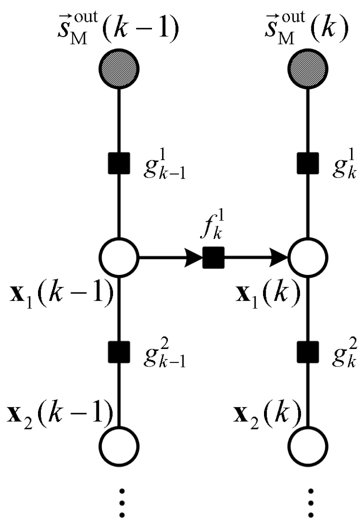
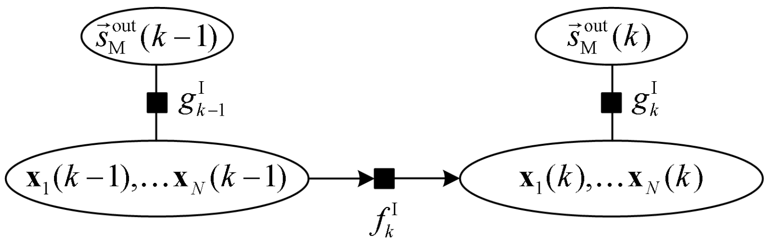


Fig. 6.8. 2TFG for clock synchronization

2TFG, i.e., two-slice temporal factor graph. Figure 6.8 illustrates 2TFG for the synchronization problem.

Section 4.3 introduced probabilistic inference algorithms for state estimation in dynamical systems, which convert a 2TBN to a junction tree. Kevin Murphy's interface algorithm produces the exact result. In this algorithm, we first identify the interface $\mathcal{I}^{(k)}$ that d-separates the past from the future. The interface from time slice $k-1$ to slice k contains hidden variables in time slice $k-1$ which have children in time slice k. In the 2TFG shown in Figure 6.8, the interface from time slice $k-1$ to k is $\{\mathrm{x}_n(k-1)\}_{n=1}^N$. According to the interface algorithm, when we construct a junction tree, all the variables in the interface should form a big clique. As a result, we should assemble $\{\mathrm{x}_n(k)\}_{n=1}^N$ into a big node, which result in a factor graph shown in Figure 6.9.



$\vec{s}_{\mathrm{M}}^{\mathrm{out}}(k-1)$
$\vec{s}_{\mathrm{M}}^{\mathrm{out}}(k)$
g_{k-1}^{I}
g_{k}^{I}
$\mathbf{x}_1(k-1),\ldots\mathbf{x}_N(k-1)$
$\mathbf{x}_1(k),\ldots\mathbf{x}_N(k)$
f_k^{I}

Fig. 6.9. Junction tree of the 2TFG in Figure 6.8

In Figure 6.9, the function $g_k^{\mathtt{I}}$ is given by:

$$\begin{aligned} & g_k^{\mathtt{I}}(\mathbf{x}_1(k), \ldots \mathbf{x}_N(k);\ \overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k)) \\ = \ & g_k^1(\mathbf{x}_1(k);\ \overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k)) \cdot \prod_{n=2}^{N} g_k^n(\mathbf{x}_n(k), \mathbf{x}_{n-1}(k)) \end{aligned} \tag{6.91}$$

and the function f_k^{I} is given by:

$$\begin{aligned} & f_k^{\mathrm{I}}(\mathbf{x}_1(k), \ldots \mathbf{x}_N(k), \mathbf{x}_1(k-1), \ldots \mathbf{x}_N(k-1)) \\ = & \prod_{n=1}^{N} f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1)) \end{aligned} \tag{6.92}$$

At each time slice k, the interface algorithm requires joint estimation of all hidden state variables. Such an algorithm must be implemented in a centralized way, i.e., all the relevant information should be transmitted to a fusion center where the junction tree shown in Figure 6.9 can be constructed to compute the joint posterior probability distribution of all the hidden state variables. Such a centralized estimation guarantees exact results. However, it is infeasible in practice because transmitting information to the fusion center is not supported by the PTP protocol. Additional transportation protocol has to be defined. Furthermore, communication between the slaves and the fusion center introduces delays, which degrades the synchronization performance. Therefore, centralized inference only provides theoretically the best achievable synchronization precision.

In order to implement a distributed state estimation method, we need to introduce approximations. We modify the 2TFG in Figure 6.8 to produce a graphical model with simpler structure. Such a simpler structure enables distributed state estimation. However, the result is not exact anymore. We should expect that the results are worse than that of the centralized inference method. Boyen and Koller proved in [10] that for appropriate approximations, the error, measured by the Kullback-Leibler distance to the true distribution, remains bounded indefinitely.

In the following sections, we introduce respectively the centralized and the distributed inference algorithms for clock synchronization.

6.5.3 State space model and centralized master time estimation

We first present an estimation method that is implemented in a centralized way. All the time stamps associated with the Sync messages are transmitted to a fusion center, where a Kalman filter is used to solve the state estimation problem. The solution is optimal in the sense of minimizing the mean square error. The results are then sent to the corresponding slave elements so that they can work out the relationship between their own clock and the master clock. It will be shown that this method is equivalent to the interface algorithm mentioned in Section 6.5.2 which computes the exact posterior probability distribution of the hidden state variables.

Usually a distributed implementation of the state estimation is preferable, in order to avoid extra network traffic and the adverse effect of additional delays. However, as explained in Section 6.5.2, exact inference requires joint estimation of all the hidden state variables. This centralized Kalman filtering implements such a centralized inference, therefore yields a lower bound to the synchronization error achieved by any other synchronization algorithm that is based on the same probabilistic model.

Using (6.60) and (6.63), we obtain the following state transtion model for the whole system:

$$\begin{bmatrix} \overrightarrow{x}_1^{\mathtt{in}}(k) \\ \vdots \\ \overrightarrow{x}_N^{\mathtt{in}}(k) \\ r_{\mathtt{MS}_1}(k) \\ \vdots \\ r_{\mathtt{MS}_N}(k) \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 0 & a_1(k) & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & a_N(k) \\ 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \overrightarrow{x}_1^{\mathtt{in}}(k-1) \\ \vdots \\ \overrightarrow{x}_N^{\mathtt{in}}(k-1) \\ r_{\mathtt{MS}_1}(k-1) \\ \vdots \\ r_{\mathtt{MS}_N}(k-1) \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \omega_1(k) \\ \vdots \\ \omega_N(k) \end{bmatrix} \tag{6.93}$$

Using $\mathbf{x}(k)$ to denote the vector of hidden state variables on the left hand side of (6.93), using $\mathbf{A}(k)$ for the first matrix on the right hand side, and $\boldsymbol{\omega}(k) = [\omega_1(k), \ldots \omega_N(k)]^{\mathrm{T}}$ for the noise vector and defining $\mathbf{E}_N = [\mathbf{0}_N, \mathbf{I}_N]$ where $\mathbf{0}_N$ is an $N \times N$ null matrix and $\mathbf{I}_N$ is an identity matrix of dimension N, we can write (6.93) as follows:

$$\mathbf{x}(k) = \mathbf{A}(k) \cdot \mathbf{x}(k-1) + \mathbf{E}_N^{\mathrm{T}} \cdot \boldsymbol{\omega}(k) \tag{6.94}$$

which gives us the *state transition model.*

Now, let us rewrite (6.77) in the following form:

$$0 = \overrightarrow{x}_n^{\mathtt{in}}(k) - d_n(k) \cdot r_{\mathtt{MS}_n}(k) - \overrightarrow{x}_{n-1}^{\mathtt{in}}(k) - b_{n-1}(k) \cdot r_{\mathtt{MS}_{n-1}}(k) + \nu_n(k) \tag{6.95}$$

Combining (6.76) with (6.95) for all elements, we obtain:

$$\begin{bmatrix} \overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & -d_1(k) & 0 & \cdots & 0 & 0 \\ -1 & 1 & \cdots & 0 & 0 & -b_1(k) & -d_2(k) & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & -1 & 1 & 0 & 0 & \cdots & -b_{N-1}(k) & -d_N(k) \end{bmatrix} \cdot \begin{bmatrix} \overrightarrow{x}_1^{\mathtt{in}}(k) \\ \overrightarrow{x}_2^{\mathtt{in}}(k) \\ \vdots \\ \overrightarrow{x}_{N-1}^{\mathtt{in}}(k) \\ \overrightarrow{x}_N^{\mathtt{in}}(k) \\ r_{\mathtt{MS}_1}(k) \\ r_{\mathtt{MS}_2}(k) \\ \vdots \\ r_{\mathtt{MS}_{N-1}}(k) \\ r_{\mathtt{MS}_N}(k) \end{bmatrix} + \begin{bmatrix} -\nu_1(k) \\ -\nu_2(k) \\ \vdots \\ -\nu_N(k) \end{bmatrix} \tag{6.96}$$

Using $\mathbf{C}(k)$ to denote the matrix on the right hand side of (6.96), $\mathbf{y}(k)$ for the left hand side, $\boldsymbol{\nu}(k)$ for the noise vector, we rewrite (6.96) as follows:

$$\mathbf{y}(k) = \mathbf{C}(k) \cdot \mathbf{x}(k) + \boldsymbol{\nu}(k) \tag{6.97}$$

which gives us the *observation model.*

Now, let us write down the state-space model:

- *state transition model* (6.94): $\mathbf{x}(k) = \mathbf{A}(k) \cdot \mathbf{x}(k-1) + \mathbf{E}_N^{\mathrm{T}} \cdot \boldsymbol{\omega}(k)$
- *observation model* (6.97): $\mathbf{y}(k) = \mathbf{C}(k) \cdot \mathbf{x}(k) + \boldsymbol{\nu}(k)$

The first formula reveals the time correlation of the state variables, so they constitute the *state transition model*. The second one reveals the space correlation of the state variables; hence they are the *observation or coupling model*. These two equations represent the *state-space-model* of the system.

The vectors and matrices in these two formulas fall into the following categories:

- known: $\mathbf{y}(k)$, $\mathbf{A}(k)$, $\mathbf{C}(k)$, $\mathbf{E}_N^{\mathrm{T}}$.
- hidden states: $\mathbf{x}(k-1)$, $\mathbf{x}(k)$.
- noise with Gaussian distribution: $\boldsymbol{\omega}(k)$, $\boldsymbol{\nu}(k)$.

In (6.94), the random vectors $\mathbf{x}(k-1)$ and $\mathbf{x}(k)$ are related via the probability density function $p_{\boldsymbol{\omega}(k)}(\boldsymbol{\omega}(k))$ of the noise $\boldsymbol{\omega}(k)$. From (6.94), we obtain a distribution of random vectors $\mathbf{x}(k)$ which is given by the probability density function $p_{\boldsymbol{\nu}(k)}(\boldsymbol{\nu}(k))$ of the noise $\boldsymbol{\nu}(k)$.

As discussed in the previous section, we assume that all the additive noise are Gaussian, i.e., $\omega_n(k) \sim \mathcal{N}(0, \sigma_\omega^2)$ and $\nu_n(k) \sim \mathcal{N}(0, \sigma_\nu^2)$ for all $n = 1, \ldots N$. Let us use $\mathbf{R}$ and $\mathbf{Q}$ to denote the covariance matrix of $\boldsymbol{\omega}(k)$ and $\boldsymbol{\nu}(k)$, so:

$$\mathbf{R} = \begin{bmatrix} \sigma_\omega^2 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_\omega^2 \end{bmatrix} = \sigma_\omega^2 \cdot \mathbf{I} \tag{6.98}$$

and

$$\mathbf{Q} = \begin{bmatrix} \sigma_\nu^2 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_\nu^2 \end{bmatrix} = \sigma_\nu^2 \cdot \mathbf{I} \tag{6.99}$$

Then:

$$p_{\boldsymbol{\omega}(k)}(\boldsymbol{\omega}(k)) = \frac{1}{(2\pi)^{N/2} |\mathbf{Q}|^{1/2}} \exp\left(-\frac{1}{2}\boldsymbol{\omega}^{\mathrm{T}}(k)\mathbf{Q}^{-1}\boldsymbol{\omega}(k)\right) \tag{6.100}$$

From (6.94):

$$\mathbf{E}_N^{\mathrm{T}} \cdot \boldsymbol{\omega}(k) = \mathbf{x}(k) - \mathbf{A}(k) \cdot \mathbf{x}(k-1) \tag{6.101}$$

Left multiplying on both sides of (6.101) with $\mathbf{E}_N$ and noting that $\mathbf{E}_N\mathbf{E}_N^{\mathrm{T}} = \mathbf{I}_N$, we obtain:

$$\boldsymbol{\omega}(k) = \mathbf{E}_N \cdot (\mathbf{x}(k) - \mathbf{A}(k) \cdot \mathbf{x}(k-1)) \tag{6.102}$$

Now we define:

$$\begin{aligned} f_k^{\mathrm{C}}(\mathbf{x}(k-1), \mathbf{x}(k)) &= p_{\boldsymbol{\omega}(k)}(\mathbf{E}_N \cdot (\mathbf{x}(k) - \mathbf{A}(k) \cdot \mathbf{x}(k-1))) \\ &= \frac{1}{(2\pi)^{N/2} |\mathbf{Q}|^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{x}(k)-\mathbf{A}(k)\cdot\mathbf{x}(k-1))^{\mathrm{T}}\mathbf{E}_N^{\mathrm{T}}\mathbf{Q}^{-1}\mathbf{E}_N(\mathbf{x}(k)-\mathbf{A}(k)\cdot\mathbf{x}(k-1))\right)} \end{aligned} \tag{6.103}$$

Likewise:

$$p_{\boldsymbol{\nu}(k)}(\boldsymbol{\nu}(k)) = \frac{1}{(2\pi)^{N/2}\,|\mathbf{R}|^{1/2}} \exp\left(-\frac{1}{2}\boldsymbol{\nu}^{\mathrm{T}}(k)\mathbf{R}^{-1}\boldsymbol{\nu}(k)\right) \tag{6.104}$$

and from (6.97):

$$\boldsymbol{\nu}(k) = \mathbf{y}(k) - \mathbf{C}(k)\cdot\mathbf{x}(k) \tag{6.105}$$

hence we define:

$$\begin{aligned} g_k^{\mathrm{C}}(\mathbf{x}(k)) &= p_{\boldsymbol{\nu}(k)}(\mathbf{y}(k) - \mathbf{C}(k)\cdot\mathbf{x}(k)) \\ &= \frac{1}{(2\pi)^{N/2}\,|\mathbf{R}|^{1/2}} e^{\left(-\frac{1}{2}(\mathbf{y}(k)-\mathbf{C}(k)\cdot\mathbf{x}(k))^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{y}(k)-\mathbf{C}(k)\cdot\mathbf{x}(k))\right)} \end{aligned} \tag{6.106}$$

Figure 6.10 shows the factor graph representation of the *state-space model*.

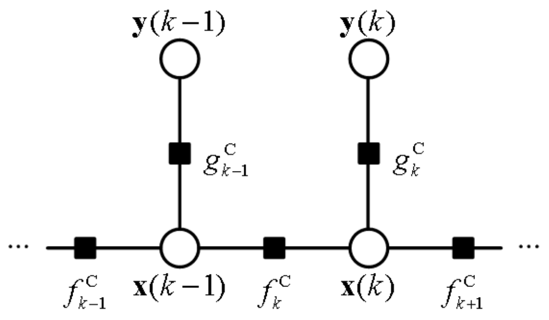


Fig. 6.10. Factor graph representation of the centralized *state-space model*

Since we assume that all the random vairables in the model are Gaussian distributed, we can use a Kalman filter to solve the state estimation problem. The calculation is summarized as follows:

- Predict:

  predict state: $\hat{\mathbf{x}}(k, k-1) = \mathbf{A}(k)\hat{\mathbf{x}}(k-1, k-1)$

  predict covariance: $\mathbf{P}(k, k-1) = \mathbf{A}(k)\mathbf{P}(k-1, k-1)\mathbf{A}^{\mathrm{T}}(k) + \mathbf{E}_N^{\mathrm{T}}\mathbf{Q}\mathbf{E}_N$

- Update:

  measurement residual: $\tilde{\mathbf{z}}(k) = \mathbf{y}(k) - \mathbf{C}(k)\hat{\mathbf{x}}(k, k-1)$

  residual covariance: $\mathbf{S}(k) = \mathbf{C}(k)\mathbf{P}(k, k-1)\mathbf{C}^{\mathrm{T}}(k) + \mathbf{R}$

  Kalman gain: $\mathbf{K}(k) = \mathbf{P}(k, k-1)\mathbf{C}^{\mathrm{T}}(k)\mathbf{S}^{-1}(k)$

  updated state: $\hat{\mathbf{x}}(k, k) = \hat{\mathbf{x}}(k, k-1) + \mathbf{K}(k)\tilde{\mathbf{z}}(k)$

  updated covariance: $\mathbf{P}(k, k) = (\mathbf{I} - \mathbf{K}(k)\mathbf{C}(k))\mathbf{P}(k, k-1)$

Figure 6.10 is very similar to the graph shown in Figure 6.9. It can also be easily proved that (6.106) and (6.103) are the expression of (6.91) and (6.92) in vector form. As a consequence, the interface algorithm is identical to the centralized Kalman filter (see the discussion in 4.3). That is to say that using the state space model and the Kalman filter, we implement the centralized inference that computes the exact posterior distribution of the hidden state variables.

As we mentioned, this Kalman filter method is a centralized estimation method. All the timing information has to be transmitted to a fusion center where a Kalman filter is implemented to jointly estimate the hidden state for all slaves. Then the results will be disseminated to the slaves. Figure 6.11 illustrates the gathering of the information from the slaves and the distribution of the estimation results.

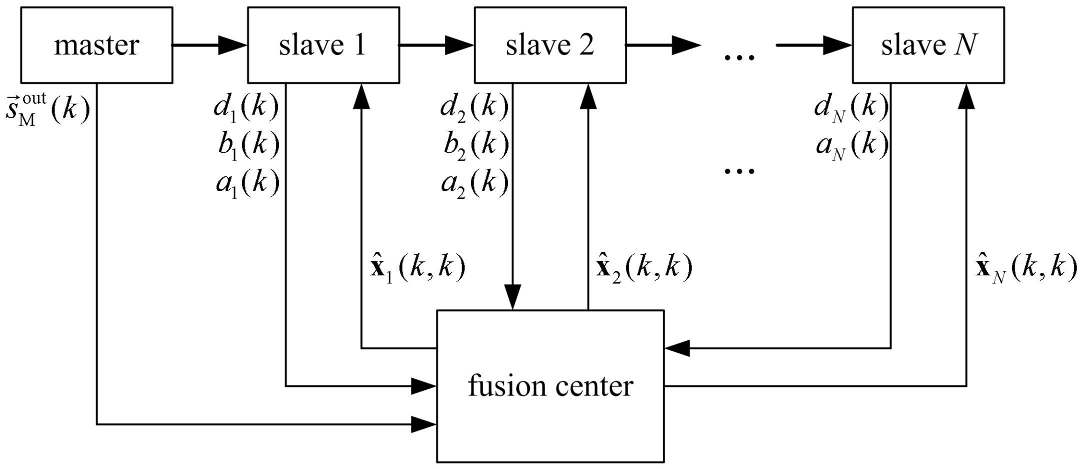

master
slave 1
slave 2
...
slave N
$\vec{s}_{\mathrm{M}}^{\,\mathrm{out}}(k)$
$d_1(k)$
$b_1(k)$
$a_1(k)$
$d_2(k)$
$b_2(k)$
$a_2(k)$
$d_N(k)$
$a_N(k)$
$\hat{\mathbf{x}}_1(k,k)$
$\hat{\mathbf{x}}_2(k,k)$
$\hat{\mathbf{x}}_N(k,k)$
fusion center

Fig. 6.11. Information transmission between network elements and the fusion center

The performance of the centralized Kalman filter is verified by simulation. The results, i.e., the synchronization error is depicted in Figure 6.12. This simulation result is obtained by choosing an appropriate value for the parameter σ_ω.

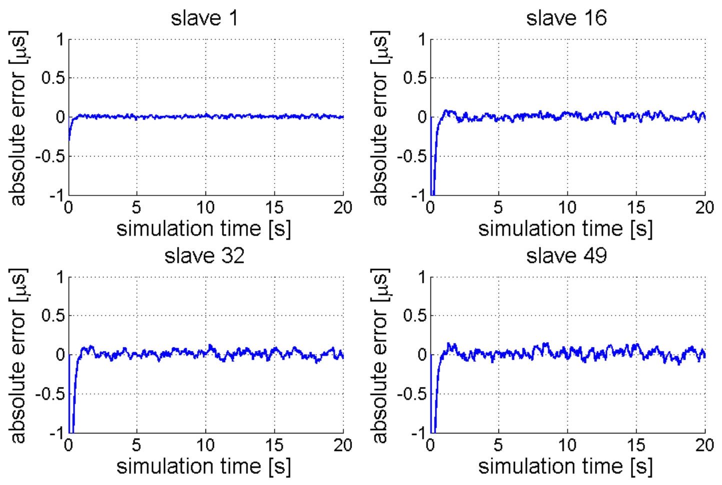


Fig. 6.12. Synchronization error of the centralized Kalman filter

6.5.4 The sum-product algorithm for master time estimation

In this section, we develop a distributed implementation of the hidden state estimation. As discussed in Section 6.5.2, computation of exact posterior probability distribution requires a centralized implementation. We modify the structure of the original probabilistic graphical model so that the resulting graph is suitable for running the sum-product algorithm, which realizes the distributed estimation of the hidden state variables.

6.5.4.1 Modified factor graph for distributed inference

The 2TFG shown in Figure 6.8 represent the following distribution:

$$\begin{aligned} & p_k(\mathbf{x}(k-1), \mathbf{x}(k), \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k) | \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(1), \ldots \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k-1)) \\ = \; & p(\mathbf{x}(k-1) | \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(1), \ldots \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k-1)) \cdot g_k^1(\mathbf{x}_1(k); \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k)) \cdot \prod_{n=2}^{N} g_k^n(\mathbf{x}_n(k), \mathbf{x}_{n-1}(k)) \\ & \cdot \prod_{n=1}^{N} f_k^n(\mathbf{x}_n(k) | \mathbf{x}_n(k-1)) \end{aligned} \tag{6.107}$$

In each time step k we want to infer the posterior distribution of $\mathbf{x}_n(k)$ and pass the belief state $p(\mathbf{x}(k)|\overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(1), \ldots \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k))$ to the next time step.

According to Section 4.1, distributed inference is based on the factorization of the joint probability. In (6.107), the problem is with the belief state $p(\mathbf{x}(k-1)|\overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(1), \ldots \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k-1))$ which is usually not factorizable. To overcome this problem, we approximate this belief state by:

$$p(\mathbf{x}(k-1)|\overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(1), \ldots \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k-1)) = \prod_{n=1}^{N} q(\mathbf{x}_n(k-1)) \tag{6.108}$$

We further assume that g_k^n represent conditional probability distributions. And now we can modify (6.107) to:

$$\begin{aligned} & p_k(\mathbf{x}(k-1), \mathbf{x}(k), \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k) | \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(1), \ldots \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k-1)) \qquad (6.109) \\ = \; & \prod_{n=1}^{N} q(\mathbf{x}_n(k-1)) \cdot g_k^1(\mathbf{x}_1(k); \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k)) \cdot \prod_{n=2}^{N} g_k^n(\mathbf{x}_n(k)|\mathbf{x}_{n-1}(k)) \cdot \prod_{n=1}^{N} f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1)) \end{aligned}$$

Based on (6.109), we construct a inference graph as shown in Figure 6.13. Arrows are introduced on the vertical edges to reveal that g_k^n now is a conditional probability function. By introducing directed edges, we regulate that message passing only follows the indicated directions.

6.5.4.2 Sum-product algorithm for distributed inference

Using the graph in Figure 6.13, we can derive the posterior probability distribution of every $\mathbf{x}_n(k)$, i.e., $q(\mathbf{x}_n(k))$ by using the sum-product algorithm.

Since all the equations in the state-space model are linear and all the random variables have Gaussian distribution, $q(\mathbf{x}_n(k))$ is also a Gaussian distribution, which is given by:

$$q(\mathbf{x}_n(k)) = \frac{1}{(2\pi)\,|\mathbf{P}_n(k)|^{1/2}} \exp\left(-\frac{1}{2} \left(\mathbf{x}_n(k) - \bar{\mathbf{x}}_n(k)\right)^{\mathrm{T}} \mathbf{P}_n(k)^{-1} \left(\mathbf{x}_n(k) - \bar{\mathbf{x}}_n(k)\right) \right) \tag{6.110}$$

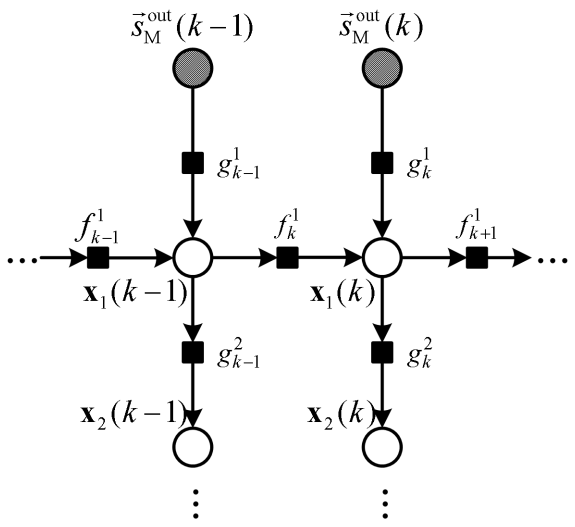


Fig. 6.13. Factor graph for distributed inference

where $\bar{\mathbf{x}}_n(k)$ is the mean of Gaussian random variable $\mathbf{x}_n(k)$. $\mathbf{P}_n(k)$ is its covariance matrix. The mean and covariance matrix are initialized with appropriate values (e.g., $E\{r_{\mathtt{MS}_n}(k)\}$ is initialized by 1, $E\{x_n^{\mathsf{in}}(k)\}$ is initialized by $\overrightarrow{s}_{\mathtt{M}}^{\mathtt{out}}(k) + \sum_{i=1}^{N} d_i(k) + \sum_{i=1}^{N-1} b_i(k)$ and $\mathbf{P}_n(k)$ is initialized by: $N \cdot \sigma_{\nu(k)}^2 \cdot \mathbf{I}_N$) in step 0. The following part derives the transition from step $k-1$ to step k. Using the sum-product algorithm, we can calculate $q(\mathbf{x}_n(k))$ in each time step k, for which it is sufficient to calculate the mean and variance. Because $q(\mathbf{x}_n(k))$ is Gaussian, it has its maximum at the mean, hence the MAP estimate of $\mathbf{x}_n(k)$ is: $\hat{\mathbf{x}}_n^{\mathsf{MAP}}(k) = \bar{\mathbf{x}}_n(k)$.

In (4.12) and (4.13), if incoming messages are Gaussian, then the outgoing message is also Gaussian. The sum-product algorithm will be initialized by Gaussian distributions, so all the messages are Gaussian. We use $\boldsymbol{\mu}_{a\to b}$ and $\boldsymbol{\Lambda}_{a\to b}$ to denote the mean and the variance of message $m_{a\to b}$.

Since now we regulate the message flow of the sum-product algorithm, i.e., messages only pass in the direction indicated by the arrows on the edges, we can discover the following relationships by applying the computation rule (equations (4.12) and (3.8)) of the sum-product algorithms (see Figure 6.16):

$$\begin{aligned}
& m_{\mathbf{x}_{n-1}(k-1)\to f_k^{n-1}}\left(\mathbf{x}_{n-1}(k-1)\right) \\
= \; & m_{\mathbf{x}_{n-1}(k-1)\to g_{k-1}^{n}}\left(\mathbf{x}_{n-1}(k-1)\right) \\
= \; & q\left(\mathbf{x}_{n-1}(k-1)\right) \\
= \; & m_{f_{k-1}^{n-1}\to \mathbf{x}_{n-1}(k-1)}\left(\mathbf{x}_{n-1}(k-1)\right) \cdot m_{g_{k-1}^{n-1}\to \mathbf{x}_{n-1}(k-1)}\left(\mathbf{x}_{n-1}(k-1)\right) \qquad (6.111)
\end{aligned}$$

We first take a look at the message passing around function node f_k^n, as depicted in Figure 6.14.

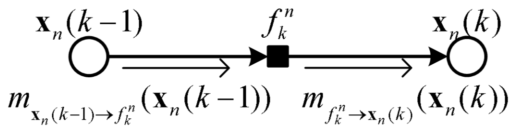

$\mathbf{x}_n(k-1)$
f_k^n
$\mathbf{x}_n(k)$
$m_{\mathbf{x}_n(k-1)\to f_k^n}(\mathbf{x}_n(k-1))$
$m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k))$

Fig. 6.14. Message passing around f_k^n

According to (4.13) and (6.111), the message from function node f_k^n to variable node $\mathbf{x}_n(k)$ can be calculated as follows:

$$m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k)) = \int_{\mathbf{x}_n(k-1)} m_{\mathbf{x}_n(k-1)\to f_k^n}(\mathbf{x}_n(k-1)) \cdot f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1)) \tag{6.112}$$

$$= \int_{\mathbf{x}_n(k-1)} q(\mathbf{x}_n(k-1)) \cdot f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1)) \tag{6.113}$$

The expression of $q(\mathbf{x}_n(k-1))$ is given by (6.110) and the expression of $f_k^n(\mathbf{x}_n(k)|\mathbf{x}_n(k-1))$ is given by (6.84). Calculating the integral in (6.112) according to the standard multiplication and integration of multivariate Gaussian distribution functions, leads to the result that $m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k))$ is a Gaussian density function, with mean:

$$\boldsymbol{\mu}_{f_k^n\to\mathbf{x}_n(k)} = \mathbf{A}_n(k)\cdot\bar{\mathbf{x}}_n(k-1) \tag{6.114}$$

and covariance matrix:

$$\boldsymbol{\Lambda}_{f_k^n\to\mathbf{x}_n(k)} = \mathbf{A}_n(k)\mathbf{P}_n(k)\mathbf{A}_n^{\mathrm{T}}(k) + \boldsymbol{\Phi}_n(k) \tag{6.115}$$

where $\boldsymbol{\Phi}_n(k) = \begin{bmatrix} 0 & 0 \\ 0 & \sigma_\omega^2 \end{bmatrix}$.

So $m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k))$ can be expressed as:

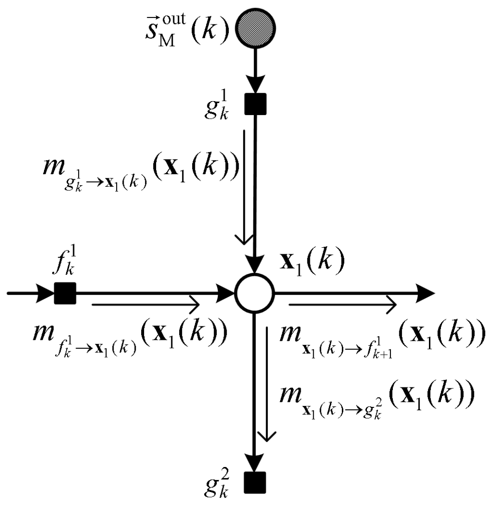
$$m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k)) \tag{6.116}$$
$$= \frac{1}{2\pi\left|\boldsymbol{\Lambda}_{f_k^n\to\mathbf{x}_n(k)}\right|} \exp\left(-\frac{1}{2}\left(\mathbf{x}_n(k)-\boldsymbol{\mu}_{f_k^n\to\mathbf{x}_n(k)}\right)^{\mathrm{T}} \boldsymbol{\Lambda}^{-1}_{f_k^n\to\mathbf{x}_n(k)} \left(\mathbf{x}_n(k)-\boldsymbol{\mu}_{f_k^n\to\mathbf{x}_n(k)}\right)\right)$$

Message Calculation Around Node $\mathbf{x}_1(k)$ As shown in Figure 6.15, the message from function node g_k^1 to variable node $\mathbf{x}_1(k)$, i.e., $m_{g_k^1\to\mathbf{x}_1(k)}$ is only determined by the function g_k^1, since no message enters this function node. According to the sum-product algorithm, $m_{g_k^1\to\mathbf{x}_1(k)}$ is given by:

$$m_{g_k^1\to\mathbf{x}_1(k)}(\mathbf{x}_1(k)) = g_k^1(\mathbf{x}_1(k)) \tag{6.117}$$

As $g_k^1(\mathbf{x}_1(k); \overrightarrow{s}_{\mathrm{M}}^{\mathrm{out}}(k))$ is given by (6.88), the expression of $m_{g_k^1\to\mathbf{x}_1(k)}(\mathbf{x}_1(k))$ is identical to (6.88), namely:

$$m_{g_k^1\to\mathbf{x}_1(k)}(\mathbf{x}_1(k)) = \frac{1}{\sqrt{2\pi}\sigma_\nu} \exp\left(-\frac{(\overrightarrow{s}_{\mathrm{M}}^{\mathrm{out}}(k) - \mathbf{c}_1^{\mathrm{T}}(k)\mathbf{x}_1(k))^2}{2\sigma_\nu^2}\right) \tag{6.118}$$



$\vec{s}_M^{out}(k)$
g_k^1
$m_{g_k^1\to x_1(k)}(x_1(k))$
f_k^1
$x_1(k)$
$m_{f_k^1\to x_1(k)}(x_1(k))$
$m_{x_1(k)\to f_{k+1}^1}(x_1(k))$
$m_{x_1(k)\to g_k^2}(x_1(k))$
g_k^2

Fig. 6.15. Message passing around $\mathrm{x}_1(k)$

According to Figure 6.15, the only incoming message at variable node $\mathrm{x}_1(k)$ are $m_{g_k^1\to \mathrm{x}_1(k)}(\mathrm{x}_1(k))$ and $m_{f_k^n\to \mathrm{x}_n(k)}(\mathrm{x}_n(k))$. Since now they have been calculated in (6.117) and (6.112), and the results are given in (6.118) and (6.116), we can calculate the estimated marginal $q(\mathrm{x}_1(k))$, according to (6.111) as follows:

$$q(\mathrm{x}_1(k)) = m_{g_k^1\to \mathrm{x}_1(k)}(\mathrm{x}_1(k)) \cdot m_{f_k^1\to \mathrm{x}_1(k)}(\mathrm{x}_1(k)) \tag{6.119}$$

Calculating the product of two Gaussian functions in (6.119), we can find out that $q(\mathrm{x}_1(k))$ is also a Gaussian function. Its covariance matrix is:

$$\mathbf{P}_1(k) = \left(\mathbf{c}_1(k)\sigma_\nu^{-2}\mathbf{c}_1^T(k) + \mathbf{\Lambda}^{-1}_{f_k^1\to \mathrm{x}_1(k)}\right)^{-1} \tag{6.120}$$

and the mean is:

$$\bar{\mathrm{x}}_1(k) = \mathbf{P}_1(k)\cdot\left(\mathbf{c}_1(k)\sigma_\nu^{-2}\vec{s}_{\mathrm{M}}^{\,\mathrm{out}}(k) + \mathbf{\Lambda}^{-1}_{f_k^1\to \mathrm{x}_1(k)}\mu_{f_k^1\to \mathrm{x}_1(k)}\right) \tag{6.121}$$

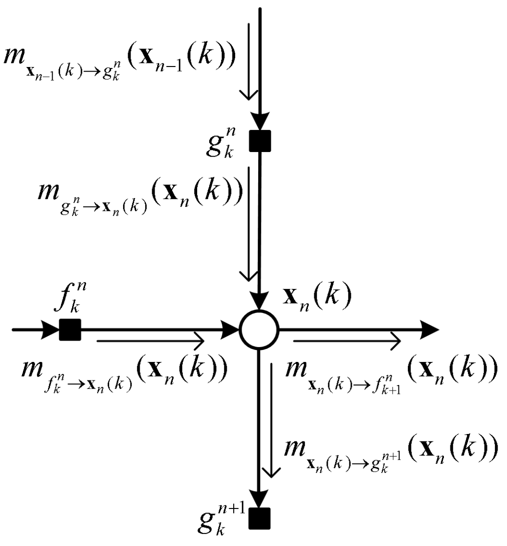
Message Calculation Around Node $\mathrm{x}_n(k)$ $(n > 1)$ The message from g_k^n to $\mathrm{x}_n(k)$ (for $n > 1$) is given by, using (4.13) and (6.111):

$$\begin{aligned} & m_{g_k^n\to \mathrm{x}_n(k)}\left(\mathrm{x}_n(k)\right) \\ &= \int\limits_{\mathrm{x}_{n-1}(k)} m_{\mathrm{x}_{n-1}(k)\to g_k^n}\left(\mathrm{x}_{n-1}(k)\right)\cdot g_k^n(\mathrm{x}_n(k)|\mathrm{x}_{n-1}(k)) \\ &= \int\limits_{\mathrm{x}_{n-1}(k)} q\left(\mathrm{x}_{n-1}(k)\right)\cdot g_k^n(\mathrm{x}_n(k)|\mathrm{x}_{n-1}(k)) \end{aligned} \tag{6.122}$$

Inserting (6.110) and (6.89) into (6.122) and computing the integration, we obtain:

$$m_{g_k^n\to \mathrm{x}_n(k)}(\mathrm{x}_n(k)) = \alpha\cdot\exp\left(-\frac{\left(\mathbf{c}_n^{\mathsf{T}}(k)\mathrm{x}_n(k) - \mathbf{b}_{n-1}^{\mathsf{T}}(k)\bar{\mathrm{x}}_{n-1}(k)\right)^2}{2\left(\sigma_\nu^2 + \mathbf{b}_{n-1}^{\mathsf{T}}(k)\mathbf{P}_{n-1}(k)\mathbf{b}_{i-1}(k)\right)}\right) \tag{6.123}$$

where α is the normalization factor.



$m_{\mathbf{x}_{n-1}(k)\to g_k^n}(\mathbf{x}_{n-1}(k))$
g_k^n
$m_{g_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k))$
f_k^n
$\mathbf{x}_n(k)$
$m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k))$
$m_{\mathbf{x}_n(k)\to f_{k+1}^n}(\mathbf{x}_n(k))$
$m_{\mathbf{x}_n(k)\to g_k^{n+1}}(\mathbf{x}_n(k))$
g_k^{n+1}

Fig. 6.16. Message passing around $\mathbf{x}_n(k)$

Then the posterior probability distribution of $\mathbf{x}_n(k)$ is calculated by:

$$q(\mathbf{x}_n(k)) = m_{f_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k)) \cdot m_{g_k^n\to\mathbf{x}_n(k)}(\mathbf{x}_n(k)) \tag{6.124}$$

Inserting the result of (6.116) and (6.123) into (6.124), it can be found out that the covariance matrix of $q(\mathbf{x}_n(k))$ is:

$$\mathbf{P}_n(k) = \left(\mathbf{c}_n(k)\left[\sigma_\nu^2 + \mathbf{b}_{n-1}^T(k)\mathbf{P}_{n-1}(k)\mathbf{b}_{n-1}(k)\right]^{-1}\mathbf{c}_n^T(k) + \mathbf{\Lambda}_{f_k^n\to\mathbf{x}_i(k)}^{-1}\right)^{-1} \tag{6.125}$$

and the mean of $q(\mathbf{x}_n(k))$ is:

$$\mathbf{x}_n(k) = \mathbf{P}_n(k)\left\{\begin{array}{c} \mathbf{c}_n(k)\left[\sigma_\nu^2 + \mathbf{b}_{n-1}^T(k)\mathbf{P}_{n-1}(k)\mathbf{b}_{n-1}(k)\right]^{-1}\mathbf{b}_{n-1}^T(k)\bar{\mathbf{x}}_{n-1}(k) \\ +\mathbf{\Lambda}_{f_k^n\to\mathbf{x}_i(k)}^{-1}\boldsymbol{\mu}_{f_k^n\to\mathbf{x}_i(k)} \end{array}\right\} \tag{6.126}$$

Using (6.110)-(6.126), the posterior probability density function for all variable nodes can be calculated.

Figure 6.17 illustrates the information exchanged between the slaves. All this information can be delivered by the Sync messages.

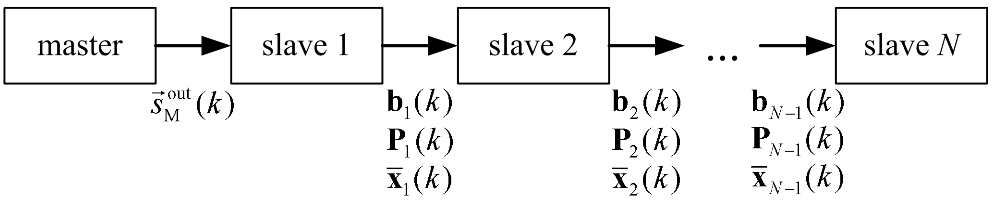


Fig. 6.17. Information transmission between network elements

The synchronization performance of this distributed Kalman filter is evaluated by simulations. The results, i.e., synchronization error is shown in Figure 6.18. This simulation result is obtained by choosing an appropriate value for the parameter σ_ω.

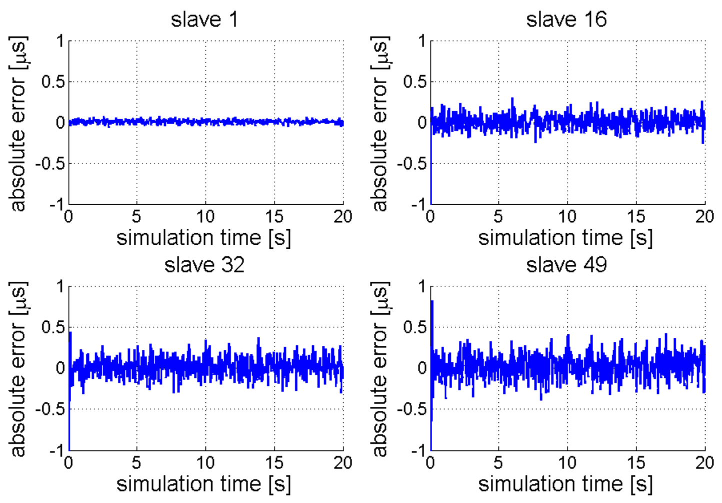


Fig. 6.18. Synchronization error of the distributed Kalman filter

6.5.5 Sequential Kalman filter for master time estimation

In the state space model derived in the previous sections, we model the frequency ratio as a hidden variable, which doubles the number of the hidden variables. In this section, we present another state-space model where the frequency ratio will be regarded as a parameter of the model. It will be estimated separately and the estimate will be used as a deterministic input to the state-space model.

6.5.5.1 State space model

As we just need to modify the modeling and the estimation of the frequency ratio, we can still use formula (6.63), (6.76) and (6.77) in the new probabilistic model. Let us summarize them here for clarity:

- (6.63): $\vec{x}_n^{\text{in}}(k) = \vec{x}_n^{\text{in}}(k-1) + a_n(k) \cdot r_{\text{MS}_n}(k)$
- (6.76): $\vec{s}_{\text{M}}^{\text{out}}(k) = \vec{x}_1^{\text{in}}(k) - d_1(k) \cdot r_{\text{MS}_1}(k) + \nu_1(k)$
- (6.77): $\vec{x}_{n-1}^{\text{in}}(k) + b_{n-1}(k) \cdot r_{\text{MS}_{n-1}}(k) = \vec{x}_n^{\text{in}}(k) - d_n(k) \cdot r_{\text{MS}_n}(k) + \nu_n(k)$

To further decouple the problem, we now define different frequency ratios and rewrite the equations as follows:

$$\vec{x}_n^{\text{in}}(k) = \vec{x}_n^{\text{in}}(k-1) + a_n(k) \cdot r_{\text{MS}_n}^a(k) \tag{6.127}$$

$$\vec{s}_{\text{M}}^{\text{out}}(k) = \vec{x}_1^{\text{in}}(k) - d_1(k) \cdot r_{\text{MS}_1}^d(k) + \nu_1(k) \tag{6.128}$$

$$\vec{x}_{n-1}^{\text{in}}(k) + b_{n-1}(k) \cdot r_{\text{MS}_{n-1}}^b(k) = \vec{x}_n^{\text{in}}(k) - d_n(k) \cdot r_{\text{MS}_n}^d(k) + \nu_n(k) \tag{6.129}$$

where $r_{\text{MS}_n}^a(k)$, $r_{\text{MS}_n}^b(k)$ and $r_{\text{MS}_n}^d(k)$ represent different values of the frequency ratio between master and slave n.

At slave 1, we estimate the frequency ratio between master clock and slave 1's clock by:

$$\hat{r}_{\mathtt{MS_1}}(k) = \frac{\overrightarrow{s}^{\mathtt{out}}_{\mathtt{M}}(k) - \overrightarrow{s}^{\mathtt{out}}_{\mathtt{M}}(k-1)}{\overrightarrow{s}^{\mathtt{in}}_{\mathtt{S_1}}(k) - \overrightarrow{s}^{\mathtt{in}}_{\mathtt{S_1}}(k-1)} \tag{6.130}$$

Usually, several last estimates are averaged to improve the estimation accuracy. Let us use $\eta_1(k)$ to denote the estimation error, i.e.,:

$$r^{\mathtt{TT}}_{\mathtt{MS_1}}(k) = \hat{r}_{\mathtt{MS_1}}(k) + \eta^{\mathtt{TT}}_1(k) \tag{6.131}$$

where $\mathtt{TT} \in \{a, b, d\}$.

The property of the frequency ratio estimation error has be analytically studied in [79]. Improvements of the estimation accuracy have been proposed in [80]. It can be seen that the final estimation uncertainty can be well modeled by a Gaussian random variable. We further assume that $\eta^a_1(k)$, $\eta^b_1(k)$ and $\eta^d_1(k)$ are three independence random variables that are generated from the same Gaussian distribution, i.e., $\eta^{\mathtt{TT}}_1(k) \sim \mathcal{N}(0, \sigma^2_\eta)$ for $(\mathtt{TT} = a, b, c)$. For simplicity, we assume that the variance of $\eta^{\mathtt{TT}}$ is time and element independent. The variance of the Gaussian distribution is related to the stamping error and the quantization error in the time stamping. The relationship was shown in [80]. In case of frequency drift, the reasonable choice of the value of σ_η also depends on the speed of the frequency drift. If the value of σ_η is small, then we assume that the frequency drift is slow. Otherwise, we assume the frequency drift is fast.

Now let us insert (6.131) into (6.127) with $\mathtt{TT} = a$, then we obtain:

$$\overrightarrow{x}^{\mathtt{in}}_1(k) = \overrightarrow{x}^{\mathtt{in}}_1(k-1) + a_1(k) \cdot \hat{r}_{\mathtt{MS_1}}(k) + a_1(k) \cdot \eta^a_1(k) \tag{6.132}$$

Let us define:

$$\epsilon_1(k) = a_1(k) \cdot \eta^a_1(k) \tag{6.133}$$

and rewrite (6.132) as follows:

$$\overrightarrow{x}^{\mathtt{in}}_1(k) = \overrightarrow{x}^{\mathtt{in}}_1(k-1) + a_1(k) \cdot \hat{r}_{\mathtt{MS_1}}(k) + \epsilon_1(k) \tag{6.134}$$

which constitutes the new *state transition model* at slave 1.

Inserting (6.131) into (6.128) with $\mathtt{TT} = d$, we obtain:

$$\overrightarrow{s}^{\mathtt{out}}_{\mathtt{M}}(k) = \overrightarrow{x}^{\mathtt{in}}_1(k) - d_1(k) \cdot \hat{r}_{\mathtt{MS_1}}(k) - d_1(k) \cdot \eta^d_1(k) + \nu_1(k) \tag{6.135}$$

Let us define:

$$\zeta_1(k) = -d_1(k) \cdot \eta^d_1(k) + \nu_1(k) \tag{6.136}$$

Then we rewrite (6.135) as follows:

$$\overrightarrow{s}^{\mathtt{out}}_{\mathtt{M}}(k) = \overrightarrow{x}^{\mathtt{in}}_1(k) - d_1(k) \cdot \hat{r}_{\mathtt{MS_1}}(k) + \zeta_1(k) \tag{6.137}$$

which constitute the *observation model* at slave 1.

At slave n, the frequency ratio between master clock and slave n's clock is estimated by:

$$\hat{r}_{\mathtt{MS}_n}(k) = \frac{\hat{x}^{\mathtt{out}}_{n-1}(k) - \hat{x}^{\mathtt{out}}_{n-1}(k-1)}{\overrightarrow{s}^{\mathtt{in}}_{\mathtt{S}_n}(k) - \overrightarrow{s}^{\mathtt{in}}_{\mathtt{S}_n}(k-1)} \tag{6.138}$$

where $\hat{x}^{\mathtt{out}}_{n-1}(k)$ is the estimate of $\overrightarrow{x}^{\mathtt{out}}_{n-1}(k)$ which can be computed by:

$$\hat{x}^{\mathtt{out}}_{n-1}(k) = \hat{x}^{\mathtt{in}}_{n-1}(k) + b_{n-1}(k) \cdot \hat{r}_{\mathtt{MS}_{n-1}}(k) \tag{6.139}$$

The estimation of $\overrightarrow{x}^{\mathtt{in}}_{n-1}(k)$, i.e., $\hat{x}^{\mathtt{in}}_{n-1}(k)$ will be explained later.

Let us use $\eta_n(k)$ to model the estimation error, then:

$$r^{\mathtt{TT}}_{\mathtt{MS}_n}(k) = \hat{r}_{\mathtt{MS}_n}(k) + \eta^{\mathtt{TT}}_n(k) \tag{6.140}$$

where $\mathtt{TT} \in \{a, b, d\}$. Again we assume that $\eta^a_n(k)$, $\eta^b_n(k)$ and $\eta^d_n(k)$ are three independence random variables that are generated from the same Gaussian distribution, i.e., $\eta^{\mathtt{TT}}_n(k) \sim \mathcal{N}(0, \sigma^2_\eta)$ for $(\mathtt{TT} = a, b, c)$.

Inserting (6.140) into (6.127), we obtain:

$$\overrightarrow{x}^{\mathtt{in}}_n(k) = \overrightarrow{x}^{\mathtt{in}}_n(k-1) + a_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) + a_n(k) \cdot \eta^a_n(k) \tag{6.141}$$

Let us define:

$$\epsilon_n(k) = a_n(k) \cdot \eta^a_n(k) \tag{6.142}$$

and rewrite (6.141) as follows:

$$\overrightarrow{x}^{\mathtt{in}}_n(k) = \overrightarrow{x}^{\mathtt{in}}_n(k-1) + a_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) + \epsilon_n(k) \tag{6.143}$$

which constitutes the new *state transition model* at slave n.

Inserting (6.66) and (6.140) into (6.129), we obtain:

$$\overrightarrow{x}^{\mathtt{out}}_{n-1}(k) = \overrightarrow{x}^{\mathtt{in}}_n(k) - d_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) - d_n(k) \cdot \eta^d_n(k) + \nu_n(k) \tag{6.144}$$

Let us define:

$$\zeta_n(k) = -d_n(k) \cdot \eta^d_n(k) + \nu_n(k) \tag{6.145}$$

Then (6.144) can be rewritten as:

$$\overrightarrow{x}^{\mathtt{out}}_{n-1}(k) = \overrightarrow{x}^{\mathtt{in}}_n(k) - d_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) + \zeta_n(k) \tag{6.146}$$

which constitutes the *coupling model* at slave n.

Now let us summarize the most important formulas of the new state space model:

- (6.134) $\overrightarrow{x}^{\mathtt{in}}_1(k) = \overrightarrow{x}^{\mathtt{in}}_1(k-1) + a_1(k) \cdot \hat{r}_{\mathtt{MS}_1}(k) + \epsilon_1(k)$
- (6.137) $\overrightarrow{s}^{\mathtt{out}}_{\mathtt{M}}(k) = \overrightarrow{x}^{\mathtt{in}}_1(k) - d_1(k) \cdot \hat{r}_{\mathtt{MS}_1}(k) + \zeta_1(k)$
- (6.143) $\overrightarrow{x}^{\mathtt{in}}_n(k) = \overrightarrow{x}^{\mathtt{in}}_n(k-1) + a_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) + \epsilon_n(k)$
- (6.146) $\overrightarrow{x}^{\mathtt{out}}_{n-1}(k) = \overrightarrow{x}^{\mathtt{in}}_n(k) - d_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) + \zeta_n(k)$

The variables in these four equations fall into the following categories:

- Known variable: $a_1(k)$, $a_n(k)$, $\overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k)$ (from the time-stamps); $d_n(k)$ (from the line delay estimation (6.46)); $\hat{r}_{\mathtt{MS}_n}(k)$ (from the frequency ratio estimation (6.130) and (6.138))
- Hidden state variables: $\overrightarrow{x}^{\,\mathtt{in}}_n(k)$, $\overrightarrow{x}^{\,\mathtt{out}}_n(k)$
- Random variable with known distributions: $\epsilon_n(k)$, $\zeta_n(k)$. These random variables contain all the random errors in the system. Each of them is a linear combination of several other random variables. As a result, their distribution can be well approximated by a Gaussian distribution. In the following derivations, we assume that $\epsilon_n(k) \sim \mathcal{N}(0, \sigma^2_{\epsilon_n(k)})$ and $\zeta_n(k) \sim \mathcal{N}(0, \sigma^2_{\zeta_n(k)})$. According to (6.133) and (6.142)

$$\sigma^2_{\epsilon_n(k)} = a_n^2(k)\sigma^2_\eta \tag{6.147}$$

According to (6.136) and (6.145),

$$\sigma^2_{\zeta_n(k)} = d_n^2(k)\sigma^2_\eta + \sigma^2_\nu \tag{6.148}$$

From the derivation, we can easily see that $\epsilon_n(k)$ and $\zeta_n(k)$ are independent random variables.

6.5.5.2 Distributed Kalman filter for state estimation

The probabilistic model derived in the previous section is linear and all the random variables therein are Gaussian. As a result, the posterior probability distribution of the hidden variables is also Gaussian, i.e., $\overrightarrow{x}^{\,\mathtt{in}}_n(k) \sim \mathcal{N}(\hat{x}^{\mathtt{in}}_n(k), v^{\mathtt{in}}_n(k))$ and $\overrightarrow{x}^{\,\mathtt{out}}_n(k) \sim \mathcal{N}(\hat{x}^{\mathtt{out}}_n(k), v^{\mathtt{out}}_n(k))$. We can use a Kalman filter to estimate the mean $\hat{x}^{\mathtt{in}}_n(k)$ and the variance $v^{\mathtt{out}}_n(k)$ of the Gaussian distribution. Let us define the estimation error:

$$\tilde{x}^{\mathtt{in}}_n(k) = x^{\mathtt{in}}_n(k) - \hat{x}^{\mathtt{in}}_n(k) \tag{6.149}$$

$$\tilde{x}^{\mathtt{out}}_n(k) = x^{\mathtt{out}}_n(k) - \hat{x}^{\mathtt{out}}_n(k) \tag{6.150}$$

Then:

$$E\{\tilde{x}^{\mathtt{in}}_n(k)\} = E\{\tilde{x}^{\mathtt{out}}_n(k)\} = 0 \tag{6.151}$$

$$E\{(\tilde{x}^{\mathtt{in}}_n(k))^2\} = v^{\mathtt{in}}_n(k) \tag{6.152}$$

$$E\{(\tilde{x}^{\mathtt{out}}_n(k))^2\} = v^{\mathtt{out}}_n(k) \tag{6.153}$$

For the convenience of a unified expression of the state-space model for slave 1 and all the other slaves, we define:

$$\hat{x}^{\mathtt{out}}_0(k) = \overrightarrow{s}^{\,\mathtt{out}}_{\mathtt{M}}(k) \tag{6.154}$$

and

$$v^{\mathtt{out}}_0(k) = 0 \tag{6.155}$$

Using (6.150) and (6.154), we can unify the observation model in (6.137) and the coupling model in (6.146) by expressing them as:

$$\hat{x}^{\mathtt{out}}_{n-1}(k) = \overrightarrow{x}^{\,\mathtt{in}}_n(k) - d_n(k)\cdot \hat{r}_{\mathtt{MS}_n}(k) + \zeta_n(k) - \tilde{x}^{\mathtt{out}}_{n-1}(k) \quad \text{for} \quad \mathrm{n} \geq 1 \tag{6.156}$$

where according to (6.155), $\tilde{x}_0^{\texttt{out}}(k) = 0$.

Similarly, we unify the state transition model for all slave elements:

$$\overrightarrow{x}_n^{\texttt{in}}(k) = \overrightarrow{x}_n^{\texttt{in}}(k-1) + a_n(k) \cdot \hat{r}_{\texttt{MS}_n}(k) + \epsilon_n(k) \quad \text{for} \quad \mathsf{n} \geq 1 \tag{6.157}$$

For $n > 1$, $\hat{x}_{n-1}^{\texttt{out}}(k)$ is computed from $\hat{x}_{n-1}^{\texttt{in}}(k)$ by:

$$\hat{x}_{n-1}^{\texttt{out}}(k) = \hat{x}_{n-1}^{\texttt{in}}(k) + b_{n-1}(k) \cdot \hat{r}_{\texttt{MS}_{n-1}}(k) \tag{6.158}$$

and the variance is computed by:

$$\begin{aligned} v_{n-1}^{\texttt{out}}(k) &= E\{(\tilde{x}_{n-1}^{\texttt{out}}(k))^2\} \\ &= E\{((\overrightarrow{x}_{n-1}^{\texttt{in}}(k) + b_{n-1}(k) \cdot r_{\texttt{MS}_{n-1}}^b(k)) - (\hat{x}_{n-1}^{\texttt{in}}(k) + b_{n-1}(k) \cdot \hat{r}_{\texttt{MS}_{n-1}}(k)))^2\} \\ &= E\{(\tilde{x}_{n-1}^{\texttt{in}}(k) + b_{n-1}(k) \cdot \eta_{n-1}^b(k))^2\} \\ &= v_{n-1}^{\texttt{in}}(k) + b_{n-1}^2(k) \cdot \sigma_\eta^2 \end{aligned} \tag{6.159}$$

where in the last line is based on the uncorrelatedness between random variables $\tilde{x}_{n-1}^{\texttt{in}}(k)$ and $\eta_{n-1}^b(k)$.

For the Kalman filtering, we initialize the state estimation by:

$$\hat{r}_{\texttt{MS}_n}(0) = 1 \tag{6.160}$$

$$\hat{x}_n^{\texttt{in}}(0) = \hat{x}_{n-1}^{\texttt{out}}(0) + d_n(0) \cdot \hat{r}_{\texttt{MS}_n}(0) \tag{6.161}$$

$$v_n^{\texttt{in}}(0) = v_{n-1}^{\texttt{out}}(0) + \sigma_{\zeta_n(k)}^2 \tag{6.162}$$

$$\hat{x}_n^{\texttt{out}}(0) = \hat{x}_n^{\texttt{in}}(0) + b_n(0) \cdot \hat{r}_{\texttt{MS}_n}(0) \tag{6.163}$$

$$v_n^{\texttt{out}}(0) = v_n^{\texttt{in}}(0) + b_n^2(0) \cdot \sigma_\eta^2 \tag{6.164}$$

In the initial phase, no enough Sync messages are available to estimate RCF. So we initialize it with 1. Equations (6.161) and (6.163) make sense according to the physics of the problem. Equation (6.162) results from the fact that the uncertainties at slave n, modeled by $\zeta_n(k)$ increase the variance by $\sigma_{\zeta_n(k)}^2$. Equation (6.164) is obtained from (6.159).

The Kalman filter contains two steps.

- In the first step, the prediction step, it computes, based on the results from the previous time step:

  Predicted mean:

  $$\hat{x}_n^{\texttt{in}}(k, k-1) = \hat{x}_n^{\texttt{in}}(k-1) + a_n(k) \cdot \hat{r}_{\texttt{MS}_n}(k) \tag{6.165}$$

  Predicted variance:

  $$v_n^{\texttt{in}}(k, k-1) = v_n^{\texttt{in}}(k-1) + \sigma_{\epsilon_n(k)}^2 \tag{6.166}$$

- The second step, the updating step, corrects the prediction by using the observation model (6.156). In this step, the following computations will be carried out, according to [108] and the derivation in Section 4.3.

  Measurement residual:

$$z_n^{\mathtt{in}}(k) = \overrightarrow{x}_{n-1}^{\mathtt{out}}(k) + d_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) + \zeta_n(k) - \tilde{x}_{n-1}^{\mathtt{out}}(k) - x_n^{\mathtt{in}}(k, k-1) \tag{6.167}$$

  Residual mean:

$$\hat{z}_n^{\mathtt{in}}(k) = \hat{x}_{n-1}^{\mathtt{out}}(k) + d_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) - \hat{x}_n^{\mathtt{in}}(k, k-1) \tag{6.168}$$

  where we have used the fact that both $\zeta_n(k)$ and $\tilde{x}_{n-1}^{\mathtt{out}}(k)$ are zero mean random variables.

  Residual variance:

$$\begin{aligned} s_n^{\mathtt{in}}(k) &= E\{(z_n^{\mathtt{in}}(k) - \hat{z}_n^{\mathtt{in}}(k))^2\} \\ &= E\{(\tilde{x}_{n-1}^{\mathtt{out}}(k))^2\} + E\{\zeta_n^2(k)\} + E\{(\tilde{x}_n^{\mathtt{in}}(k, k-1))^2\} \\ &= v_{n-1}^{\mathtt{out}}(k) + \sigma^2_{\zeta_n(k)} + v_n^{\mathtt{in}}(k, k-1) \end{aligned} \tag{6.169}$$

  Kalman gain:

$$k_n^{\mathtt{in}}(k) = \frac{v_n^{\mathtt{in}}(k, k-1)}{s_n^{\mathtt{in}}(k)} \tag{6.170}$$

  Posterior mean:

$$\hat{x}_n^{\mathtt{in}}(k) = \hat{x}_n^{\mathtt{in}}(k, k-1) + k_n^{\mathtt{in}}(k) \cdot z_n^{\mathtt{in}}(k) \tag{6.171}$$

  Posterior variance:

$$v_n^{\mathtt{in}}(k) = (1 - k_n^{\mathtt{in}}(k)) \cdot v_n^{\mathtt{in}}(k, k-1) \tag{6.172}$$

Using the above equations, all the state variables can be estimated. It can be seen that the computations are local. Each slave n can use a local Kalman filter to estimation a master time $\hat{x}_n^{\mathtt{in}}(k)$ that corresponds to its own time $\overrightarrow{s}_{\mathtt{S}_n}^{\mathtt{in}}(k)$. The estimation results, i.e., $\hat{x}_n^{\mathtt{out}}(k)$ and $v_n^{\mathtt{out}}(k)$ will be packaged in the Sync message and transmitted to the next slave element. Information exchange between the elements is illustrated in Figure 6.19.

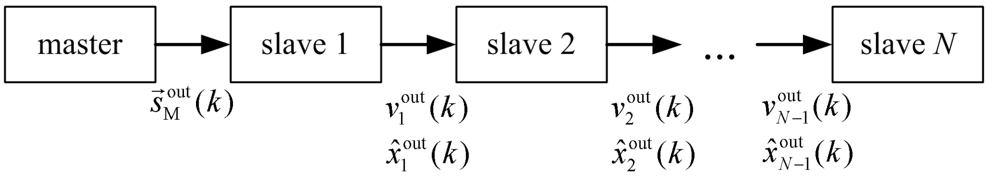

master
slave 1
slave 2
...
slave N
$\vec{s}_{\mathrm{M}}^{\mathrm{out}}(k)$
$v_1^{\mathrm{out}}(k)$
$\hat{x}_1^{\mathrm{out}}(k)$
$v_2^{\mathrm{out}}(k)$
$\hat{x}_2^{\mathrm{out}}(k)$
$v_{N-1}^{\mathrm{out}}(k)$
$\hat{x}_{N-1}^{\mathrm{out}}(k)$

Fig. 6.19. Information transmission between network elements

Simulation results in Figure 6.20 illustrate the performance of this distributed Kalman filter approach. This simulation result is obtained by choosing an appropriate value for the parameter σ_η.

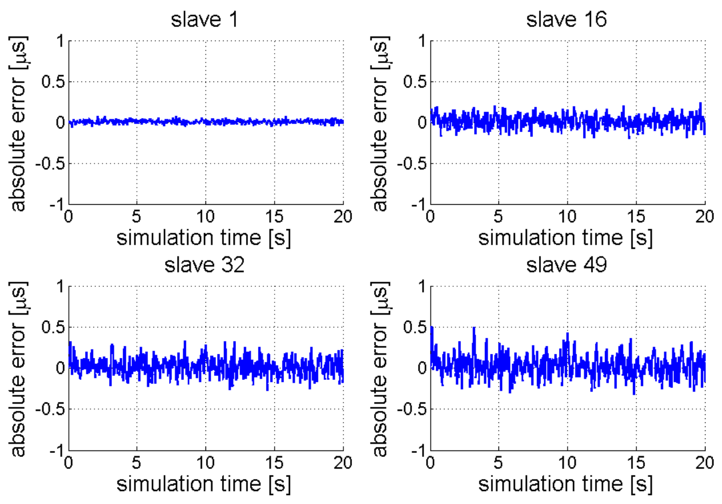


Fig. 6.20. Synchronization error of sequential Kalman filter

6.6 Numerical evaluation of synchronization algorithms

In this section, we provide more comprehensive simulation results of the synchronization algorithms introduced in the previous sections, i.e.,

- standard synchronization algorithm (standard)
- centralized Kalman filter (CKF)
- sum-product algorithm (SPA)
- sequential Kalman filter (SKF)

Our simulation parameters are obtained from the direct measurement of the industrial automation network. The simulation results are realistic.

We consider two simulation settings. In the first setting, the clocks have constant frequency. In the second setting, we consider the impact of frequency drift on different algorithms.

6.6.1 Synchronization with constant clock frequency

In the previous sections, we demonstrated the synchronization performance of different synchronization algorithms when the key parameters are carefully chosen. The centralized Kalman filter and the sum-product algorithm are based on the same probabilistic model. In that model the most important parameter is σ_ω. This parameter describes the dynamics of the frequencies of the clock. In the probabilistic model used by the sequential Kalman filter, the key parameter is the uncertainty of the frequency estimate, i.e., σ_η. This parameter also characterizes the change of the frequency. Only by choosing the appropriate values for these key parameters, the best synchronization performance can be achieved. To illustrate this, we simulate all the algorithms with different values of the parameters. We evaluate the performance by using the root mean square error (RMSE) of the master time estimates, i.e., the RMSE of $\hat{x}_n^{\mathtt{in}}(k)$.

In the simulation, we set:

$$\sigma_\omega = \sigma_\eta = \rho \quad \text{for} \quad n = 1, \dots N \quad \text{and} \quad k = 1, \dots K \tag{6.173}$$

In Figure 6.21, we plot the RMSE vs. ρ curves for all different algorithms at the first slave in the line topology with 50 elements.

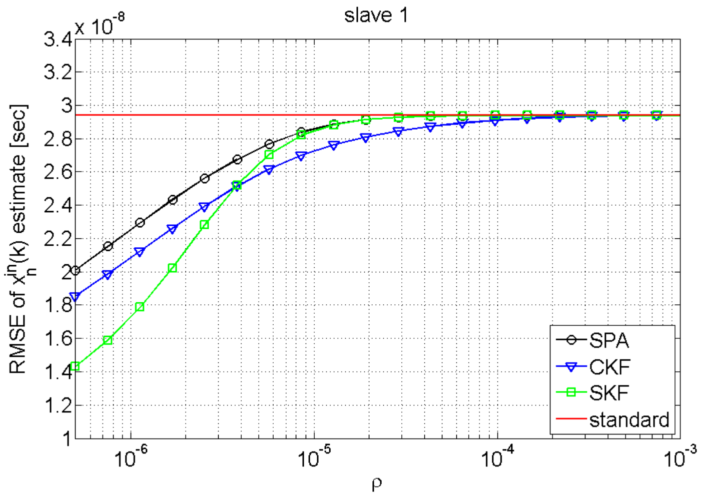


Fig. 6.21. The RMSE with different values of the key parameters (slave 1)

In this simulation scenario, the frequency ratio is always constant. To reflect this, in the probabilistic model used by the centralized Kalman filter and the sum product algorithm, the value of σ_ω should be small. In the extreme case when $\sigma_\omega = 0$, the state transition equation (6.60) becomes $r_{\mathtt{MS}_n}(k) = r_{\mathtt{MS}_n}(k-1)$, which accurately reflects the fact that the frequency ratio is constant. From the simulation results we can see that the smaller the value of ρ, the better the synchronization performance. The same principle applies for the probabilistic model used by the sequential Kalman filter.

If we look at the RMSE performance at remote slaves (e.g., slave 9 in Figure 6.22 and slave 49 in Figure6.23), the curves of sequential Kalman filter and the sum product algorithm becomes irregular. This is because we have chosen the same value of σ_ω and σ_η for all slaves which is not optimal. Both distributed methods estimate the hidden state variables in a given slave based on the results of the hidden state estimation in upstream slaves. The parameters σ_ω and σ_η not only characterize the stability of the frequency ratio, but also reflect the error inherited from the previous element. As a consequence, the optimal choice of σ_ω and σ_η should result in different values for different slave elements. However, for simplicity, we choose the same value of σ_ω and σ_η for all slave elements which leads to the RMSE curves shown in Figure 6.22. For centralized Kalman filter, such a configuration is fine since the state variables of all slave elements are estimated jointly. Finding out the optimal configuration of the parameters for all slave elements is not a trivial task, no matter numerically or analytically.

Another interesting observation is that the centralized Kalman filter and the sum-product algorithm can be worse than the standard algorithm when the value of σ_ω is too large.

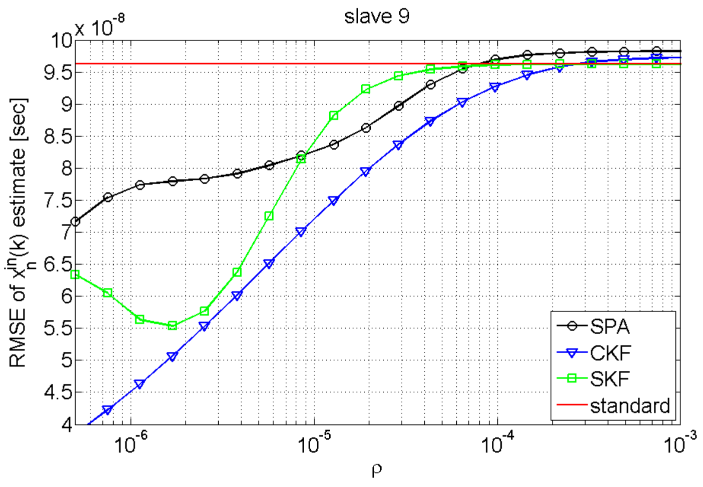


Fig. 6.22. The RMSE with different values of the key parameters (slave 9)

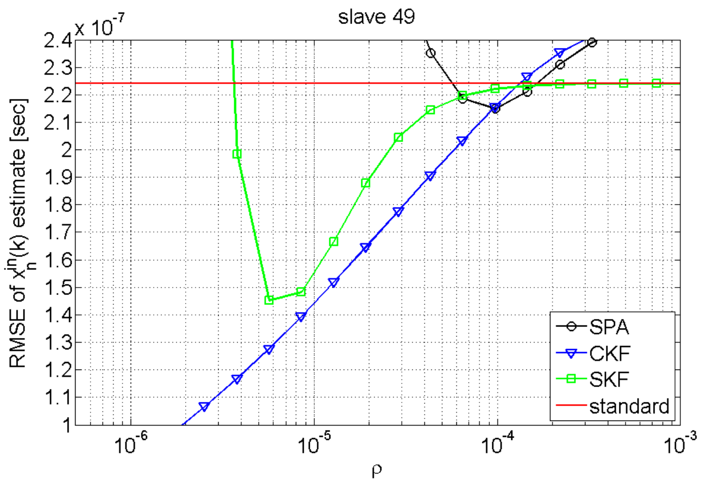


Fig. 6.23. The RMSE with different values of the key parameters (slave 49)

However, the sequential Kalman filter is never worse than the standard algorithm when the parameter $\rho = \sigma_\eta$ is in the range $[4 \times 10^{-6}, 1 \times 10^{-3}]$. Actually, it can be proved that (see Appendix 6.A) when the value of $\rho = \sigma_\eta$ is in a given range, the RMSE performance of sequential Kalman filter is identical to that of the standard algorithm.

6.6.2 Synchronization under adverse environmental effects

The environment can influence the stability of the oscillators. For example, dramatical temperature changes, mechanical shocks and vibrations may cause the frequency of the oscillators to drift. In this section, we examine the performance of different synchronization algorithms under these environmental effects.

6.6.2.1 Effect of temperature change

Temperature change may cause the oscillator's frequency to drift. In the automation and manufacturing environment, temperature may increase or decrease dramatically. We simulate the synchronization algorithms mentioned in this chapter and check their performance under temperature change.

The temperature change we investigate in the simulation is illustrated in Figure 6.24.

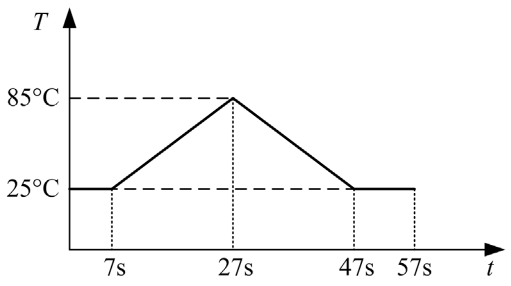


Fig. 6.24. Profile of temperature change

Detailed discussion on the relation between temperature change and the frequency drift can be found in [105]. In this chapter, we assume that the speed of the frequency drift is proportional to the speed of the temperature change. Relevant parameters are summarized in Table 6.2

Parameter	Value
Maximum temperature	85°C
Minimum temperature	25°C
Initial temperature	25°C
Speed of temperature change	3K/s
Frequency drift w.r.t. temperature change	1ppm/K
Speed of frequency drift caused by temperature change	3ppm/s

Table 6.2. Parameters of temperature change

Our first simulation scenario assumes temperature change at the master. We first study the influence of the parameters on the performance of the algorithms. We plot the RMSE versus parameter $\rho = \sigma_\omega = \sigma_\eta$ curve for each algorithm in Figure 6.25, 6.26 and 6.27 for slave 1, 9 and 49 respectively.

From the results, we can see that all the probabilistic approaches produce U-shape curves. This is due to the fact that the parameter ρ being very small indicates that the variances of $\omega_n(k)$ and $\eta_n(k)$ are small in (6.60) and (6.140), which can be interpreted that the probabilistic models assume constant frequency. However, due to the temperature change, the true frequency changes. Therefore, choosing ρ to be small introduces model mismatch problem. On the other hand, if the value of ρ is too large, then we assume an inaccurate state transition model. The estimation will give more importance to the measurements. In this case, the measurement noise cannot be well filtered out. As a consequence, there is always a tradeoff between model accuracy and noise filtering. To illustrate this, we plot the absolute synchronization error achieved by centralized Kalman filter by choosing different values for $\rho = \sigma_\omega$.

In Figure 6.28, a a small value for $\rho = \sigma_\omega$ is chosen. We observe biased estimation error when the temperature is constantly changing. The bias in the estimation is caused by the wrong assumption that the frequency ratio is constant. Due to that, the slave element

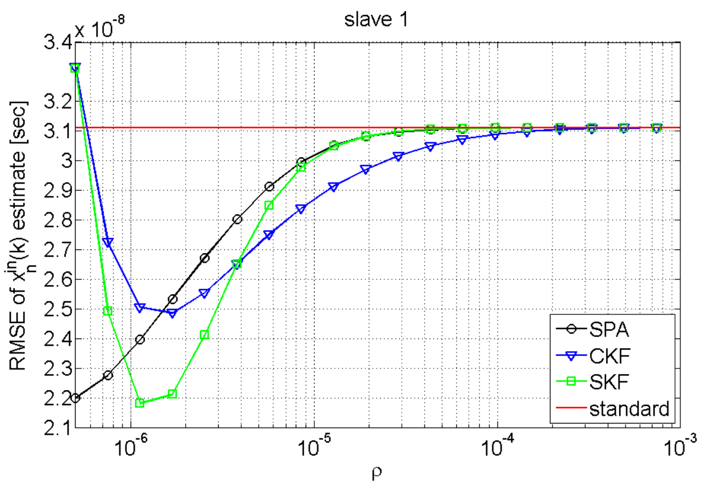


Fig. 6.25. The RMSE with different values of the key parameters (slave 1)

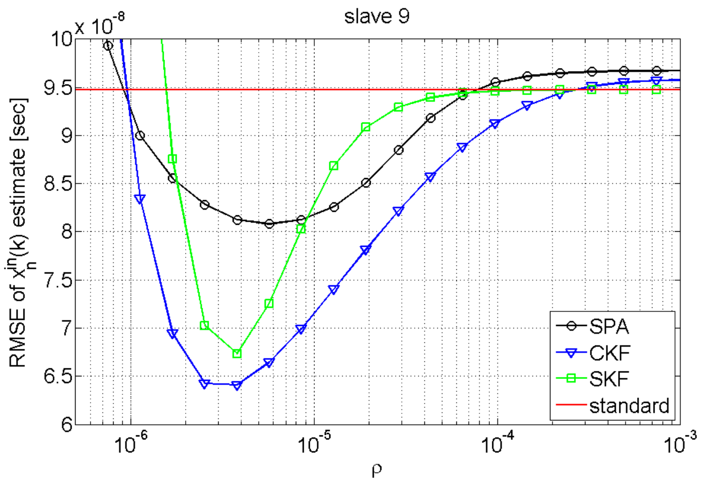


Fig. 6.26. The RMSE with different values of the key parameters (slave 9)

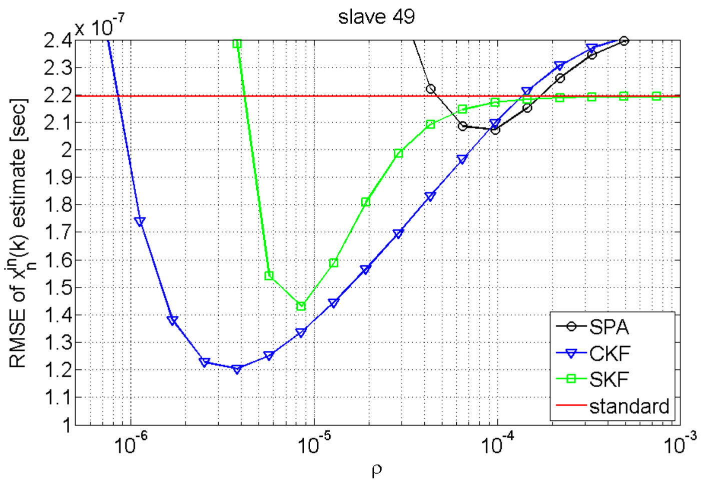


Fig. 6.27. The RMSE with different values of the key parameters (slave 49)

cannot track the actual frequency of the master clock. This phenomenon was analytically studied in [81, 97]. Methods are proposed in [82] to remove this estimation offset

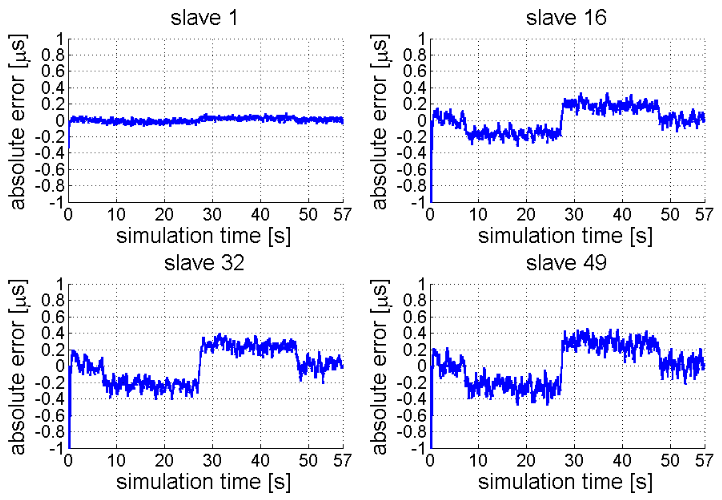


Fig. 6.28. Synchronization performance of CKF when $\rho = 7 \times 10^{-7}$ with temperature change at master

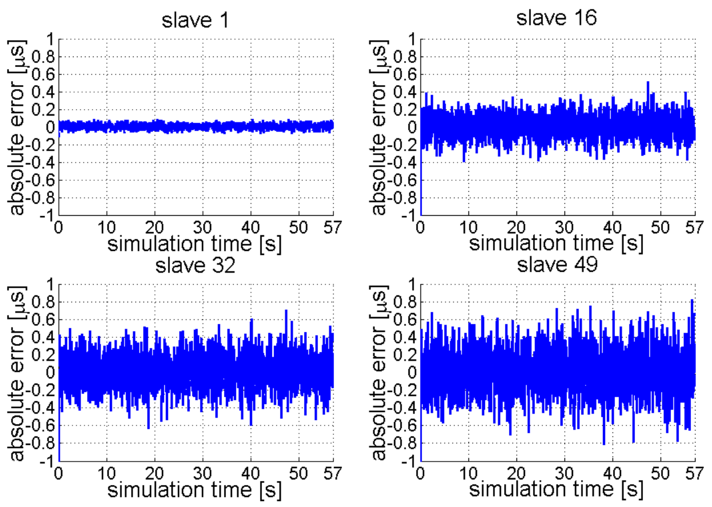


Fig. 6.29. Synchronization performance of CKF when $\rho = 1 \times 10^{-3}$ with temperature change at master

for the standard synchronization algorithm. In Figure 6.30 and Figure 6.29, the constant frequency assumption is relaxed by assigning large values to $\rho = \sigma_\omega$. Therefore the slave can track the master clock frequency change better.

High frequency fluctuations are present in the synchronization error curves, which reveal the result of filtering the measurement noise. It can be seen that the smaller the value of $\rho = \sigma_\omega$, the better the reduction of measurement noises, i.e., the smaller the fluctuations. In the case that the value of $\rho = \sigma_\omega$ is too large, the measurement noise will overtake the model mismatch problem to become the dominating error contributor, which is evidenced by the results in (6.29).

Now let us see what happens if the temperature change takes place at slave 1 only. The results of using centralized Kalman filter is shown in Figure 6.31.

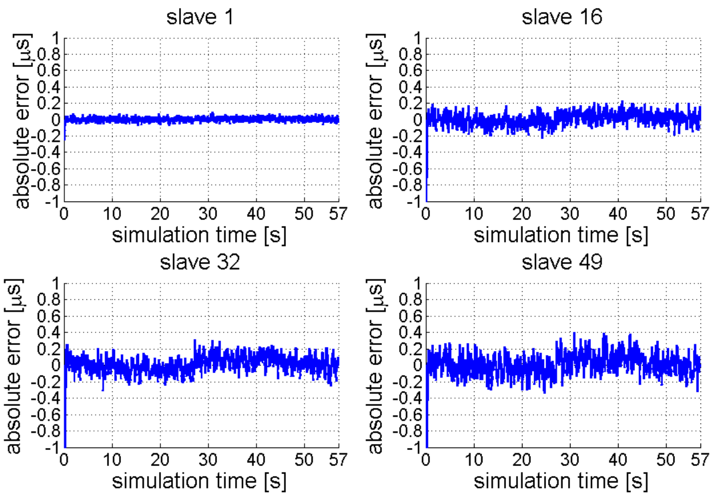


Fig. 6.30. Synchronization performance of CKF when $\rho = 3 \times 10^{-6}$ with temperature change at master

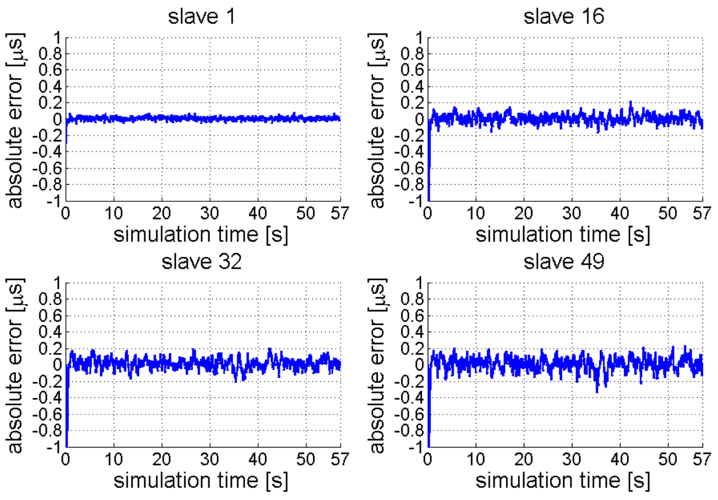


Fig. 6.31. Synchronization performance of CKF with temperature change at slave 1 when $\rho = 7 \times 10^{-7}$

Comparing Figure 6.31 with Figure 6.28 we can see that temperature change taking place at a single slave element has much less influence on the overall synchronization performance than a temperature change at the master. It has been analyzed in [98] that if only one element is heating, the estimation offset observed at the end of the line is much larger if this element is the master and not a slave. Only if all slaves exhibit identical non-zero frequency drift do they match the effect of ”master only heating”. This is verified by the simulation results shown in Figure 6.32 where all the slaves are having temperature change.

From the curves shown in Figure 6.25, 6.26 and 6.27, we can identify the best choice of the value of ρ for each synchronization algorithm. Now we plot the synchronization error performance achieved by different algorithms with the best choice of the parameters. The results are shown in Figure 6.33. As we can see from the results, the sum-product

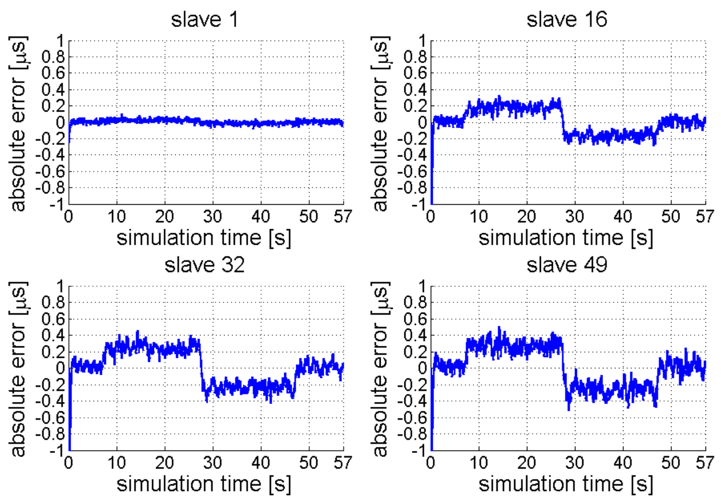
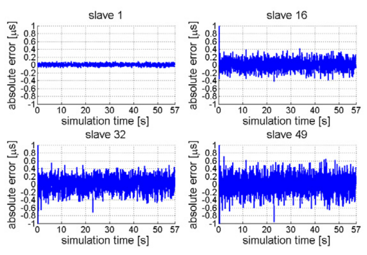
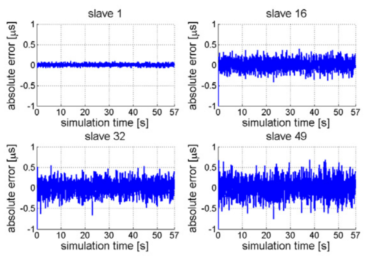


Fig. 6.32. Synchronization performance of CKF with temperature change at all slaves when $\rho = 7 \times 10^{-7}$

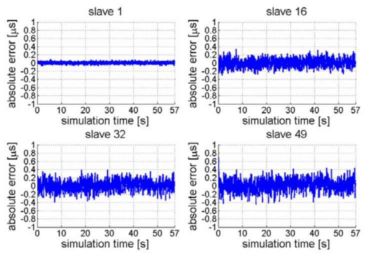
algorithm does not achieve a visible improvement over the standard algorithm. The sequential Kalman filter achieves some improvement and the centralized Kalman filter has the best performance.



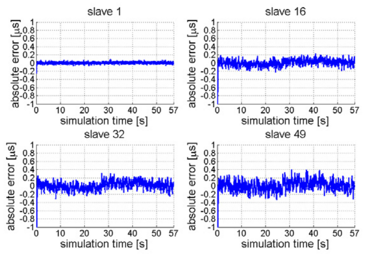
(a) standard algorithm



(b) sum-product algorithm



(c) sequential Kalman filter



(d) centralized Kalman filter

Fig. 6.33. Synchronization error with temperature change at the master

6.6.2.2 Mechanical shocks

Mechanical shocks may cause the frequency of the oscillator to drift in a short time. The drift can be modeled by a sinusoid function, which is illustrated in Figure 6.34

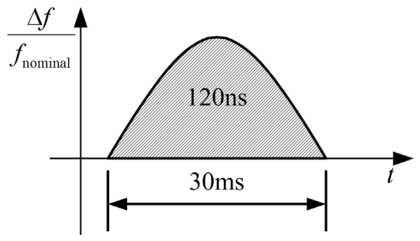


Fig. 6.34. Profile of frequency drift caused by mechanical shocks

As shown in Figure 6.34, the duration of the frequency drift caused by the mechanical shock is 30ms. The total error in time is equal to the area of the half wave, which is in this case 120ns.

In the simulation, we assume that mechanical shocks take place according to a pattern shown in Figure 6.35.

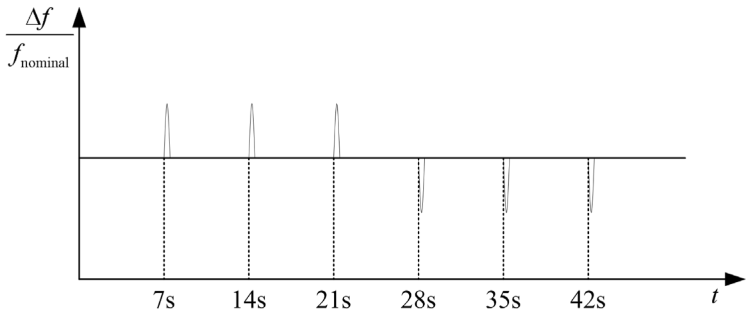


Fig. 6.35. Pattern of mechanical shocks

6.6.2.3 Vibrations

Vibrations cause the oscillators' frequency to oscillate periodically, resulting in a sinusoid function. Figure 6.36 illustrates the frequency with vibrations. Parameters of the sinusoid function are summarized in Table 6.3.

Parameters	Value
Wave length	60ms
Area of half wave	120ns
Initial phase	uniform $[-\pi, \pi]$
Start of vibration	7s
End of vibration	56s

Table 6.3. Parameters of vibrations

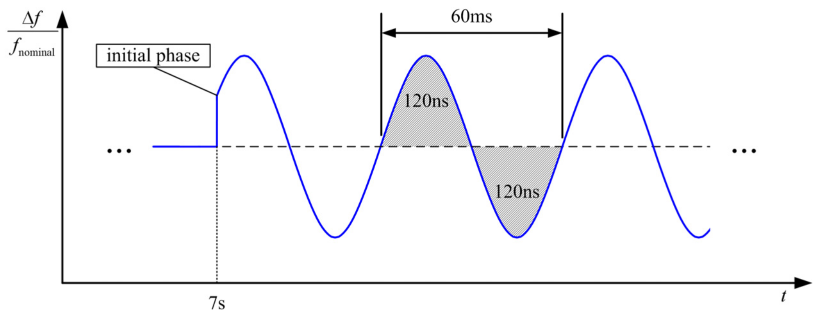


Fig. 6.36. Pattern of vibration

6.6.2.4 Synchronization performance for N elements line length

In industrial automation networks an interesting topic is the maximum line-length of the clock synchronization protocol, i.e., the number of elements that stay within a required synchronization error tolerance. Now, we evaluate and compare the performance of different synchronization algorithms by measuring the synchronization error at each slave element of a line with N elements. At the end, we achieve a RMSE vs line length curve for each algorithm, where the RMSE is achieved when using the optimal choice of the parameters. Based on these curves, we can check and predict the maximum synchronizable line length that is supported by each algorithm.

The simulation scenario is as follows. The network has a line topology with 50 elements. The first element is the master which provides reference time to the other elements. System parameters, including the delays in the network, properties of the oscillators and the parameters of PTP messaging are summarized in Table 6.1. The master element experiences temperature changes. The profile of the temperature change is shown in Figure 6.24. Mechanical shocks take place at each element and thus cause their frequency to drift. The pattern of the mechanical shocks is shown in Figure 6.35. Vibrations start at the simulation time of 7 second at all elements. Profile of the frequency drift caused by vibration is shown in Figure 6.36 and parameters are summarized in Table 6.3.

We choose for each synchronization method the optimal value of the parameter using the results in Figure 6.27, i.e., for centralized Kalman filter, $\rho = \sigma_\omega = 4 \times 10^{-6}$; for the sum-product algorithm, $\rho = \sigma_\omega = 8 \times 10^{-5}$ and for the sequential Kalman filter, $\rho = \sigma_\eta = 8 \times 10^{-6}$.

Based on 100 simulation runs, we obtained some statistics of the RMSE performance for each of the synchronization algorithms. The results are depicted in Figure 6.37.

From the simulation results, we can see that all the probabilistic methods improve the synchronization performance when the parameters are properly chosen.

6.7 Discussions

In this chapter, we studied synchronization of cascaded clocks based on the Precision Time Protocol. The standard synchronization algorithm does not take care of the stamping error. As a consequence, it can only support a limited number of consecutively connected clocks. For example for a RMSE requirement of 100ns under the situation defined

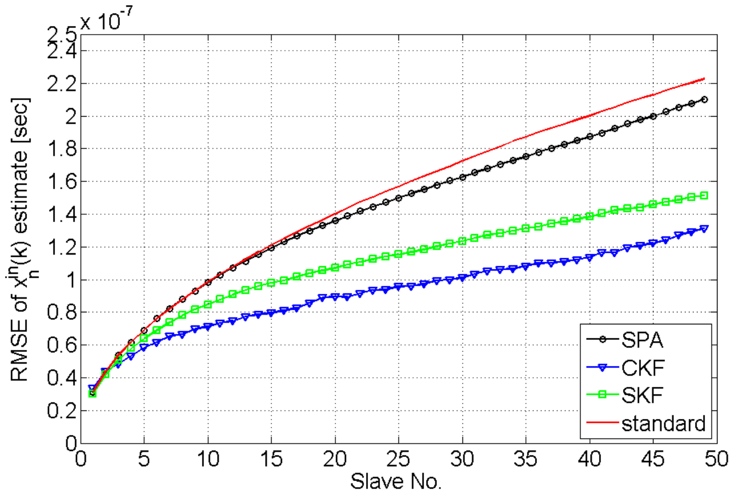


Fig. 6.37. The RMSE performance comparison

in Section 6.6.2.4, the standard algorithm can support only 10 elements according to the simulation results shown in Figure 6.37. We formulated clock synchronization as an estimation problem and used graphical models to represent the statistical relationships between the variables. Based on the probabilistic model, we developed several inference algorithms that improve the synchronization performance. With a RMSE requirement of 100ns under the situation defined in Section 6.6.2.4, the sum-product algorithm does not support more slaves than the standard algorithm. The sequential Kalman filter supports 15 slaves, i.e., a gain of 50%. If the computational complexity is out of concern, then the centralized Kalman filter achieves an improvement of 200% as it supports 30 slaves.

Synchronization in general is an important and non-trivial task in distributed systems. Besides what we discussed in this chapter, there are many other interesting topics. For example, clock synchronization in a network that is not fully compatible to the PTP protocol has to deal with the unknown delays, e.g., queuing delay incurred in the intermediated switches. Related work can be found in [65, 101]. Another interesting research direction is consensus time synchronization as discussed in [99, 111]. Synchronization in wireless networks, especially sensor networks is also intensively studied, e.g., in [20, 29, 67].

6.A Connection between SKF and the standard synchronization algorithm

In the sequential Kalman filter, the parameter σ_η is important. It characterizes the frequency variation and the estimation error made in the RCF estimation. Obviously, σ_η cannot be too small, otherwise the uncertainties cannot be fully characterized. A model mismatch problem occurs. Now we study the case when σ_η takes a very large value.

Let us study the steady state of the Kalman filter in which:

$$v_n^{\text{in}}(k) = v_n^{\text{in}}(k-1) \tag{6.174}$$

$$v_n^{\text{in}}(k, k-1) = v_n^{\text{in}}(k-1, k-2) \tag{6.175}$$

Furthermore, we assume that the parameters $a_n(k)$, $b_n(k)$, $d_n(k)$ are time-invariant, i.e., the index k can be dropped.

Let us first study the steady state of the Kalman filter at slave 1. Using (6.172), (6.170), (6.169) and (6.166), we obtain:

$$\begin{aligned} v_1^{\mathsf{in}}(k) &= (1 - k_1^{\mathsf{in}}(k)) \cdot v_1^{\mathsf{in}}(k, k-1) \\ &= (1 - \frac{v_1^{\mathsf{in}}(k, k-1)}{s_1^{\mathsf{in}}(k)}) \cdot v_1^{\mathsf{in}}(k, k-1) \\ &= \frac{(v_0^{\mathsf{out}}(k) + \sigma^2_{\zeta_1(k)}) \cdot v_1^{\mathsf{in}}(k, k-1)}{v_0^{\mathsf{out}}(k) + v_1^{\mathsf{in}}(k, k-1) + \sigma^2_{\zeta_1(k)}} \\ &= \frac{\sigma^2_{\zeta_1(k)} \cdot v_1^{\mathsf{in}}(k, k-1)}{v_1^{\mathsf{in}}(k, k-1) + \sigma^2_{\zeta_1(k)}} \end{aligned} \tag{6.176}$$

where the last step is based on the fact that $v_0^{\mathsf{out}}(k) = 0$ since $x_0^{\mathsf{out}}(k) = TS(M(k))$ is deterministic. Using (6.166), we can rewrite (6.176) as:

$$\begin{aligned} v_1^{\mathsf{in}}(k+1, k) &= v_1^{\mathsf{in}}(k) + \sigma^2_{\epsilon_1(k)} \\ &= \frac{\sigma^2_{\zeta_1(k)} \cdot v_1^{\mathsf{in}}(k, k-1)}{v_1^{\mathsf{in}}(k, k-1) + \sigma^2_{\zeta_1(k)}} + \sigma^2_{\epsilon_1(k)} \end{aligned} \tag{6.177}$$

Using the steady state condition in (6.175), we obtain:

$$v_1^{\mathsf{in}}(k, k-1) = \frac{\sigma^2_{\zeta_1(k)} \cdot v_1^{\mathsf{in}}(k, k-1)}{v_1^{\mathsf{in}}(k, k-1) + \sigma^2_{\zeta_1(k)}} + \sigma^2_{\epsilon_1(k)} \tag{6.178}$$

Solving (6.178) for $v_1^{\mathsf{in}}(k, k-1)$, we obtain:

$$\begin{aligned} v_1^{\mathsf{in}}(k, k-1) &= \frac{\sigma^2_{\epsilon_1(k)} + \sqrt{\sigma^4_{\epsilon_1(k)} + 4\sigma^2_{\epsilon_1(k)}\sigma^2_{\zeta_1(k)}}}{2} \\ &= \frac{a_1^2\sigma_\eta^2 + \sqrt{a_1^4\sigma_\eta^4 + 4a_1^2\sigma_\eta^2\sigma_\nu^2 + 4a_1^2 d_1^2 \sigma_\eta^4}}{2} \end{aligned} \tag{6.179}$$

where we applied (6.147) and (6.148) in the second line.

In a realistic PTP implementation, the parameter $a_1 \gg d_1$ e.g., the interval of sending Sync message (a_n) is in millisecond level and the line delay (d_n) is in the nanosecond level. Then if σ_η is sufficiently large, i.e., $\sigma_\eta \gg \sigma_\nu$, we have:

$$v_1^{\mathsf{in}}(k, k-1) = a_1^2\sigma_\eta^2 \tag{6.180}$$

Inserting this result in (6.170), we obtain:

$$\begin{aligned} k_1^{\mathsf{in}}(k) &= \frac{v_1^{\mathsf{in}}(k, k-1)}{v_1^{\mathsf{in}}(k, k-1) + \sigma^2_{\zeta_1(k)}} \\ &= \frac{a_1^2\sigma_\eta^2}{a_1^2\sigma_\eta^2 + d_1^2\sigma_\eta^2 + \sigma_\nu^2} \end{aligned} \tag{6.181}$$

where we applied (6.148) in the second line.

Inserting (6.181) and (6.180) into (6.172), we obtain:

$$\begin{aligned} v_n^{\text{in}}(k) &= (1 - k_n^{\text{in}}(k)) \cdot v_n^{\text{in}}(k, k-1) \\ &= \frac{(d_1^2\sigma_\eta^2 + \sigma_\nu^2) \cdot a_1^2\sigma_\eta^2}{a_1^2\sigma_\eta^2 + d_1^2\sigma_\eta^2 + \sigma_\nu^2} \end{aligned}$$

For $a_1 \gg d_1$ and $\frac{\sigma_\nu}{a_1} \ll \sigma_\eta \ll \frac{\sigma_\nu}{d_1}$, we have:

$$k_1^{\text{in}}(k) = 1 \tag{6.182}$$

and

$$v_1^{\text{in}}(k) = \sigma_\nu^2 \tag{6.183}$$

Inserting (6.182) into (6.171), we obtain:

$$\begin{aligned} \hat{x}_1^{\text{in}}(k) &= \hat{x}_1^{\text{in}}(k, k-1) + k_1^{\text{in}}(k) \cdot \hat{z}_1^{\text{in}}(k) \\ &= \hat{x}_1^{\text{in}}(k, k-1) + \hat{z}_1^{\text{in}}(k) \\ &= \hat{x}_0^{\text{out}}(k) + d_1(k) \cdot \hat{r}_{\text{MS}_1}(k) \\ &= \overrightarrow{s}_{\text{M}}^{\text{out}}(k) + d_1(k) \cdot \hat{r}_{\text{MS}_1}(k) \\ &= \overrightarrow{s}_{\text{M}}^{\text{out}}(k) + \hat{r}_{\text{MS}_1}(k) \cdot \hat{c}_{\text{S}_1}(\overrightarrow{d}_1^{\text{LD}}(k)) \end{aligned} \tag{6.184}$$

where we used (6.154) for the 4th line and (6.75) for the 5th line. The final expression in (6.184) is identical to (6.55), i.e., in this case the Kalman filter at slave 1 converges to the standard synchronization algorithm.

Then $x_1^{\text{out}}(k)$ is estimated by using (6.158), i.e.,

$$\begin{aligned} \hat{x}_1^{\text{out}}(k) &= \hat{x}_1^{\text{in}}(k) + b_1(k) \cdot \hat{r}_{\text{MS}_1}(k) \\ &= \hat{x}_1^{\text{in}}(k) + \hat{c}_{\text{S}_1}(\overrightarrow{d}_1^{\text{BD}}(k)) \cdot \hat{r}_{\text{MS}_1}(k) \end{aligned} \tag{6.185}$$

where we used (6.65) in the second line.

The calculation in (6.185) the same as (6.56), i.e., the Kalman filter coincides with the standard algorithm again. As in practice, $b_1 \ll a_1$, by choosing $\frac{\sigma_\nu}{a_1} \ll \sigma_\eta \ll \frac{\sigma_\nu}{b_1}$ for (6.159), we have:

$$v_1^{\text{out}}(k) = \sigma_\nu^2 \tag{6.186}$$

Using induction, we conclude that for slave n, if $d_n \ll a_n$, $b_n \ll a_n$, when we choose $\frac{\sigma_\nu}{a_n} \ll \sigma_\eta \ll \frac{\sqrt{n}\sigma_\nu}{d_n}$ and $\frac{\sigma_\nu}{a_n} \ll \sigma_\eta \ll \frac{\sqrt{n}\sigma_\nu}{b_n}$, then the following holds:

$$v_n^{\text{in}}(k, k-1) = a_n^2\sigma_\eta^2 \tag{6.187}$$

$$k_n^{\text{in}}(k) = 1 \tag{6.188}$$

and

$$v_n^{\mathsf{in}}(k) = n\sigma_\nu^2 \tag{6.189}$$

Inserting (6.188) into (6.171), we obtain:

$$\begin{aligned} \hat{x}_n^{\mathsf{in}}(k) &= \hat{x}_{n-1}^{\mathsf{out}}(k) + d_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) \\ &= \hat{x}_{n-1}^{\mathsf{out}}(k) + \hat{c}_{\mathtt{S}_n}(\overrightarrow{d_n^{\mathtt{LD}}}(k)) \cdot \hat{r}_{\mathtt{MS}_n}(k) \end{aligned} \tag{6.190}$$

where we used (6.75) in the last step.

Using (6.158), $\hat{x}_n^{\mathsf{out}}(k)$ is given by:

$$\begin{aligned} \hat{x}_n^{\mathtt{out}}(k) &= \hat{x}_n^{\mathtt{in}}(k) + b_n(k) \cdot \hat{r}_{\mathtt{MS}_n}(k) \\ &= \hat{x}_n^{\mathtt{in}}(k) + \hat{c}_{\mathtt{S}_n}(\overrightarrow{d}_n^{\mathtt{BD}}(k)) \cdot \hat{r}_{\mathtt{MS}_n}(k) \end{aligned} \tag{6.191}$$

where we used (6.65) in the last step.

From the results of the above derivation, we can see that if the value of σ_η fulfills $\frac{\sigma_\nu}{a_n} \ll \sigma_\eta \ll \mathsf{min}(\frac{\sqrt{n}\sigma_\nu}{d_n}, \frac{\sqrt{n}\sigma_\nu}{b_n})$ (for the simulation settings shown in Table 6.1, this corresponds to $3 \times 10^{-6} \ll \sigma_\eta \ll 10^{-3}$ approximately), then the Kalman filter implementation is closely approximated by equations (6.130), (6.184), (6.185), (6.138), (6.190) and (6.191). Comparing them with the equations used in the standard synchronization algorithm, i.e., (6.54), (6.55), (6.56), (6.57), (6.58) and (6.59), we can see that they are totally identical. In this case, the results of sequential Kalman filter coincide with the results of the standard algorithm. The simulation results shown in Figure 6.23 verified our analysis.

7. Conclusions and Future Work

This chapter briefly summarizes the main themes and the contributions of the thesis.

7.1 Summary and contributions

A networked system is usually composed of a large number of simple systems. Many applications in networked systems involve the estimation of quantities of interest. Estimation in networked systems is not trivial due to the typically high complexity of the system and the inaccurate and incomplete information. Probabilistic models provide a standard way to assemble all the information that is needed for the estimation and to represent the entire system as a joint distribution over a large number of variables. Estimation is then formulated as computing or optimizing the relevant probability functions such as likelihood or posterior distribution of certain variables of interest. Graphical models intuitively visualize the relationships between the variables. In particular, they provide a basic formalism to represent the conditional independence property or the factorization of a high dimensional function. Exploiting the independence property or the ability to factorize the problem, we can compute the probability functions in a more efficient way.

The whole thesis has discussed how to use probabilistic inference to solve the estimation problems in networked systems. Several factors complicate the direct application of standard inference algorithms. They are:

- The large number of network participants, the power or time constraint which forbid a centralized implementation of the inference.
- The high complexity of the stochastic processes involved results in very complicated distribution functions.
- Spatial and temporal correlations, in particular, spatially correlated dynamics, introduce dependences between variables, which complicate the exact inference.

The main contribution of the thesis is summarizing the existing methods and developing new solutions to the above-mentioned problems. In particular,

- we have introduced a general procedure of implementing distributed inference;

- we have reviewed sample based inference methods, i.e., particle filters and non-parametric belief propagation and we have derived inference algorithms using the Fourier series approximation;
- we have summarized the tools to solve the state estimation problem of a complicated dynamic system;
- we have applied distributed inference methods to the self-organized sensor localization problem. Both sample based and Fourier series based approximations have been considered to simplify the inference so that it fulfills the power constraints of a wireless sensor networks;
- we have formulated networked clock synchronization as a state estimation problem, then we have derived a probabilistic model and developed inference algorithms that solve the estimation problem.

7.2 Discussion on future directions

7.2.1 Measurement of the error caused by approximation

We have used approximations for both applications presented in this thesis. In the sensor localization application, we need an approximation because the distributions involved in the belief propagation do not have a closed form expression. In the clock synchronization application, we approximate the posterior distribution obtained in each time step by a distribution with a simpler structure to enable distributed implementation of inference. In both applications, we constantly introduce new errors by doing approximations in each intermediate step. Although in some cases, we can measure the error we make in each single step in some form (e.g., Hellinger metric for Fourier series approximation, Kullback-Leibler divergence for approximate inference in dynamical systems), it is very difficult to compute the accumulated error or understand the effect of error propagation for such complicated systems. However, some analytical work can be done with simple models, e.g., graphical model or distribution functions with special structure. For example, Boyen and Koller [10] studied the error propagation in approximate inference in dynamic Bayesian networks. The study was focused on discrete random variables. In the clock synchronization algorithm, all the uncertainties are modeled Gaussian. The Kullback-Leibler distance between two Gaussian distributions has a closed form [109]. Based on Boyen and Koller's idea, we can also study the error propagation effect in a dynamical system with Gaussian random variables.

7.2.2 Distributed inference

Many applications prefer a distributed implementation of inference due to the constraints on communications, computational complexity or allowable latencies. Some networked systems provide many degrees of freedom to design the inference. For example, in a meshed network there are many different choices of communication links to carry out message passing for inference. In this case, the design of an inference algorithm by itself can be posed as an optimization problem. Different cost functions can be defined, for example, the communication cost, the overall complexity, maximal complexity of the required processing at each node and so on. Sometimes, several criteria have to be simultaneously considered, making such an optimization problem more complicated.

7.2.3 Other potential applications

In this thesis, we considered two applications of distributed and approximate inference. In practice, there is a tremendous variety of other applications. Many of them can benefit from using the methodology and the results derived in the thesis.

First of all, we present in Chapter 6 a synchronization protocol. By using probabilistic modeling and distributed inference, we improve the synchronization accuracy. The results are examined through the simulation of an industrial automation network. Extended versions of the synchronization algorithms can also be applied to other networks, e.g., power grids, sensor networks, communication systems and so on.

In Chapter 5, a self-organized sensor localization problem is discussed. The problem can be complicated if the sensors can move, which becomes a tracking problem. In that case, we need a more sophisticated graphical model to represent the correlation over space and the dynamics of each moving sensor. The structure of the graph should be exploited for the design of an efficient inference algorithm. Approximations have to be introduced if it is necessary. Other constraints can further complicate the task. Interesting applications of tracking multiple moving objects include dynamic positioning of vehicles, cooperative robot team and so on.

Another interesting set of problems is tracking with indirect observations, just like the synchronization of cascaded clocks. Each slave element can not directly observe the actual state of the master clock. The indirect observation will be affected by the intermediate elements through which the slave element is connected to the master. In this case, we should exploit the model of the target dynamic process and the spatial model to filter the noisy observations.

As we mentioned at the beginning of the thesis, there is a tremendous variety of applications in networked systems that rely on the estimation. The inference methodologies discussed in this thesis can be potentially applied to the fault diagnosis in power grids [38] or automation networks [95], to the safety systems in vehicles [35] and so on.

Bibliography

[1] H. Abubakari and S. Sastry. Ieee 1588 style sychronization over wireless link. In *Proc. Of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 127–130, Ann Arbor, USA, 2008.

[2] Y. Arai, T. Agui, and M. Nakajima. A fast dct/sq scheme for images. *Transactions of the Institute of Electronics and Communication Engineering of Japan. Section E*, 71(11):1095–1097, 1988.

[3] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[4] S. Arulampalam, S. Maskell, and N. Gordon. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50:174–188, 2002.

[5] L.F. Auler and R. d'Amore. Adaptive kalman filter for time synchronization over packet-switching networks: an heuristic approach. In *Proc. Of 2nd Internatinal Conference on Communication Systems Software and Middleware (COMSWARE 2007)*, pages 1–7, Bangalore, India, 2007.

[6] M. Bayati, D. Shah, and M. Sharma. Maximum weight matching via max-product belief propagation. In *In Proc. 2005 IEEE Intl. Symp. on Information Theory*, 2005.

[7] J.A. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, ICSI, UC Berkeley, 1997.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, August 2006.

[9] A. Bletsas. Evaluation of kalman filtering for network time keeping. In *Proc. Of 1st IEEE International Conference on Pervasive Computing and Communications (PerCom03)*, page 289, Los Alamitos, USA, 2003.

[10] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *In Proc. UAI*, pages 33–42, 1998.

[11] D. Brunn, F. Sawo, and U. D. Hanebeck. Efficient nonlinear bayesian estimation based on fourier densities. In *Proc. of the 2006 IEEE Intl. Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI 2006)*, pages 317–322, Heidelberg, Ger-

many, Sept. 2006.

[12] D. Brunn, F. Sawo, and U. D. Hanebeck. Nonlinear multidimensional bayesian estimation with fourier densities. In *Proc. of the 2006 IEEE Conf. on Decision and Control (CDC 2006)*, pages 1303–1308, San Diego, USA, Dec. 2006.

[13] T. F. Chan and J. Shen. *Image processing and analysis-variational, PDE, wavelet and stochastic methods.* Society of Applied Mathematics, 2005.

[14] B. Chen, Y.P. Chen, J.M. Xie, Z.D. Zhou, and J.M. Sa. Control methodologies in networked motion control systems. In *Proc. of 2005 International Conference on Machine Learning and Cybernetics*, 2005.

[15] C. Crick and A. Pfeffer. Loopy belief propagation as a basis for communication in sensor network. In *In UAI 2003*, 2003.

[16] A. P. Dawid. Conditional independence in statistical theory. *Journal of the Royal Statistical Society Series B*, 41:1–31, 1979.

[17] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo methods in practice.* Springer, 2001.

[18] A. Doucet, S. Godsill, and C. Andrleu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208, July 2000.

[19] J. C. Eidson. *Measurement, Control and Communication Using IEEE 1588.* Springer, 2006.

[20] J. Elson and D. Estrin. Fine-grained network time synchronization using reference broadcast. In *Proc. of the 5th Symp. on Operating System Design and Implementation (OSDI)*, pages 147–163, 2002.

[21] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient belief propagation for early vision. In *In CVPR*, pages 261–268, 2004.

[22] G. D. Forney. Codes on graphs, normal realizations. *IEEE Trans. on Information Theory*, 47(2):520–548, 2001.

[23] M. Fossorier. Iterative reliability-based decoding of low-density parity check codes. *IEEE Journal on Selected Areas in Communications*, 19(5), 2001.

[24] W.T. Freeman, E.C. Pasztor, and O.T. Carmichael Y. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25â47, 2000.

[25] B. Frey. Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 257–26, San Francisco, CA, 2003. Morgan Kaufmann.

[26] B.J. Frey and N. Jojic. A comparison of algorithms for inference and learning in probabilistic graphical models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9):1392–1416, 2005.

[27] S. Funiak, C. Guestrin, M. Paskin, and R. Sukthankan. Distributed localization of networked cameras. In *In Proc. of Fifth International Conference on Information Processing in Sensor Networks (IPSN-06*, pages 34–42, 2006.

[28] S. Funiak, C. E. Guestrin, M. Paskin, and R. Sukthankar. Distributed inference in dynamical systems. *Advances in Neural Information Processing Systems*, 19, Dec. 2006.

[29] S. Ganeriwal, R. Kumar, and M. Srivastava. Timing-sync protocol for sensor networks. In *Proc. of 1st ACM Conference on Embedded Networked Sensor Systems*, pages 138–149, 2003.

[30] S. Geman and D. Geman. Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transactions on PAMI*, 6(6):721–741, 1984.

[31] Z. Ghahramani. Learning dynamic bayesian networks. *Lecture Notes in Computer Science*, 1387:168–197, 1998.

[32] R. M. F. Goodman and A. D. Green. Microprocessor-controlled soft-decision decoding of error-correcting block codes. In *Proc. of the IERE Intl. Conf. on Digital Proc. of Signals in Communications*, pages 337–349, Sept. 1977.

[33] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. In *In NIPS-14*, pages 1523–1530. The MIT Press, 2001.

[34] C. Guestrin, P. Bodik R. Thibaux, M. Paskin, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proc. of IPSN 2004*, pages 1–10, 2004.

[35] F. Gustafsson. Automotive safety systems. *IEEE Signal Processing Magazine*, 26(4):32–47, 2009.

[36] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.J. Nordlund. Particle filters for positioning, navigation, and tracking. *Signal Processing, IEEE Transactions on [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on]*, 50(2):425–437, 2002.

[37] U. D. Hanebeck. Progressive bayesian estimation for nonlinear discrete-time systems: the measurement step. In *Proc. of the 2003 IEEE Intl. Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI 2003)*, pages 173–178, Tokyo, Japan, July 2003.

[38] L. He. Application of bayesian netowkr in power grid fault diagnosis. In *Proc. of 4th Intl. Conf. on Natural Computation*, pages 61–64, Los Alamitos, USA, 2008.

[39] D. Heckerman. A tutorial on learning with bayesian networks. Technical Report MSR-TR-95-06, Microsoft Research, 1995.

[40] J. Hijmans and J. De Boer. An approximation method for order-disorder problems, i, ii, iii. *Physica*, XXI:471–516, 1955.

[41] X. Hu, E. Eleftheriou, and A. Dholakia. Efficient implementation of the sum-product algorithm for decoding ldpc codes. In *In Proc. 2001 GLOBECOM*, volume 2, pages 1036–1036E, 2001.

[42] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15:225–263, 1996.

[43] IEEE. Ieee standard for lan / man csma / cd access method, 1997.

[44] IEEE. *IEEE standard definition of physical quantities for fundamental frequency and time metrology-random instabilities*. IEEE Std 1139-1999, 1999.

[45] IEEE. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Std. 1588-2002, 2002.

[46] IEEE. *IEEE standard for local and metropolitan area networks: media access control (MAC) bridges*. IEEE Std 802.1D-2004, 2004.

[47] IEEE. *IEEE P1588TM D2.2 Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Std. 1588-2008, 2007.

[48] IEEE. Standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks, 2007.

[49] A.T. Ihler, J.W. Fisher, and R.L. Moses. Nonparametric belief propagation for self-calibration in sensor networks. In *In Proceedings of the Third International Symposium on Information Processing in Sensor Networks*, pages 225–233, 2004.

[50] International standard IEC 61588. *Precision clock synchronization protocol for networked measurement and control systems*, 2004.

[51] J. Jasperneite, K. Shehab, and K. Weber. Enhancements to the time synchronization standard ieee-1588 for a system of cascaded bridges. In *Proc. of 2004 IEEE International Workshop on Factory Communication Systems*, pages 239–244, 2004.

[52] F. V. Jensen and F. Jensen. Optimal junction trees. In *In UAI*, pages 360–366. Morgan Kaufmann, 1994.

[53] M. I. Jordan. Graphical models. *Statistical Science*, 19:140–155, 2004.

[54] M.I. Jordan, Z. Ghahramani, T. Jaakkola, and L.K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[55] S. Julier and J. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL*, 1997.

[56] R. Kikuchi. A theory of cooperative phenomena. *Phys. Rev.*, 81(6):988–1003, 1951.

[57] J.H. Kim and J. Pearl. A computational model for combined causal and diagnostic reasoning in inference systems. In *Proc. of the IJCAI-83*, pages 190–193, Karlsruhe, Germany, 1983.

[58] R. Kindermann and J. L. Snell. *Markov Random Fields and Their Applications*. AMS, 1980.

[59] R. Koetter and A. Vardy. Algebraic soft-decision decoding of reed-solomon codes. *IEEE Transactions on Information Theory*, 48(11):2809–2825, Nov. 2003.

[60] J. H. Kotecha and P. M. Djuric. Gaussian particle filtering. In *Proceedings of the 11th IEEE Signal Processing Workshop on Statistical Signal Processing*, pages 429–432, 2001.

[61] R. Kronmal and M. Tarter. The estimation of probability densities and cumulatives by fourier series methods. *Journal of the American Statistical Association*, 63(323):925–952, 1968.

[62] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519, 2001.

[63] S. Lin and D. J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.

[64] P. Loschmidt, R. Exel, A. Gagy, and G. Gaderer. Limits of synchronization accuracy using hardware support in ieee 1588. In *Proc. Of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 12–16, Ann Arbor, USA, 2008.

[65] A. Machizawa, T. Iwawma, and H. Toriyama. Software-only implementation of slave clocks with sub-microsecond accuracy. In *Proc. Of IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 17–22, Ann Arbor, USA, 2008.

[66] D.J.C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge, 2003.

[67] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding synchronization protocol. In *Proc. of 2nd ACM Conference on Embedded Networked Sensor Systems*, pages 39–49, 2004.

[68] R.J. Mceliece, D.J.C. Mackay, and J. Cheng. Turbo decoding as an instance of pearl's belief propagation algorithm. *IEEE Journal on Selected Areas in Communications*, 16:140–152, 1998.

[69] D.L. Mills. Internet time synchronization: The network time protocol. Network Working Group Request for Comments: 1129, 1989.

[70] D.L. Mills. Precision synchronization of computer network clocks, 1994.

[71] B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):696–710, 1997.

[72] J.M. Mooij and H.J. Kappen. Sufficient conditions for convergence of the sumâproduct algorithm. *IEEE Transactions on Information Theory*, 53(12):4422–4437, Dec 2007.

[73] T. Morita. Cluster variation method for nonuniform ising and heisenberg models and spin-pair correlation function. *Progr. Theor. Phys.*, 85(2):243–255, 1991.

[74] T. Morita, M. Suzuki, K. Wada, and M. Kaburagi. Foudations and applications of cluster variation method and path probability method. *Progr. Theor. Phys. Suppl.*, 115, 1994.

[75] O.L. Moses and R. Patterson. Self-calibration of sensor networks. In *in SPIE vol. 4743: Unattended Ground Sensor Technologies and Applications IV*, pages 108–119, 2002.

[76] K. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in dbns. In *In UAI*, pages 378–385, 2001.

[77] K.P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2002.

[78] K.P. Murphy, Y. Weiss, and M.I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *In Proceedings of Uncertainty in AI*, pages 467–475, 1999.

[79] C. Na, D. Obradovic, and R. L. Scheiterer. A probabilistic approach to clock synchronization of cascaded networked elements. In *Proc. Of IEEE Intl. Conf. on Acoustics, Speech and Signal Processing (ICASSP 2009)*, pages 1793–1796, Taipei, April 2009.

[80] C. Na, D. Obradovic, and R. L. Scheiterer. Probabilistic model for clock synchronization of cascaded network elements. In *Proc. Of IEEE Intl. Instrumentation and Measurement Technology Conf. (I2MTC 2009)*, pages 1594–1598, Singapore, May 2009.

[81] C. Na, D. Obradovic, R.L. Scheiterer, G. Steindl, and F.J. Goetz. Synchronization performance of the precision time protocol. In *Proc. of 2007 International IEEE Symposium on Precision Clock Synchronization (ISPCS) for Measurement, Control and Communication*, pages 25–32, Vienna, Austria, 2007.

[82] C. Na, D. Obradovic, R.L. Scheiterer, G. Steindl, and F.J. Goetz. Enhancement of the precision time protocol in automation networks with a line topology. In *Proc. of 17th IFAC World Congress (IFAC'08)*, pages 8333–8338, Seoul, South Korea, 2008.

[83] C. Na, R. L. Scheiterer, and D. Obradovic. Clock synchronization based on distributed hidden state estimation. In *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS 2009)*, Brescia, Italy, Oct. 2009.

[84] C. Na, R. L. Scheiterer, and D. Obradovic. A kalman filter approach to clock synchronization of cascaded network elements. In *Proc. Of the 1st IFAC Workshop on Estimation and Control of Networked Systems (NecSys'09)*, Venice, Italy, Sept. 2009.

[85] C. Na, H. Wang, D. Obradovic, and U. D. Hanebeck. Multisensor fusion and integration for intelligent systems. In *Proc. Of IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI 2008)*, pages 290–295, Seoul, Republic Of Korea, Aug. 2008.

[86] C. Na, H. Wang, D. Obradovic, and U. D. Hanebeck. *Multisensor Fusion and Integration for intelligent Systems*, chapter Fourier density approximation for belief propagation in wireless sensor networks. Lecture Notes in Electrical Engineering. Springer, 2009.

[87] J. Nieminen. Synchronization of next generation wireless communication systems. Master's thesis, Helsinki University of Technology, 2007.

[88] F. Orderud. Comparison of kalman filter estimation approaches for state space models with nonlinear measurements. In *In Proc. of Scandinavian Conference on Simulation and Modeling*, 2005.

[89] M. Paskin and C. Guestrin. Robust probabilistic inference in distributed systems. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 436–445, 2004.

[90] M. Paskin, C. Guestrin, and J. Mcfadden. A robust architecture for distributed inference in sensor networks. In *Information Processing in Sensor Networks, 2005. IPSN*

2005. Fourth International Symposium on, pages 55–62, 2005.

[91] J. Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proc. of the American Association of Artificial Intelligence National Conference on AI*, pages 133–136, Pittsburgh, PA, 1982.

[92] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[93] R. Perlman. An algorithm for distributed computation of a spanning tree in an extended lan. *ACM SIGCOMM Computer Communication Review*, 15(4):44–53, 1985.

[94] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[95] B. Sallans, D. Bruckner, and G. Russ. Statistical model-based sensor diagnostics for automation systems. In *Proc. of the 6th IFAC Intl. Conf. on Filedbus Systems and their Applications*, pages 239–246, Mexico, Nov. 2005.

[96] R.L. Scheiterer, C. Na, D. Obradovic, and G. Steidl. Synchronization performance of the precision time protocol in industrial automation networks. *ISPCS 2007 Special Issue of IEEE Transactions on Instrumrntation and Measurement*, 2008.

[97] R.L. Scheiterer, C. Na, D. Obradovic, and G. Steindl. Synchronization performance of the precision time protocol in industrial automation networks. *IEEE Transactions on Instrumentation and Measurement*, 58(6):1849–1857, 2009.

[98] R.L. Scheiterer, C. Na, D. Obradovic, G. Steindl, and F.J. Goetz. Synchronization performance of the precision time protocol in the face of slave clock frequency drift. In *Proc. of 4th IEEE Conference on Automation Science and Engineering(CASE2008)*, pages 554–559, Washington DC, USA, 2008.

[99] L. Schenato and G. Gamba. A distributed consensus protocol for clock synchronization in wireless sensor network. In *Proc. of 46th IEEE Conference on Decision and Control*, pages 2289–2294, Dec. 2007.

[100] B. W. Silverman. *Density estimation for statistics and data analysis*. Monographs on Statistics and Applied Probability. Chapman and Hall, London, 1986.

[101] Y. Stein and B. Stroehlein. Using synchronization over psn-does ieee 1588 really make a difference? In *Proc. WSTS'06*, March 2006.

[102] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. *Advances in Neural Info. Proc. Sys.*, 13:689–695, 2003.

[103] S. ten Brink. Convergence of iterative decoding. *Electronics Letters*, 35(10):806–808, 1999.

[104] International Telecommunication Unit. *Definition and terminology for synchronization networks*. ITU-T G.810, 1996.

[105] J.R. Vig. Quartz crystal resonators and oscillators for frequency control and timing applications-a tutorial. Technical report, US Army Communications-Electronics Command, 2007.

[106] M.J. Wainwright and M.I. Jordan. Graphical models, exponential families, and variational inference. Technical report, Dept. of Statistics, September 2003.

[107] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41, 2000.

[108] G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report TR 95-041, Univ. of North Carolina, July 2006.

[109] Wikipedia. Multivariate gaussian distribution. http://en.wikipedia.org/wiki/Multivariate_normal_distribution.

[110] M. A. Woodburv. Inverting modified matrices. Technical report, Statist. Res. Group, Mem. Rep. No. 42, Princeton, N. J., June 1950.

[111] G. Xiong and S. Kishore. Discrete-time second order distributed consensus time synchronization algorithm for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, 2009.

[112] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Generalized belief propagation. In *In NIPS 13*, pages 689–695. MIT Press, 2001.

[113] J.S. Yedidia, W.T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

[114] P. Zarchan and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach*. AIAA, Dec. 2000.