

Statistical Inference in Graphical Models

Kevin Gimpel

October 30, 2006

ABSTRACT

Graphical models fuse probability theory and graph theory in such a way as to permit efficient representation and computation with probability distributions. They intuitively capture statistical relationships among random variables in a distribution and exploit these relationships to permit tractable algorithms for *statistical inference*. In recent years, certain types of graphical models, particularly *undirected graphical models*, *Bayesian networks*, and *dynamic Bayesian networks (DBNs)*, have been shown to be applicable to various problems in air and missile defense that involve decision making under uncertainty and estimation in dynamic systems. While the scope of problems addressed by such systems is quite diverse, all require mathematically-sound machinery for dealing with uncertainty. Graphical models provide a robust, flexible framework for representing and computationally handling uncertainty in real-world problems. While the graphical model regime is relatively new, it has deep roots in many fields, as the formalism generalizes many commonly-used stochastic models, including Kalman filters [25] and hidden Markov models [47]. Statistical inference on a graphical model is NP-Hard, but there have been extensive efforts aimed at developing efficient algorithms for certain classes of models as well as for obtaining approximations for quantities of interest. The literature offers a rich collection of both exact and approximate statistical inference algorithms, several of which we will describe in detail in this report.

TABLE OF CONTENTS

	Page
Abstract	iii
List of Illustrations	vii
1. INTRODUCTION	1
1.1 Two Examples	1
2. GRAPHICAL MODELS	5
2.1 Notation and Preliminary Definitions	6
2.2 Undirected Graphical Models	6
2.3 Bayesian Networks	10
2.4 Dynamic Bayesian Networks	15
3. STATISTICAL INFERENCE	19
3.1 Variable Elimination	20
3.2 Belief Propagation	23
3.3 Inference in Dynamic Bayesian Networks	25
4. EXACT INFERENCE ALGORITHMS	29
4.1 The Junction Tree Algorithm	29
4.2 Symbolic Probabilistic Inference	37
5. ALGORITHMS FOR APPROXIMATE INFERENCE	41
5.1 The Boyen-Koller Algorithm	42
5.2 Particle Filtering	43
5.3 Gibbs Sampling	49
APPENDIX A:	53
A.1 Independence \Leftrightarrow Conditional Independence	53
A.2 Derivation of pairwise factorization for acyclic undirected graphical models	53

A.3 Derivation of belief propagation for continuous nodes with evidence applied	54
References	59

LIST OF ILLUSTRATIONS

Figure No.		Page
1	An example Bayesian network for medical diagnosis.	1
2	An undirected graphical model with five nodes.	7
3	u-separation and the global Markov property.	7
4	An acyclic undirected graphical model.	10
5	A Bayesian network with five nodes.	10
6	Serial connection.	12
7	Diverging connection.	12
8	Converging connection.	12
9	Boundary conditions.	12
10	Bayes ball algorithm example run.	12
11	The Markov blanket of a node X .	13
12	Equivalent way of specifying the local Markov property for Bayesian networks.	13
13	A 2-timeslice Bayesian network (2-TBN).	16
14	The 2-TBN for a hidden Markov model (HMM) or Kalman filter.	17
15	An acyclic undirected graphical model.	20
16	A message passed from node F to node E .	21
17	Message passes from an example run of the variable elimination algorithm to compute $p(A)$.	22
18	COLLECT-TO-ROOT and DISTRIBUTE-FROM-ROOT, the two series of message passing.	24
19	The main types of inference for DBNs.	25
20	Moralization and triangulation stages in the junction tree algorithm.	30
21	Junction tree created from the triangulated graph.	31

22	Message passing in the junction tree.	32
23	A 2-TBN showing timeslices $t - 1$ and t .	34
24	Steps to construct a junction tree from the prior B_0 .	35
25	Steps to construct a junction tree from the 2-TBN B_t .	36
26	Procedure for advancing timesteps.	37
27	A network to illustrate the set factoring algorithm.	39
28	The 2-TBN we used to demonstrate execution of the junction tree algorithm for DBNs.	43
29	Procedure for advancing timesteps in the Boyen-Koller algorithm.	44
30	The steps in particle filtering.	48
A-1	An undirected graphical model with observation nodes shown.	55
A-2	COLLECT-TO-ROOT and DISTRIBUTE-FROM-ROOT, the two series of message passing.	57

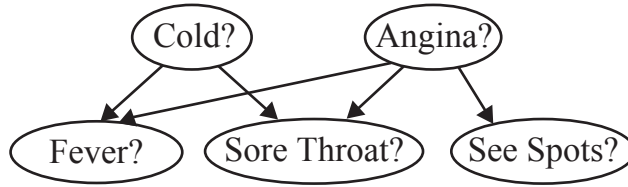


Figure 1. An example Bayesian network for medical diagnosis (adapted from [21]).

1. INTRODUCTION

Research on graphical models has exploded in recent years due to their applicability in a wide range of fields, including machine learning, artificial intelligence, computer vision, signal processing, speech recognition, multi-target tracking, natural language processing, bioinformatics, error correction coding, and others. However, different fields tend to have their own classical problems and methodologies for solutions to these problems, so graphical model research in one field may take a very different shape than in another. As a result, innovations are scattered throughout the literature and it takes considerable effort to develop a cohesive perspective on the graphical model framework as a whole.

1.1 TWO EXAMPLES

To concretize our discussion, we will first introduce examples of two types of real-world problems that graphical models are often used to solve. The first consists of building a system to determine the most appropriate classification of a situation by modeling the interactions among the various quantities involved. In particular, we will consider a graphical model that is a simplification of the types of models found in medical diagnosis. In this example, we want to determine the diseases a patient may have and their likelihoods given the reported symptoms. A natural way of approaching this problem is to model probabilistically the impact of each illness upon its possible symptoms, and then to infer a distribution over the most likely diseases when given an actual set of symptoms observed in a patient. Figure 1 shows a Bayesian network for a toy model consisting of 2 diseases and 3 symptoms. Each node of a Bayesian network corresponds to a random variable and the edges encode information about statistical relationships among the variables. In certain applications, it is natural to attribute a “causal” interpretation to the directed edges between variables. For example, in Figure 1 having a cold can be interpreted as having a direct or causal influence on the appearance of a fever or sore throat. When building a Bayesian network, a hyperparameter must be specified for each variable to concretize this influence. In particular, one must delineate the conditional probability distribution of each variable given the set of variables pointing to it. In Section 2, we will describe in detail why these parameters are sufficient to specify a Bayesian network and how we can make use of these parameters to allow for efficient representation and computation with the joint distribution represented by the network. The idea of doing medical diagnosis using graphical models has been used in several real-world systems, including the decision-theoretic formulation of the Quick Medical Reference (QMR) [56], which includes approximately

600 diseases and approximately 4000 symptoms “caused” by the diseases.

The other type of problem we shall consider is that of estimation in dynamic systems. To approach this problem, we assume the existence of an underlying stochastic process that we cannot observe directly, but for which we have a time-series of noisy observations. These observations are used to refine an estimate of the process’s hidden state. An example of this type of problem is that of automatic speech recognition, in which the process being modeled is that of human speech and the hidden state consists of the phoneme or word that is currently being spoken. We use the observation sequence, which may be the sequence of speech signal waveforms produced by the speaker, to obtain an estimate for what the speaker actually said. This type of problem can also be represented using a graphical model, as we shall see in Section 2.4. Doing so provides access to the wealth of inference techniques developed by the graphical model community for estimating the hidden state.

In general, problems such as these can be approached by representing the system being modeled as a joint probability distribution of all the quantities involved and then encapsulating this distribution in a graphical model. Frequently, the structure of the models is sparse due to statistical dependencies among the variables being modeled; the graphical model machinery exploits this sparsity to provide an efficient representative form for the distribution. Then, given observations, we can perform statistical inference on the models to obtain distributions on the desired quantities. The two examples described above use different kinds of graphical models, both of which we shall discuss in detail in this report. The expert system problem is well-suited to a *Bayesian network*, while the dynamic state estimation problem is often approached using a *dynamic Bayesian network* (DBN) model. Many commonly-used temporal models, including hidden Markov models (HMMs) and state-space models, are actually special cases of DBNs.

The purpose of graphical modeling is to exploit the statistical relationships of the quantities being modeled for representational and computational efficiency. The relevant computations we need to perform may include computing likelihoods, expectations, entropies, or other statistical quantities of interest. Obtaining any of these quantities requires performing *statistical inference* on the model under consideration. In the most general case, exact statistical inference in graphical models is NP-Hard and the computations required become intractable on a single computer for even moderately-sized problems [9]. As a result, efficient inference algorithms have been developed to compute exact results for certain subclasses of networks or for particular sets of queries, and there has also been a large focus on the design and convergence analysis of approximation schemes for general networks that use Monte Carlo sampling techniques or Markov chain-Monte Carlo methods. We will describe several exact and approximate inference algorithms in this report, including the junction tree algorithm, symbolic probabilistic inference (SPI), the Boyen-Koller algorithm, particle filtering, and Gibbs sampling.

The rest of the report is laid out as follows. In Section 2, we provide a standalone introduction to graphical models, focusing on undirected graphical models, Bayesian networks, and dynamic Bayesian networks. In Section 3, we introduce statistical inference in graphical models by describing its common incarnations in the context of models we introduce in Section 2. In addition, we develop the canonical “message-passing” algorithm for inference in graphical models known as *belief propagation* or the *sum-product algorithm*. In Section 4, we describe two algorithms for

exact inference for both static and dynamic Bayesian networks – the junction tree algorithm and symbolic probabilistic inference – and in Section 5 three algorithms for approximate inference: the Boyen-Koller algorithm, particle filtering, and Gibbs sampling.

2. GRAPHICAL MODELS

Using statistical models for solving real-world problems often requires the ability to work with probability distributions in a computationally efficient way. This is due to the complexity of the models required for tackling such problems, a complexity which can manifest itself in several ways. The first is in the number of random variables which must be modeled in order to accurately represent a real-world system. Generally speaking, accuracy of a model is proportional to the degree to which real-world phenomena are properly modeled; however, there is a trade-off between modeling as close to reality as possible and maintaining computational tractability. As the number of variables grows, the complexity of dealing with the joint distribution increases rapidly for certain types of random variables. Even just storing the full joint distribution over a set of discrete random variables can be prohibitively expensive. For n binary random variables, a table with 2^n entries is required to store the joint distribution. Fundamental computations, such as marginalizing one variable out of the distribution, require us to touch every entry in this table, thereby requiring $O(2^n)$ computations. The second way in which models can become complex is in the degree of interaction among the quantities being modeled. When several variables possess complicated statistical interdependencies, representing and performing calculations with the distribution becomes time-consuming. A third way in which complexity arises is through the choice of distribution to use for quantities of interest. Continuous real-world quantities are often approximated by Gaussian distributions for simplicity, but this may produce large errors in cases of multi-modal densities; in these cases, we would prefer to use a model which could better handle the multi-modality but this will frequently result in increased computational difficulty.

To address these complexities, one approach is to choose a form of representation for a distribution that exploits its properties to allow for efficient representation and computation. A widely-used form that uses graph theory to encode statistical relationships is a family of constructs called *graphical models*. To represent a distribution, a graphical model consists of a graph in which each node corresponds to a single random variable in the distribution and the edges between nodes encode their statistical relationships in the probabilistic domain.¹ While graphical models abstract many commonly-used stochastic models, there are two primary types, undirected and directed, which differ in the classes of distributions they can represent and in the ways they represent relationships among variables. As the name suggests, *undirected graphical models* are built on undirected graphs and have been used in machine vision [15], error correcting codes [16], and statistical physics [65], while directed graphical models, more commonly called *Bayesian networks*, are built on directed graphs and have been used for medical diagnosis [56], software troubleshooting agents, and many applications in artificial intelligence. Bayesian networks can be extended to model dynamic systems, for which they are called *dynamic Bayesian networks* (DBNs), and have been applied to problems including medical monitoring [11], object tracking [3], and speech recognition [46].

This section is laid out as follows. In Section 2.1, we will introduce the notation we will use in this report and review a few definitions from probability theory and graph theory essential to our discussion of graphical models. In Section 2.2 we will introduce the key concepts underlying graphical models by focusing on undirected graphical models. Then in Section 2.3 we will discuss

¹Since each node corresponds to exactly one variable, we will use the terms *node* and *variable* interchangeably.

Bayesian networks and the types of probability distributions they can represent. Finally, in Section 2.4, we shall discuss how Bayesian networks can be used to model dynamic systems by introducing and defining dynamic Bayesian networks (DBNs).

2.1 NOTATION AND PRELIMINARY DEFINITIONS

We will use ordinary typeface (e.g., X) to denote random variables and boldface (e.g., \mathbf{X}) for vectors of variables. Individual variables in \mathbf{X} will be subscripted by integers (e.g., where $s = 1$, X_s denotes the single variable X_1), and sets of variables will be subscripted by integer sets (e.g., where $A = \{1, 2, 3\}$, \mathbf{X}_A denotes the set $\{X_1, X_2, X_3\}$). We will occasionally refer to a set of variables solely by its integer subscript or by its integer index set, so we will refer to X_s by s or \mathbf{X}_A by A . In this report, we will assume the reader has familiarity with basic concepts of probability theory and graph theory. However, the authors feel it helpful to review a few definitions that will be vital to the discussion herein.

Definition 2.1. Independence. *Two random variables X and Y are independent if their joint pdf can be factored in the following way: $p_{X,Y}(x,y) = p_X(x)p_Y(y)$ for all values of x and y .² We denote this by $X \perp\!\!\!\perp Y$.*

Definition 2.2. Conditional Independence. *Two random variables X and Y are conditionally independent given the random variable Z if $p(x,y | z) = p(x | z)p(y | z)$ or, equivalently, if $p(x | y, z) = p(x | z)$, for all values of x, y , and z . We denote this by $X \perp\!\!\!\perp Y | Z$.*

Intuitively, if X and Y are conditionally independent given Z , the distribution of X gives no information about Y if the value of Z is known. Independence does not generally imply conditional independence and the converse is also not generally true (See A.1 for counterexamples).

Definition 2.3. Graph. *A graph is an ordered pair $(\mathcal{V}, \mathcal{E})$ in which $\mathcal{V} \triangleq \{v_k\}_{k=1}^N$ is the set of vertices or nodes in the graph, and \mathcal{E} is the set of edges in the graph, each represented as an ordered pairs of vertices. In an undirected graph, the order of the vertices in an edge is immaterial, whereas in a directed graph, the order indicates the source and destination vertices, respectively, of a directed edge. If an edge connects two vertices in a graph, they are called adjacent vertices.*

Definition 2.4. Path. *A path in a graph is a sequence of vertices in which consecutive vertices in the sequence are adjacent vertices in the graph.*

Definition 2.5. Cycle. *A cycle (or loop) in a graph is a path in which the first and last vertices are the same. A graph with no cycles is called an acyclic or tree-structured graph.*

2.2 UNDIRECTED GRAPHICAL MODELS

As stated above, a graphical model consists of a graph in which each node corresponds to a single random variable and the edges between nodes specify properties of the statistical relationships among the variables that the nodes represent. In particular, a graphical model implies a set

²We use $p_{X,Y}(x,y)$ as shorthand for $p(X = x \ \& \ Y = y)$.

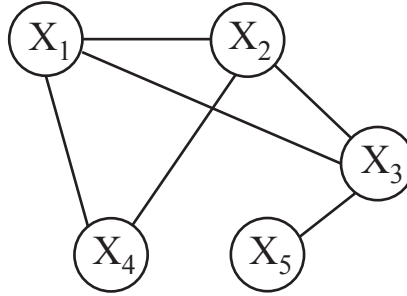


Figure 2. An undirected graphical model with five nodes.

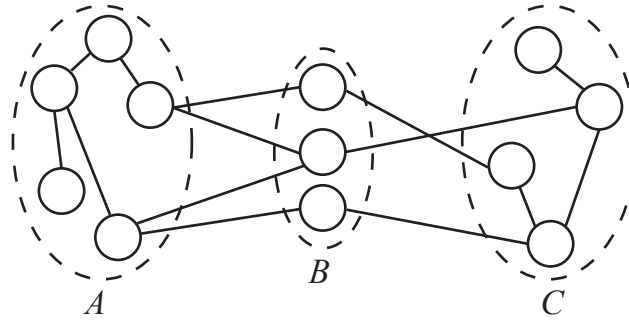


Figure 3. *u*-separation. Since all paths between A and C must pass through B , B separates A and C . Therefore, by the global Markov property, $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_C \mid \mathbf{X}_B$.

of conditional independence statements regarding its variables. The graphical modeling framework provides a means for translating graph-theoretic notions of separation and connectivity into these conditional independence statements. We will begin our description of how this is done by introducing the notion of *graph separation*, which is defined differently for directed and undirected graphs. Since an undirected graphical model (e.g., Figure 2) represents a distribution using an undirected graph, it makes use of the undirected form of graph separation, which is called *u-separation* and defined as follows:

Definition 2.6. u-separation (undirected graph separation). *Given three sets of nodes A , B , and C in an undirected graph, B is said to separate A and C if every path between A and C passes through B (See Figure 3).*

Conditional independence statements are derived from graph separation according to the *global Markov property* and its corollary, the *local Markov property*, which both hold for all types of graphical models. The former is defined as follows:

Definition 2.7. Global Markov property. *Given three sets of nodes A , B , and C in a graphical model, if B separates A and C , then $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_C \mid \mathbf{X}_B$ (See Figure 3).³*

³The global Markov property is identical for both undirected graphical models and Bayesian networks, provided the term *separates* assumes the appropriate definition of either u-separation for undirected graphical models or d-separation (Definition 2.10) for Bayesian networks.

From the global Markov property, we can begin to see how graphical models intuitively capture relationships among variables: connected variables are related in some way that involves those variables along the paths between them, while disconnected variables are clearly independent. The relationship between graph structure and variable independence will become more concrete after we define the local Markov property, which follows directly from the global for undirected graphical models:

Definition 2.8. Local Markov property. *A node X in a graphical model is conditionally independent of all other nodes in the model given its Markov blanket,*

where the *Markov blanket* of a node is defined as below for undirected graphs:

Definition 2.9. Markov blanket (for undirected graphs). *The Markov blanket of a node X in an undirected graph is the set of nodes adjacent to X .*

Therefore, a node in an undirected graphical model is conditionally independent of its non-neighbors given its neighbors, i.e., $p(X_s | \mathbf{X}_{\mathcal{V} \setminus s}) = p(X_s | \mathbf{X}_{N(s)})$, where \mathcal{V} is the set of nodes in the graph and $N(s) = \{t \mid (s, t) \in \mathcal{E}\}$ is the set of neighbors of X_s . For example, applying the Markov properties to the network in Figure 2 gives us

$$p(X_4 | X_1, X_2, X_3, X_5) = p(X_4 | X_1, X_2).$$

Along with its graphical structure, a graphical model contains a set of *potential functions*, nonnegative functions defined on subsets of its variables. For undirected graphical models, these functions are defined on the cliques of the undirected graph.⁴ Intuitively, it can be helpful to think of a potential as an “energy” function of its variables. The presence of a clique in an undirected graphical model implies that some association exists among its variables, and the clique potential can be thought of as concretizing this association, favoring certain configurations of the variables in the clique and penalizing others. For each clique C , we define a potential function $\psi_C(\mathbf{X}_C)$ which maps each possible instantiation of \mathbf{X}_C to a nonnegative real number such that the joint distribution $p(\mathbf{X})$ represented by the graphical model factorizes into the normalized product of these clique potentials:

$$p(\mathbf{X}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{X}_C), \tag{1}$$

where \mathcal{C} is the set of cliques in the graph and Z is the normalization constant

$$Z = \sum_{\mathbf{X}} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{X}_C)$$

to ensure that $\sum_{\mathbf{X}} p(\mathbf{X}) = 1$. For example, we can factorize the undirected graphical model in Figure 2 as

$$p(X_1, X_2, X_3, X_4, X_5) = \frac{1}{Z} \psi_{1,2,3}(X_1, X_2, X_3) \psi_{1,2,4}(X_1, X_2, X_4) \psi_{3,5}(X_3, X_5). \tag{2}$$

⁴A clique is a set of nodes in which each pair of nodes is connected by an edge, i.e., a completely connected subgraph.

With the slightly ethereal language of clique potentials, it may not be clear to the reader how graphical models aid in representation and computation with probability distributions. We can see from Equation (2) how the factorization afforded by the graphical model semantics brings about a reduction in the storage requirements of the joint distribution represented by the model. If all random variables are binary, the left-hand side of (2) requires $O(2^5)$ space while the right-hand side requires $O(2^3)$ space. Clearly, the space required to store the potential functions of a discrete undirected graphical model is exponential in the size of the largest clique. Essentially, the absence of edges in a graphical model allows us to make certain assumptions so that less space is needed to store the joint probability distribution represented by the model. Consider a graphical model built on a *completely connected* undirected graph, i.e., an undirected graph in which each node is adjacent to every other node. Such a graphical model makes no conditional independence statements and thus offers no reduction in the space-complexity of storing the distribution that it represents. The entire graph is a single clique and thus the joint distribution simply equals the single clique potential. Once we bring additional information to bear to this model in the way of conditional independence statements, however, we can remove certain edges and thereby reduce the space-complexity of storing the distribution.

If we can remove enough edges such that the graph becomes acyclic, we can actually define the clique potentials explicitly in terms of probability distributions over nodes in the model. To do so, we decompose the joint into a product of conditional distributions while making use of conditional independencies in the model to do so as concisely as possible. It can be easily shown (See A.2) that this procedure gives rise to the following factorization for the joint distribution $p(\mathbf{X})$ for any acyclic undirected graphical model:

$$p(\mathbf{X}) = \prod_{(s,t) \in \mathcal{E}} \frac{p(X_s, X_t)}{p(X_s)p(X_t)} \prod_{s \in \mathcal{V}} p(X_s). \quad (3)$$

Equation (3) shows that the distribution of an acyclic undirected graphical model can be factorized according to Equation (1) using only pairwise clique potentials defined on edges of the graph:

$$p(\mathbf{X}) = \frac{1}{Z} \prod_{(s,t) \in \mathcal{E}} \psi_{s,t}(X_s, X_t). \quad (4)$$

For example, consider the acyclic undirected graphical model in Figure 4. By Equation (3), we have:

$$p(ABCDEF) = p(AB) \frac{p(CB)}{p(B)} \frac{p(DB)}{p(B)} \frac{p(EB)}{p(B)} \frac{p(FE)}{p(E)} \quad (5)$$

$$= p(AB)p(C | B)p(D | B)p(E | B)p(F | E) \quad (6)$$

This formulation is useful because thinking of things in terms of probability distributions is more familiar than working with potentials over cliques. In addition, it might be a more attractive proposition to be able to specify a hyperparameter for each *variable* as opposed to specifying one for each clique, as it is arguably more intuitive to think of modeling the interaction of one variable with

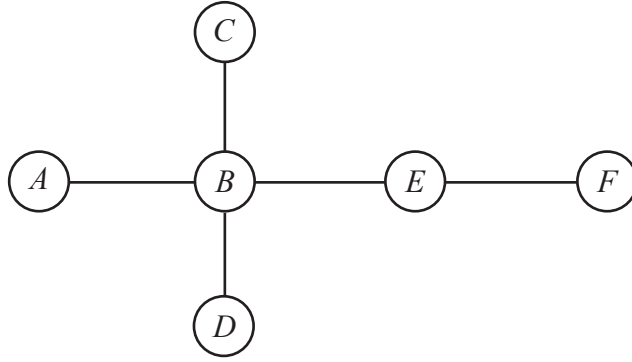


Figure 4. An acyclic undirected graphical model.

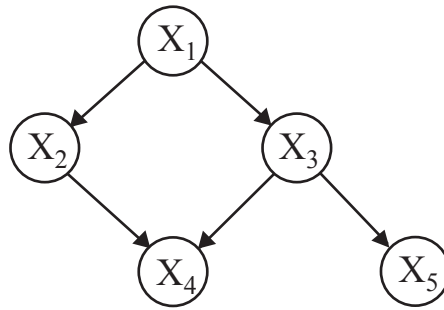


Figure 5. A Bayesian network with five nodes.

its neighbors as opposed to modeling the interactions of several variables which are all closely tied in the domain. Both features — (1) a factorization in terms of probability distributions instead of potentials, and (2) hyperparameters for variables instead of cliques — are properties of a popular class of graphical models called *directed graphical models* or *Bayesian networks*, which we shall discuss in the next section.

We began our discussion of graphical models by starting with undirected graphical models because graph separation is simpler on undirected graphs, and we showed how undirected graph separation translates into conditional independence statements involving the variables in the model. Following a similar course of discussion, we will now introduce Bayesian networks and describe the way that they use directed edges to imply conditional independence relationships to induce a factorization of the joint distribution solely in terms of marginal and conditional probability distributions.

2.3 BAYESIAN NETWORKS

A Bayesian network (Figure 5) is an acyclic directed graphical model that represents a joint probability distribution of a set of random variables using a directed graph. If a directed edge is drawn from a variable X_1 to X_2 in the Bayesian network, X_1 is said to be a *parent* of X_2 , and X_2

is in turn a *child* of X_1 .

There has been a great deal of discussion in the Bayesian network community regarding ways of properly representing causation and correlation among real-world quantities using directed graphs. Such debates often arise in medical diagnosis applications, in which certain symptoms are deemed “caused” by certain pathological conditions and thus a directed edge is drawn from the condition to the symptom to concretize this observation. Debate often arises in these and similar applications in which the line is blurred between causation and correlation. We will bypass the issue by avoiding the concept of causality and instead view the edges in a Bayesian network strictly as an abbreviation for conditional independence statements.

Similar to undirected graphical models, Bayesian networks describe statistical relationships among random variables using the structure of the underlying graph. These relationships are specified through association with graph separation via the Markov properties (Definitions 2.7, 2.8). However, the definition of graph separation is more sophisticated on directed graphs. We will define *d-separation*, or directed graph separation, through the use of the *Bayes ball algorithm* [52]. This algorithm determines whether two variables in a graph are d-separated by using the image of a ball moving through the graph to represent the spread of influence among variables. To test if two variables are d-separated, we place the ball on one and see if it can reach the other according to the rules defined below for the three types of node connections in directed graphs. The variables are d-separated if and only if the ball cannot reach the other variable.

Before we can go any further, however, we must introduce the notion of *evidence*. Evidence is any quantifiable observation of a variable in a graphical model. For discrete variables, evidence indicates that certain states or combinations of states for a set of variables are impossible. For continuous variables, evidence supplies specific values. If the variable is discrete and becomes instantiated as a result of the observation, then the evidence is called *hard*; otherwise, it is *soft*. The Bayes ball rules for each type of node connection in a directed graph and the boundary conditions are summarized in Figures 6-9. Using these rules as an illustration, we can now formally define d-separation.

Definition 2.10. d-separation (directed graph separation). *Given three sets of nodes A , B , and C in a directed graph, B is said to separate A and C if, for all paths between A and C , either (1) a serial or diverging connection exists on the path through an intermediate variable $X \in B$, or (2) a converging connection exists on the path, and neither the node at the bottom of the “v-structure” nor any of its descendants is in B . If either of these conditions holds, A and C are said to be “d-separated given B ”.*

To check for d-separation, we can shade all nodes in B and then apply the Bayes ball reachability rules. If no ball starting at any node in A can reach any node in C , or vice-versa, then A and C are d-separated given B . Figure 10 shows an example execution of the Bayes ball algorithm which determines that X_2 and X_5 are d-separated given X_3 .

Now that we have a definition for graph separation on directed graphs, we can apply the global Markov property (Def. 2.7) to Bayesian networks: $\mathbf{X}_A \perp\!\!\!\perp \mathbf{X}_C \mid \mathbf{X}_B$ whenever B separates A and C . For example, Figure 10 implies that $X_2 \perp\!\!\!\perp X_5 \mid X_3$. The local Markov property (Def. 2.8)

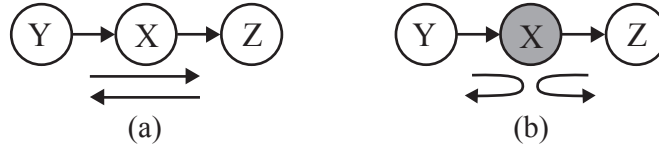


Figure 6. Serial connection. (a) If X is unknown, the ball moves freely between Y and Z . (b) If X has been instantiated, the connection is blocked. The ball bounces back in the direction from which it came.

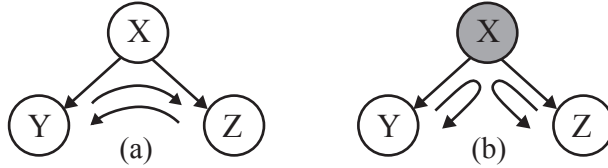


Figure 7. Diverging connection. (a) If X is unknown, the ball freely moves between X 's children. (b) If X has been instantiated, the ball is blocked.

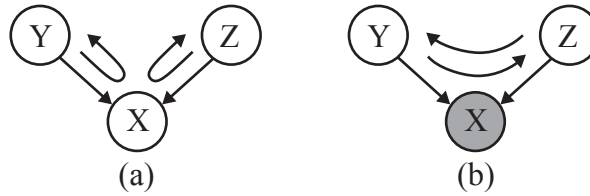


Figure 8. Converging connection. (a) If neither X nor any of its descendants has received any evidence, the ball is blocked between X 's parent nodes. (b) If X or any of its descendants has received either soft or hard evidence, the ball moves between X 's parents.



Figure 9. Boundary conditions. (a) If Y is unobserved, the ball leaves the graph. (b) If Y is observed, the ball bounces back to X .

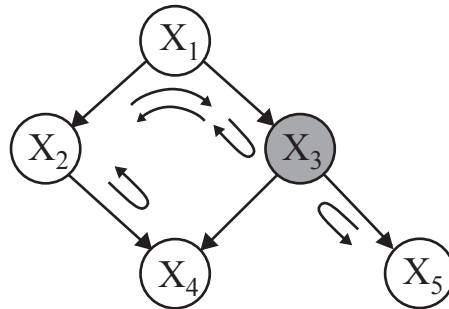


Figure 10. Bayes ball algorithm example run to check if X_3 d -separates X_2 and X_5 . The arrows are drawn to reflect the movement of the ball if it is placed on either X_2 or X_5 at the start. Clearly, the ball cannot reach one from the other, so X_2 and X_5 are d -separated given X_3 .

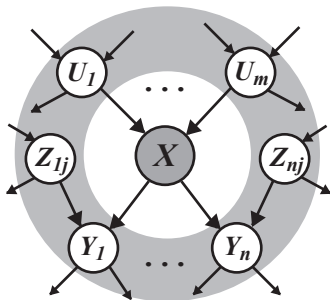


Figure 11. The shaded area is the Markov blanket of X (Adapted from [51]).

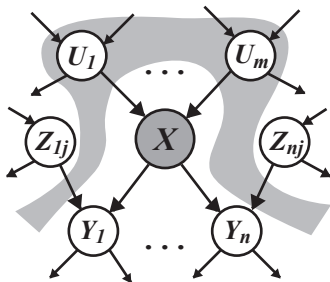


Figure 12. Equivalent way of specifying the local Markov property for Bayesian networks. A node is conditionally independent of its non-descendants given its parents (Adapted from [51]).

also holds for Bayesian networks, provided we define the Markov blanket for a node in a directed graph:

Definition 2.11. Markov blanket (for directed graphs). *The Markov blanket of a node X in a directed graph is the set of nodes containing the parents of X , the children of X , and the parents of the children of X (See Fig. 11).*

Thus, a node in a Bayesian network is conditionally independent of all other nodes given those in its Markov blanket. For example, in Figure 5, $X_2 \perp\!\!\!\perp X_5 \mid X_1, X_3, X_4$. An equivalent way of specifying the conditional independence statements implied by the local Markov property is shown in Figure 12: a node is conditionally independent of its non-descendants given its parents.

The purpose of the Markov properties is to allow us to form a compact factorization of the joint distribution represented by a graphical model using potential functions. Unlike undirected graphical models, in which the potential functions are clique potentials, we define a potential function at each node in a Bayesian network to be the conditional probability of that node given its parents. Where $pa(X_s)$ denotes the set of parents of a node X_s , the potential associated with X_s is $p(X_s \mid pa(X_s))$. If X_s has no parents, its potential is simply the prior probability $p(X_s)$. The joint distribution $p(\mathbf{X})$ factorizes into the product of the potentials at each of the nodes:

$$p(\mathbf{X}) = \prod_i p(X_i \mid pa(X_i)). \tag{7}$$

For example, the distribution represented by the network in Figure 5 factorizes as follows:

$$p(\mathbf{X}) = p(X_1)p(X_2 | X_1)p(X_3 | X_1)p(X_4 | X_2, X_3)p(X_5 | X_3).$$

For an example of how the graphical model machinery allows for efficient representation of a distribution, consider a distribution with n binary variables. We would have to specify 2^{n-1} parameters for this distribution’s joint probability table.⁵ However, for a Bayesian network representing this same distribution, we only need to specify $O(n2^k)$ parameters, where k is the maximum number of parents (“fan-in”) of any node. This number comes from the fact that there is one cpt to specify for each node and each cpt has $O(2^k)$ entries.

2.3.1 Distributions

Bayesian networks can have both discrete and continuous variables, even within the same network, in which case they are called *hybrid Bayesian networks*. Some problems can be solved using purely discrete networks, such as the simplified medical diagnosis problem described in Section 1. Using discrete quantities simplifies inference algorithms by allowing potential functions to be represented simply by conditional probability tables, but real-world systems often include continuous quantities, so applicability of purely discrete networks is limited. A common practice is to discretize continuous variables through binning, which involves dividing the range of values in a continuous distribution into a finite number of bins, each representing a range of continuous values. This allows continuous quantities to be treated as discrete variables, but results in very large cpts when several connected variables all must be binned finely enough to minimize accuracy loss. More precisely, the cpt of a variable with n parents has $O(k^{n+1})$ entries, where k is the maximum number of states of any variable. We will often wish to retain the continuous distribution representation of a variable, and fortunately there are many possibilities for conveniently specifying cpds in continuous and hybrid networks.

If a node X has continuous parents Z_1, \dots, Z_k , its cpd can be modeled as a *linear Gaussian* distribution: for parent values z_1, \dots, z_k , this amounts to $p(X | z_1, \dots, z_k) = \mathcal{N}(a_0 + a_1z_1 + \dots + a_kz_k; \sigma^2)$, where the a_0, \dots, a_k and σ^2 are fixed parameters of the cpd. It is well known that a Bayesian network in which every cpd is a linear Gaussian represents a multivariate Gaussian distribution, and that every multivariate Gaussian can be represented by a Bayesian network with all linear Gaussian cpds [54]. The cpd of a continuous node X with discrete parents \mathbf{Z} can be specified as a *conditional Gaussian* (CG) distribution; that is, each possible instantiation \mathbf{z} of the parent variables specifies the mean $\mu_{\mathbf{z}}$ and covariance $\sigma_{\mathbf{z}}^2$ of a Gaussian for X , making the cpd $p(X | \mathbf{z}) = \mathcal{N}(\mu_{\mathbf{z}}; \sigma_{\mathbf{z}}^2)$. If X has both discrete and continuous parents, we can specify a separate linear Gaussian as a function of the continuous parents for each instantiation \mathbf{z} of the discrete parents. This is known as the *conditional linear Gaussian* (CLG) distribution. We can allow discrete nodes to have continuous parents by using a *softmax* cpd [29] or a *Mixture of Truncated Exponentials* (MTE) distribution [8, 41].

Naturally, many of these ideas can be applied to any type of distribution for X , including mixture models. In practice, Gaussians are often used because many inference algorithms can

⁵The table would actually have 2^n entries, but since probabilities must sum to one, only half of the entries must be specified (if all variables are binary).

handle them without difficulty. Other distributions, including mixture models and non-parametric models, can be supported by Bayesian networks, but typically require inference algorithms that use Monte Carlo sampling or Markov chain-Monte Carlo (MCMC), both of which will be discussed in Section 5.

2.4 DYNAMIC BAYESIAN NETWORKS

Many applications in science and engineering require the ability to work with data that arrives sequentially, e.g., speech recognition, multi-target tracking, biosequence analysis, medical monitoring, and highway surveillance. In each of these applications, the observed data is used to determine properties of an underlying hidden process that is executing and producing the observable phenomena. This is the well-known *filtering* problem, which is often modeled using a *state-space model* approach. A state-space model maintains a hidden state vector which changes according to the parameters of some underlying process and of which observations are made at each timestep. State-space models contain two components: the *dynamic model*, which describes how the state changes over time, and the *observation model*, which describes how observations arise from the hidden state.

We shall use \mathbf{X}_t to represent the hidden state vector of a process at a particular time t .⁶ We shall also find it useful to represent sequences of state vectors corresponding to blocks of time; for example, we shall denote a sequence of state vectors from the start of the process up to time t as $\mathbf{X}_{0:t}$. The dynamic model specifies how the state of the process evolves over time. For generality, we will think of the dynamic model as a conditional probability distribution of the new state given all preceding states: $p(\mathbf{X}_t | \mathbf{X}_{0:t-1})$. However, we will restrict our attention to processes that satisfy the *Markov property*, which states that the current state depends on the preceding state alone and no earlier states, i.e., $p(\mathbf{X}_t | \mathbf{X}_{0:t-1}) = p(\mathbf{X}_t | \mathbf{X}_{t-1})$.⁷ Since this definition of the dynamic model is a recursive definition, we also need to define the value of the state vector at the start of the process, which again we specify as a distribution for purposes of generality. Thus, this initial state distribution, which we call the *prior*, is $p(\mathbf{X}_0)$. Therefore, the dynamic model can be fully encapsulated by $p(\mathbf{X}_t | \mathbf{X}_{t-1})$ and $p(\mathbf{X}_0)$.

We will use $\mathbf{Y}_{1:t}$ to represent the sequence of observation vectors containing observations of phenomena arising from the state vectors at the corresponding times. As with the dynamic model, while the observations may arise from a linear or otherwise functional relationship with the state, we will use distributions to represent the observation model. However, we will pose the constraint that the observations at a particular time depend only on the state at that time and on no previous states nor observations, i.e., $p(\mathbf{Y}_t | \mathbf{X}_{0:t}, \mathbf{Y}_{1:t-1}) = p(\mathbf{Y}_t | \mathbf{X}_t)$. Thus, the observation model can be characterized by the distribution $p(\mathbf{Y}_t | \mathbf{X}_t)$. In addition, we assume that the process being modeled is *stationary*, meaning, among other things, that the dynamic and

⁶In this report, we shall restrict our attention to discrete-time processes, that is, situations in which t is a natural number.

⁷A parallel can be drawn to the *local Markov property* for graphical models (Def. 2.8), by defining the Markov blanket of the current state to be simply the previous state. Thus, the current state is conditionally independent of earlier states given the one immediately preceding it.

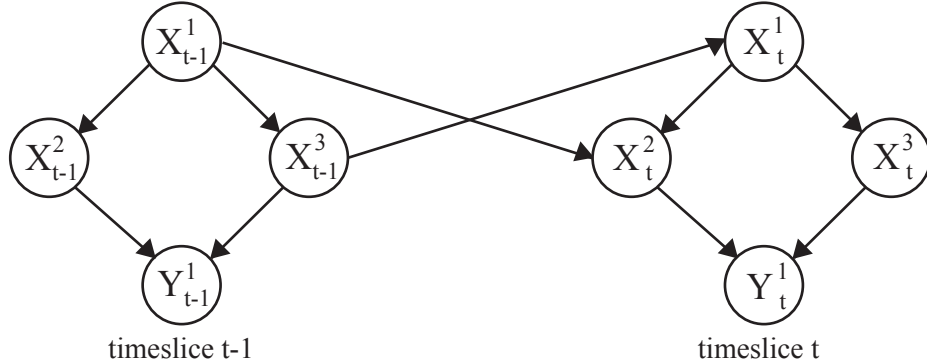


Figure 13. A 2-timeslice Bayesian network (2-TBN), which shows the dynamic and observation models using two consecutive timeslices of a process.

observation models do not depend on t . Our goal is to estimate the state sequence of the process given all the observations that have occurred up to the present, a quantity called the *posterior* distribution and given by $p(\mathbf{X}_{0:t} \mid \mathbf{Y}_{1:t})$. In particular, we are often interested in the current state given all observations up to the present, i.e., the quantity $p(\mathbf{X}_t \mid \mathbf{Y}_{1:t})$. Since this distribution represents our current beliefs about the hidden state, it is commonly called the *belief state* of the process, and also referred to as the *marginal* or *filtering* distribution.

A *dynamic Bayesian network* (DBN) is a formalism for efficiently representing such a process. More precisely, since graphical models represent *distributions*, a DBN provides a compact factorization for the posterior distribution of a process. Since we only consider processes that are stationary and Markovian, and since the observations at a given time are only dependent on the state at that time, two consecutive timeslices are sufficient to graphically depict the dynamic model and observation models using the Bayesian network machinery discussed above (Section 2.3). In doing so, we use an auxiliary structure called a *2-timeslice Bayesian network* (2-TBN), a directed acyclic graph containing two static Bayesian networks modeling the state of a process at consecutive timeslices with directed edges linking the first to the second.⁸ These directed edges are called *temporal* edges, since they span timeslices of a process. Figure 13 shows an example of a 2-TBN for a simple process with three state variables and one observation variable. The Markov blanket for a node in a 2-TBN is simply its set of parent variables; i.e., a variable is conditionally independent of all other variables in the past given values for its parents. Therefore, a 2-TBN gives the following factorization for the dynamic model:

$$p(\mathbf{X}_t \mid \mathbf{X}_{t-1}) = \prod_{i=1}^n p(X_t^i \mid pa(X_t^i)). \quad (8)$$

Given this definition, we can now formally define a DBN:

Definition 2.12. Dynamic Bayesian Network (DBN). A *dynamic Bayesian network* is an ordered pair (B_0, B_t) , where B_0 is a Bayesian network representing the prior distribution $p(\mathbf{X}_0)$

⁸To differentiate when necessary, we will use the term *static Bayesian network* to refer to any Bayesian network that is not a DBN.

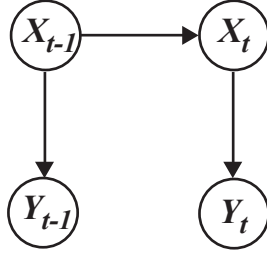


Figure 14. The 2-TBN for a hidden Markov model or Kalman filter.

of a process and B_t is a 2-timeslice Bayesian network (2-TBN) which defines the dynamic model $p(\mathbf{X}_t | \mathbf{X}_{t-1})$ of the process. Using these models, a DBN provides a compact factorization for the posterior distribution $p(\mathbf{X}_{0:T})$ for any T :

$$p(\mathbf{X}_{0:T}) = p(\mathbf{X}_0) \prod_{t=1}^T p(\mathbf{X}_t | \mathbf{X}_{t-1}). \quad (9)$$

In equations (8) and (9), we have ignored the observation variables and only focused on how the state variables and the dynamic model are represented by the Bayesian network formalism. In effect, the observation variables can be “swallowed” up into the state variables in the 2-TBN and thus become embedded in the dynamic model factorization, making the DBN construct more general than traditional state-space models. It is merely a subclass (albeit a rich and useful one) of DBNs that use explicitly-represented observation variables. Two members of this subclass are worth mentioning. Hidden Markov models (HMMs) [47] are DBNs in which \mathbf{X} is a discrete vector and \mathbf{Y} may be discrete or continuous. Kalman filters [25] are DBNs in which all variables are continuous and all cpds are linear Gaussian. In addition, these two types of models have a simple graphical structure which makes no assumptions about statistical relationship among the components of \mathbf{X} or among the components of \mathbf{Y} . This structure is depicted as a 2-TBN in Figure 14. Any DBN with only discrete state variables can be represented as an HMM by collecting all of the state variables into a single discrete state vector \mathbf{X} , though any independency assumptions implicit in the (lack of) edges in the 2-TBN will be lost in the process. These models (and their extensions) are widely used because their simple structure and distribution requirements allow for tractable algorithms for statistical inference, which we will introduce in the next section.

3. STATISTICAL INFERENCE

We have described in detail how graphical models offer a more efficient framework for computing with probability distributions by attributing a statistical independence semantics to graphical structure. In particular, this allows us write a factorization of the full joint probability distribution in terms of functions on subsets of nodes. But thus far we have only discussed how various types of graphical models aid in representing distributions. We have not yet discussed how to do any computations involving the distributions we are modeling. To show what we mean by computing with a distribution, we will return to the medical diagnosis network from Section 1. Considering this model, we may have observed that a patient has a sore throat and we want to see the likelihoods of the various diseases represented by the model. Suppose in particular that we are interested in the probability that the patient has angina. That is, we are interested in the conditional distribution $p(\text{angina} \mid \text{sorethroat} == \text{'true'})$. How do we obtain this distribution from the model?

Statistical inference consists of performing calculations on a probability distribution to obtain statistical quantities of interest. In the context of graphical models, inference is generally the computation of a particular distribution in a graphical model given evidence. For static Bayesian networks, this distribution could be either the marginal probability of a node, the joint probability of a set of nodes, or the conditional probability of one set of nodes given another. To obtain these distributions, we can compute the joint probability of the entire network given the evidence and then marginalize out the variables in which we are not interested.⁹ However, one can choose multiple orders in which to marginalize out unwanted variables and the different orders do not, in general, require the same numbers of computations. Exact inference algorithms describe efficient ways of performing this marginalization while handling the intermediate terms that arise as efficiently as possible. We will discuss two exact inference algorithms in detail in Section 4. However, exact inference is NP-hard in general, implying the existence of cases for which no exact algorithm will be able to efficiently perform inference. Therefore, there have been many efforts in recent years aimed at the design and convergence analysis of algorithms for approximate inference, which we will discuss in Section 5.

For dynamic Bayesian networks, there are more possibilities to consider in terms of applying evidence and querying distributions. This is shown in the variety of nomenclature for these different situations, which include *filtering*, the various types of *smoothing*, and *Viterbi decoding*. These computations can all be performed by doing inference on a graphical model. Many of the same approaches and techniques from inference with static Bayesian networks can be applied to the dynamic setting.

The purpose of this section is to introduce the various problems and fundamental challenges associated with statistical inference in order to make the descriptions of exact and approximate inference algorithms in Sections 4 and 5 more approachable. In Section 3.1, we will introduce inference by starting with the simplest exact algorithm for computing a single distribution, *variable elimination*. Then, in Section 3.2, we will show how variable elimination can be extended

⁹Joints and marginals are obtained directly through marginalization of the full joint, and since conditional distributions are nothing more than quotients of joint distributions, a conditional can be obtained through two executions of this inference algorithm followed by a divide operation.

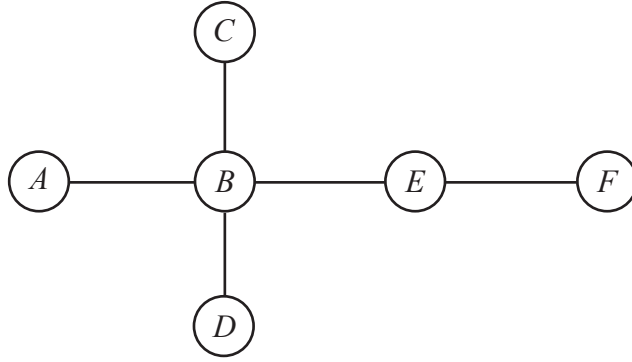


Figure 15. An acyclic undirected graphical model.

to an algorithm which computes the marginal distributions of all variables in a graphical model simultaneously. This algorithm, known as *belief propagation* or the *sum-product algorithm*, forms the basis of several inference algorithms for graphical models, including the *junction tree algorithm* which will be discussed in Section 4.1. Finally, in Section 3.3, we will discuss the various dynamic inference problems in the setting of dynamic Bayesian networks.

3.1 VARIABLE ELIMINATION

To describe the variable elimination algorithm, we proceed by example. Suppose we want to compute the marginal distribution $p(A)$ from a joint distribution $p(ABCDEF)$. We can do so by marginalizing out all other variables from the full joint:

$$p(A) = \sum_{BCDEF} p(ABCDEF). \quad (10)$$

Assuming all variables are discrete for simplicity, this algorithm requires storing a table of size exponential in the number of variables in the distribution. However, suppose that the distribution is represented by the undirected graphical model in Figure 15. We shall use the factorization afforded by the graphical model semantics to reduce the time and space complexity of this computation. In particular, noting the absence of cycles in the model in Figure 15, we can make use of the pairwise factorization formula for acyclic undirected graphical models (Equation (3)) to obtain an expression for the joint probability of the nodes in the model explicitly in terms of probability distributions. Applying this formula to the undirected graphical model in Figure 15, we obtain:¹⁰

$$p(ABCDEF) = p(AB)p(C | B)p(D | B)p(E | B)p(F | E). \quad (11)$$

Returning to the computation of the marginal $p(A)$, we can use the factorization from (11)

¹⁰Naturally, there are multiple ways of using (3) to factorize $p(ABCDEF)$; we chose the one in (11) because it seemed reasonable given the graph.

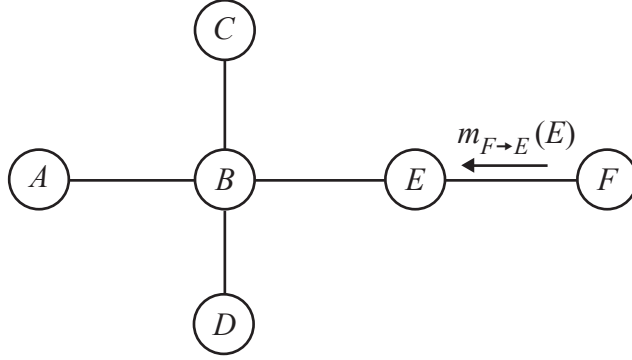


Figure 16. A message passed from node F to node E .

and then “push” sums inside of products to reduce the number of required computations:

$$p(A) = \sum_{BCDEF} p(AB)p(C | B)p(D | B)p(E | B)p(F | E) \quad (12)$$

$$= \sum_B p(AB) \sum_C p(C | B) \sum_D p(D | B) \sum_E p(E | B) \sum_F p(F | E). \quad (13)$$

In Equation (13), we have pushed the summations to the right as far as possible in order to reduce the number of computations needed. If all variables are binary, Equation (12) requires 120 multiplications and 62 additions while Equation (13) requires only 16 multiplications and 10 additions. Proceeding with the computation, we can show how the terms in Equation (13) correspond to a type of local message passing between nodes in the graph. Beginning with the rightmost term, when we marginalize out F from $p(F | E)$, we obtain a function over E which we will denote as $m_{F \rightarrow E}(E)$:¹¹

$$p(A) = \sum_B p(AB) \sum_C p(C | B) \sum_D p(D | B) \sum_E p(E | B) m_{F \rightarrow E}(E). \quad (14)$$

The notation comes from the observation that the term $m_{F \rightarrow E}(E)$ can be viewed as a “message” from the summation over F to the summation over E since it implicitly contains all the information needed about F to do the desired inference but only depends explicitly on E . This message has the graphical interpretation of being sent from node F to node E , as shown in Figure 16. Therefore, once E receives this message from F , E encapsulates all information about the variable F that is needed to compute the desired marginal. The variable F is *eliminated* from the summations.

¹¹Actually, $\sum_F p(F | E) = 1$, but we will avoid simplifying for purposes of generality.

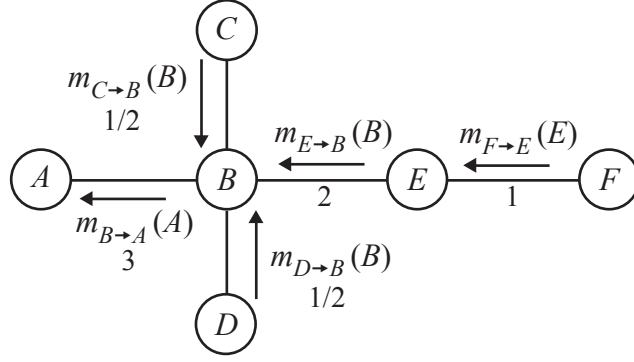


Figure 17. Message passes from an example run of the variable elimination algorithm to compute $p(A)$. A node x can only send a message to a neighbor y after x has received a message from each of its neighbors (besides y). The numbers associated with each message indicate the order in which the message must be computed. If two messages have the same number, they can be computed at the same time in parallel. The label “1/2” indicates that the message can be computed either during step 1 or during step 2.

Continuing in the same way, we have

$$p(A) = \sum_B p(AB) \sum_C p(C | B) \sum_D p(D | B) m_{E \rightarrow B}(B) \quad (15)$$

$$= \sum_B p(AB) m_{E \rightarrow B}(B) \sum_C p(C | B) \sum_D p(D | B) \quad (16)$$

$$= \sum_B p(AB) m_{E \rightarrow B}(B) \sum_C p(C | B) m_{D \rightarrow B}(B) \quad (17)$$

$$= \sum_B p(AB) m_{E \rightarrow B}(B) m_{D \rightarrow B}(B) \sum_C p(C | B) \quad (18)$$

$$= \sum_B p(AB) m_{E \rightarrow B}(B) m_{D \rightarrow B}(B) m_{C \rightarrow B}(B) \quad (19)$$

$$= m_{B \rightarrow A}(A). \quad (20)$$

Note that we specified the term $m_{E \rightarrow B}(B)$ as being a message from E to B since it only depends on B ; in general, we will try to push messages as far to the left as possible in the product of summations for reasons of efficiency. Note that for the same reason we move the term $m_{D \rightarrow B}(B)$ to the left of the summation over C . The procedure we followed above is generally referred to as the *variable elimination* algorithm, and the series of message passes for our example run is depicted graphically in Figure 17.

Looking back at Equations 12 and 13, it becomes clear that there can be multiple orderings in which sums can be pushed inside of products. In general, these different *elimination orderings* result in different numbers of required computations. Unfortunately, finding the optimal elimination ordering – the ordering which results in the fewest required computations – is NP-Hard [2]. Consequently, a family of inference algorithms has arisen which focuses solely on obtaining as near-optimal an elimination ordering for a particular query as possible. Such algorithms are query-driven

and include algorithms such as variable elimination, symbolic probabilistic inference (SPI) [35, 53], and bucket elimination [13]. We shall discuss an early SPI algorithm called *set factoring* in Section 4.2. Set factoring essentially frames the problem of choosing an elimination ordering as a combinatorial optimization problem, thereby providing a framework for algorithms which use heuristics to come up with orderings that approach the minimum number of computations required.

The idea of pushing sums inside of products is not new. It can be applied to any commutative semi-ring, yielding some very well-known algorithms in diverse fields depending on the choice of semi-ring, including the Hadamard and fast Fourier transforms, the Baum-Welch algorithm, and turbo decoding, among others [1]. For an example that will be useful in the graphical model framework, we can obtain Viterbi’s algorithm for finding the most likely configuration of states for the variables in a graphical model by simply changing “sum” to “max” in the equations above. Naturally, the algorithms that we will describe for performing exact inference in a graphical model can be suitably molded to perform any of these tasks as well.

3.2 BELIEF PROPAGATION

Having computed $p(A)$ via variable elimination, now suppose that we wish to compute the marginal $p(D)$. We can use variable elimination again, but we will find that we are re-computing messages. In particular, the terms $m_{F \rightarrow E}(E)$, $m_{E \rightarrow B}(B)$, and $m_{C \rightarrow B}(B)$ will be re-computed. Since variable elimination takes $O(n)$ time, where n is the number of nodes in the graph, calling it to compute each marginal will take $O(n^2)$ time altogether. Caching schemes can be implemented to store intermediate results for future queries, but we would prefer to develop a single algorithm to compute all marginals of a model simultaneously. In fact, we can use dynamic programming to essentially run variable elimination for all variables simultaneously to compute all n marginals in only $O(n)$ time. The algorithm that we will develop is called *belief propagation* or the *sum-product* algorithm.

To do so, we formalize the operations we used above in Equations (14)-(20) by observing the following formula for computing a message from a node Y to a neighboring node X :

$$m_{Y \rightarrow X}(X) = \sum_Y \psi(XY) \prod_{Z \in \mathcal{N}(Y) \setminus X} m_{Z \rightarrow Y}(Y) \quad (21)$$

where $\psi(XY)$ is a potential function over X and Y and $\mathcal{N}(Y)$ is the set of neighbors of node Y . The reader can verify that each step in the computation of $p(A)$ above fits this formula. The formula for the marginal distribution of a node X then becomes

$$p(X) \propto \prod_{Z \in \mathcal{N}(X)} m_{Z \rightarrow X}(X). \quad (22)$$

Equations (21) and (22) comprise the *belief propagation* or *sum-product* algorithm.¹² So far, however, nothing is new; we have merely formalized the operation of the variable elimination algorithm. However, as Equation (21) suggests, a node can only send a message to a particular neighbor once

¹²The nomenclature *sum-product* comes from Equation (21).

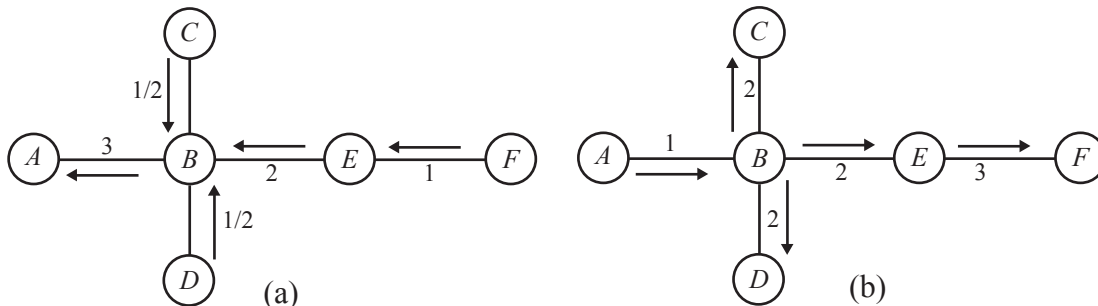


Figure 18. (a) COLLECT-TO-ROOT, the first series of message passes, where A is designated (arbitrarily) as the root node. The numbers indicate the order of message passes. If two have the same number, it means that they can occur at the same time. (b) DISTRIBUTE-FROM-ROOT, the second series of message passing.

it has received messages from all other neighbors. This suggests a “divide-and-conquer” strategy of computing messages to ensure that no messages are computed twice. In particular, dynamic programming can be used as a mechanism for storing results of computations in subsequent queries. We can compute all marginal distributions of the model in only two stages of message passing, called COLLECT-TO-ROOT and DISTRIBUTE-FROM-ROOT and shown in Figure 18(a) and (b), respectively. Each stage takes $O(n)$ time, making the entire belief propagation algorithm $O(n)$. One node is arbitrarily chosen as the root node and, in the first stage, all other nodes send messages to their neighbors towards the root, starting with the leaf nodes. A node only computes and sends its message after receiving messages from all but one of its neighbors, and then sends its message to this neighbor. In the second stage, the root begins the process by sending messages to all of its neighbors, which in turn send messages to each of their neighbors, etc. When message passing is completed, the marginals of all nodes in the network have been computed. For simplicity, we have not considered cases in which evidence is applied to one or more of the nodes in the model. In Appendix A.3, we include a derivation following [58] that handles evidence.

We have developed the belief propagation algorithm for exact inference on acyclic undirected graphical models with discrete nodes, but the same basic algorithm can be applied to many other types of graphical models. For instance, with straightforward modifications, the algorithm can handle Gaussian distributions in acyclic undirected graphical models [58]. In addition, belief propagation can be adapted with minimal change to *singly-connected Bayesian networks*, that is, Bayesian networks without any undirected cycles. This well-known exact inference algorithm is due to Pearl [45]. For models with cycles, belief propagation can still be applied – in which case it is called *loopy belief propagation* – but only as an approximate algorithm. For certain models, loopy belief propagation does not converge and produces endless oscillation, but in many cases the algorithm provides excellent results [44]. To perform exact inference in models with cycles, some variant of the *junction tree algorithm* is commonly used [34]. This algorithm, which we discuss in Section 4.1, converts a Bayesian network into an acyclic undirected graphical model called a *junction tree* and then executes a modified version of belief propagation on it. Also, the junction tree algorithm has been extended to perform exact inference in Bayesian networks which contain both discrete variables and continuous variables from the exponential family [32]. Finally, belief propagation has been extended for approximate inference in graphical models with continuous,

non-Gaussian distributions, with or without cycles. This algorithm, called *nonparametric belief propagation* (NBP) [59], combines techniques from the standard belief propagation algorithm developed above along with sampling techniques similar to those we will discuss in Section 5.

3.3 INFERENCE IN DYNAMIC BAYESIAN NETWORKS

Since DBNs model dynamic systems, inference can take on several forms, as shown in Figure 19 from [42]. We describe each of these problems in the sections below.

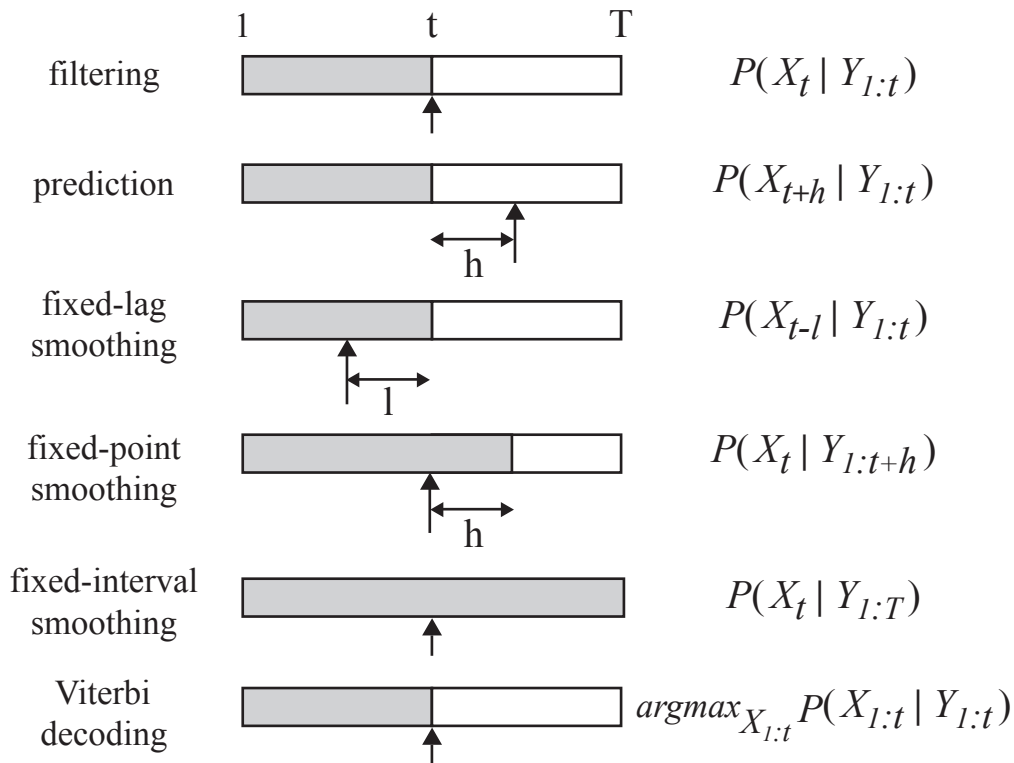


Figure 19. The main types of inference for DBNs. The shaded portions show the amount of time for which we have data. All inference types except fixed-interval smoothing are online. The arrows indicate the times at which we infer the state vector. The current time is t and T is the ending time of the process. The prediction horizon is h and the smoothing lag is l (Adapted from [42]).

3.3.1 Filtering

In many applications, we want to estimate the belief state of a process at each timestep during its execution. This task is called *filtering* and can be solved exactly through a recursion that we shall describe below. This will give us the exact solution, but we must keep in mind that the integrals are not tractable in general but can only be computed if certain assumptions can be

made.

Given the belief state $p(\mathbf{X}_{t-1} | \mathbf{Y}_{1:t-1})$ at time $t - 1$, we can obtain the belief state at time t by proceeding for a single iteration through two stages: *prediction* and *update*. From the belief state at time $t - 1$, the prediction phase uses the dynamic model to predict the next state of the process at time t using the Chapman-Kolmogorov equation:

$$p(\mathbf{X}_t | \mathbf{Y}_{1:t-1}) = \int p(\mathbf{X}_t | \mathbf{X}_{t-1})p(\mathbf{X}_{t-1} | \mathbf{Y}_{1:t-1})d\mathbf{X}_{t-1}. \quad (23)$$

In this step, we have made use of the assumption that the process is first order Markov. From the predicted state, the update phase uses the measurement model to refine the prediction and obtain the belief state at time t using Bayes' rule:

$$p(\mathbf{X}_t | \mathbf{Y}_{1:t}) = \frac{p(\mathbf{Y}_t | \mathbf{X}_t)p(\mathbf{X}_t | \mathbf{Y}_{1:t-1})}{\int p(\mathbf{Y}_t | \mathbf{X}_t)p(\mathbf{X}_t | \mathbf{Y}_{1:t-1})d\mathbf{X}_t}. \quad (24)$$

Recursive filtering using these formulas will obtain the exact posterior density of the belief state of the process at any desired time. However, it is not tractable in general; only in certain cases can we be sure that the optimal solution is computable, such as when the state-space model equations are known linear functions and the densities are Gaussian. That is, if $p(\mathbf{X}_{t-1} | \mathbf{Y}_{1:t-1})$ is Gaussian, it can be shown that $p(\mathbf{X}_t | \mathbf{Y}_{t-1})$ is also Gaussian so long as the dynamic and measurement models are known linear functions with additive Gaussian noise. That is, the models are given respectively by the following two equations:

$$\mathbf{X}_t = F_t\mathbf{X}_{t-1} + \mathbf{v}_{t-1}$$

$$\mathbf{Y}_t = H_t\mathbf{X}_t + \mathbf{n}_t,$$

where F_t and H_t are known matrices which define the linear functions and \mathbf{v}_{t-1} and \mathbf{n}_t are independent and identically-distributed (iid) samples from Gaussian distributions with known parameters. In this case, the Kalman filter [25] provides the optimal solution. When the system does not satisfy such constraints, however, exact inference is often intractable, so we must resort to approximate algorithms, such as the Extended Kalman Filter (EKF) or sequential Monte Carlo sampling techniques.

Prediction and Viterbi Decoding We have overloaded the term *prediction* by using it to refer both to the inference problem of prediction shown in Figure 19 and to the first stage of the filtering recursion above. However, this nomenclature turns out to be quite reasonable, since the inference problem of prediction can be solved through repeated application of the Chapman-Kolmogorov equation until the belief state at the desired horizon h is obtained.

Viterbi decoding determines the most likely sequence of states for the given observations, i.e.: $\arg \max_{x_{1:t}} p(x_{1:t} | y_{1:t})$. This computation is closely linked with filtering and indeed it is well-known that any filtering algorithm can be converted to a Viterbi algorithm essentially by replacing integration or summation with a “max” operation.

3.3.2 Smoothing

Smoothing is a family of operations that estimate the state at some point in the past in light of subsequent observations. Formulas can be derived for computing smoothing distributions in a fashion similar to the derivation of the filtering equations above; again, the integrals may not be computable unless certain assumptions are made. In particular, there are analogous equations to the Kalman filter for smoothing, generally called *Kalman smoothing* equations.

There are several special types of smoothing which may be particularly appropriate for one case or another, but the algorithms to perform the computations are very similar. *Fixed-lag smoothing* consists of estimating the state vector at some varying point in the past given observations up to the present. This requires computing the distribution $p(\mathbf{X}_{t-l} | \mathbf{Y}_{1:t})$, where $l > 0$ is some time lag back into the past in which we are interested and t is the current time. *Fixed-point smoothing* is estimation of the state vector at a particular time in the past with a varying range of subsequent observations to consider, i.e., computing the distribution $p(\mathbf{X}_t | \mathbf{Y}_{1:t+h})$, where $h > 0$ is the observation horizon. *Fixed-interval smoothing* is a special case of fixed-point smoothing which is performed offline with a full set of observations. That is, the computation of $p(\mathbf{X}_t | \mathbf{Y}_{1:T})$ where the horizon is taken to be T , the final time of the process.

4. EXACT INFERENCE ALGORITHMS

Most exact inference algorithms for Bayesian networks fall into one of two categories. The first type consists of query-driven algorithms which retain a symbolic representation of the computation to be performed and then use several techniques to simplify it, including pruning ineffectual nodes and approximating an optimal ordering in which to marginalize out unwanted variables from the given joint distribution for the query. Such algorithms typically run in $O(n)$ time given a query for a particular distribution. Thus, computing all n marginal distributions of a graphical model requires $O(n^2)$ time, but caching schemes may be used in practice to allow sequential queries to re-use computations. This category of algorithms includes variable elimination (described in Section 3.1), bucket elimination [13], and *symbolic probabilistic inference* (SPI) [35, 53] algorithms. The other major category of exact inference algorithms is composed of message-passing algorithms [20] which compute all marginals simultaneously using a series of local message passes, taking $O(n)$ time to do so. Such algorithms (e.g., junction tree) are based on the belief propagation algorithm for acyclic undirected graphical models, which we described in Section 3.2. In addition, both categories of algorithms have been adapted to perform inference on dynamic Bayesian networks [42].

In Section 4.1, we will describe the junction tree algorithm for static Bayesian networks and, in Section 4.1.3, we will show how it can be extended for inference in dynamic Bayesian networks. In Section 4.2, we will introduce SPI for static networks as we discuss one set factoring algorithm in detail.

4.1 THE JUNCTION TREE ALGORITHM

The junction tree algorithm [34] is based on the belief propagation algorithm for inference in acyclic undirected graphical models, but is designed for Bayesian networks with or without undirected cycles. To perform inference, the algorithm converts the Bayesian network into an acyclic secondary structure (called a *junction tree*) which encapsulates the statistical relationships of the original network in an acyclic, undirected graph. Inference is then performed through message-passing on the junction tree according to the belief propagation scheme. Several forms of the junction tree algorithm have appeared in the literature, but the algorithm described below is the *Hugin architecture* and most closely follows the notation of [20]. This architecture is for discrete networks, but extensions of the junction tree algorithm have been developed for exponential families and other well-behaved distributions [33].

In Section 4.1.1, we will discuss the creation of a junction tree from a Bayesian network. While we will not describe each algorithm in the process, we will outline each step and provide references for commonly-used algorithms for each task. In Section 4.1.2, we will describe the message-passing scheme of the algorithm and show how it follows the belief propagation algorithm derived in Section 3.2. Finally, in Section 4.1.3, we will adapt the junction tree algorithm for inference in dynamic Bayesian networks.

4.1.1 Creating a Junction Tree

In building an undirected secondary structure for inference, we need to ensure that we preserve the statistical relationships implicated by the original Bayesian network. The first step in junction tree construction consists of connecting nodes in the Bayesian network that share a child and dropping directional arrows off all edges, resulting in the so-called *moral graph* (Figure 20(b)). This process, called *moralization*, ensures that the Markov blanket for a given node in the original

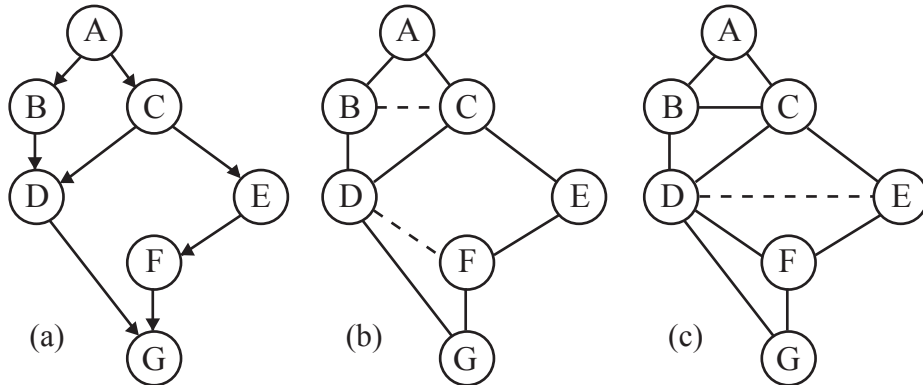


Figure 20. (a) A Bayesian network. (b) The moral graph. Dotted edges were created during moralization. (c) The triangulated moral graph. The dotted edge was created during triangulation. Note that adding an edge perpendicular to the dotted one above would have resulted in a different triangulated graph.

Bayesian network is identical to its Markov blanket in the moral graph (See Definitions 2.9 and 2.11). Thus, by the local Markov property, conditional independencies in the original network are preserved during moralization.

We then modify the moral graph by adding edges to it so as to make it *triangulated* (Figure 20(c)), that is, so that every cycle of length greater than three contains an edge between two non-adjacent nodes in the cycle. Intuitively, every cycle in a triangulated graph is either a triangle or encloses a triangle. In general, there may be multiple ways to triangulate a graph (Figure 20(c)). The optimal triangulation minimizes the sum of the state space sizes of the cliques in the triangulated graph; however, obtaining an optimal triangulation is NP-complete [2, 64]. One common choice is to use a greedy, polynomial-time approximation algorithm from Kjaerulff [27]. The purpose of triangulation is so that we can create a junction tree from the triangulated graph: it can be shown that every triangulated graph has a junction tree associated with it, but this is not true of undirected graphs in general. Given a triangulated graph, we are now ready to build the junction tree.

A junction tree is an acyclic undirected graph in which every node corresponds to a nonempty set of variables and is called a *cluster*. Each edge, named by the nonempty set of variables in common between the two clusters at its endpoints, is called a *separator set*, or *sepset* for short. So, each sepset is a subset of both of its endpoint clusters, and two clusters connected by an edge must have a non-empty intersection. In fact, we go further to require that the clusters satisfy the following

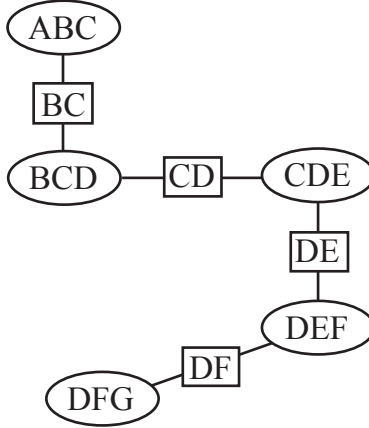


Figure 21. Junction tree created from the triangulated graph in Figure 20(c). Ovals are clusters and are labeled with cliques in the triangulated graph, and rectangles are sepsets, labeled by the intersection of the clusters at their endpoints.

running intersection property:

Definition 4.1. Running intersection property (RIP). For any two clusters R_1 and R_2 in a junction tree, all clusters on the path from R_1 to R_2 contain $R_1 \cap R_2$.

Intuitively, this means that a variable cannot disappear and then reappear along a path in the junction tree. Figure 21 shows a junction tree constructed from the triangulated graph in Figure 20(c). The cliques from the triangulated graph have become the clusters of the junction tree. An algorithm from Golumbic [18] identifies cliques in the triangulated graph and an algorithm from [22] can be used to connect the cliques together in such a way as to satisfy the junction tree property and with consideration for optimality. The links (sepsets) created to connect cliques are labeled by the overlapping nodes.

A junction tree also contains a potential ϕ associated with each cluster and with each sepset that satisfy the following constraints:

- For each cluster R and each sepset S ,

$$\sum_{R \setminus S} \phi_R = \phi_S \tag{25}$$

When the above holds for a cluster R and a neighboring sepset S , ϕ_S is *consistent* with ϕ_R . When every cluster-sepset pair is consistent, the junction tree is *locally consistent*.

- Where ϕ_{R_i} and ϕ_{S_j} are cluster and sepset potentials, respectively,

$$p(\mathbf{X}) = \frac{\prod_i \phi_{R_i}}{\prod_j \phi_{S_j}} \tag{26}$$

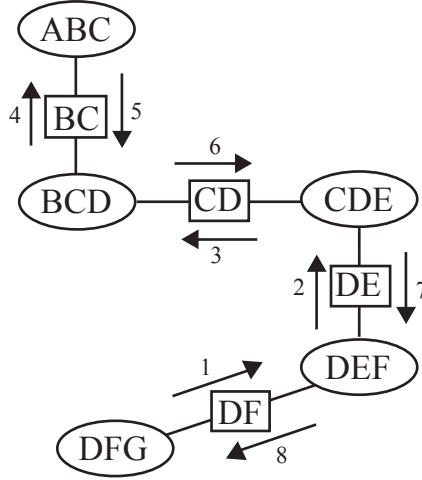


Figure 22. Message passing in the junction tree. The root is the cluster ABC.

To initialize cluster and sepset potentials in the junction tree, we begin by setting each to the identity element (all 1's in the discrete case). Then, for each variable X in the original Bayesian network, we choose a cluster R in the junction tree which contains X and X 's parents and multiply $p(X | pa(X))$ onto ϕ_R . After initialization, Equation (25) is not satisfied for all pairs and thus the junction tree is inconsistent, but Equation (26) is satisfied since it evaluates to the factorization for Bayesian networks (Equation 7), as shown below:

$$\frac{\prod_i \phi_{R_i}}{\prod_j \phi_{S_j}} = \frac{\prod_k p(X_k | pa(X_k))}{1} = p(\mathbf{X}).$$

4.1.2 Inference Using the Junction Tree

Now that we have a junction tree constructed from the original network, we can perform inference through a series of message passes. Similar to belief propagation (Section 3.2), this is done in two phases, COLLECT-TO-ROOT and DISTRIBUTE-FROM-ROOT, as shown in Figure 22. The purpose of message passing is to achieve local consistency (Equation (25)) in the junction tree while preserving the truth of Equation (26). Once completed, a variable's marginal can be obtained from any cluster containing that variable through simple marginalization of the cluster potential.

A message pass from one cluster to its neighbor proceeds as follows, where R is the sender, T the receiver, and S the sepset between them.

1. Save the current sepset potential and assign a new value for it by marginalizing down from R .

$$\phi'_S = \phi_S$$

$$\phi_S = \sum_{R \setminus S} \phi_R$$

2. Update ϕ_T using ϕ_S , ϕ'_S , and ϕ'_T .

$$\phi'_T = \phi_T$$

$$\phi_T = \phi'_T \frac{\phi_S}{\phi'_S}$$

Equation (26) remains satisfied, as shown below:

$$\left(\frac{\prod_i \phi_{X_i}}{\prod_j \phi_{S_j}} \right) \frac{\phi'_S \phi_T}{\phi_S \phi'_T} = \left(\frac{\prod_i \phi_{X_i}}{\prod_j \phi_{S_j}} \right) \frac{\phi'_S}{\phi_S} \frac{\phi'_T \frac{\phi_S}{\phi'_S}}{\phi'_T} = p(\mathbf{X})$$

This message-passing procedure is known as the *Hugin architecture*. It essentially stores the product of all messages at each clique and accommodates a new message by multiplying it onto the product and dividing out the “old” version of the message. Another popular variant is *Shafer-Shenoy* message passing [55], which does not use a division operation (though fine for some continuous distributions, like Gaussians, division is problematic for Gaussian mixtures) and requires less space since it does not store a product, but takes longer since it repeatedly remultiplies messages. Other variants include Lazy Propagation [38] and multiply sectioned Bayesian network (MSBN) inference. Extensions to support inference in Bayesian networks with conditional Gaussian distributions can also be implemented straightforwardly [33].

4.1.3 Junction Tree Algorithms for Dynamic Bayesian Networks

There are several ways to use junction trees for implementing inference in DBNs. The naïve approach is to “unroll” the DBN for the desired number of timeslices and then perform inference on the resulting model as if it were a static Bayesian network. However, this will obviously be too time-consuming or memory-intensive, particularly in an application such as filtering or online smoothing. Intuitively, we should be able to do better by making use of the assumption that all processes being modeled are stationary and Markovian. Here, we describe the *interface algorithm* from Murphy [42] which essentially runs the static junction tree algorithm on the 2-TBN for a single timeslice pair, then “advances” the algorithm one timestep, saving all information about the process up to that point needed to do inference in the next timeslice.

To show why we can do this, we need to introduce some definitions. Where \mathbf{X}_t is the set of nodes in timeslice t , the set of *temporal edges* between timeslices $t-1$ and t is denoted $\mathcal{E}^{tmp}(t)$ and given by $\mathcal{E}^{tmp}(t) = \{(u, v) \in \mathcal{E} \mid u \in \mathbf{X}_{t-1}, v \in \mathbf{X}_t\}$. The *outgoing interface* I_{t-1} of timeslice $t-1$ is the set of nodes with children in timeslice t and given by

$$I_{t-1} = \{u \in \mathcal{V}_{t-1} \mid (u, v) \in \mathcal{E}^{tmp}(t), v \in \mathcal{V}_t\}.$$

Figure 23 shows an example of this definition. Murphy showed that the outgoing interface I_t d-separates the past from the future and therefore is a *sufficient statistic* of the past for performing inference in future timesteps [42]. Here, “past” refers to the nodes in timeslices before t along with the non-interface nodes in timeslice t , while “future” refers to all nodes in timeslices after t .

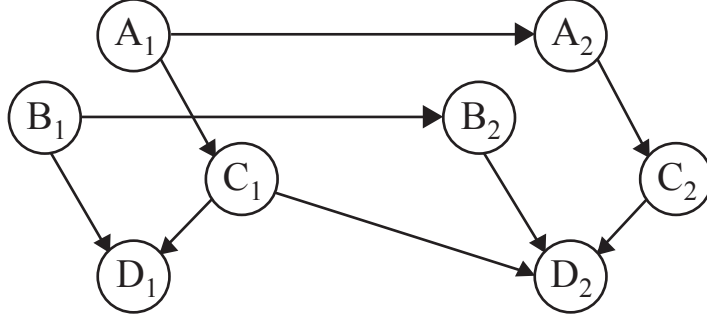


Figure 23. A 2-TBN showing timeslices $t - 1$ and t . The outgoing interface for timeslice $t - 1$ is $I_{t-1} = \{A, B, C\}$.

From this result, it is easy to see that we only need to maintain the distribution over the outgoing interface at a given timestep in order to fully represent what happened in the past. In the context of junction trees, this translates to maintaining the clique potential on the clique containing the outgoing interface from timestep to timestep.

Recall that a DBN is an ordered pair (B_0, B_t) , where B_0 is a Bayesian network representing the prior distribution $p(\mathbf{X}_0)$ of a process and B_t is a 2-timeslice Bayesian network (2-TBN) which defines the dynamic model $p(\mathbf{X}_t | \mathbf{X}_{t-1})$. The interface algorithm begins with some initialization steps which build the junction trees that will be used for inference. In particular, a junction tree J_0 is built from B_0 according to the process specified above in Section 4.1.1 with some modifications that we will describe below, and another junction tree J_t is built from B_t , also with several key modifications. To do filtering, we only need to perform static junction tree inference on the current timestep, using the clique potential from the outgoing interface from the previous timestep to encapsulate all required information of the process up to the present. Thus, we need to ensure that the outgoing interface is fully contained within at least one clique of each junction tree. The steps in the creation of J_0 and J_t include modifications to enforce this constraint.

The steps for creating J_0 from B_0 are shown in Figure 24, with B_0 shown in Figure 24(a) along with its outgoing interface labeled. First, B_0 is moralized, producing the moral graph in Figure 24(b). Then, in a departure from the procedure outlined in Section 4.1.1, edges are added so that the outgoing interface becomes a clique, as shown in Figure 24(c). As mentioned above, the reason for this is so that, when we advance timesteps, the potential on the outgoing interface nodes will be fully contained in at least one clique. Finally, triangulation, junction tree formation, and clique potential initialization proceed as before, producing the junction tree in Figure 24(d). The clique containing the outgoing interface is labeled as the *out-clique*.

The steps for creating a junction tree J_t from the 2-TBN B_t are shown in Figure 25. The 2-TBN must first be changed into a *1.5DBN*, which contains all nodes in the second timeslice of the 2-TBN but only those nodes from the first slice which have children in the second, that is, nodes in the outgoing interface I_{t-1} . The resulting 1.5DBN for the 2-TBN from Figure 23 is shown in Figure 25(a). Then, we moralize the 1.5DBN, as shown in Figure 25(b). Again, we must ensure that the outgoing interface is fully contained within a clique of the junction tree; however, there are two

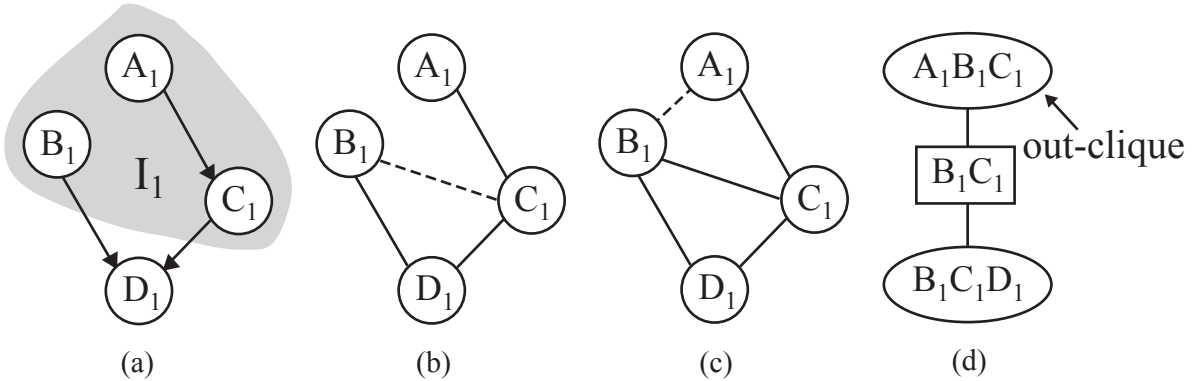


Figure 24. Steps to construct a junction tree from the prior B_0 . (a) B_0 for the DBN from Figure 23, with outgoing interface labeled. (b) The moral graph for B_0 . (c) All nodes in I_1 must be in a single clique in the junction tree, so we add edges to connect them. (d) Finally, we triangulate, form the junction tree, and initialize clique potentials according to the procedure in Section 4.1.1. The clique containing the outgoing interface nodes is labeled as the out-clique.

instances of the outgoing interface, one in timeslice $t - 1$ and one in timeslice t , and we want each of them fully contained within a clique (not necessarily the same clique). So, we connect together all nodes in I_{t-1} and do the same for the nodes in I_t ; the result is shown in Figure 25(c). Then we triangulate as before, resulting in the model in Figure 25(d). The junction tree is then created as before, but clique potential initialization proceeds slightly differently from the static case. When initializing clique potentials, only cpts of nodes in timeslice 2 of the original 2-TBN are multiplied onto cliques in the junction tree.

Once the junction trees have been constructed and initialized, inference is performed through two stages of message-passing, as before. The cluster containing the outgoing interface in timeslice $t - 1$ is called the *in-clique*, while the cluster containing the outgoing interface in slice t is called the *out-clique*. Once inference has been completed on the junction tree for timeslices $(t - 1, t)$ and the algorithm is ready to advance, the out-clique potential is marginalized down to the outgoing interface potential α , the 2-TBN is “advanced” to timeslices $(t, t + 1)$, and α is multiplied onto the in-clique potential in the new 2-TBN. This procedure is shown in Figure 26. Since we would be repeating the same junction tree construction steps for each timestep, we can simply build the junction tree once and use it for all timesteps for which inference is performed. Thus, when time is incremented in the advance step above, the junction tree is re-initialized to its initial clique potentials.

As stated above, the reason why we only need the out-clique potential for filtering (and no other potentials in the 1.5DBN junction tree) is because the outgoing interface d-separates the past from the future. Clearly, this algorithm will run into difficulty when the outgoing interface contains many discrete nodes. This will require that we perform marginalization and multiplication operations with large potentials whenever time is advanced, which could easily occur at up to 25 Hz given the data rates in real-world applications. So, approximation schemes have been developed to perform faster inference for DBNs. The *Boyen-Koller algorithm*, which we discuss in Section 5.1,

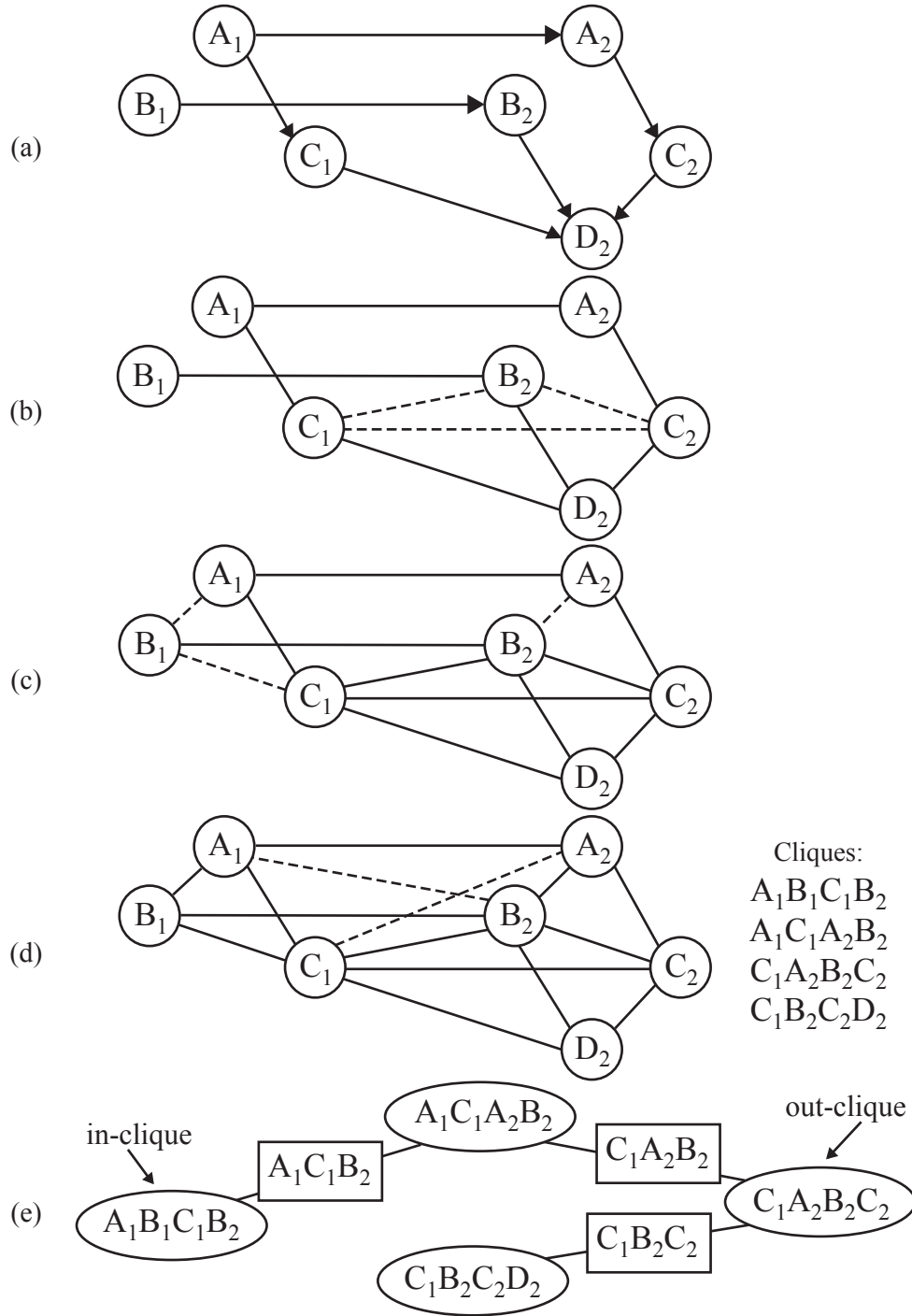
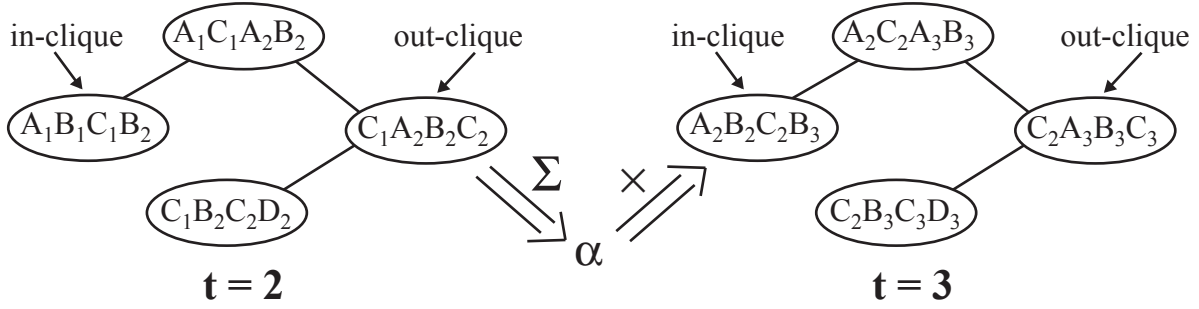


Figure 25. Steps to construct a junction tree J_t from the 2-TBN B_t . (a) Non-outgoing-interface nodes (and their edges) have been removed from timeslice $t-1$ to create a 1.5DBN. (b) The moral graph for the 1.5DBN. (c) All nodes in I_{t-1} must be in a single clique in the junction tree, so we add edges to connect them, and do the same for I_t . (d) Next, we triangulate as before. (e) Finally, a junction tree can be created and inference can be performed as with the static junction tree algorithm.



1. $\alpha = \phi_{A_2B_2C_2} = \sum_{C_1} \phi_{C_1A_2B_2C_2}$
2. time incremented, potentials reset
3. $\phi_{A_2B_2C_2B_3} = \alpha * \phi_{A_2B_2C_2B_3}$

Figure 26. Procedure for advancing timesteps. The potential on the out-clique is marginalized down to the potential on the outgoing interface (which we call α), the 2-TBN is advanced and clique potentials are re-initialized, and α is multiplied onto the potential for the in-clique in the new 2-TBN.

assumes independence among subsets of the outgoing interface in order to factorize the outgoing interface potential into a product of smaller potentials.

4.2 SYMBOLIC PROBABILISTIC INFERENCE

Symbolic probabilistic inference (SPI) [35, 53] algorithms employ various techniques to pare down the inference problem in response to a particular query, while keeping the problem in symbolic terms as long as possible to avoid unnecessary computations. In Section 3 above, we stated that any statistical inference computation on a graphical model can be performed by computing the joint probability distribution of the model and then marginalizing out irrelevant variables. We showed in Section 3.1 how different elimination orderings – also called *factorings* – for marginalizing out the variables can result in different numbers of required computations. Since finding the optimal factoring is NP-hard [2], a fundamental part of typical SPI algorithms involves finding as near-optimal a factoring as possible. *Set factoring* [35] is an algorithm which frames this task as a combinatorial optimization problem called the *optimal factoring problem*.

In Section 4.2.1, we will introduce the optimal factoring problem and define the notation we will use. Then, in Section 4.2.2, we shall present the set factoring algorithm below from [35].

4.2.1 Optimal Factoring Problem

The optimal factoring problem (OFP) uses the following definitions.

Given:

1. \mathbf{X} , a set of m random variables

2. $S = \{S_{\{1\}}, S_{\{2\}}, \dots, S_{\{n\}}\}$, a set of n subsets of \mathbf{X}
3. $\mathbf{Q} \subseteq \mathbf{X}$, a set of query variables

We define the following:

1. Where $I, J \subseteq \{1, 2, \dots, n\}$, $I \cap J = \emptyset$, we define the *combination* $S_{I \cup J}$ of two subsets S_I and S_J as follows:

$$S_{I \cup J} = S_I \cup S_J - \{x : x \notin S_K \text{ for } K \cap I = \emptyset, K \cap J = \emptyset, x \notin \mathbf{Q}\}$$

That is, $S_{I \cup J}$ consists of all variables shared by the subsets except those which are not in any of the other subsets nor among the query variables, i.e., if a variable is only in $S_I \cup S_J$ (and not a query variable), then remove it from $S_I \cup S_J$ when creating the combination $S_{I \cup J}$. These variables can be dropped out because they will not show up in any other subsets, and since they are not query variables, we don't ultimately care about them anyway.

2. We define the *cost function* μ of combining the two subsets:

$$\mu(S_{\{i\}}) = 0, \text{ for } 1 \leq i \leq n, \text{ and}$$

$$\mu(S_{I \cup J}) = \mu(S_I) + \mu(S_J) + 2^{|S_I \cup S_J|}$$

The base in the exponential expression above is the number of possible states in each variable, which we assume to be 2 without loss of generality.

This definition of the cost function is not complete. Suppose $|I| > 2$. Then $\mu(S_I)$ can have multiple breakdowns into its composite subsets, each potentially producing a different cost. So, we have to include an indication of how the subsets in a term are joined together in order to properly evaluate its cost. We denote this combining, or *factoring*, by a subscript α ; therefore, $\mu_\alpha(S_I)$ gives the cost of evaluating S_I using the factoring α . The optimal factoring problem is to find a factoring α such that $\mu_\alpha(S_{\{1,2,\dots,n\}})$ is minimized.

4.2.2 The Set Factoring Algorithm

Efficient factoring algorithms have been found for certain types of Bayesian networks [35], but we will present an approximation algorithm that can be used for any arbitrary discrete Bayesian network. Below, we will use the term *factor* to refer to a set of variables in a Bayesian network and a *marginal factor* for a set containing a single variable. On each iteration of the algorithm, a heuristic is used which combines the pair of factors which will produce the smallest-dimensional table as a result. We do this by testing different combinations to find the number of multiplications needed for each, then pick the one which requires the fewest.

The algorithm proceeds as follows:

1. Initialize two sets A and B for the algorithm. The set A is called the *factor set* and is initially filled with *factors* corresponding to all relevant network distributions, each represented as a

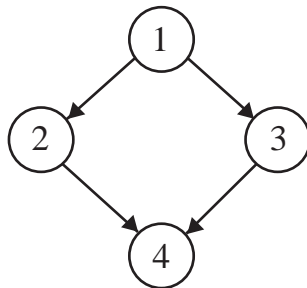


Figure 27. A simple Bayesian network.

set of variables. Throughout the algorithm, A will always contain the set of factors that can be chosen for the next combination. B is called the *combination candidate set*, and is initially empty.

2. Add all pairwise combinations of factors in A to B (as long as they are not already in B), except the combinations in which each factor is marginal and the two factors have no common child. Compute $u = (x \cup y)$ and $sum(u)$ of each pair, where x and y are factors in A and $sum(u)$ is the number of variables in u which can be summed over when the product which includes the combined factors is computed. When x and y are combined, the number of multiplications needed is $2^{|x \cup y|}$.
3. Build a new set C from B such that $C = \{u \in B : minimum_B(|u| - sum(u))\}$. Here, $|u|$ is the size of u with observed nodes removed. This operation favors combinations which have few variables in their union and also contain many variables which can be summed out, i.e., variables which do not appear in any other factors. If C only contains one element, then choose x and y as the factors for the next combination; otherwise, build a new set D from C such that $D = \{u \in C : maximum_C(|x| + |y|), x, y \in u\}$. If D contains only one element, x and y are the terms for the next multiplication; otherwise, choose any one in D . If multiple potential combinations appear in C , we favor those that have the larger number of variables being summed over; it is typically preferred to sum over variables as early as possible.
4. Create a new factor by combining the two factors chosen above in step 3, calling it the *candidate pair*. Delete the two chosen factors from the factor set A and add the candidate pair to A .
5. Delete any factor pair in B that has a non-empty intersection with the candidate pair.
6. Repeat steps 2 to 5 until only one element is left in A ; this element is the factoring.

We will now present an example to illustrate this algorithm. Consider the network in Figure 27. We assume that each variable is binary and we want to compute $p(X_4)$. Initially, we fill A with the elements $\{1, 2, 3, 4\}$ and B is empty¹³.

First loop iteration:

¹³We will abbreviate a variable X_i by just naming it by its index i

After step 2: $A = \{1, 2, 3, 4\}$, $B = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$.

After step 3: Multiple combinations appear in C , so we choose the combination $(1, 2)$ arbitrarily.

After step 4: $A = \{(1, 2), 3, 4\}$, $B = \{(3, 4)\}$.

Second loop iteration: After step 2: $A = \{(1, 2), 3, 4\}$, $B = \{((1, 2), 3), ((1, 2), 4), (3, 4)\}$.

After step 3: We pick $((1, 2), 3)$.

After step 4: $A = \{(1, 2, 3), 4\}$, $B = \{\}$

Third loop iteration: After step 2: $A = \{(1, 2, 3), 4\}$, $B = \{((1, 2, 3), 4)\}$.

After step 3: We pick $((1, 2, 3), 4)$.

After step 4: $A = \{(1, 2, 3, 4)\}$, $B = \{\}$

A has only one element, so we have reached our answer, the factoring $(1, 2, 3, 4)$.

The set factoring algorithm outperforms the simple heuristic of finding the pair with the minimum amount of multiplication required at each step. It is not optimal, however, because it only takes one step in the future into account, whereas the optimal algorithm must consider the entire process.

5. ALGORITHMS FOR APPROXIMATE INFERENCE

For many graphical models used in real-world applications, performing exact statistical inference is infeasible. This could be the case for a variety of reasons. First, it often happens that exact inference on a particular model can not be performed quickly enough to keep up with the data rates in real-world scenarios. This is frequently the case in BMD applications, in which streams of data can be received in real-time from multiple sensors with high data rates. In addition, the need to accurately model physical dynamics necessitates a certain minimum number of nodes and edges in a graphical model. Combining large, densely-connected models with high update rates poses severe challenges to any exact inference algorithm. Consequently, algorithms that obtain approximately-correct results have been developed to address such cases. One class of approximate algorithms consists of those which make certain approximations within the framework of exact inference algorithms in order to perform inference tractably while sacrificing some accuracy. For example, recall that the belief propagation algorithm discussed in Section 3.2 is an exact inference algorithm for acyclic undirected graphical models. The same algorithm has been applied to graphical models with cycles, for which it only obtains approximate results. This algorithm is called *loopy belief propagation* [44]. Another example, the *Boyen-Koller algorithm*, is a simple modification to the junction tree algorithm for dynamic Bayesian networks described in Section 4.1.3 and will be discussed in Section 5.1.

In addition to the constraints imposed by real-time scenarios, there are other situations which require the use of approximate inference algorithms. For example, the presence of particular distributions in a model may prevent the ability to perform the calculations required by any exact inference algorithm even if an unlimited amount of time was available. In particular, models containing continuous, non-Gaussian random variables or non-linear statistical relationships among variables may cause the integrals that arise in belief propagation to be impossible to compute. In these situations, one is forced to make use of approximations in order to perform any calculations of interest. One possibility is to use an approximation for problematic distributions of continuous quantities such that an exact inference algorithm can be used. For example, any continuous random variable can be made discrete through the use of binning, as we discussed in Section 2.3.1, or can be approximated by a Gaussian or other continuous distribution that an exact inference algorithm can handle. This sort of approximation will introduce error, especially in cases of multi-modal distributions, but it will allow us to apply the exact inference algorithms discussed in Section 4 to compute results. If one desires to preserve the continuous densities as modeled, there are a variety of algorithms for dealing with arbitrary continuous distributions in graphical models. An important class of such algorithms includes *particle filters*, a family of sequential Monte Carlo sampling methods which we discuss in Section 5.2. Another important class of approximate algorithms is the family of Markov chain-Monte Carlo (MCMC) methods. We will discuss one MCMC algorithm, the Gibbs Sampler, in Section 5.3. There are several other important families of approximate algorithms that we will not discuss in this report, including *variational methods* [24], *expectation propagation* [40], and *nonparametric belief propagation* (NBP) [59].

In this section, we will describe in detail three approximate inference algorithms. We will discuss the Boyen-Koller algorithm in Section 5.1 and show how it can be used within the context

of the junction tree algorithm for DBNs described in Section 4.1.3. Then, in Section 5.2, we will discuss the canonical particle filtering algorithm for inference in dynamic systems. Finally, in Section 5.3, we introduce the MCMC method and describe a popular algorithm in the MCMC family — the Gibbs sampler.

5.1 THE BOYEN-KOLLER ALGORITHM

The Boyen-Koller (BK) algorithm performs approximate statistical inference in the context of dynamic processes. We will describe the algorithm by only considering processes that can be modeled by dynamic Bayesian networks (DBNs). As discussed in Section 2.4, a DBN provides an efficient representation of the belief state (or filtering distribution) of a process, which is written as $p(\mathbf{x}_t \mid \mathbf{y}_{1:t})$. The Boyen-Koller algorithm approximately computes the belief state at each time by assuming that the process being modeled is composed of independent subprocesses. These independence assumptions allow a reduced number of computations at each time step and provide some impressive bounds for the induced error from truth in the belief state. We will begin by discussing, in the realm of stochastic processes, the intuition behind the BK algorithm and the situations to which it is well-suited. Then, we will describe the algorithm itself by showing specifically how its ideas can be applied in the junction tree algorithm.

Boyen and Koller [4] showed that it is possible to use an approximation for the belief state of a stochastic process such that the belief state’s deviation from truth remains bounded indefinitely over time. In particular, they showed that the rate at which divergence from truth increases actually contracts exponentially over time due to the stochastic nature of the process dynamics and the informative nature of the observations. So, this approximation scheme works best when the dynamic model is structured such that the current state depends as little as possible on previous states. The extreme case occurs when the current state does not depend at all on the state history, i.e., if $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = p(\mathbf{x}_t)$. Intuitively, it makes sense that performance would be best on such a model, since the magnitude of the error incurred in the belief state at one timestep will be dampened or removed entirely by a dynamic model that largely or entirely ignores the previous state. Along the same vein, the approximation error is greatest in situations in which the dynamic model is a deterministic function of the preceding state. With regards to the observation model, the BK algorithm works best when the observations are produced by a noiseless, deterministic function of the state, but has more trouble in situations in which observations are either absent or uninformative. Boyen and Koller also showed how to use their approximation scheme for inference in DBNs in the context of the junction tree algorithm (Section 4.1.3), and it is this technique that we will describe here as an approximate inference algorithm for DBNs.

The BK algorithm for DBNs approximates the outgoing interface potential at each timestep by breaking up the outgoing interface into sets of nodes and assuming that these sets of variables are independent. The breakdown of these sets, which are called *BK clusters*, is specified as an input to the algorithm. With these independence assumptions, the joint distribution on the outgoing interface can be decomposed into the product of the joint distributions on the BK clusters. The clusters must be disjoint sets and their union must form the original outgoing interface, i.e., together they must form a *partition* of the outgoing interface. Also, for best results, no cluster should “affect”

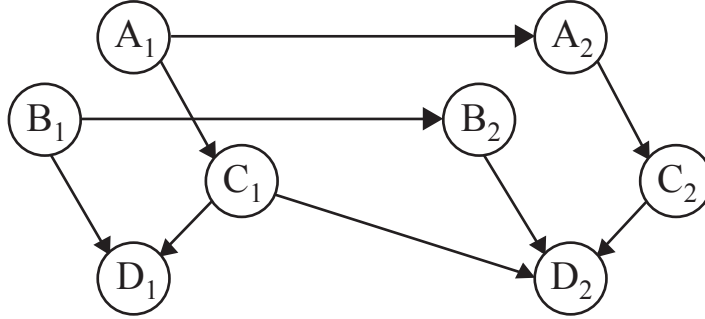


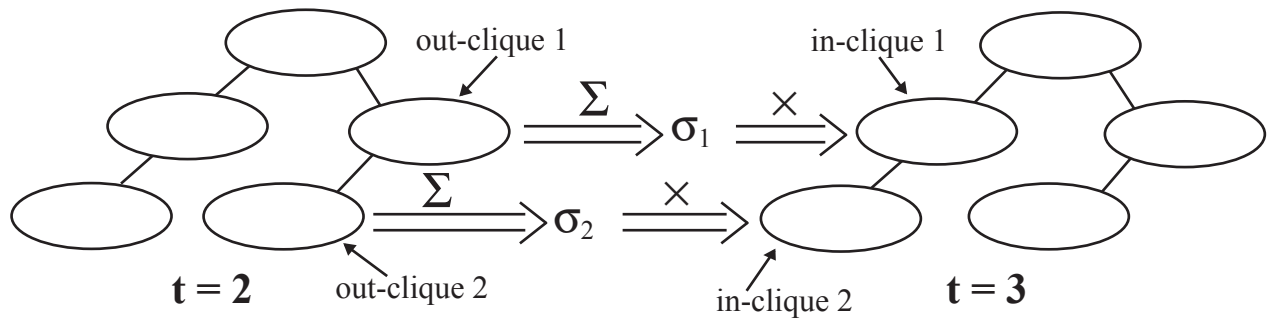
Figure 28. The 2-TBN we used to demonstrate execution of the junction tree algorithm for DBNs. The outgoing interface for timeslice $t - 1$ is $I_{t-1} = \{A, B, C\}$ and a possible set of BK clusters is $\{\{A, C\}, \{B\}\}$.

any other cluster within the same timeslice. That is, within a timeslice, no node in any one cluster should have as parent a node in a different cluster. For example, the outgoing interface in the 2-TBN in Figure 28 can be factored using two BK clusters: $\{A, C\}$ and $\{B\}$. If the BK clusters are all singleton sets, we obtain the *fully factorized* approximation, the most aggressive approximation possible. Conversely, if we use just one BK cluster which contains all the nodes in the outgoing interface, we will obtain the same results as the junction tree algorithm for DBNs from Section 4.1.3. So, the junction tree algorithm is simply a special case of the BK algorithm with the choice of BK clusters that gives the exact solution, i.e., with no extra independence assumptions made.

Inference within a timeslice remains unchanged, but junction tree creation and the procedure for advancing timesteps have small modifications from the standard dynamic junction tree algorithm described in Section 4.1.3. Previously while creating junction trees, we added edges to ensure that the outgoing interfaces in each timeslice were cliques. With the BK algorithm, we effectively have multiple outgoing interfaces in the form of the separate BK clusters, so we must add edges to ensure that each BK cluster is a clique. When advancing timesteps, we follow the procedure shown in Figure 29. In a departure from standard dynamic junction tree inference, we now have multiple out-cliques (one for each BK cluster), so the out-clique potentials are marginalized down to their respective BK cluster potentials, stored in a vector σ , the 2-TBN is “advanced” to timeslices $(t, t + 1)$ and potentials are re-initialized as before, and the elements of σ are multiplied onto the appropriate in-clique potentials in the new 2-TBN.

5.2 PARTICLE FILTERING

5.3 GIBBS SAMPLING



1. $\sigma_1 = \sum \phi_{\text{out-clique 1}}$
 $\sigma_2 = \sum \phi_{\text{out-clique 2}}$
2. time incremented, potentials reset
3. $\phi_{\text{in-clique 1}} = \sigma_1 * \phi_{\text{in-clique 1}}$
 $\phi_{\text{in-clique 2}} = \sigma_2 * \phi_{\text{in-clique 2}}$

Figure 29. Procedure for advancing timesteps in the Boyen-Koller algorithm. The potentials on the out-cliques are marginalized down to the potentials on the BK clusters (we call this potential vector σ), the 2-TBN is advanced, and the elements of σ are multiplied onto their respective in-cliques in the new 2-TBN.

APPENDIX A

A.1 INDEPENDENCE \Leftrightarrow CONDITIONAL INDEPENDENCE

A.1.1 Independence \Rightarrow Conditional Independence

Independence does not generally imply conditional independence, as shown by the following counter-example.

Example A.1. Let X and Y be independent Bernoulli random variables with parameters $\frac{3}{4}$ and $\frac{1}{2}$, respectively.¹⁴ By independence, $P_{X,Y}(1,1) = P_X(1)P_Y(1) = \frac{3}{8}$. We introduce a third Bernoulli random variable Z with parameter $p = \frac{1}{3}$ and we let $P(X=1 | Z=1) = P(X=1 | Z=0) = P(Y=1 | Z=1) = \frac{3}{4}$ and $P(Y=1 | Z=0) = \frac{3}{8}$. It can easily be verified that these conditional probabilities are consistent with our initial point-mass function definitions for X and Y . Furthermore, we define $P(X=1, Y=1 | Z=1) = \frac{5}{8}$ and $P(X=1, Y=1 | Z=0) = \frac{1}{4}$, which are consistent with our original $P_{X,Y}(1,1)$. Now, when we test to see if $X \perp\!\!\!\perp Y | Z$, we find that X and Y are not conditionally independent given Z :

$$P(X=1, Y=1 | Z=1) = \frac{5}{8} \neq \frac{3}{4} \times \frac{3}{4} = P(X=1 | Z=1)P(Y=1 | Z=1).$$

A.1.2 Conditional Independence \Rightarrow Independence

Conditional independence does not necessarily imply independence, as the following counter-example shows.

Example A.2. Let X , Y , and Z be Bernoulli random variables such that $X \perp\!\!\!\perp Y | Z$. We define Z with the parameter $\frac{1}{2}$, and we let $P(X=1 | Z=1) = \frac{1}{3}$, $P(X=1 | Z=0) = \frac{1}{2}$, $P(Y=1 | Z=1) = \frac{1}{2}$, and $P(Y=1 | Z=0) = \frac{1}{3}$. By conditional independence, $P(X=1, Y=1 | Z=1) = P(X=1 | Z=1)P(Y=1 | Z=1) = \frac{1}{6}$ and, similarly, $P(X=1, Y=1 | Z=0) = \frac{1}{6}$. Given these probabilities, we can derive that $P_X(1) = P_Y(1) = \frac{5}{12}$ as well as $P(X=1 | Y=1) = P(X=1, Y=1 | Z=1)P_Z(1) + P(X=1, Y=1 | Z=0)P_Z(0) = \frac{1}{6}$. So, we have

$$P(X=1, Y=1) = \frac{1}{6} \neq \frac{5}{12} \times \frac{5}{12} = P_X(1)P_Y(1),$$

and, therefore, X and Y are not independent.

A.2 DERIVATION OF PAIRWISE FACTORIZATION FOR ACYCLIC UNDIRECTED GRAPHICAL MODELS

If the undirected graph associated with an undirected graphical model is connected and acyclic, we can derive a general form for the factorization of the joint probability explicitly in

¹⁴A Bernoulli random variable X takes values from $\{0,1\}$ and has a parameter p which equals $P_X(1)$. Since $P_X(0) = 1 - P_X(1)$, p is sufficient to define the point-mass function of X .

terms of joints and marginals of the nodes in the model. This factorization provides us with the vocabulary to specify the clique potentials in an acyclic model entirely in terms of probability distributions. To do so, we shall decompose the joint distribution into a product of conditional distributions, making use of conditional independencies in the model to do so in as compact a way as possible.

We begin by initializing our decomposition π to 1 and initializing copies \mathcal{V}' and \mathcal{E}' of the vertex and edge sets, respectively, as follows: $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' = \mathcal{E}$.

Then, we repeat the following two steps until $\mathcal{E}' = \emptyset$:

1. Choose a node $s \in \mathcal{V}'$ that has only one neighbor t and set $\pi = \pi * P(s | t)$, then remove s from the graph along with its only edge (s, t) (i.e., set $\mathcal{V}' = \mathcal{V}' - \{s\}$ and $\mathcal{E}' = \mathcal{E}' - \{(s, t)\}$).¹⁵
2. Add the removed edge as an ordered pair (s, t) to a set \mathcal{E}_O , so-named because it is a set of undirected edges in which the order of the two vertices in each edge matters.

Since the number of vertices in a connected, acyclic graph is $1 + q$, where q is the number of edges, there will now be exactly one node left in \mathcal{V}' , which we call u . We then multiply this marginal $P(u)$ onto π , i.e., $\pi = \pi * P(u)$.

Now, π is the product of $P(u)$ and q conditional probabilities, one for each edge in the graph. Rewriting the conditionals as quotients of joints and marginals, we obtain

$$\pi = P(X_u) \prod_{(s,t) \in \mathcal{E}_O} \frac{P(X_s, X_t)}{P(X_t)}. \quad (\text{A-1})$$

Note that this form depends on the edges being represented as ordered pairs, which is not the case for undirected graphs, where the nodes in an edge can be written in either order. Furthermore, we know that there is one term in the above product for each node in the graph, since a term is multiplied onto π for each node that is eliminated from the graph in step (1) above. So, if we multiply by $\prod_{s \in \mathcal{V}} P(X_s) / \prod_{s \in \mathcal{V}} P(X_s)$, we will be able to move the denominator of this identity into the product from equation A-1, giving us the following factorization for the joint distribution $P(\mathbf{X})$:

$$P(\mathbf{X}) = \prod_{(s,t) \in \mathcal{E}} \frac{P(X_s, X_t)}{P(X_s)P(X_t)} \prod_{s \in \mathcal{V}} P(X_s). \quad (\text{A-2})$$

A.3 DERIVATION OF BELIEF PROPAGATION FOR CONTINUOUS NODES WITH EVIDENCE APPLIED

Belief propagation (BP) is the general term for a family of message-passing algorithms for inference in graphical models. In Section 3.2 we showed heuristically how belief propagation arises

¹⁵In general, decomposition of a joint distribution into conditionals would involve multiplying π by $P(s | \mathcal{V}' \setminus s)$ on each iteration instead of $P(s | t)$. In our case, $P(s | \mathcal{V}' \setminus s) = P(s | t)$ due to the Markov properties and the fact that t is the only neighbor of s . Since the graph is acyclic and removing an edge on each iteration will never cause it to become cyclic, we will always be able to choose a node s with only one neighbor.

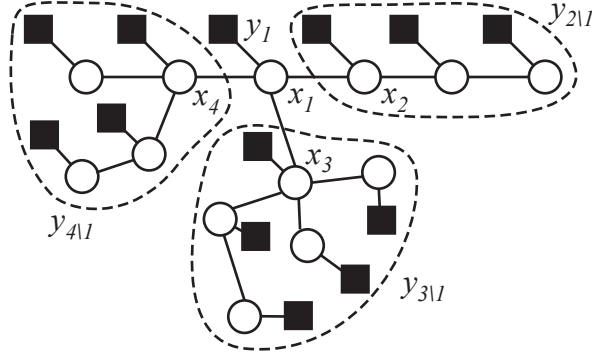


Figure A-1. An undirected graphical model with observation nodes shown (Adapted from [58]).

from the ordinary action of pushing sums over products in variable elimination. However, we only considered the case of performing inference in the case of no evidence being applied and all nodes being discrete. Below, while we still only consider acyclic graphical models, we show how to explicitly handle evidence and additionally allow all nodes to be either continuous or discrete (essentially just by changing sums to integrals). BP computes all N marginal distributions of a graphical model in $O(N)$ time, using a series of local, recursive message passes between nodes.

Where \mathbf{X} is the set of variables in the network, we let \mathbf{Y} denote the set of observed values for the corresponding variables, i.e., Y_i is the observed value for X_i . Our goal is to compute $P(X_s | \mathbf{Y})$ for all $s \in \mathcal{V}$. For the purposes of describing the BP algorithm, we will include the observation nodes graphically when drawing an undirected graphical model, as shown in Figure A-1. We assume that the observations are pairwise conditionally independent given \mathbf{X} and that, conditioned on X_i , an observation Y_i is independent of the rest of the nodes in \mathbf{X} , giving us $P(\mathbf{Y} | \mathbf{X}) = \prod_{s \in \mathcal{V}} P(Y_s | X_s)$. For any $s \in \mathcal{V}$ and any $t \in N(s)$, we define $\mathbf{Y}_{s \setminus t}$ to be the set of all observation nodes in the tree rooted at node s , excluding those in the subtree rooted at node t . Figure A-1 shows examples of this definition.

Our derivation of the belief propagation algorithm will follow Sudderth [58]. We begin with the quantity in which we are interested, $P(X_s | \mathbf{Y})$ for some $s \in \mathcal{V}$. Using Bayes' rule and the fact that the observations \mathbf{Y} are conditionally independent given X_s , we obtain

$$P(X_s | \mathbf{Y}) = \frac{P(\mathbf{Y} | X_s)P(X_s)}{P(\mathbf{Y})} = \alpha P(X_s)P(Y_s | X_s) \prod_{t \in N(s)} P(\mathbf{Y}_{t \setminus s} | X_s). \quad (\text{A-3})$$

From equation A-3 we can begin to see how inference can be reduced to a series of message passes by noting that, for each neighbor t of s , $P(\mathbf{Y}_{t \setminus s} | X_s)$ encapsulates all the necessary information about t and its subtree to compute the marginal $P(X_s | \mathbf{Y})$. The quantity $P(\mathbf{Y}_{t \setminus s} | X_s)$ can be interpreted as a “message” that effectively gets “sent” from X_t to X_s . We will now show how these messages can be expressed in terms of their neighboring nodes, and in doing so, show that this message passing can be a local, recursive series of operations:

$$P(\mathbf{Y}_{t \setminus s} | X_s) = \alpha \int_{X_t} \frac{P(X_s, X_t)}{P(X_s)P(X_t)} P(X_t)P(Y_t | X_t) \prod_{u \in N(t) \setminus s} P(\mathbf{Y}_{u \setminus t} | X_t) dX_t. \quad (\text{A-4})$$

We can see from this equation the appearance of the terms that form the factorization of an acyclic undirected graphical model (Equation (3)). If we use (A-4) repeatedly to "unroll" (A-3), we obtain

$$P(X_u | \mathbf{Y}) = \alpha \int_{\mathbf{X}_{\mathcal{V} \setminus u}} \prod_{(s,t) \in \mathcal{E}} \frac{P(X_s, X_t)}{P(X_s)P(X_t)} \prod_{s \in \mathcal{V}} P(X_s)P(Y_s | X_s) d\mathbf{X}_{\mathcal{V} \setminus u}. \quad (\text{A-5})$$

Using Equation (3), we can see that this decomposition of the marginal $P(X_u | \mathbf{Y})$ is equivalent to computing the joint distribution of the network given the observations and then marginalizing out all variables except X_u . Reproducing A-5 below:

$$\begin{aligned} P(X_u | \mathbf{Y}) &= \alpha \int_{\mathbf{X}_{\mathcal{V} \setminus u}} \prod_{(s,t) \in \mathcal{E}} \frac{P(X_s, X_t)}{P(X_s)P(X_t)} \prod_{s \in \mathcal{V}} P(X_s)P(Y_s | X_s) d\mathbf{X}_{\mathcal{V} \setminus u} \\ &= \alpha \int_{\mathbf{X}_{\mathcal{V} \setminus u}} \prod_{(s,t) \in \mathcal{E}} \psi_{s,t}(X_s, X_t) \prod_{s \in \mathcal{V}} P(Y_s | X_s) d\mathbf{X}_{\mathcal{V} \setminus u} \\ &= \alpha \int_{\mathbf{X}_{\mathcal{V} \setminus u}} P(\mathbf{X})P(\mathbf{Y} | \mathbf{X}) d\mathbf{X}_{\mathcal{V} \setminus u} = \alpha \int_{\mathbf{X}_{\mathcal{V} \setminus u}} P(\mathbf{X} | \mathbf{Y}) d\mathbf{X}_{\mathcal{V} \setminus u} = P(X_u | \mathbf{Y}). \end{aligned}$$

Thus, through equations (A-3) and (A-4), we have worked out a scheme of local, recursive message-passing from the ordinary approach of marginalizing out unwanted variables from the joint distribution represented by the model. We can generalize (A-5) to the potential function factorization by using equation (4) and through some straightforward manipulation obtain the following two equations which comprise the belief propagation algorithm:

$$P(X_s | \mathbf{Y}) = \alpha P(Y_s | X_s) \prod_{t \in N(s)} m_{ts}(X_s) \quad (\text{A-6})$$

$$m_{ts}(X_s) = \alpha \int_{X_t} \psi_{s,t}(X_s, X_t) P(Y_t | X_t) \prod_{u \in N(t) \setminus s} m_{ut}(X_t) dX_t \quad (\text{A-7})$$

In these equations, we have replaced the probabilistic term $P(\mathbf{Y}_{t \setminus s} | X_s)$ with $m_{ts}(X_s)$ to show that this quantity is the message that gets passed from X_t to X_s .

As suggested by the equivalence to computation of the full joint followed by marginalization, BP will not provide us any gain in efficiency if we perform the message passing in an inefficient manner. Instead, we use dynamic programming to obtain all N marginals through only two stages of message passing, called COLLECT-TO-ROOT and DISTRIBUTE-FROM-ROOT and shown in Figure A-2(a) and (b), respectively. Each stage takes $O(N)$ time, making the entire belief propagation algorithm $O(N)$. One node is arbitrarily chosen as the root node and, in the first stage, all other nodes send messages to their neighbors towards the root, starting with the leaf nodes. A node only computes and sends its message after receiving messages from all but one of its neighbors, and then sends its message to this neighbor. In the second stage, the root begins the process by sending messages to all of its neighbors, which in turn send messages to each of their neighbors, etc. When message passing is completed, the marginals of all nodes in the network have been computed.

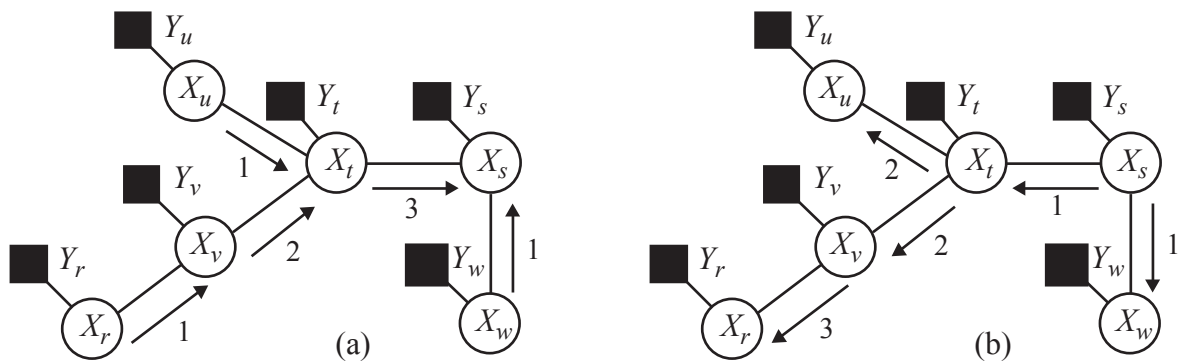


Figure A-2. (a) COLLECT-TO-ROOT, the first series of message passes, where X_s is designated (arbitrarily) as the root node. The numbers indicate the order of message passes. If two have the same number, it means that they can occur at the same time. (b) DISTRIBUTE-FROM-ROOT, the second series of message passing.

REFERENCES

- [1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.
- [2] S. Arnborg, D.G. Corneil, and A. Proskurowski. Complexity of finding embedding in a k-tree. *SIAM Journal of Algebraic and Discrete Methods*, 8(2):177–184, 1987.
- [3] A. Blake, M. Isard, and D. Reynard. Learning to track the visual motion of contours. *Artificial Intelligence*, 78(1-2):179–212, 1995.
- [4] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI)*, pages 33–42, 1998.
- [5] Jessica Bradley. *BNET Quick Start User’s Guide*. Decisive Analytics Corporation, Rosslyn, VA, 2005.
- [6] W. L. Buntine. Learning with graphical models. Technical Report FIA-94-02, NASA Ames Research Center, 1994.
- [7] E. Castillo, J. Gutierrez, and A. Hadi. *Expert Systems and Probabilistic Network Models*. Springer-Verlag, New York, 1996.
- [8] B. R. Cobb and P. P. Shenoy. Inference in hybrid Bayesian networks with mixtures of truncated exponentials. In *6th Workshop on Uncertainty Processing*, pages 47–63, Hejnice, Czech Republic, 2003.
- [9] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2-3):393–405, 1990.
- [10] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York, 1999.
- [11] P. Dagum and A. Galper. Forecasting sleep apnea with dynamic network models. In *Proceedings of the 9th Annual Conference on Uncertainty in AI (UAI)*, pages 64–71, 1995.
- [12] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.
- [13] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Annual Conference on Uncertainty in AI (UAI)*, pages 211–219, 1996.
- [14] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, New York, 2001.
- [15] P. W. Fieguth, W. C. Karl, and A. S. Willsky. Efficient multiresolution counterparts to variational methods for surface reconstruction. *Computer Vision and Image Understanding: CVIU*, 70(2):157–176, 1998.

- [16] R. G. Gallager. Low-density parity-check codes. 1963.
- [17] W.R. Gilks and C. Berzuini. Following a moving target – Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society Series B*, 63(1):127–146, 2001.
- [18] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.
- [19] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Bayesian state estimation for tracking and guidance using the bootstrap filter. *Journal of Guidance, Control and Dynamics*, 18(6):1434–1443, 1995.
- [20] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.
- [21] F. V. Jensen. *Statistics for Engineering and Information Science*. Springer-Verlag, New York, 2001.
- [22] Finn V. Jensen and Frank Jensen. Optimal junction trees. In *Proceedings of the 10th Annual Conference on Uncertainty in AI (UAI)*, pages 360–366. Morgan Kaufmann, 1994.
- [23] M. I. Jordan and Y. Weiss. *The handbook of brain theory and neural networks*, 2002.
- [24] Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.
- [25] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(D):35–45, 1960.
- [26] R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Transactions of the ASME–Journal of Basic Engineering*, 83(D):95–108, 1961.
- [27] U. Kjaerulff. Triangulation of graphs - algorithms giving small total state space. Technical Report R-90-09, Dept. of Math and Comp. Sci., Aalborg University, Denmark, 1990.
- [28] D. Koller and U. Lerner. *Sequential Monte Carlo Methods in Practice*, chapter Sampling in Factored Dynamic Systems, pages 445–464. Springer, 2001.
- [29] D. Koller, U. Lerner, and D. Angelov. A general algorithm for approximate inference and its application to hybrid Bayes nets. In *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI)*, pages 324–333, 1999.
- [30] Daphne Koller and Avi Pfeffer. Object-oriented bayesian networks. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313, 1997.
- [31] A. Krogh, M. Brown, I. S. Mian, K. Sjlander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

- [32] S. L. Lauritzen. Propagation of probabilities, means and variances in mixed graphical association models. *Journal of the American Statistical Association*, 87:1098–1108, 1992.
- [33] S. L. Lauritzen and Frank Jensen. Stable local computation with conditional Gaussian distributions. *Statistics and Computing*, 11(2):191–203, 2001.
- [34] S.L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.
- [35] Z. Li and B. D’Ambrosio. Efficient inference in Bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1):55–81, 1994.
- [36] J.S. Liu, W.H. Wong, and A. Kong. Covariance structure of the Gibbs sampler with applications to the comparisons of estimators and augmentation schemes. *Biometrika*, 81(1):27–40, 1994.
- [37] D. Madigan and J. York. Bayesian graphical models for discrete data. *International Statistical Review*, 63:215–232, 1995.
- [38] Anders L. Madsen and Finn V. Jensen. Lazy propagation in junction trees. In *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI)*, pages 362–369.
- [39] Suzanne M. Mahoney and Kathryn B. Laskey. Network engineering for complex belief networks. In *Proceedings of the 12th Annual Conference on Uncertainty in AI (UAI)*, pages 389–396, San Francisco, CA, 1996. Morgan Kaufmann.
- [40] T. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Ph.D. thesis, MIT Media Lab, 2001.
- [41] S. Moral, R. Rumi, and A. Salmeron. Mixtures of truncated exponentials in hybrid Bayesian networks. In P. Besnard and S. Benferhart, editors, *Sixth European Conference on Symbolic and Quantitative Approaches to Reasoning under Uncertainty*, pages 156–167, 2001.
- [42] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Ph.D. thesis, U.C. Berkeley, 2002.
- [43] Kevin Murphy. Software packages for graphical models / Bayesian networks, 2005.
- [44] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI)*, pages 467–475, 1999.
- [45] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.
- [46] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–285, 1989.

- [47] L. R. Rabiner and Biing-Hwang Juang. An introduction to hidden Markov models. *IEEE Acoustics, Speech, and Signal Processing Magazine*, 3(1):4–16, 1986.
- [48] B. Ristic, S. Arulampalam, and N. Gordon. *Beyond the Kalman Filter - Particle Filters for Tracking Applications*. Artech House, Boston, MA, 2004.
- [49] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer-Verlag, 2004.
- [50] D. Rudoy. Project Hercules Decision Architecture development DRAFT (SECRET). Technical Report PR DA-01, MIT Lincoln Laboratory, 2006.
- [51] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, second edition, 2003.
- [52] R. D. Shachter. Bayes-Ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI)*, pages 480–487, San Francisco, CA, 1998. Morgan Kaufmann.
- [53] R. D. Shachter, B. D’Ambrosio, and B. Del Favero. Symbolic probabilistic inference in belief networks. In *Proceedings of the 8th National Conference on AI*, pages 126–131, 1990.
- [54] R. D. Shachter and C. R. Kenley. Gaussian influence diagrams. *Management Science*, 35:527–550, 1989.
- [55] G. Shafer and P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, (2):327–357, 1990.
- [56] M. Shwe, B. Middleton, D. Heckerman, M. Henrion, E. Horvitz, H. Lehmann, and G. Cooper. Probabilistic diagnosis using a reformulation of the INTERNIST-1/QMR Knowledge Base I. the probabilistic model and inference algorithms. *Methods of Information in Medicine*, 30:241–255, 1991.
- [57] P. Smyth. Belief networks, hidden Markov models, and Markov random fields: a unifying view, 1998.
- [58] E. Sudderth. *Embedded trees: Estimation of Gaussian processes on graphs with cycles*. PhD thesis, SM thesis, MIT Dept. of EECS, 2002.
- [59] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. Technical Report LIDS 2551, MIT, 2002.
- [60] Norsys Software Corp. (www.norsys.com). Netica, 1996-2005.
- [61] Y. Xiang and F. V Jensen. Inference in multiply sectioned Bayesian networks with extended Shafer-Shenoy and lazy propagation. pages 680–687.
- [62] Y. Xiang, D. Poole, and M. P. Beddoes. Exploring localization in Bayesian networks for large expert systems. In *Proceedings of the 8th Annual Conference on Uncertainty in AI (UAI)*, pages 344–351, Stanford, CA, 1992. Morgan Kaufmann.

- [63] Y. Xiang, D. Poole, and M. P. Beddoes. Multiply Sectioned Bayesian Networks and junction forests for large knowledge based systems. *Computational Intelligence*, 9(2):171–220, 1993.
- [64] M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal of Algebraic and Discrete Methods*, 2(1):77–79, 1981.
- [65] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. In *NIPS*, pages 689–695, 2000.