An Overview of Dynamic Graphical Models

Jeff Bilmes
bilmes@ee.washington.edu
University of Washington, Seattle
Department of Electrical Engineering
Seattle WA, 98195-2500

1 Introduction

A graphical model consists of a graph $G = (V, E)$ and a set of properties that determine a family of probability distributions. There are many different types of graphs and properties, each determining a family. It is common to be able to develop algorithms that work for all members of the family by considering only a graph and its properties. Thus, solving difficult problems (such as deriving an approximation to an NP-complete optimization problem) might become worthwhile only because a solution can be applied many times for different problem instances.

Given a probability distribution $p(x_V)$ over a set of random variables $X_V = \{X_{v_1}, X_{v_2}, \ldots, X_{v_N}\}$, where $N = |V|$, one often wishes to compute quantities of the form $p(x_S|x_U)$ where $S, U \subseteq V$ are disjoint. Such computations are known as *probabilistic inference*, or sometimes just *inference*. If it is known that p is a member of a graphical model's family, and if we already have an algorithm for any member of this family, we then have a solution for $p(x_S|x_U)$. Computing such probabilistic quantities is the core of many important statistical inference and machine learning algorithms and their applications.

This paper discusses a form of graphical model that is amenable to sequential data, including speech and language strings, and biological and economic series. At the heart of dynamic models lie ''static'' graphical models so we will briefly review these first.

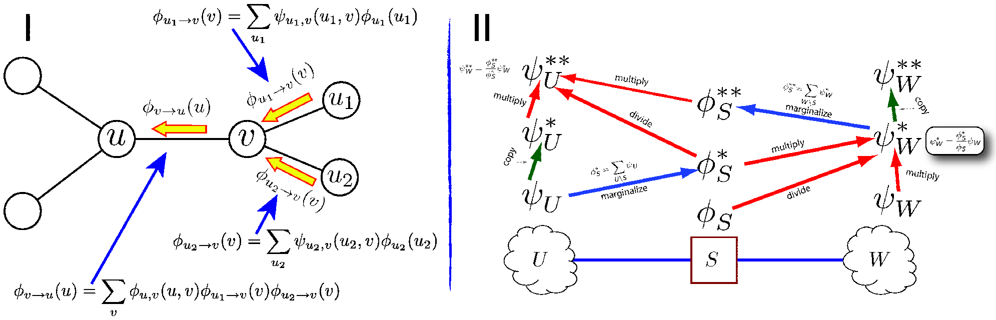

I
$\phi_{u_1\to v}(v) = \sum_{u_1} \psi_{u_1,v}(u_1,v)\phi_{u_1}(u_1)$
$\phi_{v\to u}(u)$
$\phi_{u_1\to v}(v)$
u
v
u_1
u_2
$\phi_{u_2\to v}(v)$
$\phi_{u_2\to v}(v) = \sum_{u_2} \psi_{u_2,v}(u_2,v)\phi_{u_2}(u_2)$
$\phi_{v\to u}(u) = \sum_{v} \phi_{u,v}(u,v)\phi_{u_1\to v}(v)\phi_{u_2\to v}(v)$
II
ψ_U^{**}
ψ_U^{*}
ψ_U
ϕ_S^{**}
ϕ_S^{*}
ϕ_S
ψ_W^{**}
ψ_W^{*}
ψ_W
multiply
divide
marginalize
copy
U
S
W

Figure 1: I: standard belief propagation messages on a tree. Each yellow arrow corresponds to a message, which is defined by the associated equation. II: Messages sent on a clustered graph, where each cloud shape is a cluster of variables in the original graph, and the messages are in the Hugin [10] style. The clique potentials ψ_U and ψ_W are initialized to the product of factors that are contained within cliques U and W respectively. The separator potential ϕ_S is initialized to unity. The messages are shown in data-flow fashion, so that the arrows show message dependency. Green arrows are message copies. Blue arrows are marginalization operations, and red arrows are message multiplies or divides. The cost of this process is exponential in the size of the cluster. More details may be found in [10, 13, 11].

2 Static Graphical Models

Let $p = p(x_1, x_2, \ldots, x_N)$ be a probability distribution over $N = |V|$ random variables and let $\mathcal{U}$ be the set of all such distributions. A graphical model consists of a graph $G = (V, E)$ and a set of properties $\mathcal{M}$ (often called *Markov properties* [16]) that together determine a sub-family $\mathcal{F}(G, \mathcal{M}) \subseteq \mathcal{U}$ of probability distributions. There are many types of graphical model (such as Markov random fields (MRF) or Bayesian networks (BN)). The type of graphical model is determined by both the set of allowable graphs and the set of properties.

One such simple graphical model is the Markov random field (MRF), where any undirected graph over N vertices is allowed (one vertex for each random variable), and p is a member of the family whenever p's conditional independence properties obey the separation properties $\mathcal{M}^{\text{sep}}$ in the graph. Given three sets of graph vertices $A, B, C \subset V(G)$, it is said that A is separated from B by C in G if all paths (i.e., sequences of adjacent vertices) from any node in A to any node in B must intersect some node in C, and in such case C is called a *separator*. If C is such a separator, then for any $p \in \mathcal{F}(G, \mathcal{M}^{\text{sep}})$ we must have that $p(x_A, x_B|x_C) = p(x_A|x_C)p(x_B|x_C)$ for all values x_A, x_B, x_C. This conditional independence property is denoted $X_A \perp\!\!\!\perp X_B | X_C$. Any $p \in \mathcal{F}(G, \mathcal{M}^{\text{sep}})$ must satisfy the conditional independence properties corresponding to all separation properties in G. Another way to characterize MRFs is via factorization properties $\mathcal{M}^{\text{fac}}$ [16]. We have that $p \in \mathcal{F}(G, \mathcal{M}^{\text{fac}})$ if p factorizes with respect to a set of factors, $\{\phi(\cdot)\}$, known as *potential functions*, defined on the maximal cliques[1] of G. It is always the case that $\mathcal{F}(G, \mathcal{M}^{\text{fac}}) \subseteq \mathcal{F}(G, \mathcal{M}^{\text{sep}})$ but the reverse is true only for strictly positive p [16]. An example of separation and factorization properties is depicted in Figure 2-left.

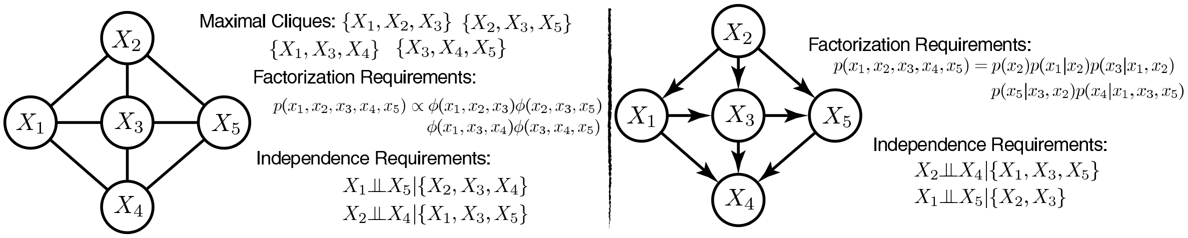

X_1
X_2
X_3
X_4
X_5
Maximal Cliques: $\{X_1, X_2, X_3\}$ $\{X_2, X_3, X_5\}$
$\{X_1, X_3, X_4\}$ $\{X_3, X_4, X_5\}$
Factorization Requirements:
$p(x_1, x_2, x_3, x_4, x_5) \propto \phi(x_1, x_2, x_3)\phi(x_2, x_3, x_5)$
$\phi(x_1, x_3, x_4)\phi(x_3, x_4, x_5)$
Independence Requirements:
$X_1 \perp\!\!\!\perp X_5 | \{X_2, X_3, X_4\}$
$X_2 \perp\!\!\!\perp X_4 | \{X_1, X_3, X_5\}$
X_1
X_2
X_3
X_4
X_5
Factorization Requirements:
$p(x_1, x_2, x_3, x_4, x_5) = p(x_2)p(x_1|x_2)p(x_3|x_1, x_2)$
$p(x_5|x_3, x_2)p(x_4|x_1, x_3, x_5)$
Independence Requirements:
$X_2 \perp\!\!\!\perp X_4 | \{X_1, X_3, X_5\}$
$X_1 \perp\!\!\!\perp X_5 | \{X_2, X_3\}$

Figure 2: Left: A 4-cycle with an extra node. There are two separation (and thus conditional independence) properties. There are four maximal cliques and thus a minimum of four factors. Any p in the graph's family must obey these independence properties, and must also validly factorize as shown. Right: The edges are now directed as in a BN. The required factorization properties, as well as the conditional independence properties, are shown. Notice how the required properties have changed relative to the undirected (left) case.

A Bayesian network (BN) is another type of graphical model, where G is always directed and acyclic (a DAG). All edges are said to point from *parent* to *child*. Like before, a BN corresponds to a family of probability distributions $\mathcal{F}(G, \mathcal{M}^{\text{bn}})$ based on G and a set of properties $\mathcal{M}^{\text{bn}}$ specific to BNs. BN properties are a bit more complicated than those of MRFs, but the essential concept is the same: if there is a form of separation encoded in the graph, then there must be a corresponding independence property. In the BN case, separation can be based on d-separation [23, 16]. There is also a corresponding directed factorization property $\mathcal{M}^{\text{dfac}}$, namely: if x_i is a variable, and π_i is the set of parents of i, then if $p \in \mathcal{F}(G, \mathcal{M}^{\text{dfac}})$, we may factor p as $p(x) = \prod_i p(x_i | x_{\pi_i})$. An example is depicted in Figure 2-right.

A benefit of graphical models is that it is possible to automatically develop algorithms that can compute probabilistic quantities of interest (exactly or approximately) based on knowledge only of $(G, \mathcal{M})$ and without having knowledge of a particular instance $p \in \mathcal{F}(G, \mathcal{M})$. Given such an algorithm, it is then applicable to any member of the family, thereby amortizing the algorithm's development cost over all $p \in \mathcal{F}(G, \mathcal{M})$. The algorithm can of course also be applied to any $p \in \mathcal{F}(G', \mathcal{M}')$ where $\mathcal{F}(G', \mathcal{M}') \subseteq \mathcal{F}(G, \mathcal{M})$. One simple such algorithm for computing probabilistic quantities exactly is belief propagation and its generalization, the junction tree algorithm. We give a brief overview of this algorithm here. First, the algorithm operates on

---

[1] A clique is a fully connected set of vertices, and with a maximal clique, the vertex set is no longer fully connected if any additional vertex is added.

MRFs rather than directly on BNs. Therefore, for any BN, it must be possible to find a MRF that includes all members of its family. That is, given that G is a directed graph, we need an operation (let's call it $m(G)$) that transforms the BN G into an MRF $m(G)$ such that $\mathcal{F}(G, \mathcal{M}^{\text{bn}}) \subseteq \mathcal{F}(m(G), \mathcal{M}^{\text{sep}})$ holds. Then, any inference algorithm developed for $\mathcal{F}(m(G), \mathcal{M}^{\text{sep}})$ will be valid for any member of $\mathcal{F}(G, \mathcal{M}^{\text{bn}})$. We also wish $m(G)$ to be minimal, in that performing inference for a member of $\mathcal{F}(m(G), \mathcal{M}^{\text{sep}})$ should not be significantly more costly than inference for members of $\mathcal{F}(G, \mathcal{M}^{\text{bn}})$. The operation $m(G)$ (called moralization) is a simple graph-theoretic operation of converting a DAG into an undirected graph: connect with an undirected edge the unconnected parents of any child, and then drop all remaining edge directions. Henceforth, we assume the graph is an MRF.

If the graph G is a tree (meaning, all pairs of nodes are connected by a unique path), then we are guaranteed that there are at least two random variables (i.e., leaf nodes) that have the property that they are involved in a factor with only one additional node. Such random variables can then be marginalized away (the corresponding nodes have been *eliminated*, and the graph-theoretic counterpart of belief propagation is known as the *elimination algorithm*). This leaves us with a residual graph that is still a tree, meaning that once again there must be at least two nodes with the desired leaf property. Performing such marginalizations repeatedly, but only on leaf nodes, ensures that computing any node marginal, or any marginal along two neighboring nodes connected by a tree edge, is computationally inexpensive. Each such marginalization can be seen as a message passed between two nodes along a connecting edge. The message passing equations are summarized in Figure 1-I. Thus we can see that marginalizing away all nodes corresponds to sending messages starting at the leaves, and towards a designated root of the tree. In fact, these marginalization messages can be passed along both directions of each edge, and since there are $N-1$ edges, after $2(N-1)$ messages the state of the tree will reach the point where each node (and edge) holds the marginal distribution for that node (edge). This is sufficient for the majority of probabilistic queries needed on a tree.

If the graph $G = (V, E)$ is not a tree, then we can make use of a generalization of belief propagation known as the *junction tree algorithm*. A junction tree is a particular kind of tree in which the nodes are overlapping clusters of the original nodes V. Let $\mathcal{T} = (\mathcal{C}, \mathcal{E})$ be this tree, where for each $C \in \mathcal{C}$, $C \subseteq V$. The edge set $\mathcal{E}$ is such that $\mathcal{T}$ is a tree with subsets of V as nodes. The tree of clusters can then be treated exactly like the tree of nodes discussed in our presentation of belief propagation, where we eliminate leaf nodes by marginalizing them away, thereby producing messages that are passed on the junction tree. In this case, however, marginalizing away a leaf node means summing over more than just one variable. In fact, multiple variables are summed over simultaneously. Essentially, the cluster of nodes C can be treated as a single node with a large enough domain size. That is, the variable, say Y_u, can represent the set of variables

X_C if Y_u has as many possible values as the Cartesian product $\times_{c \in C} |\mathsf{D}_{X_c}|$ where D_{X_v} is the domain of X_v and $|\mathsf{D}_{X_v}|$ its size.

A tree of cluster nodes is not a junction tree unless it possesses certain properties. Those properties are as follows: first, the original graph must be *triangulated* (see the next ¶). If the graph is not triangulated, then one must add edges until it is triangulated. Only the class of triangulated graphs can be clustered into a junction tree. Second, the tree of clusters must be such that each tree node cluster corresponds to a clique in the (triangulated) graph. Third, for every two clusters in the tree, and for the necessarily unique path between these two clusters, the nodes that lie in the intersection of these extremal cliques must exist within every cluster along this path. This is called the *running intersection property*. If the above three properties are true, then eliminating leaf clusters one at a time will produce a correct inference algorithm for the original graph. We note, however, that the cost of marginalizing away the nodes in a cluster is exponential in the cluster's size, so it is imperative to have as small a cluster size as possible.

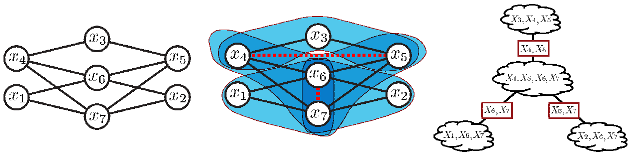


Figure 3: A graph with seven nodes is eliminated in node order $(x_1, x_2, \ldots)$ to produce the middle triangulated graph with fill in edges (dotted red) and cliques (shaded blue). Right: the resulting junction tree.

A graph is triangulated whenever any cycle in the graph has an edge connecting two non-consecutive nodes in that cycle. Running the elimination algorithm on the original graph is one way to produce a triangulated graph. In this case, however, the elimination algorithm must be modified: to eliminate a node v with more than one neighbor, we first connect together all of v's neighbors in the graph and then we remove v and its immediately adjacent edges from the graph. This step is called *node elimination*, and any new edges are called *fill-in edges* (see Figure 3). Given a graph $G = (V, E)$ we can depict its triangulation as $G^{\text{tr}} = (V, E \cup F)$ where F are the fill-in edges. To triangulate the graph, one way is to eliminate all of the nodes, and then ‘‘reconstitute’’ all nodes along with any edges that were added along the way. Once done, we are guaranteed that the graph is triangulated. Since a triangulation can only add edges, $\mathcal{F}(G, \mathcal{M}^{\text{sep}}) \subseteq \mathcal{F}(G^{\text{tr}}, \mathcal{M}^{\text{sep}})$, so any inference algorithm for all members of $\mathcal{F}(G^{\text{tr}}, \mathcal{M}^{\text{sep}})$ will work for any member of $\mathcal{F}(G, \mathcal{M}^{\text{sep}})$. Any of the $N!$ orders in which nodes might be eliminated (each known as an *elimination order*) will yield a triangulated graph. The graphical elimination procedure corresponds to summing out a variable from a factored probability distribution.

One very simple triangulated graph has all variables clustered into a single large clique, but this is typically not useful since it entails a cost that is exponential in the number of variables N. In general, the goal is to find the triangulation that minimizes the size of the largest maximal clique. Unfortunately, this is itself an NP-complete optimization problem, so heuristics are often used [16, 10, 13, 11].

Our focus in this paper is on dynamic models and how inference methodology can be tailored for this subclass of models, to which we next turn our attention.

3 Dynamic Graphical Models

A dynamic graphical model consists of a graph $G = (V, E)$ template, a set of properties $\mathcal{M}$, and an integer expansion parameter $T \geq 0$. The family associated with a DGM expanded to length T is denoted as $\mathcal{F}(G, \mathcal{M}, T)$. Any $p \in \mathcal{F}(G, \mathcal{M}, T)$ must be a distribution over a set of random variables whose size is some function of T, and must obey all properties $\mathcal{M}$ when applied to G expanded by T. The complete family of distributions associated with a DGM considers all T, namely $\bigcup_{T>0} \mathcal{F}(G, \mathcal{M}, T)$.

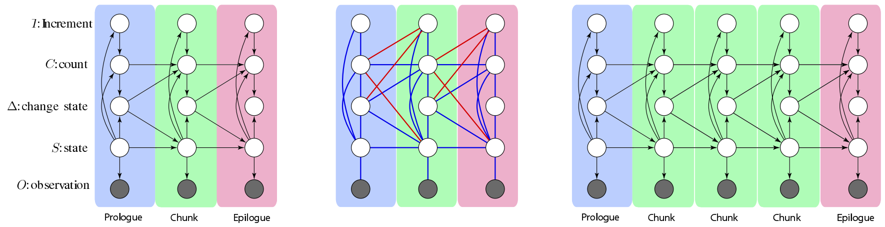


Figure 4: Left: A DBN template that consists of a prologue, chunk, and epilogue, and that forms a simple segment model useful in bioinformatics [24]. Middle: The moralized template where fill-in edges are shown as red. Right: The template unrolled 2 times.

Often, the template G is partitioned into three sections, an prologue $\mathcal{G}^p = (V_p, E_p)$, a chunk $\mathcal{G}^c = (V_c, E_c)$ (which is to be repeated in time), and an epilogue $\mathcal{G}^e = (V_e, E_e)$. Given an integer $T > 0$, an ‘‘unrolling’’ of the template T times is an instantiation where $\mathcal{G}^p$ appears once (on the left), $\mathcal{G}^c$ appears $T + 1$ times in succession, and $\mathcal{G}^e$ appears once (on the right). An unrolling consists of repeating the nodes of the chunk along with the edges connecting to the adjacent left and right chunks. Figure 4 shows an example of a template and its unrolled expansion where there are three copies of the chunk.

Each section within a template has connectivity rules. The prologue is a graph over nodes in V_p but also may contain edges to nodes in V_c. It may not contain edges to nodes in V_e. The chunk is a graph over nodes in V_c but can also in some cases include edges to nodes in either V_p or V_e or both. Symmetrically mirroring the prologue, the epilogue is a graph over variable in V_e and may also include edges to variables in V_c but may not include edges to variables in V_p. These rules ensure that any expansion leads to a valid graphical model and also ensure the existence of a temporal independence property --- that is, some notion of the present renders some notion of the future, and of the past, independent. For example, this property in a first-order Markov chain $Q_{1:T}$ means that each Q_t renders $Q_{1:t-1}$ and $Q_{t+1:T}$ independent. In a DGM, since each chunk cannot reach indefinitely into the past or future, there will always be some version of this property as well. Given the above definition of a chunk, the temporal independence property can be generalized so that one must condition on more than a single time slice to ensure conditional independence of past and future.

The template and connectivity rules also mean that for any expansion, the model still has a finite-length description. That is, the number of parameters in the model is on the order of the size of the template G, not the length T. This is similar to a time homogeneity assumption in a Markov chain. The number of parameters not increasing with T means that different factors must share the same parameters.

DGMs generalize dynamic Bayesian networks (DBNs) [3], dynamic (or temporal) Markov random fields (DMRFs) (both of which themselves generalize HMMs and hierarchical HMMs [6]), Boltzmann chains [27], sequential segmental models [7], dynamic conditional random fields (CRFs) [15] and segmental conditional random fields. If the template and all expanded graphs are Bayesian networks, then the DGM corresponds to a DBN. If the template and its expansions always produce a MRF, then we have a DMRF. If the expanded template corresponds to a conditional distribution, the DGM corresponds to a CRF or its generalizations. Any type of static graphical model, in fact, including factor graphs [14] and chain graphs [16] has a dynamic counterpart. In this article we concentrate on only DBNs and DMRFs, but the methods apply to any type of dynamic graphical model.

In practice, DGMs have a quite different general shape than their static cohorts. Since DGMs are sequential, the expanded graphs are typically much wider than higher since T can be arbitrarily large. It is clear, therefore, that probabilistic inference should follow a form similar to the standard forward-backward inference algorithm in hidden Markov models (HMMs). However, unlike with an HMM, the state space is structured, and this structure offers some unique opportunities for performing inference not available to the HMM. Moreover, in many domains (such as speech recognition) the state space at each time point can be quite large (in the tens of millions).

4 Example: Segment Modeling in Bioinformatics

Bioinformatics is an important research area involving learning and reasoning about biological molecules (e.g., DNA and proteins). For example, one might wish to predict the 3-D structure of a protein using only the sequence of constituent amino acids, or given a DNA sequence the goal might be to decide which sub-segments code for currently undiscovered proteins and to determine how one can automatically infer their ultimate function. DGMs are quite appropriate for this domain, and there have already been many successful instances of HMMs and its variants in bioinformatics [11].

We next give an example DGM that is useful in sequential modeling of segments. The DGM (which is actually a DBN, see Figure 4) corresponds to a modular reusable DBN component that allows for non-geometric state duration distributions. Each state, represented by variable S_t, may have its own state duration and determines the distribution over observations O_t like an HMM. Unlike an HMM, however, this model has quite different state duration behavior. A state change is triggered by a binary state change indicator variable Δ; that is, $S_t = S_{t-1}$ when $\Delta_{t-1} = 0$, but when $\Delta_{t-1} = 1$, then S_t is chosen randomly based on $P(S_t|S_{t-1})$. The actual duration distributions of each state are determined by variables S, I, and C, and may be one of: 1) a fixed length distribution; 2) an arbitrary multinomial distribution, and 3) a state-dependent negative binomial distribution. To implement a fixed (non-probabilistic) duration, when $\Delta_{t-1} = 1$, then C_t gets set to the desired length and it decrements deterministically (i.e., $C_t = C_{t-1} - 1$ with probability 1) until $C_t = 0$ at which point it triggers a state transition. In this case, C_t ignores I_t. To produce an arbitrary multinomial duration distribution, rather than deterministically assigning C_t to a length, C_t gets a random length based on a multinomial distribution $p(C_t|S_t, \Delta_{t-1} = 1)$. C_t then decrements as in the first case until it reaches zero. To produce a negative binomial distribution corresponding to the sum of k geometric distributions, C_t is deterministically set to k, but then C_t decrements only when I_t (a state-dependent binary indicator variable) is unity. Of course if $k = 1$, then we recover the standard HMM geometric duration, which would also be an option for some states. As can be seen, by explicitly encoding the various length distributions within the structure of the DBN, there is considerable flexibility for duration modeling.

The above example is a modified and simplified version of a DBN used as a method to predict the topology of transmembrane proteins [24]. Transmembrane protein prediction is a sequential labeling task, where each amino acid of a protein is labeled as belonging to one of the following four classes: cytoplasmic loops, membrane-spanning segments, non-cytoplasmic loops, and signal peptides. Different classes have very different duration properties.

5 HMMs and Graphical Models

An HMM is a well known instance of a DGM that, for an instance of length T, consists of $2T$ variables: a sequence of T discrete state variables $Q_{1:T}$ organized as a first order Markov chain, and a set of T observed variables $X_{1:T}$ each conditionally independent of everything else when conditioned on the corresponding

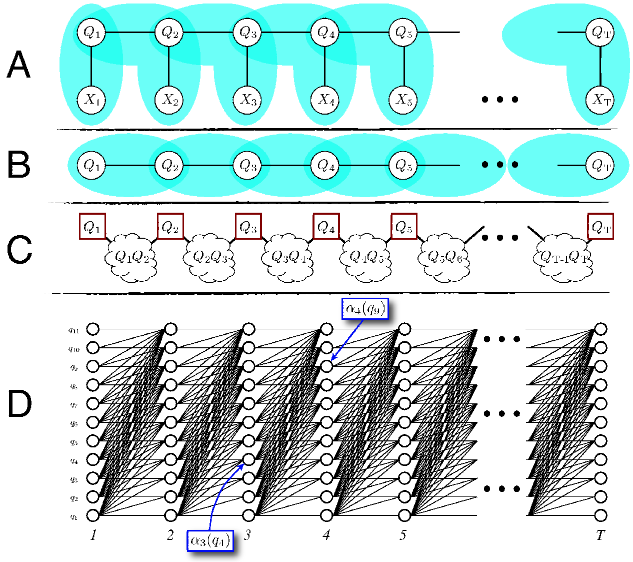


Figure 5: A: An HMM as a DGM. B: The DGM for the effective state space of the HMM. C: a junction tree corresponding to the HMM, where each clique has two random variables (shown as cloud shapes) and each separator has one random variable (shown as squares). D: the trellis graph corresponding to the search space of the HMM, and also to the junction tree. The trellis is shown aligned so that the separators and cliques in the junction tree vertically line up with the columns or transitions of the trellis.

state variable (see Figure 5-A). It is also well known that for a time signal of length T, and with an HMM that has N states, the time complexity of HMM inference is $O(N^2T)$ and the memory complexity is $O(NT)$. We briefly review the inference procedures in HMMs that lead to these complexities in order to compare to their analogues in the DGM case. We also describe the junction tree and inference algorithm for HMMs, and how it compares to several standard methods of HMM inference (namely *synchronous* and *asynchronous* decoding), and this will allow us, later in the paper, to describe how these approaches relate to DGM triangulation.

An HMM is a joint distribution with the following factorization properties:

$$p(\bar{x}_{1:T:}, q_{1:T}) \propto \prod_{t=1}^{T} \phi(\bar{x}_t, q_t) \prod_{t=2}^{T} \phi(q_{t-1}, q_t) \tag{1}$$

This factorization property is also conveyed in Figure 5-A. The observation variables $\bar{x}_t$ only contribute multiplicatively to the HMM scores and not to its state space or complexity. It is therefore possible to simplify the HMM by defining factors $\phi'(q_{t-1}, q_t) \triangleq \phi(q_{t-1}, q_t)\phi(\bar{x}_t, q_t)$ yielding factorization $p(\bar{x}_{1:T:}, q_{1:T}) \propto \prod_{t=1}^{T} \phi'(q_{t-1}, q_t)$. This is shown in Figure 5-B. The junction tree for this graphical model is shown in Figure 5-C and we see that the tree is just a sequential chain of repeated cliques and separators. In the junction tree, each clique has two variables and there are T such cliques leading to the time complexity $O(N^2T)$ of

HMMs. On the other hand, the memory complexity is only $O(NT)$ so this must mean that the cliques are never stored. Indeed, consider the standard forward (or alpha) recursion for HMMs:

$$\alpha_t(q) \triangleq p(\bar{x}_{1:t}, Q_t = q) = p(\bar{x}_t|q) \sum_r p(Q_t = q|Q_{t-1} = r)\alpha_{t-1}(r) \tag{2}$$

The forward recursion is often described using a rectangular trellis graph (not a graphical model) as shown in Figure 5-D. Each recursion $\{\alpha_t(q)\}_q$ corresponds to a column of nodes in this graph --- in the figure, the elements corresponding to $\alpha_3(q_4)$ and $\alpha_4(q_9)$ are both marked with arrows.

From the figure, we see that the separators in the junction tree correspond to the columns of the trellis, while the cliques in the junction tree correspond to two successive columns (i.e., the transitions between columns). The reason for the $O(NT)$ memory usage is that the forward computation is only implicitly computing the cliques. Each set of computations for $\alpha_t(q)$ over all q corresponds to a clique *expansion* (i.e., to an enumeration of all of the values of the variables in the clique) and then immediate reduction. Since the cliques are never stored, it is not necessary to use $O(N^2)$ memory. Forward recursion thus corresponds to a form of message in the junction tree between successive separators. Of course, the computation is still $O(N^2T)$ since each $\alpha_t(q)$ requires $O(N^2)$ steps.

In general, in exact HMM inference, the goal is to ''visit'' each node in the HMM trellis, and this can be viewed as a form of search procedure [26]. The problem of ''search'' in artificial intelligence corresponds to expanding a very large space of possible elements as efficiently as possible. For example, given a factored function over variables indexed by set V, such as $f(x_V) = \prod_C \phi(x_C)$, where each $C \subset V$, the goal might be to identify a maximum element $\text{argmax}_{x_V} f(x_V)$. An alternative might be to sum f for all values of x_V. One can imagine doing either of these naively using a set of nested loops, where we first iterate over all values x_1, and then x_2, and so on, and at the deepest level, when all elements of x_V are instantiated, we can evaluate f. In general, any partial or complete set of variable assignments is called a ''hypothesis'' and along with each (partial) hypothesis is a (partial) score based on the set of factors that may currently be evaluated. There are many types of search, including standard depth, breadth, best first, and A* procedures all of which are detailed in [26]. These algorithms can in general perform much better than naive nested loops, and they do so by making decisions based on scores of partial hypotheses.

The use of search methods in speech recognition in fact has a long history [21, 9]. Most search methods are applied to domains where the search space expands exponentially with the depth of the search, so the search expansion takes the form of a tree. With an HMM, however, the search space has quite a different general shape, one that is more akin to a very wide parallelogram (see Figure 6). When t (which is the time variable, but acts also as the depth variable when considering HMM inference as search) is small, the state space can (in general) expand exponentially in t but at some critical point, the state space saturates. After this point, the number of states per time step is fixed at N and this continues until t is close to T. At that point, the search space essentially contracts since there are only a few states that are allowed to end the search (e.g., in speech recognition, one must end the search at the end but not in the middle of a word). Another difference from standard search is that with an HMM, the depth T is very large.

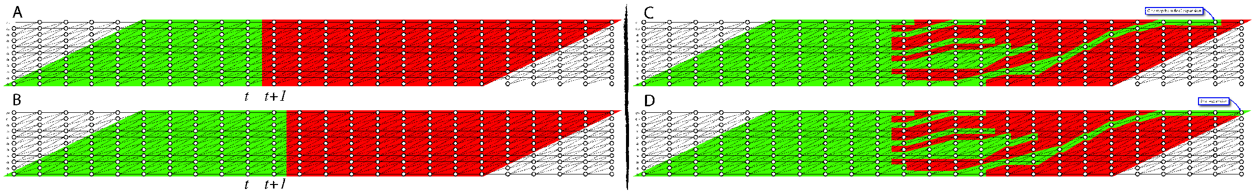


Figure 6: The parallelogram search space of HMMs. Left A & B: asynchronous search: A: All states up to and including but nothing beyond time t have been expanded. B: all states at time $t + 1$ are expanded synchronously. Right C & D: asynchronous search. Much like best first or A* search, arbitrary partial hypotheses may be expanded at any given time. In C, we are one step from the final expansion, which is shown to occur in D.

Search in HMMs can for the most part be broken down into one of two approaches: synchronous vs. asynchronous. In synchronous (also called Viterbi) search, hypotheses are expanded in temporal lock-step, where partial hypotheses that end at time t are expanded one at a time so that new partial hypotheses end at time $t + 1$. This continues for all t so that at any given time there are never extant partial hypotheses with end-points at more than two successive time points. This is essentially breadth-first search and is shown in Figure 6 A and B.

Asynchronous search, on the other hand, is such that the end-points of partial hypotheses can be at arbitrarily different time locations. Each partial hypothesis h has its own end-point consisting of a time and state pair, and its own current score s. The hypotheses may be expanded without needing to abide by any temporal constraint (see Figure 6 C and D). Asynchronous search is also called stack decoding, the reason being that one often keeps a priority-queue (implemented as a stack) of hypotheses and the hypotheses expansion occurs in a best-first order. It is also useful in stack decoding to have a form of continuation heuristic (a value that estimates the score of continuing from the current point to the end of the utterance) so that the best-first decisions are based on the combined current hypothesis score and its continuation. If the continuation heuristic is optimistic (e.g., it is an upper bound of the true probability), then it is ‘‘admissible’’ and we have an A*-search procedure [22, 8].

Both synchronous and asynchronous search procedures have been widely used in automatic speech recognition in the past. Synchronous procedures have for the most part bested their asynchronous brethren perhaps due to their overall efficiency, simplicity, and performance [9]. One of the advantages of synchronous search is that pruning (which is a form of approximate inference) is quite simple, both conceptually and to implement. Assuming that M_t is the maximum score of a set of states at time t, two simple widely used pruning strategies [21] are as follows: with *beam pruning*, we remove all partial hypotheses that are some fraction below M_t, and with K-*state pruning* (sometimes called histogram pruning), only the top K states are allowed to proceed. K-state pruning is particularly attractive since it can be efficiently implemented using either a histogram [21], or by using a fast quick-sort like algorithm to select in $O(N)$ time the top $K < N$ out of N entries. Another reason for the popularity of synchronous search is that, when the state space is very ‘‘deep’’ as shown in Figure 6, it can be a challenge to find good continuation heuristics. Much research has gone into mitigating these problems, such as fast-match heuristics [8] (where a simpler structure is used to obtain continuation scores which are then applied to complex structures), and coarse to fine based dynamic programming (where coarse-grained version of the problem is solved first, and then refined). It is challenging for such heuristics to perform well, perhaps because they need to make inferences and decisions based on events that are quite distant in the future. In general, A* search seems to work better for short and fat search problems than for long and thin search problems.

6 DGM Inference

In its most general form, performing inference in a graphical model corresponds to computing marginal probabilities for all cliques in the graph given the evidence. Concretely, let $\mathcal{C}$ be the set of all maximal cliques in the triangulated graph, so that for each $C \in \mathcal{C}$, we have that $C \subseteq V$. The set of random variables is $X_1, X_2, \ldots, X_N$ where $N = |V|$ and some subset $E \subset V$ have known values, meaning that they are ‘‘evidence’’ nodes. The joint distribution then becomes $p(x_{V \setminus E}, \bar{x}_E)$ which is a function only of the non-evidence nodes $V \setminus E$. The goal of inference is to produce clique marginal probabilities $p(x_{C \setminus E}, \bar{x}_E)$ for all $C \in \mathcal{C}$. In the case of an HMM, the goal is to produce $p(Q_{t-1}, Q_t, \bar{x}_{1:T})$ for all t since pairs of successive state variables constitute all maximal cliques (see Figure 5). Computing these clique potentials is needed both for parameter estimation, and for finding the most likely assignments to the hidden variables. Now the belief propagation algorithm, mentioned earlier in this article, when applied to a junction tree will indeed produce all maximal clique marginals. But we are interested in aspects of this algorithm that are unique to the DGM case.

A key goal is to deduce an inference algorithm for any $p \in \bigcup_{T>0} \mathcal{F}(G, \mathcal{M}, T)$ using the information

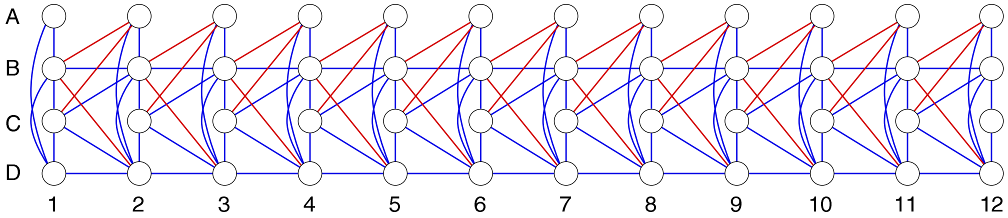


Figure 7: The MRF from Figure 4-middle unrolled to 12 frames.

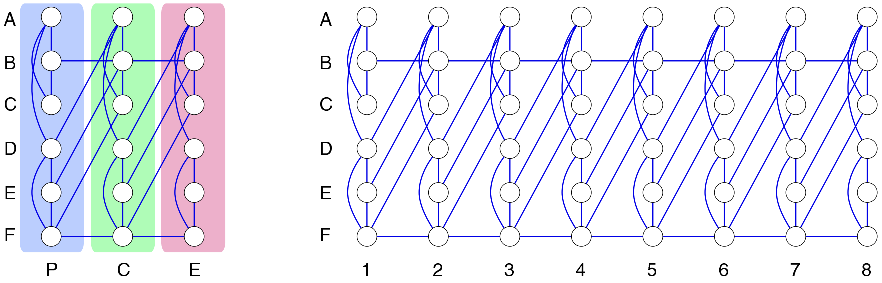


Figure 8: A DGM where eliminating one slice before the next (in either left-to-right or the right-to-left order) is not optimal. On the left is the DGM template and the right is the template unrolled to eight frames. If we use only the left interface method or only the right interface method, the largest clique size in the optimal triangulation is five. If, on the other hand, we find the optimal interface separator using the max-flow based vertex-cut procedure mentioned in the text, then the size of the largest clique in the optimal triangulation reduces to four.

only in the template $\mathcal{F}(G, \mathcal{M}, 0)$. That is, the template alone should be sufficient to deduce the computational properties and inference procedure for any amount of unrolling. Hence, the cost of deducing such an inference algorithm is amortized over the use of $(G, \mathcal{M})$. Given a DGM template G, for any fixed T, any $p \in \mathcal{F}(G, \mathcal{M}, T)$ factors with respect to a fixed graphical model, namely G unrolled T times. One option is to unroll the graph for each desired $T \geq 0$ and treat them as static graphical models, and then deduce a separate inference procedure (using, say, belief propagation) for each unrolling without any sharing of information with other unrolled models. This naive approach, however, does not achieve the aforementioned goal. We will show, however, using only $\mathcal{F}(G, \mathcal{M}, T')$ for $T' = 0$, how to deduce an inference algorithm for $\mathcal{F}(G, \mathcal{M}, T)$ with $T > 0$.[2]

If the DGM is a Bayesian network template (i.e., the model is a DBN), the first step is a template moralization step (as in Figure 4-middle), so that any factor in the BN template can find a containing clique in the resulting undirected model. We henceforth assume that all DGMs are undirected and that we are working with the family of MRFs (this assumption does not make the procedures any less applicable to other families of models, such as temporal CRFs).

---

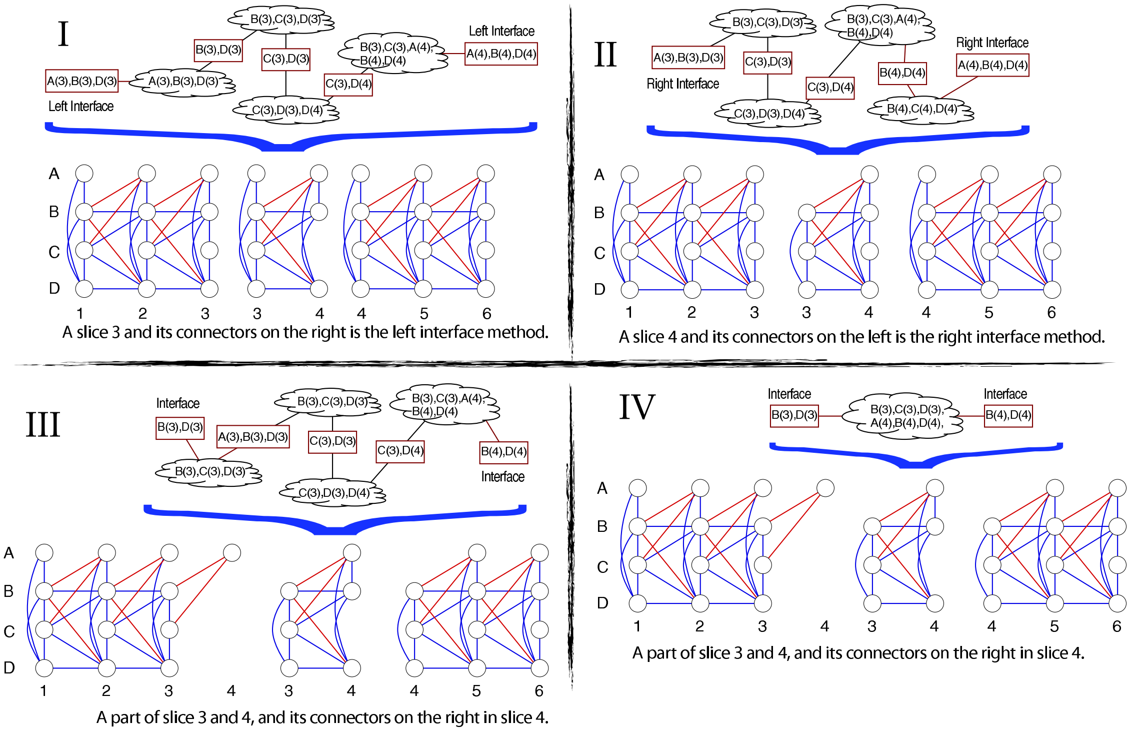
[2] We consider unrolling a small and constant number of times, $T' < k$, a reasonable step in deducing inference properties for all $T >= 0$. We need not unroll for all T.

The following techniques generalize those mentioned in Section 5, and consist of two phases. The first phase is graph-theoretic and organizational, and allows the DGM to be turned into portions of a junction tree in which inference can be performed. More importantly, these sub-junction trees can be spliced together in time to allow for any length of temporal signal. This can be done with no loss of optimality relative to the optimal exact DGM inference algorithm. The next phase is a form of message passing that also generalizes HMMs. It is a hybrid synchronous/asynchronous algorithm where the degree of asynchrony is dependent on the triangulation and underlying junction tree. The message passing algorithm, moreover, is formulated so that the many approximate beam-pruning options that have been highly successful in HMM inference can be applied here as well.

6.1 Unroll and Compute

Since it is not viable to unroll the graph for each T and then re-deduce an inference algorithm, we describe a way to deduce an inference strategy based on only the template.

Consider first the DGM from Figure 4-middle that has been unrolled a large number of times, as shown in Figure 7. While this will not lead directly to an inference algorithm, it introduces a number of ideas that will be beneficial later. Also, the observations have been removed since they do not contribute to the state space, and the factors containing the observations can be absorbed into any of the factors containing the state variable S (as we did in Figure 5). There are four variables per frame and 12 frames in the figure --- for easy naming of variables, the nodes now have a column name (the integer frame number) and a row name (A, B, C, or D). If we consider the elimination algorithm applied to this graph (seen as a static graphical model), there will be $48! \approx 10^{61}$ different possible elimination orders. As mentioned above, the goal of elimination is to produce the smallest clique in the reconstituted triangulated graph. With a dynamic graph we can rule out a number of elimination orders immediately. For example, consider starting the elimination at node D(6). That will immediately couple together its neighbors D(5), C(5), B(5), A(6), B(6), C(6), and D(7), leading to a clique of size eight. If we were to next eliminate C(6), that would yield yet another size-eight clique. Given that a naive triangulation and junction tree consisting of a chain of cliques of the form B(t), C(t), D(t), A$(t+1)$, B$(t+1)$, C$(t+1)$, D$(t+1)$ has a maximum clique of size seven, eliminating node D(6) first is a poor initial choice. If we instead were to first eliminate A(6), this would yield the clique A(6), B(5), C(5), D(6), B(6) of size five, but then if we next eliminated B(6) that would lead to a clique of size eight and


I
B(3),C(3),D(3)
B(3),D(3)
C(3),D(3)
B(3),C(3),A(4),B(4),D(4)
Left Interface
A(4),B(4),D(4)
A(3),B(3),D(3)
A(3),B(3),D(3)
C(3),D(4)
Left Interface
C(3),D(3),D(4)
A
B
C
D
1 2 3 3 4 4 5 6
II
B(3),C(3),D(3)
B(3),C(3),A(4),B(4),D(4)
A(3),B(3),D(3)
C(3),D(3)
Right Interface
Right Interface
B(4),D(4)
A(4),B(4),D(4)
C(3),D(4)
C(3),D(3),D(4)
B(4),C(4),D(4)
III
Interface
B(3),D(3)
B(3),C(3),D(3)
B(3),C(3),A(4),B(4),D(4)
A(3),B(3),D(3)
C(3),D(3)
C(3),D(4)
B(4),D(4)
Interface
B(3),C(3),D(3)
C(3),D(3),D(4)
1 2 3 4 3 4 4 5 6
IV
Interface
B(3),D(3)
B(3),C(3),D(3),A(4),B(4),D(4),
Interface
B(4),D(4)
1 2 3 4 3 4 4 5 6

A slice 3 and its connectors on the right is the left interface method.

A slice 4 and its connectors on the left is the right interface method.

A part of slice 3 and 4, and its connectors on the right in slice 4.

A part of slice 3 and 4, and its connectors on the right in slice 4.

Figure 9: I: when elimination is performed strictly left-to-right in slice-by-sliced constrained order, then nodes A,B, and D are forcibly completed. When we create a modified chunk (shown here as the excised slices 3 and 4), those nodes become a clique. The junction tree segment corresponding to this modified chunk is shown on the top. This junction tree can be stitched together in time to create a long junction tree corresponding to any degree of unrolling of the original DGM. II: a similar case when elimination is performed strictly right-to-left and slice-by-slice. In both case I and II, the junction tree segment has interfaces to neighboring junction tree segments of size 3, leading to a memory complexity of $O(N^3T)$ where N is the domain size of the random variables. III: an improved situation where nodes in one slice need not all be eliminated before proceeding to the next slice. An improved modified chunk (shown as the excised slices 3 and 4) can be discovered in low-order polynomial time by any minimum vertex-cut algorithm [20, 3]. Here the junction tree segment has interfaces to neighboring junction tree segments of size 2, leading to a memory complexity of only $O(N^2T)$. IV: An example of the interfaces in case III, but where the junction tree is just one clique with six variables, analogous to the HMM junction tree shown in Figure 5.

if we next eliminated D(6) it would lead to a clique of size seven.

The model above only has 12 frames, but an unrolled graph can have orders of magnitude more frames. It seems intuitive, therefore, that some form of left-to-right inference procedure is most promising. This means that variables should be eliminated starting at the left and moving to the right (forward in time), or alternatively starting from the right and moving to the left. For example, if we were to eliminate all variables in frame one in Figure 7, then no matter what the order, the largest clique will be no more than size seven. If moreover we were smarter about the elimination order, we could do much better --- consider the following elimination order: A(1), D(1), B(1), C(1), A(2), D(2), B(2), C(2), A(3), $\ldots$, where the largest clique is of size five. Eliminating in right-to-left order as follows: A(12), C(12), B(12), D(12), A(11), C(11), B(11), D(11), $\ldots$ also results in a size-five maximum clique size. In both cases, we see a periodic pattern in the elimination ordering. In the left-to-right case, we eliminate nodes in slice t in order A, D, B, C before eliminating any nodes in slice $t + 1$. In the right-to-left case, we eliminate nodes in slice t in order A, C, B, D before eliminating any nodes in slice $t - 1$.

From the above example, it might seem like we can thus constrain the elimination order such that it always eliminates one slice of variables before the next one --- the example shows that if we are performing a left-to-right elimination, we can optimally eliminate all variables in time t before we eliminate any in time $t + 1$ (and analogously in the right-to-left case). But this need not be the case as shown in Figure 8. Here, no matter what the ordering, if we eliminate nodes at time frame t before starting to eliminate nodes in time $t + 1$ (or eliminating at slice $t + 1$ before slice t) the smallest maximum clique size will be of size five. If the elimination order is allowed a bit more flexibility, the smallest maximum clique size is four --- consider the elimination order (C(1), A(1), B(1), E(1), D(1), C(2), A(2), F(1), B(2), E(2), D(2), C(3), A(3), F(2), $\ldots$). It can be shown that this corresponds to the best possible elimination order achievable in this graph. Note that after we eliminated the first two nodes C(1) and A(1), the pattern B(t), E(t), D(t), C$(t + 1)$, A$(t + 1)$, F(t) repeats. In this case, the periodic pattern did not appear until we eliminated a few ‘‘burn in’’ nodes at the left of the graph. Moreover, we allowed a less restrictive elimination order (a few later variables are eliminated before an earlier one).

Of course, our goal is to produce an inference procedure based on the template, rather than unrolling and then finding some elimination order in the unrolled graph. Moreover, we want a more systematic way to identify the periodicity in the ordering, and one can then use this periodicity to form a chunk of junction tree which itself can be unrolled to any length. Also, like in the HMM case where computational cost is $O(N^2T)$ and memory cost is $O(NT)$ we wish to achieve good computational and memory performance. The next section shows how this can be done.

6.2 Finding Periodicity

Considering again Figure 7, we see that the unrolled graph will have a periodicity. For example, as we repeat copies of $\mathcal{G}^c$, then $\mathcal{G}^c$ itself defines the beginning and end of a period --- the variables in $\mathcal{G}^c$ that connect to the left are the beginning, and the variables in $\mathcal{G}^c$ that connect to the right are the end. As with any periodic signal, one has a choice as to where the beginning and ending of a period should be. Normally these choices are arbitrary. Moreover, any periodic signal with period $\mathcal{T}$ is also periodic with period $k\mathcal{T}$ for any positive integer k. Normally, we wish to use the smallest integer k leading to what is known in the signal processing literature as the fundamental period. In the case of DGM inference, however, the choice of the location of the start and end of a period, and the use of periods other than the fundamental period can have significant consequences for inference.

As demonstrated in the previous section, like the DGM itself, the elimination orders were periodic, and this was true even for an elimination order that was not strictly left-to-right. This suggests that whatever processing is done, it can be decided only once, within a periodic segment, and then reused for all repeated segments in an unrolling. The question is what periodic segment to use and how to find it. We can view the problem as excising a portion of the fully unrolled graph based on flanking splices, and we will call the portion of the graph that is excised in such a fashion the ‘‘modified chunk.’’

The processing that is done within a modified chunk can typically be viewed as an elimination order (although there are important exceptions to this rule [2], and see Section 6.3). If the elimination corresponds strictly to a slice-by-slice left-to-right (or right-to-left) elimination order, the modified chunk might as well be the original chunk. The reason is that, in this case, the elimination ordering corresponds to a permutation of the chunk nodes, so all decisions regarding within-period processing are made within a single chunk. On the one hand, this might seem like a beneficial property since there are many fewer permutations of nodes in a single chunk than there are permutations of nodes in multiple chunks. On the other hand, there are unfortunately detrimental consequences, and this relates to how an elimination order adds edges to a graph. In a graphical model, it is the missing edges that provide the ‘‘structural’’ property in the family of distributions, and that allow for efficient inference schemes to be derived. When elimination is performed, it can only add edges which in turn can only reduce structure. Unfortunately, when performing such elimination, any underlying structure can be lost. This follows from an important theorem by Rose (Lemma 4 in [25]) that states the following: Let $\mathcal{G} = (V, E)$ be an undirected graph with a given elimination ordering that maps $\mathcal{G}$ to $\mathcal{G}' = (V, E')$ where $E' = E \cup F$, and where F are the fill-in edges added during elimination. Then $uv \in E'$ is an edge in $\mathcal{G}'$ **iff** there is a path with endpoints u and v, and where all nodes on the path other than u and v are eliminated before u and v. Thus, if there is a path between two nodes $u, v \in V_t$ where all the path nodes (except the endpoints u and v) lie entirely earlier in time, and if all such earlier nodes are eliminated, then u and v will be connected. This will couple together variables that otherwise might have nicely factored.

When all nodes earlier than chunk $t + 1$ are eliminated and when there is one connected component per chunk, there will be a set of nodes that are forcibly completed in chunk $t+1$, namely those nodes entirely in chunk $t+1$ that have neighbors in chunk t. In a DBN those nodes have been called the *interface* [12, 4], *backward interface* [28], or the *incoming interface* [19]. We will call this simply the ‘‘left interface.’’ The left interface is shown in Figure 9-I, where nodes A(t), B(t), and D(t) for all t are necessarily completed when doing a slice-by-slice left-to-right elimination ordering. Now it might seem that this would not matter since it can be seen in the graph that these nodes are already connected. However, it is often not the case that the interface nodes are already complete (consider, for example, the graph given in Figure 8). Therefore, such unnecessary adding of edges can significantly increase the state space of the resulting junction tree.

Moreover, even if the nodes are already completed, a poor interface can have a detrimental effect on the memory complexity. Recall the junction tree for the HMM in Figure 5. Inference, corresponding to this junction tree, has memory complexity $O(NT)$ and this is determined by the separator size (which is one). The separators, moreover, separate two junction tree sections that each consist only of a clique of size two. The separator in the HMM junction tree corresponds precisely to the interface in the DGM junction tree. In fact the interface is a separator in the underlying MRF, rendering those variables on the left and right conditionally independent, analogous to the way in which a state variable in an HMM renders the left and right conditionally independent. In the sequel, we will sometimes use the phrases *interface separator* or just *interface* interchangeably. In Figure 9-I, the size of the corresponding separator between two junction tree segments is three, but we can plainly see within the junction tree that if we were to define periodicity starting from a different clique, we could find another interface separator that would be of size two. Thus, memory complexity could be reduced from $O(N^3T)$ to $O(N^2T)$. The question is how to identify this separator automatically.

One option is to make sure that all nodes later than chunk t are eliminated before those in chunk t. Here, certain nodes in chunk t will be completed, again by Rose’s Theorem. These are the nodes in slice t that have neighbors in slice $t+1$, and have been called the *forward interface* [28, 4] or *outgoing interface* [19]. We will call this simply the ‘‘right interface’’ and it is again a separator in the DGM rendering its leftward and rightward variables conditionally independent. This is shown in Figure 9-II. However, we see once again that the size of the separator between two successive junction tree segments is still of size three. Moreover, if we were to try this strategy on Figure 8, it would still produce excessive additional edges thereby increasing the state space of the model.

Many more options than this are available [3]. As we can see from Figures 9-I or II, the interior (i.e., non-interface) separators of the junction trees are only of size two, while the interface separators are of size three. If we define a modified chunk as shown in Figure 9-III, then the resulting junction tree is separated by its neighboring junction tree by interfaces of only size two, thereby achieving the desired $O(N^2T)$ memory cost. Figure 9-IV shows another example with the same interface cost (size two) but with a very different junction tree consisting of only one clique with all variables. This is akin to the junction tree for the HMM shown in Figure 5 except that rather than one variable in the separator and two variables in the clique, we have two variables in the separator and six variables in the clique.

Now, the highest-quality such separator might only exist if it were allowed to span multiple, say M, of the original DGM chunks. In the above examples, $M = 1$ sufficed but we may allow any $M \geq 1$. Consider next forming a sequence of M chunks ${\mathcal{G}^c}_1, {\mathcal{G}^c}_2, \ldots, {\mathcal{G}^c}_M$ and assume also there exist a virtual left chunk ${\mathcal{G}^c}_0$ and right chunk ${\mathcal{G}^c}_{M+1}$. Note that M never needs to be larger than the number of variables in a given original chunk although the resulting best separator might span many fewer than M chunks. Thus $M \leq |V_c|$. Let the initial left interface I_l be the nodes in ${\mathcal{G}^c}_1$ that connect to nodes in ${\mathcal{G}^c}_0$ (i.e., $I_l = \{u \in V({\mathcal{G}^c}_1) : \exists v \in V({\mathcal{G}^c}_0) \text{ with } (u,v) \in E(G)\}$ where $V(G) = V$ and $E(G) = E$ when $G = (V, E)$ is a graph). Let the initial right interface I_r be the nodes in ${\mathcal{G}^c}_M$ that connect to a node in ${\mathcal{G}^c}_{M+1}$ (with an analogous mathematical definition). The problem of finding the best separator is then to find the minimum cost (I_l, I_r) separator in the graph $({\mathcal{G}^c}_1, {\mathcal{G}^c}_2, \ldots, {\mathcal{G}^c}_M)$. Assuming that S is this minimum separator, then the memory cost of exact inference on a sequence of T chunks becomes $O(T \prod_{v \in S} |\mathsf{D}_{X_v}|)$. Moreover, if a new chunk that lies between two successive separators, say S_t and S_{t+1} is now $\mathcal{G}^{c\prime}_t$ then the time complexity is at most $O(T \prod_{v \in V(\mathcal{G}^{c\prime}_t)} |\mathsf{D}_{X_v}|)$. Therefore, our goal should be to find an interface separator that minimizes $\prod_{v \in S} |\mathsf{D}_{X_v}|$ (to reduce memory requirements) and $\prod_{v \in V(\mathcal{G}^{c\prime}_t)} |\mathsf{D}_{X_v}|$ (to reduce an upper bound on computational demands).

The first problem, minimizing memory usage, corresponds to the HMM's memory complexity discussed in Section 5. In the more general DGM case, however, the optimal separator must be discovered, and solving this problem depends on how we measure the cost of the separator. If the separator cost is measured as the number of nodes in the separator, or as the *weight* of the separator (i.e., log state space $\log \prod_{v \in S} |\mathsf{D}_{X_v}|$), then the problem can be solved optimally in polynomial time using a max-flow algorithm. The approach is to transform the graph $({\mathcal{G}^c}_1, {\mathcal{G}^c}_2, \ldots, {\mathcal{G}^c}_M)$ into a flow network by adding two nodes (s, t) and where s is connected to all nodes in I_l, and t is connected to all nodes in I_r. All edges in the graph are set to have infinite capacity and then we solve the max-flow problem in a transformed network where the nodes may have a cost (achieved by duplicating each node and connecting them with an edge corresponding to the node cost), also known as the vertex cut problem [20]. None of the resulting edges from the original graphical model, therefore, will be selected as part of the cut since they have infinite capacity, and the cut will consist only of edges corresponding to the separator with the minimal cost and minimal memory requirements, as desired.

At this point, our modified chunk renders the future and the past independent, as does the modified interface separators between chunks. Since the separator was optimized, the notion of the left or right interface [12, 4, 28, 19, 3] is no longer relevant since after the aforementioned max-flow optimization they are identical. Therefore, we are free to consider next only a modified chunk which contains its left and right interface separators that (due to Rose's theorem) have been completed and that are minimal.

6.3 Static Inference in the Resulting Chunk

The resulting DGM chunk is like any normal static graphical model: it consists of a set of nodes and edges. Any inference method that can be used for a static graphical model therefore can be used for the chunk. For example, an exact inference procedure that forms a triangulation of the chunk and a resultant junction tree can be used --- since the separators produced via the results of Section 6.2 are complete, and since any triangulation of the chunk will not be capable of reaching beyond the boundaries of a chunk, the DGM separation property will still hold regardless of the triangulation used. Alternatively, approximate schemes for inference can be used [11, 13].

This is where the notion that from the template one can infer an upper bound on the cost of inference (as mentioned in Section 6) comes into play. The chunk can be triangulated using any off-line triangulation scheme [25, 2], and the resulting maximum clique size of the triangulation then provides an upper bound on the cost of inference. That is, analogous to the static case, if the resultant maximum clique size of the chunk is $\omega + 1$ then the time cost of inference will become $O(TN^{\omega+1})$. Moreover, the up-front cost spent triangulating the chunk is amortized over the life of the DGM, since the computational properties of the triangulation will hold for any T.

The above approach, of first finding the optimal separator and then triangulating the modified chunk, has the potential to find the optimal inference procedure for any DGM. The reason can be seen by considering the alternative, unrolling for a fixed T and then finding the optimal inference procedure in that unrolled graph for the given T. Since T can grow unboundedly, it makes sense only to eliminate nodes in some (roughly) left-to-right order, but as mentioned above, it need not be the case that one chunk $\mathcal{G}^c$ should be fully eliminated before moving onto the next chunk. However, there would never be a utility in a succession of more than $|V(\mathcal{G}^c)|$ partially eliminated chunks since at that point the benefit of eliminating the latest node could be applied to one of the earlier ones. When the search for the best separator, as mentioned above, is found in a sequence of no more than M chunks, this would allow for an optimal elimination scheme in a pre-unrolled graph. And since the elimination scheme also corresponds to a triangulation, finding first the separator and then triangulating has exactly the same potential to discover the optimal inference procedure.

6.4 Clique-based Limited Extent Asynchrony

In Section 5, we discussed synchronous vs. asynchronous decoding for HMMs. In this section, we discuss how there can be hybrid approaches that straddle the fence between these two extremes. For example, a constraint can be placed on the maximum temporal distance between the end-point of any two partial hypotheses. This, in fact, is something a junction tree can easily do. Normally in a junction tree, the form of message passing is based either on the Hugin or the Shenoy-Schafer architectures [18]. The Hugin style message passing is summarized in Figure 1-II. Each of these approaches, however, assume that the cost of a single message is itself tractable, which is often not the case in speech recognition due to the very large number of possible random variable values (e.g., the typical vocabulary size of a large vocabulary system might be more than 250,000). Therefore, even individual exact messages in a junction tree might be computationally infeasible and this is where search methods become useful.

We consider the case where a search procedure is used to compute the junction tree message by expanding the clique. The expansion is constrained to occur only within the clique so that the decoding is ‘‘synchronous’’ but only between maximal cliques (we are not allowed to expand a clique C_{t+1} until clique C_t has been expanded). Within a maximal clique, however, the expansion can occur in any variable order, much like an asynchronous decoding procedure. The cliques in the junction tree may effectively limit the extent of the hypothesis expansion. For example, consider the junction tree for the HMM given in Figure 5, where each clique consists only of successive variables $\{Q_t, Q_{t+1}\}$. If we were to perform constrained asynchronous search in such a junction tree, where cliques are expanded one after another but expansion may be asynchronous within a clique, then we have recovered synchronous search in HMMs.

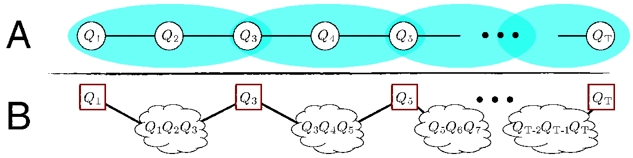


Figure 10: A: The DGM for the effective state space of the HMM. B: a junction tree corresponding to the HMM, where each clique has three random variables. D: the HMM trellis.

Cliques can consist of any number of random variables, however, and even span over short stretches of more than two time steps [3]. For example, consider the alternative junction tree for an HMM, shown in Figure 10. In this case, each clique consists of three rather than two random variables. In the triangulated graph corresponding to this junction tree, the triangulation is no longer minimal (meaning that some of the fill-in edges may be removed and the triangulation property still holds). In some cases, however, non-minimal triangulations can be useful [2]. In this case, in fact, the non-minimal triangulation is used to restrict the degree of asynchrony in an HMM expansion. By forming various triangulations (and corresponding junction trees) we can experiment with an even wider variety of different search expansion constraints for an HMM.

Such flexibility is even more prevalent in the general DGM case, especially where the M parameter mentioned in the previous section is greater than one. We will explain this using Figure 9-IV. As given, the standard way to perform such a message would be to execute the following computation

$$\phi(b_4, d_4) = \sum_{b_3,c_3,d_3,a_4,b_4,d_4} \phi(b_3, d_3)\psi(b_3, c_3, d_3)\psi(c_3, d_3, d_4)\psi(b_3, c_3, a_4, b_4, d_4) \tag{3}$$

where $\phi(b_3, d_3)$ is the initial incoming and $\phi(b_4, d_4)$ is the final outgoing interface factor, and where $\psi(b_3, c_3, d_3)$, $\psi(c_3, d_3, d_4)$, and $\psi(b_3, c_3, a_4, b_4, d_4)$ are functions corresponding to the cliques of the modified chunk. Such an approach can be wasteful, however, since it is oblivious to any sparsity that might exist in the factors and moreover it is not amenable to approximation.

The way a search procedure would approach the problem would be to perform a tree-expansion of the variables B(3), C(3), D(3), A(4), B(4), and D(4), and the factors $\phi(b_3, d_3)$, $\psi(b_3, c_3, d_3)$, $\psi(c_3, d_3, d_4)$, and $\psi(b_3, c_3, a_4, b_4, d_4)$ would act as soft constraints to produce partial hypothesis scores. Now the traditional notion of synchrony would mean that the variables a_4, b_4, and d_4 are not expanded until variables b_3, c_3, and d_3 are instantiated. A clique-based limited-extent asynchronous approach would allow the variables to expand in any order, and could even be value specific [26, 5, 1, 17]. This means that the order that the variables are expanded might depend on the values of earlier instantiated variables. This can sometimes yield enormous speedups during search.

What turns this into a form of search-based message is the following: at the leaf nodes of the search tree, all variables are expanded. Rather than computing the max, or summing the resulting scores, the set of values is used as an index into the outgoing interface separator and the resultant score is accumulated into that indexed entry. For example, with variables $b_3, c_3, d_3, a_4, b_4, d_4$ instantiated with score s, we would accumulate $\phi(b_4, d_4) \leftarrow \phi(b_4, d_4) + s$. This would occur at each leaf node visitation.

As can be imagined, the clique may span multiple time slices and if this happens so may the degree of asynchrony. The clique might span multiple time frames if $M > 1$ and the interface separator also spans multiple time slices. Alternatively, the clique might just correspond to a grouping of more than one modified chunk --- consider, as an example, Figure 9-IV, but where the clique in the junction tree contains 12, rather than six random variables, namely: $\bigcup_{t\in\{3,4\}}\{\mathrm{B}(t), \mathrm{C}(t), \mathrm{D}(t), \mathrm{A}(t+1), \mathrm{B}(t+1), \mathrm{D}(t+1)\}$. The limiting case is when the clique expands to the entire length of the sequence, and we have recovered the fully asynchronous search procedures mentioned in Section 5.

7 Discussion

We have described how a DGM can be spliced into repeatable segments, and how one can deduce a DGM inference algorithm by applying to these segments methods developed for static graphical models. As the previous pages have shown, there are many ways to deduce a final algorithm, including options for choosing the interface separators, triangulating the resulting modified chunk, organizing the separators, organizing search within cliques, limiting the temporal extent of each clique, and deciding the approximations that become necessary when the clique state space gets large. Variations of these options can lead to significant real-world differences in the computational properties of the resulting inference algorithm, but finding the optimal set of options is itself intractable. In practice, intelligent heuristics can be used to get large real-world problems to run in a quite reasonable amount of time.

Acknowledgments: The author wishes to thank Chris Bartels, Gang Ji, Mukund Narasimhan, Karim Filali, and Karen Livescu, for useful discussions.

References

[1] F. Bacchus, S. Dalmao, and T. Pitassi. Value elimination: Bayesian inference via backtracking search. In *Proceedings of the Conf. on Uncertainty in Artificial Intelligence*, pages 20--28. Morgan Kaufmann, 2003.

[2] C. Bartels and J. Bilmes. Non-minimal triangulations for mixed stochastic/deterministic graphical models. In *Proceedings of the Conf. on Uncertainty in Artificial Intelligence*, pages 15--22, Cambridge, MA, July 2006. AUAI.

[3] J. Bilmes and C. Bartels. On triangulating dynamic graphical models. In *Proceedings of the Conf. on Uncertainty in Artificial Intelligence*, pages 47--56, Acapulco, Mexico, 2003. Morgan Kaufmann.

[4] A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5--41, 2001.

[5] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[6] Z. Ghahramani and M. Jordan. Factorial Hidden Markov Models. *Machine Learning*, 29, 1997.

[7] J. R. Glass. A probabilistic framework for segment-based speech recognition. *Computer Speech & Language*, 17(2-3):137--152, 2003.

[8] P. S. Gopalakrishnan and L. R. Bahl. Fast match techniques. In C.-H. Lee, F.K. Soong, and K.K. Paliwal, editors, *Automatic Speech and Speaker Recognition: Advanced Topics*, pages 413--428. Kluwer, 1996.

[9] X. D. Huang, A. Acero, and H.-W. Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice Hall, 2001.

[10] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2001.

[11] M. I. Jordan. Graphical models. *Statistical Science*, 19(1):140--155, 2004.

[12] U. Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Conf. on Uncertainty in Artificial Intelligence*, pages 121--129, San Francisco, 1992. Morgan Kaufmann.

[13] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, August 2009.

[14] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2):498--519, 2001.

[15] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conf. on Machine Learning*, pages 282--289. Morgan Kaufmann, San Francisco, CA, 2001.

[16] S. L. Lauritzen. *Graphical Models*. Oxford Science Publications, 1996.

[17] C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf. (DAC)*, Las Vegas, June 2001.

[18] A. L. Madsen and D. Nilsson. Solving influence diagrams using HUGIN, Shafer-Shenoy and Lazy propagation. In *Proceedings of the Conf. on Uncertainty in Artificial Intelligence*, volume 17, pages 337--345. Morgan Kaufmann, 2001.

[19] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, U.C. Berkeley, Dept. of EECS, CS Division, 2002.

[20] H. Nagamochi and T. Ibaraki. *Algorithmic Aspects of Graph Connectivity*. Cambridge University Press, 2008.

[21] H. Ney, S. Ortmanns, and T. H. Aachen. Progress in dynamic programming search for LVCSR. *Proceedings of the IEEE*, 88(8):1224--1240, 2000.

[22] D. B. Paul. An efficient A* stack decoder algorithm for continuous speech recognition with a stochastic language model. *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, 1992.

[23] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 2nd printing edition, 1988.

[24] S. M. Reynolds, L. Kall, M. E. Riffle, J. Bilmes, and W. Noble. Transmembrane topology and signal peptide prediction using dynamic bayesian networks. *PLoS Computational Biology*, 4(11), November 2008.

[25] D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal Computing*, 5(2):266--282, 1976.

[26] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach, 2nd Ed.* Prentice Hall, 2003.

[27] L. K. Saul and M. I. Jordan. Boltzmann chains and Hidden Markov Models. *Neural Information Processing Society (NIPS)*, pages 435--442, 1995.

[28] Y. Xiang. Temporally invariant junction tree for inference in dynamic Bayesian network. *Lecture Notes in Computer Science*, 1600:473--487, 1999.