Constant-Space Reasoning in Dynamic Bayesian Networks

Adnan Darwiche
Computer Science Department
University of California, Los Angeles, Ca 90095
darwiche@cs.ucla.edu

November 13, 2000

Abstract: Dynamic Bayesian Networks (DBNs) have been receiving increased attention as a tool for modeling complex stochastic processes, especially that they generalize the popular Hidden Markov Models (HMMs) and Kalman filters. Since DBNs are only a subclass of standard Bayesian networks, the structure-based algorithms developed for Bayesian networks can be immediately applied to reasoning with DBNs. Such structure-based algorithms, which are variations on elimination algorithms, take $O(N \exp(w))$ time and space to compute the likelihood of an event, where N is the number of nodes in the network and w is the width of a corresponding elimination order. DBNs, however, pose two specific computational challenges that require DBN-specific solutions. First, DBNs are typically heavily connected, therefore, admitting only elimination orders of high width. Second, even if one can find an elimination order of a reasonable width, one cannot afford the space complexity of $O(N \exp(w))$ since $N = nT$ in this case, where n is the number of variables per time slice and T is the number of time slices in the DBN. For many applications, T is very large, making the space complexity of $O(nT \exp(w))$ unrealistic. Therefore, one of the key challenges of DBNs is to develop efficient algorithms which space complexity is independent of the time span T, leading to what is known as *constant-space* algorithms. We study one of the main algorithms for achieving this constant-space complexity in this paper, which is based on "slice-by-slice" elimination orders, and then suggest improvements on it based on new classes of elimination orders. We identify two topological parameters for DBNs and use them to prove a number of tight bounds on the time complexity of algorithms that we study. We also observe (experimentally) that the newly identified elimination orders tend to be better than ones based on general purpose elimination heuristics, such as min-fill. This suggests that constant-space algorithms, such as the ones we study here, should be used on DBNs even when space is not a concern.

Keywords: Dynamic Bayesian networks, elimination orders, structure-based algorithms, variable elimination, space complexity.

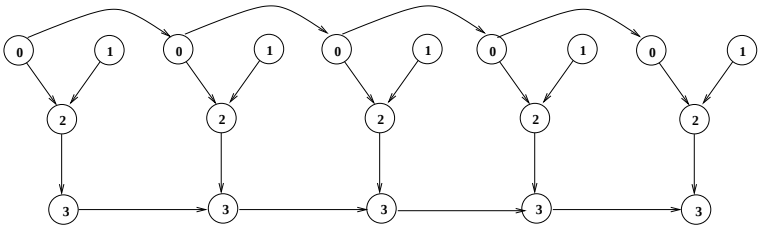

Figure 1: A Dynamic Bayesian Network (DBN).

1 Introduction

A dynamic Bayesian network (DBN) is a special kind of a Bayesian network used to model dynamic stochastic processes — see Figure 1 [2, 1, 10]. Each variable X in a DBN is associated with a time slice t and, hence, denoted by X^t. A key characteristic of DBNs is the number of time slices needed to model a particular problem, which we call the *time span* T. Another key characteristic is the number of variables associated with each time slice, which we call the *slice size* n.

A DBN is almost always assumed to satisfy the following conditions: (a) it has the same structure at any time slice t and (b) the only cross-slice edges allowed are those that extend from slice t to slice $t+1$. *We also assume here that the variables at each time slice are connected,* and talk about a T-expansion of a DBN to mean a DBN with T slices.

One of the key usages of DBNs is that of *forecasting* or *prediction:* computing the probability of some future events given evidence about the past: $Pr(X^{t_3} \mid \mathbf{o}^{t_1}, \mathbf{o}^{t_1+1}, \ldots, \mathbf{o}^{t_2})$ where $t_1 \leq t_2 < t_3$.[1] Typically, the time span $t_3 - t_1$ is large and our goal is to perform this computation in space which is independent of this span. Another task is that of diagnosis, computing the probability of some past events given evidence about the future: $Pr(X^{t_1} \mid \mathbf{o}^{t_2}, \mathbf{o}^{t_2+1}, \ldots, \mathbf{o}^{t_3})$ where $t_1 < t_2 \leq t_3$. Again, a key requirement is to perform this computation in space which is independent of the time span $t_3 - t_1$. Finally, we have the task of *monitoring:* at any point in time t, we need to have the probability of some present events given evidence about the past and present: $Pr(X^t \mid \mathbf{o}^1, \mathbf{o}^2, \ldots, \mathbf{o}^t)$. Moreover, as we proceed to the next time step $t+1$, we want to update such probabilities in time and space which are independent of t.

The complexity of reasoning in general Bayesian networks is typically characterized by two key parameters: the number N of variables in the network and the width w of a given elimination order. For a T-expansion of a DBN, the number of variables N would be nT, where n is the number of variables at each

[1] We are using the standard notation: variables are denoted by upper–case letters (A) and their values by lower–case letters (a). Sets of variables are denoted by bold–face upper–case letters ($\mathbf{A}$) and their instantiations are denoted by bold–face lower–case letters ($\mathbf{a}$).

time slice. Standard algorithms for inference in Bayesian networks [5, 11, 8, 9], which are variations on peeling algorithms [3], have a time and space complexity of $O(N \exp(w))$. We can always reason about DBNs using a standard structure-based algorithm by constructing a T-expansion such that T covers all events of interest. This will always work, but would require time and space complexity of $O(nT \exp(w))$. This turns out to be unacceptable for two reasons. First, DBNs are typically heavily connected, therefore, admitting only elimination orders of high width w. Second, even if the width w is acceptable, the space complexity of $O(nT \exp(w))$ is typically unrealistic given that the time span T is usually very large. One direction to address the first challenge is to appeal to approximate inference algorithms [2, 1, 6]. As for the second challenge, one wants to develop algorithms which space complexity is independent of T, $O(n \exp(w))$, while minimizing the time complexity as much as possible. Such algorithms are sometimes referred to as *constant-space* algorithms.

One should stress that recursive algorithms for HMMs and Kalman filters do have this property; that is, their space complexity is independent of the time span T. Therefore, one straightforward solution to the second challenge is to collapse every time slice in a DBN into a single variable leading to a chain structure, which can be handled using recursive algorithms such as those used for HMMs or Kalman filters. This approach, however, is exponential in $2n$, where n is the size of a time slice, so it works well only when $2n$ is small enough.[2]

Another more sophisticated solution to the second challenge is to use structure-based algorithms, which are variations on peeling algorithms, while restricting oneself to a subclass of elimination orders. Specifically, with an appropriate restriction on the elimination order, one can indeed reduce the space complexity of such methods from $O(nT \exp(w))$ to $O(n \exp(w))$. A popular class of elimination orders that can achieve this space complexity is known informally as *slice-by-slice orders,* and requires that one always eliminate variables at time slice t before eliminating variables at time slice $t+1$. In fact, the most comprehensive algorithm for structure-based reasoning in DBNs, that of dHUGIN [10], is based on this class of elimination orders. We analyze this class of elimination orders in this paper, highlighting two important facts:

1. The width w of any elimination order in this class is tightly bounded as follows: $w_i \leq w < w_i + n$, where w_i is a DBN structural parameter known as the *interface width.*

2. There are other classes of elimination orders that lead to constant space complexity, some of which outperform the above class on certain network structures.

Our contribution here is two-fold. First, we make formal and explicit the computational limits of algorithms based on slice-by-slice elimination orders. Second,

[2] Since each slice has n variables, each collapsed variable will have $\exp(n)$ values and the transition probability matrix will be of size $\exp(2n)$.

we open the door for the identification of other classes of elimination orders that can achieve constant-space complexity. In this regard, we identify two new classes of elimination orders and study their properties using a new DBN structural parameter known as the *temporal bandwidth*. We show that the temporal bandwidth is never greater than the interface width for the class of DBNs in which persistence edges are the only ones that extend from one time slice to another. We even show that it tends to be much smaller on a number of randomly generated networks.

Our final contribution in this paper is a surprising observation regarding heuristics for constructing elimination orders. Specifically, although constrained elimination orders cannot in principle outperform unconstrained ones, it appears that some constrained versions of the popular min-fill heuristic for constructing elimination orders tend to outperform their unconstrained counterpart. This encouraging observation means that, contrary to what one would expect, one does not seem to pay a computational price in practice as a result of constraining elimination orders so as to achieve a constant space complexity in DBNs. In fact, our experimental results suggest that one should use such constrained orders even if one is not concerned with constant-space reasoning.

This paper is structured as follows. We start by reviewing the variable elimination algorithm in Section 2 and its time and space complexity of $O(nT \exp(w))$. We then show in Section 3 how the space complexity of this algorithm can be dropped to $O(n \exp(w))$ for the case of temporal prediction, given that we restrict ourselves to slice-by-slice elimination orders. We analyze this class of elimination orders and then propose a new one, studying also its properties and comparing it to slice-by-slice orders. We then turn to the problem of temporal diagnosis in Section 4, generalizing the results to a combined algorithm for both diagnosis and prediction. Section 5 is then dedicated to experimental results, which compare the classes of elimination orders in addition to comparing the constrained and unconstrained versions of the min-fill elimination heuristic. Section 6 closes with with some concluding remarks. Proofs of all Theorems are available in Appendix A.

2 Variable Elimination

The variable elimination algorithm [5, 11] is quite simple, especially if we only want to use it for computing the probability of some evidence $\mathbf{e}$ with respect to a Bayesian network containing conditional probability tables (CPTs) $\phi_1, \ldots, \phi_N$. There are a number of variations on variable elimination, here is one:

- Let Σ be initialized to the set of tables $\phi_1, \ldots, \phi_N$.
- For each table ϕ_i in Σ, set $\phi_i(\mathbf{x})$ to 0 if $\mathbf{x}$ is inconsistent with evidence $\mathbf{e}$ (from here on, we refer to this step as *entering evidence*).
- Consider variables X according to some elimination order π:

- Let ϕ be the multiplication of all tables in Σ that mentions variable X (table multiplication is the standard operation defined on CPTs/potentials, see [8] for example).
- Sum–out variable X from ϕ to yield ϕ' (summing-out is the standard operation defined on CPTs/potentials, see [8] for example).
- Replace tables mentioning X in Σ with ϕ' (we say in this case that the tables mentioning X have been *consumed* as a result of eliminating variable X).

At the end of the algorithm, Σ is guaranteed to contain a single table ϕ with a single entry which contains the probability of evidence $\mathbf{e}$. One way of computing marginals is to compute $Pr(x, \mathbf{e})$ for each x and then $Pr(x \mid \mathbf{e}) = Pr(x, \mathbf{e}) / \sum_x Pr(x, \mathbf{e})$. This would require $O(N)$ invocations of the algorithm though.

An important notion throughout the paper is the *width* of a variable elimination order π with respect to a Bayesian network. There are a number of ways for defining this notion—here are two:

1. Suppose that J is a jointree constructed based on the elimination order π [8], and let m be the number of variables in the largest clique of J. The width of π is then $m - 1$.
2. Suppose that T is the largest table constructed when applying the variable elimination algorithm according to order π, and let m be the number of variables appearing in T. The width of π is then $m - 1$.

The more standard definition, however, is more direct and is given in terms of the moral graph of a Bayesian network [4]. We will appeal to the second definition above in this paper though, from which one can easily show that the complexity of variable elimination on a network of N nodes and using an elimination order of width w is $O(N \exp(w))$.

We close this section by noting that variable elimination algorithms, as known in the Artificial Intelligence literature, are structurally similar to peeling algorithms as known in the pedigree analysis literature [3].

3 Prediction

We will now utilize the above elimination algorithm to reason about DBNs. We start with the problem of prediction and then generalize to other tasks in Section 4.

We first present an algorithm that uses a slice-by-slice elimination order, and show how the choice of such orders allows one to drop its space complexity from $O(nT \exp(w))$ to $O(n \exp(w))$, where w is the width of given order.

Suppose we have an observation sequence $\mathbf{o} = \mathbf{o}^1, \ldots, \mathbf{o}^t$. Suppose further that we are interested in computing the marginals over variables at slice $T \geq t$ given $\mathbf{o}$. Then we will simply construct a DBN over the span T, eliminate variables at all slices except for T, then eliminate variables at T to compute marginals:

1. Let Σ contain the CPTs of slices $1, \ldots, T$. Enter evidence into tables Σ.
2. For $i = 1$ to $T - 1$: eliminate variables at slice i.
3. Apply variable elimination to Σ to compute marginals for variables X^T.

A number of observations are in order about this algorithm, which we refer to as VE_PD'. First, note that in Step 3, Σ will only contain variables at slice T. Second, the algorithm is non-deterministic in that it can eliminate variables in Steps 2 and 3 in different ways. But all such elimination orders satisfy the following property: all variables at slice t are eliminated before variables at slice $t + 1$ are eliminated. This is key to achieving a constant-space complexity as shown by the following observation:

Proposition 1 *In iteration i of Step 2 of* VE_PD*', the tables corresponding to time slices $i + 2, \ldots, T$ are irrelevant to the elimination process.* □

This basically means that there is no need to initialize Σ to the set of all tables. Instead we can dynamically introduce the tables as we need them, leading to the following constant-space version of VE_PD':

1. Let Σ contain the CPTs of slice 1. Enter evidence into tables Σ.
2. For $i = 1$ to $T - 1$:
   (a) generate the CPTs of slice $i + 1$, enter evidence into them, and add them to Σ.
   (b) eliminate variables at slice i.
3. Apply variable elimination to Σ to compute marginals for variables X^T.

The exact space and time complexity of VE_PD' depends on the specific order in which variables are eliminated in Steps 2 and 3, but we can establish tight bounds on the width of any such order. We first need the following notion:

Definition 1 *[10] The interface of a DBN is the set of variables X at a time $t > 1$ such that X, or one of its children, has a parent at time $t - 1$. The interface width of a DBN is the size of its interface.*

In the DBN of Figure 1, the interface contains three variables, X_0, X_2, X_3.

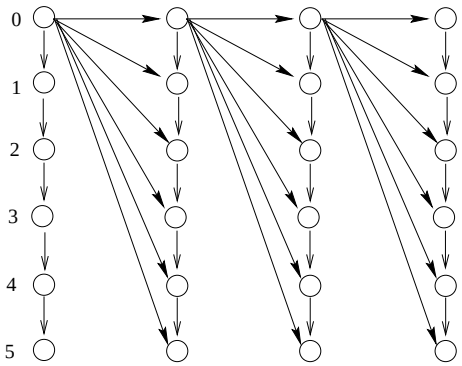

Figure 2: The interface contains all slice variables.

Theorem 1 *Let w_i be the interface width of a DBN and let π be the elimination order used by algorithm* VE_PD*' on that DBN. Then the width of order π, w, is bounded as follows: $w_i \leq w < w_i + n$, where $T > 1$ and the bounds are tight.*

We stress here two points. First, that the dHUGIN algorithm [10] constructs jointrees based on slice-by-slice elimination orders and its complexity is, therefore, subject to the established bounds.[3] The use of jointrees in dHUGIN leads to a more sophisticated notion of variable elimination, which reduces the complexity when answering multiple queries. This better complexity, however, comes at the expense of substantial algorithmic subtlety especially that the algorithm is based on the dynamic editing of such jointrees. We shall see later that in the context of DBNs, one can attain the improved complexity in case of multiple queries by only a limited and simple usage of jointrees.

We next propose a new class of elimination orders that has different properties than is shown above, yet continues to allow for constant-space reasoning in DBNs.

The new class of elimination orders is based on the following observation. Consider the DBN in Figure 2 and suppose that each slice has n variables. Then the interface width is also n and any slice-by-slice elimination order will have width $\geq n$. Note, however, that this DBN has a treewidth of 2, which is independent of n. One therefore wonders whether there exists another constant-space elimination order which width is independent of n. The answer turns to be yes, but we need the following definition first.

Definition 2 *The forward interface of a DBN with span T is the set of variables at time $t < T$ having children at time $t + 1$. The temporal bandwidth of a DBN is the size of its forward interface.*

[3] Slice-by-slice elimination is referred to as "constrained elimination" in [10], where it is also shown that a jointree constructed using such an order will have a separator that includes the DBN interface.

In Figure 1, the forward interface contains variables X_0 and X_3. In Figure 2, it contains variable X_0.

The basic idea is to restrict the forward elimination process such that when processing time slice t, we eliminate every variable except those belonging to the forward interface of slice t. Such variables will be eliminated when processing slice $t+1$. This leads to the following algorithm, which we call VE_PD:

1. For $i = 1$ to $T - 1$:
   (a) generate the CPTs at slice i, enter evidence into them, and add them to Σ.
   (b) eliminate variables in Σ except those belonging to the forward interface of slice i.
2. Generate the CPTs of slice T, enter evidence into them, and add them to Σ. Apply variable elimination to Σ to compute marginals for variables X^T.

For example, in the DBN of Figure 2, variables will be eliminated as follows in Step 1: $i = 1 : \{X_1^1, \ldots, X_5^1\}$; $i = 2 : \{X_0^1, X_1^2, \ldots, X_5^2\}$; and $i = 3 : \{X_0^2, X_1^3, \ldots, X_5^3\}$. In the DBN of Figure 1, variables will be eliminated as follows: $i = 1 : \{X_1^1, X_2^1\}$; $i = 2 : \{X_0^1, X_3^1, X_1^2, X_2^2\}$; $i = 3 : \{X_0^2, X_3^2, X_1^3, X_2^3\}$; and so on.

We will compare in Section 5 the two classes of elimination orders embodied by algorithms VE_PD and VE_PD', showing that algorithm VE_PD dominates VE_PD' on a number of benchmark and randomly generated networks. We next present a more general algorithm for prediction and diagnosis, which is based on a more general class of elimination orders, and show that its complexity is bounded from below (exponentially) by the temporal bandwidth.[4]

4 Diagnosis

We now present another constant-space elimination algorithm for temporal diagnosis in DBNs, which is based on *backward slice-by-slice elimination orders.* We then combine the prediction and diagnosis algorithms to obtain an algorithm for more general inference, based on a more general class of elimination orders.

[4]Note that algorithm VE_PD—and VE_PD' for that matter—can be easily modified so it can monitor a dynamic process, as long as it is able to execute each of its cycles in Step 1 fast enough. In particular, suppose that we have predicted the state at time t and we are now at time t. As we move forward to time $t+1$, we obtain evidence $\mathbf{o}^{t+1}$. Our goal is to compute probabilities of variables at time $t+1$ in time and space which are independent of t. As long as we have saved the state of Σ at Step 1 of the the previous time slice t, all we have to do is execute another cycle of Step 1, followed by Step 2 to compute marginals over variables X^{t+1}. This can be clearly accomplished in time and space which are independent of time t.

Suppose we have an observation sequence $\mathbf{o} = \mathbf{o}^1, \ldots, \mathbf{o}^T$. Suppose further that we are interested in computing the marginals over variables at slice 1 given $\mathbf{o}$. Then we will simply construct a DBN over the span T, eliminate variables at all slices except for 1, then eliminate variables at slice 1 to compute marginals. This leads to algorithm VE_DG:

1. Let Σ contain CPTs of slices $1, \ldots, T$. Enter evidence into tables Σ.
2. For $i = T$ to 2: eliminate variables at slice i.
3. Apply variable elimination to Σ to compute marginals for variables X^1.

Note that the algorithm takes memory which is linear in T, but the following observation tells us how it can be modified so its memory requirements are independent of T:

Proposition 2 *In iteration i of Step 2 of* VE_DG*, the tables corresponding to time slices $1, \ldots, i-1$ are irrelevant to the elimination process.* □

Again, this means that there is no need to initialize Σ to the set of all tables. Instead we can dynamically introduce the tables as we need them, leading to the following constant-space version of VE_DG:

1. For $i = T$ to 2:
   (a) generate CPTs of slice i, enter evidence into them, and add them to Σ.
   (b) eliminate variables at slice i.
2. Generate CPTs of slice 1, enter evidence into them, and add them to Σ. Apply variable elimination to Σ to compute marginals for variables X^1.

This algorithm employs backward slice-by-slice elimination orders, which width can be bound as follows.

Theorem 2 *Let w_t be the temporal bandwidth of a DBN and let π be the elimination order used by algorithm* VE_DG *on that DBN. Then the width of order π, w, is bounded as follows: $w_t \leq w < w_t + n$, where $T > 1$ and the bounds are tight.*[5]

A *temporal edge* in a DBN is one which extends from one time slice to another. A *persistence edge* is a temporal edge that extends from one variable at a time slice to the same variable at the next time slice. When all temporal edges are persistence edges, the temporal bandwidth is never greater than the interface width.

[5] It is possible for the width of an order used by algorithm VE_PD to be less than w_t.

Theorem 3 *Suppose that w_t and w_i are the temporal bandwidth and interface width, respectively, of a DBN where all temporal edges are persistence edges. Then $w_t \leq w_i$.*

We now combine algorithms VE_PD and VE_DG, leading to algorithm DVE. That is, given an observation sequence $\mathbf{o}_1, \ldots, \mathbf{o}_T$, we compute $Pr(X^t|\mathbf{o}_1, \ldots, \mathbf{o}_T)$ for all X^t, where $1 < t < T$.

1. For $i = 1$ to $t - 1$:
   (a) generate the CPTs of slice i, enter evidence into them, and add them to Σ.
   (b) eliminate variables in Σ except those belonging to the forward interface of slice i.
2. For $i = T$ downto $t + 1$:
   (a) generate the CPTs of slice i, enter evidence into them, and add them to Σ.
   (b) eliminate variables at slice i.
3. Generate the CPTs of slice t, enter evidence into them, and add them to Σ. Apply variable elimination to Σ to compute marginals for a total of m variables X^t.

Algorithm DVE is then simply a classical variable elimination algorithm for computing marginals of m variables at slice t in a DBN. The only difference between this algorithm and a standard elimination algorithm is that it uses a constrained class of elimination orders. The advantage of using this class of elimination orders is that it makes the space complexity independent of the DBN temporal span. Moreover, we can establish the following about the algorithm's complexity:

Theorem 4 *Let π be the elimination order used by algorithm DVE, w be its width. The space complexity of the algorithm is then $O(n \exp(w))$ and its time complexity is $O(n \exp(w)(T + m))$, where n is the number of variables at each time slice. Moreover, $w_t \leq w < w_t + n$, where w_t is the temporal bandwidth of given DBN.*

Note that contrary to a jointree algorithm (such as the one used in dHUGIN [10]), the time complexity of DVE depends on the number of marginals m that we wish to compute at time slice t since Step 3 takes $O(nm \exp(w))$ time. We can deal with this, however — therefore, dropping the time complexity of Step 3 to $O(n \exp(w))$—by a restricted usage of jointrees. All we have to do is perform Step 3 using jointrees, by first constructing an undirected graph G as follows:

- G will contain a node for each variable that appears in the set of tables Σ.
- G will include an edge between two variables whenever they appear in the same table in Σ.

We can then construct a jointree J of G, quantify it using the tables in Σ, and then perform jointree propagation (which will take only $O(n \exp(w))$), therefore, dropping the total running time to $O(nT \exp(w))$. One should note also that for a given DBN, one can construct the jointree structure in a pre-processing stage, leaving only the quantification to run-time, therefore reducing the overhead considerably.[6]

5 Experimental Results

We have two main goals in this section. First, to give an idea of what kind of DBNs are accessible to structure-based algorithms, at least as compared with a simple collapsing algorithm that is exponential in double the slice size. Second, to provide an experimental comparison between: the min-fill heuristic as applied to an "unrolled" DBN, and a constrained min-fill heuristic applied in the context of algorithms VE_PD′, VE_PD and VE_DG. According to the min-fill heuristic, the weight of a variable is the number of edges we have to add in order to pairwise connect all its neighbors. To generate an elimination order according to this heuristic, we start with an undirected graph representing the moral graph of given Bayesian network. The heuristic first eliminates the variable with the smallest weight, connects all its neighbors, and then repeats the process until all variables have been eliminated. According to the constrainted min-fill heuristic, one proceeds according to the ordering proposed by the corresponding algorithm, appealing to the min-fill heuristic only when a choice of elimination is possible (for example, when eliminating the variables at one particular time slice in VE_PD').

We conducted three sets of experiments. In the first, we considered a number of DBNs that have been reported in the literature. In the second, we generated randomly a static Bayesian network, and then added a random set of persistence edges to construct a DBN. In the third set of experiments, we also generated static networks, but temporal edges were not restricted to persistence edges—that is, a variable at time t was connected to a variable chosen randomly at time $t-1$. For each set of experiments, we report the width of the elimination order constructed by the unconstrained min-fill heuristic; and the min-fill heuristic in

[6] Algorithm DVE can be modified to perform smoothing in constant space, but that would increase its time complexity to quadratic in T. We can adopt the technique given in [1], however, which allows us to use $O(\log T)$ space, and only increase the time complexity to $O(T \log T)$.

Network	min-fill	VE_PD'	VE_PD	$2n$	INTERFACE WIDTH	TEMPORAL BANDWIDTH
BAT [7]	14	9	8	52	9	8
DNA [1]	5	5	5	16	5	4
WATER [2]	11	10	10	16	8	8
MILDEW [10]	3	4	3	18	4	5

Table 1: Experimental results on realistic DBNs. We used 50-expansions in the experiments. n is the DBN slice size.

the context of VE_PD', VE_PD and VE_DG. We also report the interface width and temporal bandwidths for the considered networks.

Table 1 reports such widths for a few networks that have appeared in the literature on DBNs [2, 1, 7]. As is clear from these results, the VE_PD algorithm dominates VE_PD' and the standard elimination algorithm using min-fill.[7] Moreover, in all cases, the structure–based approach is clearly much better than a slice-collapsing approach.

We then considered randomly generated DBNs. Given the size of a time slice n, we randomly generate a static network where an edge from node i to node $j > i$ is added randomly with a probability of .1. Given a persistence factor $0 < p \leq 1$, we then add persistence edges to $\lfloor pn \rfloor$ nodes chosen randomly at each time slice. Table 2 reports our results for varying n and p. As is clear from this table, VE_PD dominates VE_PD' and the standard elimination algorithm using min-fill. In fact, the difference is quite significant, leading to exponential savings in many cases. Note also that VE_PD and VE_DG are very close in all networks.

In the last set of experiments, we added temporal edges at random between slices but we did not restrict such edges to persistence ones. That is, a set of variables were selected at random at slice t, according to probability p, and each one was made a child of a randomly-chosen variable at slice $t-1$. Table 3 reports our results for varying n and p. As is clear from this table, VE_PD also dominates VE_PD' and the standard elimination algorithm. The difference between VE_PD' and VE_PD is even much more significant for this class of DBNs, although the standard elimination algorithm does much better than VE_PD' in this case. Again, VE_PD and VE_DG are very close in all networks.

The difference in quality between elimination orders constructed using the popular min-fill heuristic, and elimination orders computed in the context of VE_PD and VE_DG is sometimes surprisingly very large. The difference in quality suggests that one may use constrained elimination orders in this context, even when space complexity is not an issue.

[7] We have actually observed that the quality of elimination orders computed by min-fill are much worse for larger time spans than they are for smaller spans.

n	p	min-fill	VE_PD'	VE_PD	VE_DG	INTERFACE WIDTH	TEMPORAL BANDWIDTH
20	0.3	9.1 (4.4)	10.3 (3.2)	7.4 (2.7)	7.3 (2.6)	10.3 (3.2)	5.9 (2.3)
30		15.5 (6.7)	16.4 (3.0)	12.2 (2.5)	12.1 (2.4)	16.2 (2.9)	8.4 (2.5)
40		32.1 (12.0)	29.1 (5.5)	21.7 (2.8)	21.5 (3.1)	27.6 (4.2)	13.6 (3.2)
20	0.5	18.4 (4.0)	14.8 (2.5)	11.2 (1.5)	11.2 (1.5)	14.6 (2.5)	10.1 (1.6)
30		30.8 (10.6)	21.2 (4.0)	18.3 (3.5)	18.2 (3.9)	20.9 (3.4)	14.5 (3.1)
40		59.1 (16.9)	34.7 (3.5)	27.2 (3.1)	27.5 (2.6)	31.5 (2.8)	20.7 (2.8)
20	1.0	48.6 (8.7)	20.4 (0.7)	20.4 (0.7)	20.1 (0.3)	20.0 (0.0)	20.0 (0.0)
30		78.5 (13.7)	31.9 (1.1)	31.9 (1.1)	30.6 (0.9)	30.0 (0.0)	30.0 (0.0)
40		109.1 (16.1)	46.4 (2.4)	46.4 (2.4)	43.0 (1.9)	40.0 (0.0)	40.0 (0.0)

Table 2: The orders were computed using 20-expansions. We are reporting the average and standard deviation (in parenthesis) over 10 cases.

6 Conclusion

We have achieved several goals in this paper. First, we spelled out precisely one of the simplest constant-space algorithms for predictive and diagnostic inference in DBNs and analyzed its complexity using a new structural parameter known as the temporal bandwidth. The algorithm is based on the combination of two new classes of elimination orders, one for prediction and another for diagnosis. Second, we compared the quality of these new classes of elimination orders with the more well-known class of slice-by-slice elimination orders employed by the dHUGIN algorithm [10]. We made explicit the computational limits of any algorithm based on these classes of elimination orders, and showed that algorithms based on the proposed elimination orders have better performance on a number of realistic and randomly generated networks. Finally, we presented experimental results suggesting that some constrained versions of the min-fill elimination heuristic tend to outperform the unconstrained version of the same heuristic. This is both surprising and encouraging. It means that, contrary to what one would expect, constrained eliminated orders that achieve constant-space complexity in DBNs do not seem to practically entail a higher computational price. Going further with this observation: Even if one is not concerned with constant-space complexity, it sometimes make more sense to use constrained elimination orders instead of unconstrained ones in the context of DBNs.

n	p	min-fill	VE_PD'	VE_PD	VE_DG	INTERFACE WIDTH	TEMPORAL BANDWIDTH
20	0.3	5.7 (2.1)	8.9 (3.3)	5.5 (1.7)	5.6 (1.9)	8.9 (3.3)	4.6 (1.7)
30		10.6 (1.8)	14.9 (2.9)	10.2 (1.7)	10.3 (1.7)	14.9 (2.9)	7.6 (1.3)
40		16.8 (3.7)	20.1 (4.0)	15.9 (2.8)	16.2 (3.0)	19.8 (4.2)	9.1 (1.6)
20	0.5	10.7 (3.3)	14.5 (2.4)	9.5 (1.9)	9.6 (2.0)	14.5 (2.4)	8.0 (2.0)
30		15.5 (3.1)	22.1 (4.0)	14.5 (3.0)	14.5 (3.1)	22.1 (4.0)	11.2 (1.7)
40		22.8 (3.3)	29.6 (4.2)	21.1 (1.8)	21.5 (2.8)	29.1 (4.0)	15.0 (2.4)
20	1.0	17.5 (5.1)	20.0 (0.0)	14.3 (2.0)	14.7 (2.0)	20.0 (0.0)	11.9 (1.8)
30		28.0 (4.5)	30.2 (0.6)	23.0 (1.8)	23.1 (1.7)	30.0 (0.0)	19.1 (2.1)
40		37.6 (2.3)	41.1 (1.2)	32.8 (2.1)	33.0 (1.9)	40.0 (0.0)	25.1 (1.8)

Table 3: The orders were computed using 20-expansions. We are reporting the average and standard deviation (in parenthesis) over 10 cases.

Acknowledgement

This work has been partially supported by a MURI research grant number N00014-00-1-0617.

A Proofs

We will use $\mathbf{V}^i$ to denote the variables at slice i, $\mathbf{I}_f^i$ to denote the forward interface variables at slice i, and $\mathbf{I}^i$ to denote the interface variables at slice i.

Definition 3 *Given a set of tables $\phi_1, \ldots, \phi_n$, the $\mathbf{X}$–graph is obtained by including a node N_i for each table ϕ_i and adding an edge between two nodes whenever their corresponding tables share a variable in $\mathbf{X}$. A set of tables is $\mathbf{X}$-connected iff its $\mathbf{X}$-graph is connected.*

The following lemma is central to most of the proofs.

Lemma 1 *Let $\phi_1, \ldots, \phi_n$ be a set of $\mathbf{X}$–connected tables defined over variables $\mathbf{X} \cup \mathbf{Y}$ where $\mathbf{X} \cap \mathbf{Y} = \emptyset$. After eliminating all variables in $\mathbf{X}$, we will be left with a single table over variables $\mathbf{Y}$. And if X is the last variable eliminated in $\mathbf{X}$, then a table over $\mathbf{Y} \cup \{X\}$ will be constructed during the elimination process.*

Let G be the $\mathbf{X}$-graph of a set of tables Σ and let G' be the $\mathbf{X}$-graph of $\Sigma \setminus \{X\}$: the set of tables that result from eliminating variable X from the tables in Σ.

To prove the first part, it suffices to show that if G is connected, then G' is connected. If we can show this, then starting with an $\mathbf{X}$-connected graph G, and eliminating the variables in $\mathbf{X}$ one by one, we will end up with an $\mathbf{X}$-connected

graph G' of $\Sigma \setminus \mathbf{X}$. Since $\Sigma \setminus \mathbf{X}$ contains no variables in $\mathbf{X}$, then G' must contain a single node. Therefore, $\Sigma \setminus \mathbf{X}$ must contain a single table (over variables $\mathbf{Y}$).

Let G be the $\mathbf{X}$-graph of tables Σ and let G' be the $\mathbf{X}$-graph of tables $\Sigma \setminus \{X\}$. Moreover, let $\mathbf{N}$ be the nodes in G corresponding to tables $\Gamma \subseteq \Sigma$ that contain variable X. Nodes $\mathbf{N}$ must form a clique in G. Moreover, G' must be the result of:

1. replacing nodes $\mathbf{N}$ in G by a new node N which corresponds to table $\mathsf{sumout}(\prod_{\phi \in \Gamma} \phi, X)$;
2. replacing every edge in G between a node in $\mathbf{N}$ and a node $M \notin \mathbf{N}$ by an edge in G' between nodes N and M (by definition of an $\mathbf{X}$-graph).

Hence, if G is connected, then G' must be connected.

To prove the second part of the lemma, note that after eliminating every variable in $\mathbf{X} \setminus \{X\}$, we will have a set of tables which $\mathbf{X}$-graph is connected. Hence, each of the remaining tables must contain X. Therefore, when eliminating X, all of these tables must be multiplied together, leading to a single table over $\{X\} \cup \mathbf{Y}$ (before X is summed-out from the table). □

Lemma 2 *Let Σ be the CPTs of variables $\mathbf{X}$ in a Bayesian network. If variables $\mathbf{X}$ are connected in the Bayesian network, then tables Σ are $\mathbf{X}$-connected.*

Let G be the $\mathbf{X}$-graph of tables Σ. There is a one-to-one correspondence between the nodes of G and variables $\mathbf{X}$: each node in G corresponds to the CPT of a variable in $\mathbf{X}$. Moreover, for every edge $X_i \rightarrow X_j$ that connects variables $X_i, X_j \in \mathbf{X}$ in the Bayesian network, we must have an edge that connects their corresponding tables in G since X_i must belong to these two tables. Hence, the nodes of G must be connected since the variables in $\mathbf{X}$ are connected. □

Lemma 3 *Let Σ be the CPTs of variables at slice i, $\mathbf{V}^i$, and the CPTs of their children at slice $i+1$. Then tables Σ are $\mathbf{V}^i$-connected.*

Let G be the $\mathbf{V}^i$-graph of tables Σ. The subgraph of G corresponding to the CPTs of variables $\mathbf{V}^i$ is connected by Lemma 2 since variables $\mathbf{V}^i$ are connected in the Bayesian network. Each CPT at slice $i+1$ which belongs to a child of a variable V^i at slice i must mention variable V^i. Hence, the CPT of this child must be connected to the CPT of V^i in graph G. Hence, G is connected. □

Proof of Proposition 1

When eliminating variable X^i, the only tables that matter are those in which X^i appears. These are the tables associated with X^i and its children. The children of X^i can only appear in slices i and $i+1$. Hence, no table at slices $i+2, \ldots$ will ever contain X^i. These slices and their tables are therefore irrelevant to the elimination of variables X^i. □

Proof of Theorem 1

In Step 2, the elimination process will only involve variables $\mathbf{V}^i \cup \mathbf{I}^{i+1}$, and in Step 3, it will only involve variables $\mathbf{V}^T$. Hence, $w < w_i + n =\mid \mathbf{V}^i \cup \mathbf{I}^{i+1} \mid$. Consider now a DBN where a node at slice $i+1$ has all other nodes at slices $i+1$ and i as its parents. In this case, $w_i = n$. Since the treewidth of this network is $2n-1$, we must have $w = w_i + n - 1$. Hence, the upper bound is tight.

In Step 2, the elimination process is guaranteed to construct a table over $\mid \mathbf{I}^2 \mid +1$ variables. This can be shown as follows. In the first iteration of Step 2, the set of tables Σ involved in the elimination process are the CPTs of variables $\mathbf{V}^1$ and their children at slice 2. The variables appearing in these tables are $\mathbf{V}^1 \cup \mathbf{I}^2$. Invoking Lemma 1, with $\mathbf{X} = \mathbf{V}^1$ and $\mathbf{Y} = \mathbf{I}^2$, a table over $\mid \mathbf{I}^2 \mid +1$ variables is constructed in the process of eliminating variables $\mathbf{V}^1$ since tables Σ are $\mathbf{X}$-connected (by Lemma 3). Hence, $\mid \mathbf{I}^2 \mid = w_i \leq w$ and the bound is tight if we consider a chain network, where $w_i = w = 1$. □

Proof of Proposition 2

The CPTs associated with variables at slices $j < i$ can never mention variables at slice i. Hence, they can never be involved when eliminating variables at slice i. □

Proof of Theorem 2

In Step 1, the only variables involved in the elimination process are $\mathbf{V}^i \cup \mathbf{I}_f^{i-1}$, and in Step 2, the only variables involved in the elimination process are $\mathbf{V}^1$. Hence, $w < w_t + n =\mid \mathbf{V}^i \cup \mathbf{I}_f^{i-1} \mid$. Consider now a DBN where a node at slice $i+1$ has all other nodes at slices $i+1$ and i as its parents. In this case, $w_t = n$. Since the treewidth of this network is $2n-1$, we must have $w = w_t + n - 1$. Hence, the upper bound is tight.

In Step 1, the elimination process is guaranteed to construct a table over $\mid \mathbf{I}_f^{T-1} \mid +1$ variables. This can be shown as follows. In the first iteration of Step 1, the tables Σ involved in the elimination process are the CPTs of variables $\mathbf{V}^T$ and they contain the variables $\mathbf{V}^T \cup \mathbf{I}_f^{T-1}$. By invoking Lemma 1, with $\mathbf{X} = \mathbf{V}^T$ and $\mathbf{Y} = \mathbf{I}_f^{T-1}$, a table must be constructed over $\mid \mathbf{I}_f^{T-1} \mid +1$ variables after eliminating variables $\mathbf{V}^T$, since tables Σ are $\mathbf{X}$-connected (by Lemma 2). Hence, $\mid \mathbf{I}_f^{T-1} \mid = w_t \leq w$ and the bound is tight if we consider a chain network where $w_t = w = 1$. □

Proof of Theorem 3

When all temporal edges are persistent edges, the interface at time t will contain all variables in the forward interface of time t in addition to their parents at time t. Hence, $w_t \leq w_i$. □

Proof of Theorem 4

In Step 1, the elimination process will involve variables $\mathbf{V}^1$ in the first iteration, and variables $\mathbf{I}_f^{i-1} \cup \mathbf{V}^i$ for $i > 1$. In Step 2, the elimination process will involve tables over variables $\mathbf{V}^i \cup \mathbf{I}_f^{i-1}$, and is guaranteed to construct a table over $\mid \mathbf{I}_f^{i-1} \mid +1$ variables (see proof of Theorem 2). In Step 3, the elimination process will involve tables over variables $\mathbf{I}_f^{t-1} \cup \mathbf{V}^t$. Hence, we are guaranteed to construct a table over $w_t + 1$ variables, and no constructed table will have more than $n + w_t$ variables (since $\mid \mathbf{V}^i \cup \mathbf{I}_f^{i-1} \mid = w_t + n$), leading to $w_t \leq w < n + w_t$.

We now show that the number of tables maintained by the algorithm during any stage of its execution is $O(n)$, which establishes the space complexity (since the size of each table is $O(\exp(w))$). Initially, we have no tables and hence the number of tables is $O(n)$. We will now show that this is maintained during the algorithm execution:

- Step 1(a) will introduce n tables. Step 1(b) will consume all such tables except those whose variables are in $\mathbf{I}_f^i$ and will introduce $O(n)$ tables over variables $\mathbf{I}_f^i$. In iterations $i > 1$, Step 1(b) will in addition consume all tables that are left over from iteration $i - 1$. Specifically, any CPT introduced by Step 1(a) of iteration $i - 1$, or constructed by Step 1(b) of iteration $i - 1$, and not consumed by that iteration must have all its variables in $\mathbf{I}_f^{i-1}$ and will be consumed when such variables are eliminated in iteration i. Hence, the number of tables will continue to be $O(n)$ after each iteration of Step 1.

- When $i = T$, Step 2(a) will introduce n tables and Step 2(b) will consume them, while introducing a table over variables $\mathbf{I}_f^{T-1}$. When $i < T$, Step 2(a) will introduce n tables, and Step 2(b) will consume them, in addition to the table over $\mathbf{I}_f^i$ constructed by iteration $i + 1$. Step 2(b) will also introduce a table over variables $\mathbf{I}_f^{i-1}$. Hence, the number of tables will continue to be $O(n)$ after each iteration of Step 2.

- If we enter Step 3 with $O(n)$ tables, then we will continue to have $O(n)$ tables after CPTs have been introduced in this step. Moreover, variable elimination will never increase the number of given tables. Hence, the number of tables will continue to be $O(n)$ during the elimination process.

The time complexity follows since we eliminate $O(nT)$ variables in Steps 1 and 2, and we eliminate $O(nm)$ variables in Step 3, each creating a table of size $O(\exp(w))$. □

References

[1] J. Binder, K. Murphy, and S. Russell. Space–efficient inference in dynamic probabilistic networks. In *Proc. International Joint Conference on Artifical*

Intelligence (IJCAI), 1997.

[2] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998.

[3] C. Cannings, E. A. Thompson, and M. H. Skolnick. Probability functions on complex pedigrees. *Adv. Appl. Prob.*, 10:26–61, 1978.

[4] Rina Dechter. Constraint networks. In S. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, pages 276–285. John Wiley and Sons, 1992.

[5] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 211–219, 1996.

[6] Rina Dechter. Mini-buckets: a general scheme for approximation in automated reasoning. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI)*, pages 1297–1302, 1997.

[7] J. Forbes, Tim Huang, Keiji Kanazawa, and Stuart Russell. The batmobile: Towards a bayesian automated taxi. In *Proc. International Joint Conference on Artifical Intelligence (IJCAI)*, 1995.

[8] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[9] Finn V. Jensen. *An Introduction to Bayesian Networks.* Springer Verlag, New York Inc., 1996.

[10] U. Kjaerulff. A computational system for dynamic time-sliced Bayesian networks. *International Journal of Forecasting*, 1995.

[11] Nevin Lianwen Zhang and David Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research*, 5:301–328, 1996.