# Anytime Anyspace Probabilistic Inference

**Fabio Tozeto Ramos**

**Fabio Gagliardi Cozman**

*Escola Politécnica, Universidade de São Paulo*
*Av. Prof. Mello Moraes 2231, Cidade Universitária*
*05508-900,*
*São Paulo, SP - Brazil*
*{fabioram,fgcozman}@usp.br*

## ABSTRACT

*This paper investigates methods that balance time and space constraints against the quality of Bayesian network inferences — we explore the three-dimensional spectrum of "time × space × quality" trade-offs. The main result of our investigation is the* adaptive conditioning *algorithm, an inference algorithm that works by dividing a Bayesian network into sub-networks and processing each sub-network with a combination of exact and anytime strategies. The algorithm seeks a balanced synthesis of probabilistic techniques for bounded systems. Adaptive conditioning can produce inferences in situations that defy existing algorithms, and is particularly suited as a component of bounded agents and embedded devices.*

## 1. Introduction

One of the central characteristics of bounded systems is their flexibility to cope with simultaneous limitation in several resources [31, 60]. In this paper we concentrate on probabilistic reasoning for bounded systems, exploring algorithms for Bayesian network inference under time *and* space constraints. We require that such algorithms produce a solution at any

International Journal of Approximate Reasoning 1994 11:1–158

655 Avenue of the Americas, New York, NY 10010 0888-613X/94/$7.00

given stopping time (they must be *anytime*) and that they make the best possible use of available memory (they must be *anyspace*). We therefore look into a three-dimensional spectrum of "time × space × quality" trade-offs. Existing methods, reviewed in Section 4, usually face either "time × space" trade-offs or "time × quality" trade-offs, typically fixing one of the dimensions as more important. This paper tries to build a more complete picture of bounded probabilistic inference — we want to encode a number of trade-offs in an organized set of rules.

The main result of our investigation is the *adaptive conditioning* algorithm, described in Section 5. The algorithm decomposes a Bayesian network into smaller networks and combines conditioning, clustering and anytime operations in the sub-networks. These strategies are used together to explore, in an organized fashion, the vast space of "time × space × quality" trade-offs. In doing so, adaptive conditioning provides a useful panoramic view covering many facets of Bayesian network algorithms.

Adaptive conditioning is particularly suited for bounded agents that engage in time-sensitive negotiations, and to embedded devices found in robots and smart appliances. As every computing system has limitations in memory and available time, our methods should be of use in connection to any "large" probabilistic model. In fact, we show later that adaptive conditioning can produce exact inferences for Bayesian networks that defy existing algorithms.

The paper is organized as follows. Sections 2, 3 and 4 review concepts, ideas and relevant literature; together these sections present the background against which the adaptive conditioning algorithm is developed. Section 5 describes the adaptive conditioning algorithm itself. Section 6 contains several experiments with the algorithm, and Section 7 presents our concluding comments.

## 2. Probabilistic reasoning with Bayesian networks

Bayesian networks provide both a compact method to represent probability distributions and a powerful tool for uncertainty management. Examples of Bayesian networks can be found in expert systems for medical decisions [1, 2], technical support troubleshooters [34], decision-theoretic systems to interpret live telemetry [33], genetic research [24], speech recognition systems [67], data compression methods [17], and diagnostic systems in industrial plants [53].

A Bayesian network $\mathcal{N}$ consists of a directed acyclic graph, a set of variables and a set of conditional probability distributions (a few graph-theoretic concepts are used in this paper: nodes, edges, directed and undirected graphs, paths and cycles, and polytrees). Given a directed acyclic

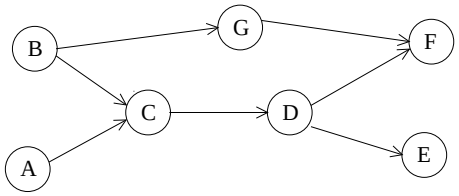



**Figure 1.** A Bayesian network associated with distributions $Pr(A)$, $Pr(B)$, $Pr(C|A,B)$, $Pr(D|C)$, $Pr(E|D)$, $Pr(F|D,G)$, and $Pr(G|B)$.

graph, the *parents* of node $\alpha$ (the nodes with directed edges pointing to $\alpha$) are indicated by $pa(\alpha)$.

In a Bayesian network every node is associated with a variable $X_i$. In this paper every variable is categorical (has a finite number of values), and we use the terms "node" and "variable" interchangeably. Every variable in a Bayesian network is assumed to be independent of its nonparents nondescendants given its parents, implying the following joint probability distribution [48]:

$$Pr(X_1, \ldots, X_n) = \prod_{i=1}^{n} Pr(X_i|pa(X_i)). \tag{1}$$

That is, a Bayesian network represents a unique joint distribution that factorizes as Expression (1). Every variable is thus associated with a single conditional distribution $Pr(X_i|pa(X_i))$. Figure 1 shows an example network and indicates the probability distributions.

Given a Bayesian network, the computation of a posterior probability distribution is usually called an *inference*. That is, we select a set of *query* variables $\mathbf{X}_Q$ and a set of *observed* variables $\mathbf{X}_E$, and we must compute

$$\begin{aligned} Pr(\mathbf{X}_Q|\mathbf{X}_E) &= \frac{\sum_{X_i \notin \{\mathbf{X}_Q, \mathbf{X}_E\}} \prod_i Pr(X_i|pa(X_i))}{\sum_{X_i \notin \{\mathbf{X}_E\}} \prod_i Pr(X_i|pa(X_i))} \\ &\propto \sum_{X_i \notin \{\mathbf{X}_Q, \mathbf{X}_E\}} \prod_i Pr(X_i|pa(X_i)). \end{aligned} \tag{2}$$

We assume that $\mathbf{X}_Q$ are $\mathbf{X}_E$ are disjoint, and we note that in Expression (2) the values of variables in $\mathbf{X}_E$ are observed and therefore fixed. For any given inference, it is possible to identify in polynomial time a set of variables that do not affect Expression (2), using *d-separation* [27].

The general problem of computing inferences (even approximate ones) in Bayesian networks is NP-hard [7, 12]. Significant special cases are inference in polytrees [48] and approximate inference by sampling methods in networks with non-zero probabilities [12].

Inference algorithms are reviewed in Section 4. Several of these algorithms rely on *junction trees* [36, 10]. Take a directed acyclic graph $\mathcal{G}$ with a set of nodes $V$. A *junction tree* of $\mathcal{G}$ is an undirected graph where nodes are subsets of $V$, such that every node $\alpha$ of $\mathcal{G}$ and the parents of $\alpha$ are contained in some node of the junction tree, and such that the following property holds: Given nodes $\gamma_i$ and $\gamma_j$ of the junction tree, the intersection $\gamma_i \cap \gamma_j$ is contained in every node of the junction tree in the unique path from $\gamma_i$ to $\gamma_j$. Each node of a junction tree is called a *cluster*; if an edge directly connects nodes $\gamma_1$ and $\gamma_2$ in a junction tree, then $\gamma_1 \cap \gamma_2$ is a *separator*. Figure 3 shows a number of junction trees.

## 3. Anytime anyspace behavior

Bounded systems have been the object of much attention in the artificial intelligence literature. A general observation is that bounded systems must settle for satisficing solutions [60]. To obtain such satisficing solutions, one strategy is to employ meta-reasoning [54], for example to select reasoning algorithms using decision theoretic principles [31]. Another strategy is to produce a list of algorithms that can solve a problem (each algorithm representing different trade-offs between time, space, and quality), and then to choose the algorithm that seems best suited for any set of constraints [26]. Yet another strategy to cope with boundedness is to design algorithms that can adapt themselves to varying levels of computational resources — *anytime* algorithms follow this strategy [18]:

DEFINITION 1. *An algorithm is* anytime *if it can produce a solution in a given time $T$ and the quality of solutions improve with time after $T$.*

An anytime algorithm may need some "bootstrapping" time $T$, but after $T$, the more time, the better [18]. Anytime algorithms seem particularly well suited for real-time systems and embedded devices, where soft and hard time constraints are routinely employed [26].

In many situations, memory may be as scarce as time, either because we must solve a large problem, or because we can only use small computing devices (such as handhelds or industrial controllers). We must therefore consider algorithms that use their available space with flexibility (again we allow a "bootstrapping" quantity $M$):

DEFINITION 2. *An algorithm is* anyspace *if it can improve its performance with increasing space, assuming that the available memory is larger than some minimal amount $M$.*

Definitions 1 and 2 capture important differences in the concepts of anytime and anyspace behavior. An anytime algorithm must dynamically improve results as time becomes available, while an anyspace algorithm is usually informed about memory availability in its starting phase, and does not have to handle memory changes during operation.

The focus of this paper is a combination of the previous situations. We assume that a bounded system must perform an inference within a given time $T$ using memory $M$, with the understanding that more time may become available as the inference is processed. An approximation may be generated at first, but the quality of the approximation should improve with time. Time, space, and quality should be properly balanced.

## 4. Inferences in Bayesian networks

This section presents a review of existing inference algorithms from the perspective of bounded systems, as we will later use ideas from most algorithms in our own methods (Section 5). We start with a brief overview of general exact and approximate algorithms; in Sections 4.4 and 4.5 we discuss a few algorithms that are closely related to this work.

### 4.1. Exact algorithms

Exact algorithms can be classified in two groups: algorithms based on conditioning, and algorithms based on clustering — with a "third group" represented by Pearl's propagation algorithm for polytrees, the only polynomial exact inference algorithm for Bayesian networks [48].

The *cutset conditioning* algorithm, also known as the *loop cutset* algorithm, exploits the fact that edges out of a node are "broken" if the node is observed (Section 5.2 formalizes such operations). The algorithm selects a set of nodes (the *loop cutset*) that, once observed, "breaks" every cycle in a graph. Every instantiation of the cutset is then considered; for each one of them, Pearl's propagation algorithm is employed. The result is an algorithm that uses a relatively small amount of memory, but takes exponential time on the size of the loop cutset. A few algorithms address this exponential growth by organizing loop cutsets in various forms [14, 22, 50, 58]. All of them essentially compute probability values of the form $Pr(x, c)$, where $x$ is an instance of variables of interest and $c$ is an instance of the loop cutset; the probability $Pr(x)$ is then computed through

$$\sum_{c} Pr(x, c). \tag{3}$$

In clustering algorithms, variables are grouped in potentially large clusters, a junction tree is built, and a propagation scheme on the junction tree produces inferences. The Lauritzen-Spiegelhalter algorithm [43] and the Shafer-Shenoy algorithm [59] are two different ways to organize this propagation. Many variants of clustering methods have appeared since these two basic algorithms were derived (several variants are discussed in [29]); all of them use considerable memory to cut processing time. A few algorithms also proceed by "grouping" variables but are not directly related to the Lauritzen-Spiegelhalter or the Shafer-Shenoy algorithms: the family of variable elimination algorithms (discussed in Section 4.4), Li and D'Ambrosio's *SPI* algorithm [44], Shachter's arc-reversal/node-reduction algorithm [57], and differential inference algorithms [15] are examples.

### 4.2. Approximate algorithms

Approximate algorithms for Bayesian network inference can be divided in a few groups. Most approximate algorithms have an "anytime" character, as results can be refined when additional time is available.

- *Stochastic approximations* are widely used in large, dense networks. Methods are generally divided into forward sampling and MCMC methods [6, 12, 23, 25, 28, 55]. They can offer polynomial time approximations when probability values are non-zero [12], but they display poor performance when probability values are extreme.
- *Model simplifications* range from the removal of weak dependencies [40] to cardinality reduction in probability distributions [62, 5]. Simplifications may also affect secondary structures such as junction trees, as demonstrated by the the mini-buckets framework [20].
- *Partial instantiation* algorithms approximate the summation in Expression (2) using only a number of terms. Examples are *bounded conditioning* [32], and *term computation* [13] (which we use and discuss in more detail later), Poole's conflict-based [51] and Henrion's search-based methods [30].
- *Loopy propagation* uses Pearl's propagation algorithm in networks with cycles, attempting to gradually improve the quality of inferences [47, 61, 64]. Little is known about convergence of loopy propagation, and lack of convergence has been observed in some situations [47, 61].

### 4.3. Combinations of exact and approximate inferences

There has been some effort in combining exact and approximate algorithms; for example, the use of Gibbs sampling inside clusters [41], the combination of clustering and stochastic approximations in dynamic models [23], and some of the anytime algorithms discussed later.

### 4.4. Variable elimination and adaptive variable elimination

Given our later use of the variable elimination algorithm, we briefly sketch the algorithm and its associated terminology. This algorithm has appeared in artificial intelligence in several forms [19, 66], and has roots in pedigree analysis in genetics [4].

Variable elimination computes Expression (2) by interchanging summations and products. First, select an ordering for all variables that must be summed out in Expression (2). Eliminate one of these variables at a time; to eliminate the first variable, select all those probability distributions that contain the first variable, multiply these functions together and sum the first variable out. Repeat this process until all variables in the ordering have been eliminated. We can imagine that every variable is associated with a *bucket* of functions and the buckets are processed sequentially [19]. The complexity of these operations depends on the ordering of variables; finding the best ordering is NP-hard, so heuristic methods are used in practice [37, 65]. Variable elimination can be generalized to incorporate properties of the Shafer-Shenoy algorithm [3] and of the Lauritzen-Spiegelhalter algorithm [11].

Variable elimination potentially consumes large amounts of memory. The first attempt to explicitly trade time and space in probabilistic inference was Dechter's conditioning-plus-variable-elimination scheme, which we call *adaptive variable elimination* [21]. The idea of adaptive variable elimination is simple: if the size of the functions in a bucket becomes too large, we must condition on some of the variables and handle smaller functions [21].[1] In the limit, the algorithm is reduced to brute force enumeration of instances. Adaptive variable elimination offers a "time $\times$ space" trade-off: For a given space, it takes a certain time; the more space, the less time is needed.

### 4.5. Conditioning with anytime and anyspace behavior

*Bounded conditioning* is inspired by the fact that Expression (3) can be approximated by an incomplete summation [32]; after computing a number of instances, we can bound Expression (3). This procedure is anytime as terms can always be computed and added to the summation if time is available. *Term computation* follows the same basic strategy, even though it does not directly rely on conditioning [13]: term computation uses heuristic techniques to find the "best" instantiations to compute, as we do in Section 5.6.

The most radical use of conditioning is represented by the *recursive decomposition* [46] and *recursive conditioning* [16] algorithms. These algorithms split a network into sub-networks, using conditioning to "break"

[1] Dechter also proposes an interesting variant: we can run a loop cutset algorithm *inside* a bucket, to save as much memory as possible for that bucket.

edges (as in Section 5.2). The sub-networks are recursively split, until networks containing a single variable are reached. The algorithm organizes the combination of conditioned sub-networks using tree structures called *dtrees*. Recursive decomposition is particularly relevant as it has been extended to *bounded recursive decomposition*, an anytime algorithm that produces probability bounds. The algorithm has an initialization phase, where intermediate results are produced and stored in caches; when an inference is requested, the algorithm uses some of the values in the caches to produce bounds. It would actually be possible to add anyspace behavior to anytime bounded conditioning by a more intense use of caches — in fact, the present paper can be understood as taking this very route.

Recursive conditioning expands the basic ideas of recursive decomposition, with a focus on anyspace behavior. If a dtree is "balanced", recursive conditioning use $\mathcal{O}(n)$ space and $\mathcal{O}(n \exp(w))$ time ($n$ is the number of variables and $w$ corresponds to the size of the largest separator in a clustering algorithm). Note that this time complexity is smaller than the time complexity of brute force instantiation, so that the introduction of balanced dtrees does present advantages. Second, if space beyond $\mathcal{O}(n)$ is available, recursive conditioning uses caches to store intermediate conditioning results, attaining complexity $\mathcal{O}(n \exp(w))$ when $\mathcal{O}(n \exp(w))$ space is available — exactly the complexity of standard variable elimination. The algorithm offers a "time $\times$ space" trade-off: For a given time, it takes a certain space; the more time, the less space is needed. Recursive conditioning is a truly flexible algorithm, possibly the most successful application of conditioning in an exact algorithm.

## 5. Adaptive conditioning

We cannot arbitrarily constrain time *and* space *and* then ask for exact answers; to look into situations that simultaneously require anytime and anyspace behavior, we must be prepared to trade inference quality for time and space.

The algorithms reviewed in the previous section suggest an endless number of strategies to trade time, space and quality. For example: we could use adaptive variable elimination to save space and, if we also had constraints in time, we could use sampling approximations in some buckets. Or we could start with recursive conditioning and add anytime behavior to it. Is there any way to organize this maze of options and produce a compact and coherent framework?

### 5.1. Sketch of adaptive conditioning

We wish to produce an inference algorithm that receives a Bayesian network, a constraint on space and a constraint on time, and produces an inference. The algorithm must adapt its operations to the available amount of space and promptly produce an answer (possibly of low quality) that can be improved if more time is available. The *adaptive conditioning* algorithm attempts to address these requirements in an organized fashion. In short, the idea is to divide a network using conditioning (to guarantee that memory constraints are met), and then to use clustering algorithms and anytime techniques to process sub-networks (to guarantee that time constraints are met). The following sketch is a starting point:

1. Use d-separation to discard variables that cannot affect the inference, obtaining a network with *requisite variables* only [56].

2. Based on space constraints, use conditioning to decompose the resulting network into sub-networks. The decomposition must guarantee that clustering algorithms can be run in every sub-network within available memory, but it need not decompose up to single nodes. The decomposition process is discussed in Section 5.3.

3. If there is some memory left after the division of the network, create caches to store intermediate results. The caching procedure is discussed in Section 5.4.

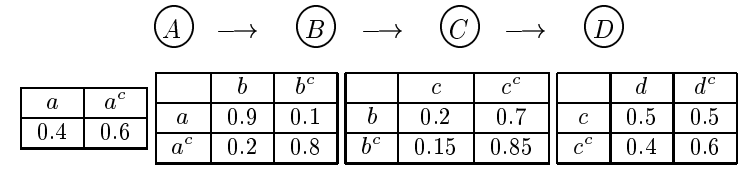
4. Now consider time constraints. If all sub-networks can be exactly processed, for all instantiations of conditioning variables, in the available time, process them with a clustering algorithm. Otherwise, process some sub-networks and instantiations in an anytime scheme for the available time (these comments are discussed in detail in Sections 5.5 and 5.6).

5. Combine instantiations, returning an exact or approximate answer.

The algorithm basically operates in two phases. The *planning phase* is responsible for steps 1, 2 and 3 (Sections 5.3 and 5.4). The *execution phase* is responsible for steps 4 and 5 (Sections 5.5 and 5.6). Before we look into these matters, Section 5.2 discusses some mathematical facts about conditioning.

### 5.2. The mathematics of adaptive conditioning

It is convenient to consider conditioning as an abstract operation that can "break" edges and "split" networks. When a node is observed, the edges off of the node are said to be *broken*. If an edge starts at node $X$, then the edge *is broken* by $X$. A Bayesian network $\mathcal{N}$ can be split in two



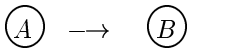

| $a$ | $a^c$ |
|---|---|
| 0.4 | 0.6 |

| | $b$ | $b^c$ |
|---|---|---|
| $a$ | 0.9 | 0.1 |
| $a^c$ | 0.2 | 0.8 |

| | $c$ | $c^c$ |
|---|---|---|
| $b$ | 0.2 | 0.7 |
| $b^c$ | 0.15 | 0.85 |

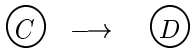
| | $d$ | $d^c$ |
|---|---|---|
| $c$ | 0.5 | 0.5 |
| $c^c$ | 0.4 | 0.6 |

(a) Network before conditioning.




| $a$ | $a^c$ |
|---|---|
| 0.4 | 0.6 |

| | $b$ | $b^c$ |
|---|---|---|
| $a$ | 0.9 | 0.1 |
| $a^c$ | 0.2 | 0.8 |




| $c$ | $c^c$ |
|---|---|
| 0.15 | 0.85 |

| | $d$ | $d^c$ |
|---|---|---|
| $c$ | 0.5 | 0.5 |
| $c^c$ | 0.4 | 0.6 |

(b) Network split after conditioning on $B = b^c$.

**Figure 2.** Decomposing a simple network by conditioning.

sub-networks $\mathcal{N}_1$ and $\mathcal{N}_2$ when we identify a set of nodes $\mathbf{C}$ such that every edge between $\mathcal{N}_1$ and $\mathcal{N}_2$ is broken by $\mathbf{C}$. The set $\mathbf{C}$ is called the *cutset* for $\mathcal{N}_1$ and $\mathcal{N}_2$, or simply the cutset, if no ambiguity can occur. The cutset $\mathbf{C}$ *splits* $\mathcal{N}$ into $\mathcal{N}_1$ and $\mathcal{N}_2$. For a sub-network $\mathcal{N}_i$, obtained by splitting a network $\mathcal{N}$ with cutset $\mathbf{C}$, the *local cutset* $\mathbf{C}_i$ is the set of variables in $\mathbf{C}$ and in $\mathcal{N}_i$. The symbol $Pr^{\mathcal{N}_i}(\cdot)$ denotes the probability $Pr(\cdot|\mathbf{C}\backslash\mathbf{C}_i)$ — that is, the probability in the sub-network $\mathcal{N}_i$ taken as a unit. Figure 2 shows an example.

The following theorem is a direct generalization of Expression (3).

THEOREM 1. *Let $\mathbf{C}$ be a cutset that splits a Bayesian network $\mathcal{N}$ into sub-networks $\mathcal{N}_i$, and $\mathbf{C}_i$ be the local cutset for $\mathcal{N}_i$. If $\mathbf{Q}$ and $\mathbf{E}$ are disjoint and contain respectively the query variables and the observed variables, with $\mathbf{Q}_i$ and $\mathbf{E}_i$ indicating the variables in $\mathbf{Q}$ and in $\mathbf{E}$ in $\mathcal{N}_i$, then*

$$Pr^{\mathcal{N}}(\mathbf{Q}|\mathbf{E}) = \sum_{\mathbf{C}\backslash\mathbf{Q}\cap\mathbf{C}} \prod_i Pr^{\mathcal{N}_i}(\mathbf{Q}_i \cup \mathbf{C}_i|\mathbf{E}_i). \tag{4}$$

This theorem indicates precisely the operations that must be repeated by adaptive conditioning. The first step of adaptive conditioning is to find a cutset; then, for each instantiation of the cutset, take each sub-network, compute $Pr^{\mathcal{N}_i}(\mathbf{Q}_i \cup \mathbf{C}_i|\mathbf{E}_i)$, and multiply these probabilities; at the end, add all products.

The theorem is completely general in that query variables can be distributed among various sub-networks; the result can be easily generalized to handle observed variables in the various sub-networks (compare this

discussion to recursive decomposition and recursive conditioning, where an inference is centered in a single variable).

### 5.3. Planning phase: computing a cutset

We now look into the planning phase of adaptive conditioning. This phase takes a Bayesian network and a memory constraint, and produces a cutset. We assume that our target is a cutset such that sub-networks can be processed by clustering algorithms. The rationale is that clustering algorithms are efficient in terms of running time; by guaranteeing that these algorithms can be used in sub-networks, we make the best use of available memory. We also avoid the trap of "saving too much memory" (using less than the available memory while incurring a large penalty in running time).

Our strategy is to form cutsets from the separators of the junction tree for the whole network, as separators do have the property of splitting networks. This strategy effectively controls memory consumption, as the memory required by clustering algorithms can be restricted to some constant amount plus the largest separator in the junction tree.[2] Suppose then that, while building the whole junction tree, we find that a separator violates memory constraints. We then include the separator in the cutset, and recursively analyze the resulting sub-networks. The cutset is produced when this process does not find any violating separator. The result is a set of sub-networks with the property that every sub-network can be processed by a clustering algorithm within the space constraints. Even though finding an optimal cutset and an optimal junction tree are NP-hard problems [8, 38, 63], good heuristics are available [37]; we have found in our tests that finding a good cutset takes about 0.5% of overall running time.

Figure 3 shows a small junction tree and the sub-networks obtained from it, assuming a constraint on separators (maximum size of just 4 floating point values) and supposing all variables are binary. The separator $ADF$ violates the constraint, so $ADF$ are included in the cutset. Two networks are generated by this cut; one of them induces clusters $ABC$ and $ACDF$, while the other contains the remainder of the original network. The decomposition process is then applied to these two sub-networks recursively until no separator has size larger than 4.

### 5.4. Planning phase: handling caches

Even though the goal of the decomposition process is to use as much memory as possible in the sub-networks (within memory constraints), it

[2] It is possible to code the variable elimination algorithm so that memory consumption is linearly related to the largest separator. The implementation of adaptive conditioning discussed in Section 6 uses an implementation of variable elimination that satisfies this linear relationship.

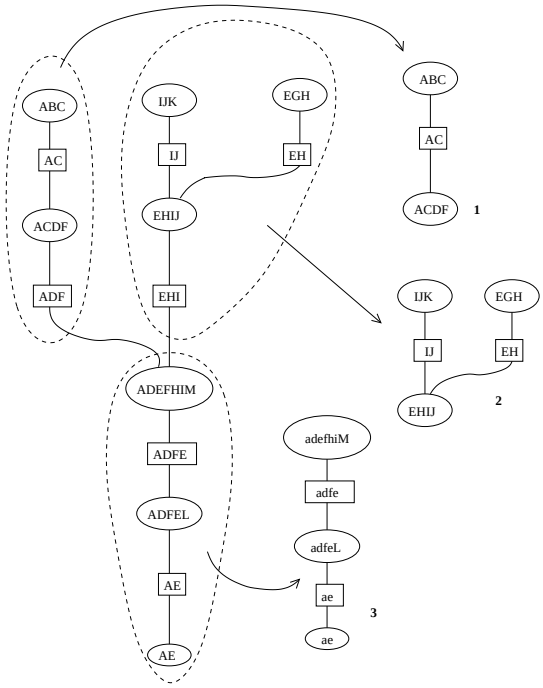



**Figure 3.** Junction tree and resulting decomposition.

may happen that the sub-networks do not use *exactly* all available memory. For example, we may have a million floating-point values at our disposal and a network where the largest separator requires ten million floating-point values; we then condition on this separator and realize that the remaining separators require at most five hundred thousand floating-point values — we now can use the remaining five hundred thousand values as we please. Following the basic anyspace technique used in recursive conditioning [16], we could use available memory to cache and reuse inferences.

Consider a simple example. Suppose that a network $\mathcal{N}$ is decomposed into $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$, such that $\mathcal{N}_2$ and $\mathcal{N}_3$ do not have common variables. Suppose also that $\mathcal{N}_1$ contains the query variable, and $\mathcal{N}_2$ and $\mathcal{N}_3$ contain observed variables. We could then cache inferences from $\mathcal{N}_3$ while we go over instantiations of $\mathcal{N}_1$ and perform inferences in $\mathcal{N}_2$.

Caches lead to a fine control of memory use, but finding a method for efficient cache allocation is a very challenging problem in itself. We have tested several strategies for cache allocation and found that the following method is quite satisfactory. We simply assign a *cache unit* to each sub-network in decreasing order of network size (number of variables), where a *cache unit* contains the amount of memory necessary to store $Pr^{\mathcal{N}_i}(\mathbf{Q}_i \cup \mathbf{C}_i|\mathbf{E}_i)$ (the

result of a particular inference in the sub-network $\mathcal{N}_i$ given a configuration of $\mathbf{C} \setminus \mathbf{C}_i$; remember that sub-networks may contain query variables in adaptive conditioning). This process is repeated until we exhaust available memory.[3] If we find that every cutset instantiation can be stored in memory, we essentially obtain a clustering method where the separators among sub-networks are gradually computed and stored.

As shown in Section 6, caching is an extremely effective strategy to refine anyspace behavior. Adaptive conditioning benefits greatly from the "smoothness" in memory consumption provided by caches — however, adaptive conditioning tries to minimize the importance of caches by using as much memory as possible for separators of sub-networks, thus easing the difficult problem of generating a caching strategy.

Another problem in handling caches is how to update the information stored when new results become available. For instance, suppose a sub-network has a cache unit (storing the result of an inference for a particular configuration of the cutset), and an inference (with a different configuration) is requested by the execution phase. Should the cache unit store the new result or keep the previous one? If the result is kept, when should it be updated? This problem is also complex and is closely related to how cutset instantiations are organized (discussed in Section 5.6). To tackle this problem, we use a simple heuristic that has proved to be efficient, particularly when combined to the strategy we use to organize cutset instantiations. Basically, we update the information of cache units as soon as new inferences become available for the sub-network.

To get a sense of the relevant cache $\times$ separator $\times$ time trade-offs, consider the following experiment with the Alarm network, shown in Figure 4. Consider the variable *BP* and no evidence (this is the query that requires most computational effort without evidence), and suppose that a very small amount of memory is available — only 36 floating-point values. The time required for inference is much more sensitive to the amount of memory allocated to separators than to caches — as the amount of memory for separators increases, the time for inference drops sharply; this is not observed as the amount of memory for caches increases. We leave for future work a precise quantification of the complex trade-offs involved in strategies for caching probabilistic inference.

### 5.5. Execution phase: anytime inference in sub-networks

After adaptive conditioning decomposes a network and assigns caches to sub-networks, the algorithm must decide how to process each sub-network. If there are no constraints on processing time, the obvious choice is to run

[3] In our implementation, the space available for caches is essentially the difference between the largest possible separator and the maximum separator actually obtained through decomposition.

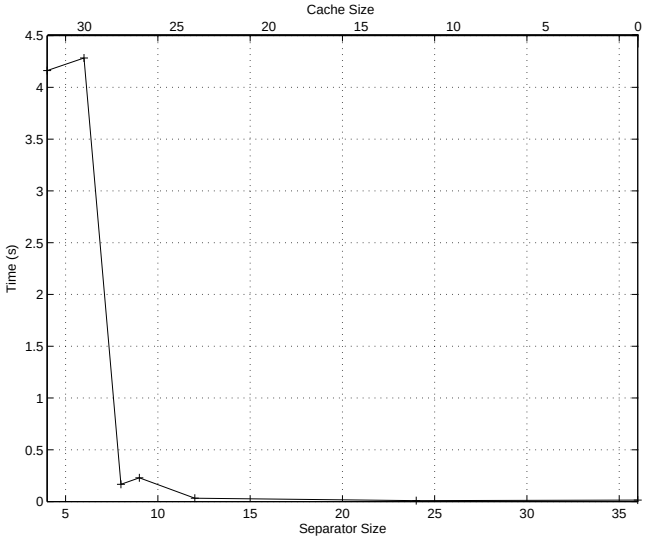



**Figure 4.** A cache × separator × time trade-off in the Alarm network, querying variable *BP* without evidence. The same fixed amount of memory is distributed between separators and caches.

a clustering algorithm in each sub-network. If instead there are limitations on processing time, several possibilities can be conceived.

Consider the possibility that some sub-networks must be assigned exact algorithms, while other sub-networks must be assigned approximate algorithms. A simple anytime procedure is to assign clustering algorithms to as many sub-networks as possible, and to leave approximate algorithms to other sub-networks. We have tested approximations based on stochastic algorithms.

We have found, after extensive tests, that Gibbs sampling algorithms take longer to produce a reasonably accurate inference than variable elimination takes to produce an *exact* inference, even in rather large networks [52].[4] Clearly these statements should be taken in the proper context. First, Gibbs sampling and other stochastic algorithms are particularly valuable in the presence of continuous variables; we stress that here we deal only with categorical variables. Second, there is a limit to the applicability of variable elimination; for very dense and large networks, one cannot hope to use straight variable elimination — however we have observed that in

[4]Such findings were corroborated by empirical evidence mentioned by Bruce D'Ambrosio at the Workshop on Real-time Decision Support and Diagnostic Systems at AAAI2002. We feel that the average performance of other stochastic algorithms should be comparable to the performance of Gibbs sampling.

those cases the anytime conditioning strategies anytime we discuss next can yield accurate approximations faster than stochastic algorithms do.

The alternative we have pursued is to use a search-based algorithm, such as bounded conditioning, in some sub-networks. Here we are left with several problems. Bounded conditioning uses *very* little memory; we may end up "saving too much memory" in the process, leaving too many memory for complex caching decisions. For example: If we combine bounded conditioning and caching, should the decomposition step be revised once memory is available? Another question is, Which sub-networks should run exact algorithms and which should run bounded conditioning? It seems very difficult to answer such questions in any sort of optimal manner.

Instead of using exact and approximate algorithms in different sub-networks, we have concluded that there exists a simpler yet more effective strategy. Observe that adaptive conditioning can be directly turned into an anytime algorithm by running a subset of all possible instantiations, thus generating bounds for the complete summation in Expression (4) — the same idea used in bounded conditioning and bounded recursive decomposition. If we stop instantiating cutset variables, we obtain lower bounds for probabilities, denoted by $\underline{Pr}(\mathbf{X}_Q, \mathbf{X}_E)$. To produce an upper bound, we use:

$$\overline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E) = 1 - \sum_{\mathbf{X}_Q \neq \mathbf{x}_Q} \underline{Pr}(\mathbf{X}_Q, \mathbf{X}_E). \tag{5}$$

As an example, suppose that we wish to compute the marginal probability for a ternary variable $X$, and we stop computation when $Pr(X = x_0) = 0.12$, $Pr(X = x_1) = 0.56$, $Pr(X = x_2) = 0.17$. Probability bounds are: $Pr(X = x_0) \in [0.12, 0.27]$, $Pr(X = x_1) \in [0.56, 0.71]$, $Pr(X = x_2) \in [0.17, 0.32]$.

Bounds for conditional probability can be easily obtained [46]:

$$\underline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q|\mathbf{X}_E) = \frac{\underline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E)}{\underline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E) + \sum_{\mathbf{X}_Q \neq \mathbf{x}_Q} \overline{Pr}(\mathbf{X}_Q, \mathbf{X}_E)}, \tag{6}$$

$$\overline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q|\mathbf{X}_E) = \frac{\overline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E)}{\overline{Pr}(\mathbf{X}_Q = \mathbf{x}_Q, \mathbf{X}_E) + \sum_{\mathbf{X}_Q \neq \mathbf{x}_Q} \underline{Pr}(\mathbf{X}_Q, \mathbf{X}_E)}.$$

As an alternative approach, we have observed that a straightforward normalization of incomplete results often provides an excelent approximation to the complete inference. To illustrate this possibility, suppose again we have $Pr(X = x_0) = 0.12$, $Pr(X = x_1) = 0.56$, $Pr(X = x_2) = 0.17$. An approximate inference can be produced by normalization: $Pr(X = x_0) \approx 0.1\underline{3}$, $Pr(X = x_1) \approx 0.6\underline{2}$, $Pr(X = x_2) \approx 0.1\underline{8}$.

The main problem is how to organize cutset instantiations, so that most

of the probability mass is quickly generated.[5] The next section describes a method that is suited to adaptive conditioning. Note that the order of cutset instantiations makes inferences in some sub-networks to be updated more often than in others — thus we obtain a method that automatically distributes the computational effort among sub-networks.

### 5.6. Execution phase: generating cutset instantiations

To generate instantiations, we exploit the intuition that the "farther" a sub-network is from the query variables, the smaller the effect of the sub-network in the inference of interest. If a sub-network $\mathcal{N}_i$ has little effect on the inference, relatively few instances of $\mathcal{N}_i$ should be visited when producing probability bounds. Such an effect is obtained by varying the cutset variables of $\mathcal{N}_i$ more slowly than the cutset variables for more critical sub-networks. The following procedure emerges: (i) order the sub-networks from "closest" to "farthest" from the query variables; (ii) order the cutset variables so that the variables for the "closest" network vary more quickly; (iii) generate and process instances until time is exhausted.

The challenge in this procedure is to formalize a "distance" between sub-networks. Our solution is inspired by results on *conditional mutual information* [40]. Take a Bayesian network $\mathcal{N}$ over variables $\mathbf{X}$. The *conditional mutual information* of variables $X$ and $Y$ in $\mathcal{N}$, denoted by $I(X;Y)$, quantifies uncertainty reduction by random variables [9]:

$$I(X;Y) = \sum_{X,Y} Pr(X,Y) \log \frac{Pr(X,Y)}{Pr(X) \cdot Pr(Y)}.$$

The mutual information is symmetric and represents a measure of the dependence between two random variables. A natural idea is to evaluate the "distance" between a sub-network and query variables by computing the mutual conditional information between query variables and variables in the sub-network cutsets (keeping all variables conditional on observed variables). However, mutual conditional information is very expensive to compute (time spent is $\mathcal{O}(m \exp(n))$ for $n$ variables, $m$ of which are query variables). We thus propose a heuristic method that relies on the monotonic relationship between mutual conditional information and shortest-path distance: Kjaerulff has proved that mutual conditional information between $X$ and $Y$ decreases with increases in the shortest path (in $\mathcal{N}$) between $X$ and $Y$ [39]. Suppose then that we want to measure the influence of $X$ in a query variable $Y$. A quick metric is to take the shortest-path algorithm, and find the number of edges between $X$ and $Y$. If instead we have a set

[5] Bounded conditioning has a built-in mechanism to order instantiations [32], while bounded recursive decomposition resorts to Gibbs sampling to decide which instantiations must be computed and which must be retrieved from an initialization phase [46].

of variables $\mathbf{X}$ and a set of query variables $\mathbf{Y}$, we take the average of all shortest-paths between variables in $\mathbf{X}$ and the set $\mathbf{Y}$ — we call the resulting quantity by Minimal Mean Distance (MMD):

$$MMD(\mathbf{X}, \mathbf{Y}) = \sum_{i}^{|\mathbf{X}|} \frac{d(X_i, \mathbf{Y})}{|\mathbf{X}|},$$

where $d(X, Y)$ is length of the shortest-path between $X$ and $Y$.

Once we obtain the MMD of every cutset variable, we sort the variables so that variables with larger MMD are modified less often than variables with smaller MMD. In addition to the sorting cutset variables, we can improve the speed of convergence of probability bounds by paying attention to the order of instantiations for categories in each cutset variable. For example, if a cutset contains binary variables $X$ and $Y$, we may choose to visit $x_1$ before $x_0$, regardless of the order in which we visit $y_0$ and $y_1$. We must first visit instantiations that potentially contain the most probability mass, looking for good instantiations (as in Henrion's search method [30]). To find an order for the values of a cutset variable $X$ we compute the posterior probability of $X$ with respect to the sub-network that contains $X$; we then visit first the values of $X$ with highest posterior probability. We have observed that this technique often increases dramatically the speed of convergence for probability bounds.

At this point, the original network has been decomposed, caches have been allocated, cutset variables and their categories have been properly sorted. Should we now consider distributing caches *after* sorting instantiations? One could argue that the cache allocation strategy should take into account the order of cutset instantiations — caches should be more useful for those variables that change less often. However we have found empirically that it is more important to allocate caches based on the size of sub-networks than on cutset orderings. Again we face a situation where many alternatives could be conceived, with no obvious "optimal" solution for the caching strategy. We conjecture that the most efficient (in terms of time) scheme should dynamically modify caches during inference, assigning memory to those large cutsets that change more often. In any event, we have decided to follow the simple yet efficient caching strategy described in Section 5.4.

### 5.7. The complete algorithm

Section 5.1 sketched the main steps of adaptive conditioning, leaving undefined several aspects of the algorithm. In fact, it is profitable to think of adaptive conditioning as a generic strategy: divide a network to satisfy space constraints, then process sub-networks as required to meet time constraints. However at this point we can present a more detailed description

1. Use d-separation to discard variables that cannot affect the inference.
2. Use conditioning to recursively divide the resulting network into sub-networks, until every separator requires less space than the available memory (Section 5.3). To do so, recursively produce junction trees for the various sub-networks and "break" them whenever separators become larger than a certain limit.
3. If there is memory left after the division of the network, assign caches to store intermediate results (Section 5.4): Assign a cache unit to each sub-network in decreasing order of network size, until available memory is exahusted.
4. If there are time constraints:
   (a) Order cutset variables using Mean Minimal Distance, and order categories of cutset variables by local posterior probability.
   (b) Apply Expression (4), performing local inferences for as much time as possible. Before executing each inference, verify whether this "sub-inference" is in the cache; if yes, then reuse it; if no, then apply a clustering method to obtain the necessary sub-inference and update the cache with the new result.
5. Obtain probability bounds (or return a single distribution when inference is actually completed) using Expressions (5) and (6).

**Figure 5.** Adaptive conditioning.

of several design decisions that, by analysis and experimentation, we regard as most adequate for implementation. Figure 5 contains a detailed description.

The execution phase is responsible for instantiating the cutset variables in the predefined order, running clustering algorithms in each sub-network, caching results whenever possible, and computing Expression (4). When time is exhausted, probability bounds are produced. Note that the number of inferences grows exponentially with the number of variables in cutsets; given a Bayesian network with $n$ variables and cutsets of width $w_c$ that decompose the network into $w_s$ sub-networks, the number of inferences performed by adaptive conditioning is $\mathcal{O}\left(w_s \cdot \exp(w_c)\right)$.

As an example, consider the network $\mathcal{N}$ in Figure 6, containing only binary variables. The figure shows a decomposition of $\mathcal{N}$ into three sub-networks, by conditioning on $C$ and $B$. Dashed nodes represent "dummy" variables that are always observed and do not change the complexity of inferences in the corresponding sub-networks. We wish to compute the joint probability of $E$ and $F$. We have to compute the following probabil-

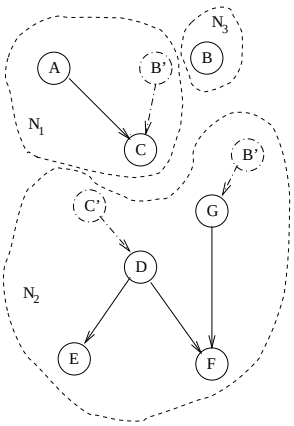



**Figure 6.** A decomposition for the Bayesian network in Figure 1.

ities: $Pr^{\mathcal{N}_1}(C \mid B' = b')$ (computed twice), $Pr^{\mathcal{N}_2}(E, F \mid C' = c', B' = b')$ (computed four times), and $Pr^{\mathcal{N}_3}(B)$ (computed only once).

As discussed in Section 5.5, we have discarded the possible strategy of distributing different exact and approximate algorithms through sub-networks. We have found mixture-of-algorithms strategies to be less effective, for anytime purposes, than just applying the same variable elimination algorithm across sub-networks. However, we conjecture that such a strategy could be interesting in various situations, for example in parallelized engines with different processing characteristics.

### 5.8. Comparison to anyspace algorithms

A comparison between adaptive conditioning and adaptive variable elimination or recursive conditioning necessarily depend on how we are to introduce anytime behavior into the latter two algorithms. These comparisons can illuminate several aspects of adaptive conditioning.

The obvious way to obtain anytime behavior with adaptive variable elimination is to run approximate algorithms inside buckets — for example, to run Gibbs sampling (as in [41]) or bounded conditioning (similarly to Dechter's loop cutset suggestion [21]). However, we are left with a problem: if intermediate results in one bucket are improved, how should the new results be propagated to other buckets? The solution would be to apply anytime algorithms in such a way that different portions of a network could be processed independently — a solution that paves the way to adaptive conditioning. It is actually easier to think of adaptive variable elimination as a derivative of adaptive conditioning, because the first algorithm is a special case of the second one (obtained when the conditioning operations are not "wide" enough to actually "cut" the network into sub-networks). We have found that adaptive conditioning is easier to understand and im-

plement than other possible combinations of adaptive variable elimination plus anytime algorithms.

Recursive conditioning is a clever algorithm with many possible variants. It could become an anytime algorithm by computing a limited number of terms in Expression (2). However this partial computation scheme is not easy to implement in recursive conditioning, as the power of the algorithm comes just from the way the computation of many terms is "entangled" in a dtree. We are again led to the conclusion that we must "cut" some portions of the network from others, so as to organize partial sums. That is, instead of splitting networks until single-node sub-networks, we must stop splitting earlier. In fact, adaptive conditioning can be understood as a close cousin of recursive conditioning in the following sense: the inference process in adaptive conditioning can be represented as a dtree where leaves are sub-networks (and sub-networks are processed in an anytime fashion).

Despite the similarity between adaptive and recursive conditioning, there are significant differences between them. The obvious, and possibly the most important difference is that adaptive conditioning directly allows anytime behavior, as discussed in the previous paragraph. Note that there is a price to pay for anytime behavior: while adaptive conditioning degrades, in the limit of scarce memory, to brute force instantiation of Expression (2), recursive conditioning takes $\mathcal{O}(n \exp(w \log n))$ time in the same circumstances. A second notable difference between adaptive and recursive conditioning is that the first algorithm can handle arbitrary sets of query variables, while the second one focuses on the computation of a single probability value for a single variable. A third difference is that adaptive conditioning tries to use as much memory as possible before it considers the use of caches (networks are divided until memory constraints are satisfied, but not more than that); recursive conditioning instead moves the whole inference to a very thin structure and then uses the available memory for caching. Because finding a reasonable caching strategy is a non-trivial problem, it makes sense to reduce its importance.

### 5.9. Comparison to anytime algorithms

Adaptive conditioning offers some significant advantages over existing anytime algorithms. The algorithm produces enclosing bounds as approximations, unlike stochastic approximations and loopy propagation algorithms. Experiments show that convergence of these bounds is very fast, even within relatively stringent memory constraints (Section 6). We should add that adaptive conditioning is *much* faster than standard stochastic approximation algorithms, at least for the kinds of "large" networks that can be found in the literature; that is, in our tests we observed that excellent bounds were obtained long before a similar approximation was produced by Gibbs sampling and similar schemes. Adaptive conditioning also fares

well against bounded conditioning and search-based anytime techniques, because adaptive conditioning essentially contains such methods and adds various improvements. Instead of raw bounded conditioning, adaptive conditioning tries to use all the available memory; instead of searching for probability terms in the whole network, adaptive conditioning tries to distribute the search on sub-networks in an organized fashion.

Adaptive conditioning can be easily employed if a purely anytime inference algorithm is required (that is, if there are no memory constraints, just time constraints). The planning phase now has to select a cutset so as to obtain the fastest convergence of bounds. Our strategy in such situations is to simply divide a network in its largest separator (more refined strategies can be devised in future work). We note an important property of such explicit decomposition: as we obtain truly independent sub-networks, we can easily apply different levels of computational effort to distinct portions of a network. It would be difficult to do so using any straightforward anytime variant of adaptive variable elimination.

## 6. Tests and results

We have implemented adaptive conditioning as described in Section 5.7, using the standard variable elimination algorithm to process sub-networks. We have tested real and simulated networks with a variety of space and time constraints.[6] We illustrate our results with inferences in real networks. For each network, we produce inferences for the variables whose set of d-connected variables are the largest — that is, we select the hardest queries without observations. The inclusion of observations does not change the properties of the algorithm but would introduce several complexities into the testing procedure (which variables to observe, which values to set as observed), so we decided not to take observations into account.

### 6.1. The Alarm network

Consider first the Alarm network [2], with memory constraints on separators. We limited separators to contain from 3 to 24 floating point values (note the very stringent constraints). We also imposed time constraints from 1 to 3 seconds (time constraints are imposed on overall running time,

[6] We run tests in a Pentium 4 1.7Ghz with 1GByte of memory running Linux 2.4.7-10; the algorithm was coded in the Java language and tested with the JVM 1.3.1_01 from Sun Microsystems. Libraries for the variable elimination algorithm are based on the inference engine for the JavaBayes system, freely available at `http://www.cs.cmu.edu/~javabayes`.

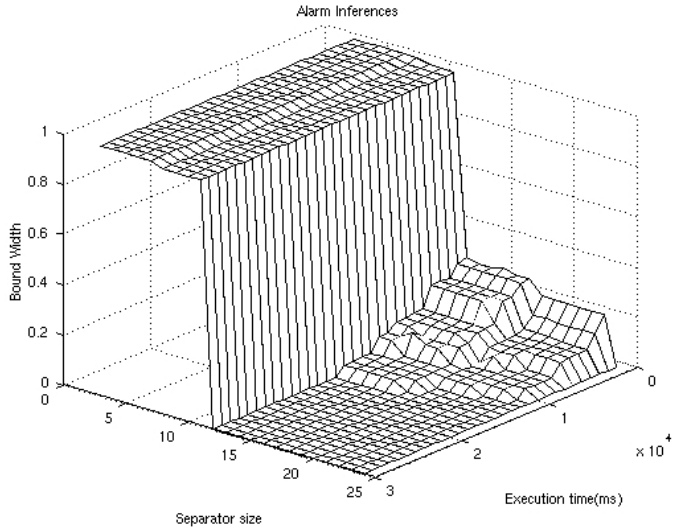



**Figure 7.** Interval width for inferences with the Alarm network (query variable is *BP*).

just as it would be the case in a real-time system). For the Alarm network we run tests with almost every possible memory configuration, as this network is relatively small and serves well as a benchmark. In the Alarm network, exact inference for *BP* requires a separator of size 25 — that is, memory beyond this quantity is useless. However we observed that excelent answers can be obtained if size larger than 13 is allowed.

Figure 7 is a graph of "quality × space × time" for the marginal probability of variable *BP*. "Quality" is represented by the interval between lower and upper probability bounds for one of the categories of *BP*. Note the dramatic increases in quality (decreases in interval length) for some small differences in memory — a little more memory sometimes leads to great improvements in the decomposition process.

We would like to stress that a graph such as the one in Figure 7 can hardly be built with existing techniques, and the great appeal of adaptive conditioning is exactly the possibility of balancing time and space constraints simultaneously while controlling quality.

Figure 8 shows a different "quality × space × time" graph; here we plot the *Kullback-Leibler divergence* or relative entropy $D(Pr \parallel \hat{Pr})$ between the probability of the exact inference $Pr(X_q)$ and the approximation based on normalizing an incomplete inference $\hat{Pr}(X_q)$:

$$D(Pr \parallel \hat{Pr}) = \sum_{X_q} Pr(X_q) \log \frac{Pr(X_q)}{\hat{Pr}(X_q)}$$

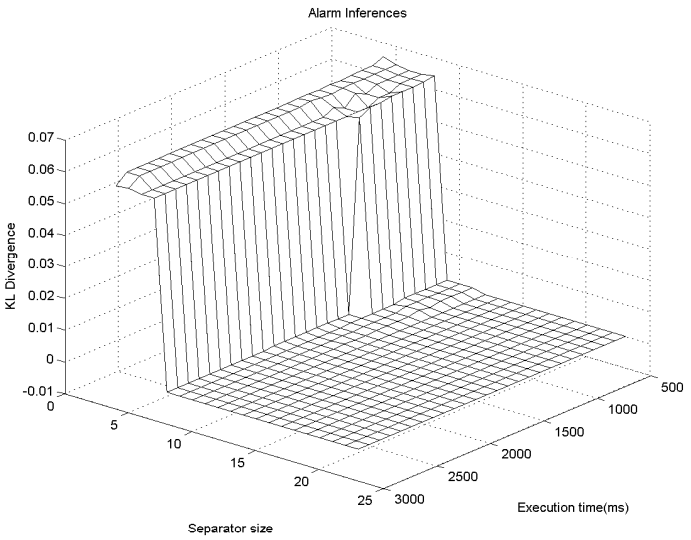



**Figure 8.** Relative Entropy for the Alarm network (query variable is *BP*).

In the case of the Alarm network, $X_q = BP$. Note the quality of inferences for relatively scarce memory and time resources. Again we see that quality varies somewhat discontinuously.

### 6.2. The Link network

Consider now the Link network [35], a large network with 724 nodes (almost all of them binary), representing linkage between two genes. Figure 9 shows interval length for query variable *DO_56_d_p*. This variable is appropriate because inferences with it require a very large number of requisite variables. Figure 10 shows the error in approximating by normalization of incomplete results, again for variable *DO_56_d_p*.

Our tests were run with memory constraints that should be close to stripped-down embedded systems. We varied separator size from *only* 65 floating point values to 129 floating point values. We note the enormous memory savings that can be obtained with adaptive conditioning: we can obtain almost exact answers within 3 seconds with a maximum separator of just 80 floating point values.

In Figures 9 and 10 we observe regions where errors increase dramatically. They indicate operation points that should be avoided in real applications with stripped down bounded agents and embedded systems. We can also observe the effect of caches in the inference process. In Figure 7 for example, for separator sizes bigger than 12 we see a smooth region where the performance increases with time and memory. As the decomposition of the

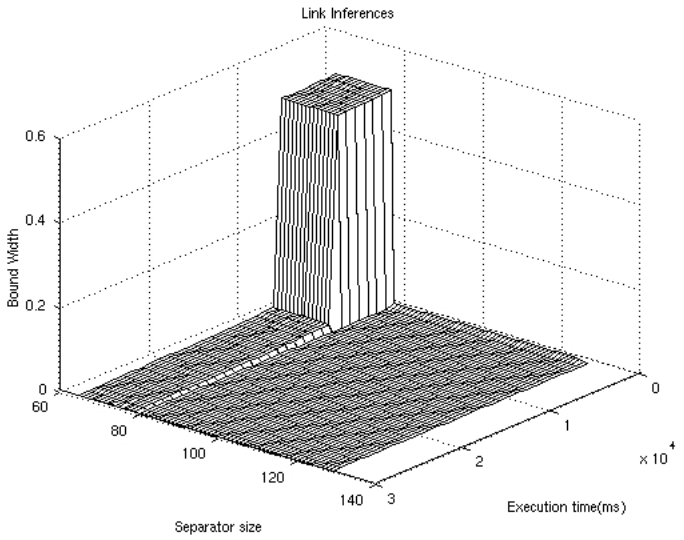



**Figure 9.** Bound width for Link inferences.

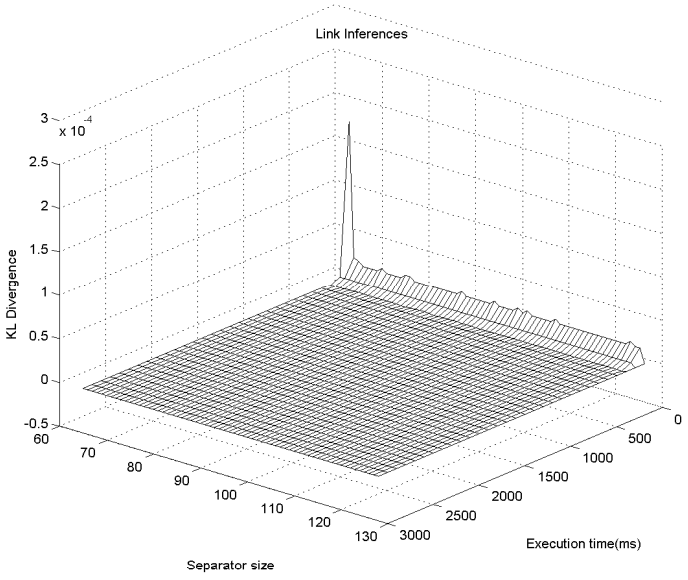



**Figure 10.** Relative Entropy for Link inferences.

network remains almost the same for separator sizes wider than 12, the performance increases with memory is due to cache allocation.

### 6.3. The Diabetes network

The experiments just reported used very stringent space constraints; it could be argued that typical probabilistic inference employs larger memory resources. In this section we move to networks with huge memory requirements for inference.

We have conduct tests with models that follow the usual pattern of dynamic Bayesian network; that is, networks with a regular structure containing repeating blocks. Our results are illustrated using the Diabetes network.[7] The structure we used was an expansion of Diabetes into 24 slices, each containing 17 variables. The model is particularly interesting because Diabetes contains some "linking" variables that are connected to all slices, and is therefore harder to handle than purely repeating dynamic Bayesian networks. The goal was to produce inferences for the variable *bg_24* (at the "bottom" of the 24th slice). The largest separator for this network (using a maximum weight heuristic) contains 64 variables. As variables have six categories on average, we would need an astronomically large amount of memory to conduct exact inference with standard variable elimination. Adaptive conditioning instead faces no difficulties, and can produce the *exact answer* in less than 3 seconds, using a separator size of 1500 floating point values. We ran tests in Diabetes using separator sizes of 1300 to 4000 and time constraints from 1000 ms to 5000 ms. As we see in Figure 11 and in Figure 12, changes in separator sizes from 1500 to 4000 did not affect the quality significantly. However, for separator sizes less than 1500, the network decompositons changed and the quality degraded considerably.

We close by noting that the experiments reported here are not the only ones we have conducted, and were not selected as successful cases — rather, similar behavior was met in a large variety of tests.

## 7. Conclusion

This paper presents a discussion of algorithms that simultaneously display anytime and anyspace characteristics in Bayesian network inference. We have attempted to provide a relatively broad description of the many factors involved in such inferences, while keeping the exposition as simple

[7] Diabetes is available for download on Bayesian Network Repository: http://www.cs.huji.ac.il/labs/compbio/Repository/networks.html.

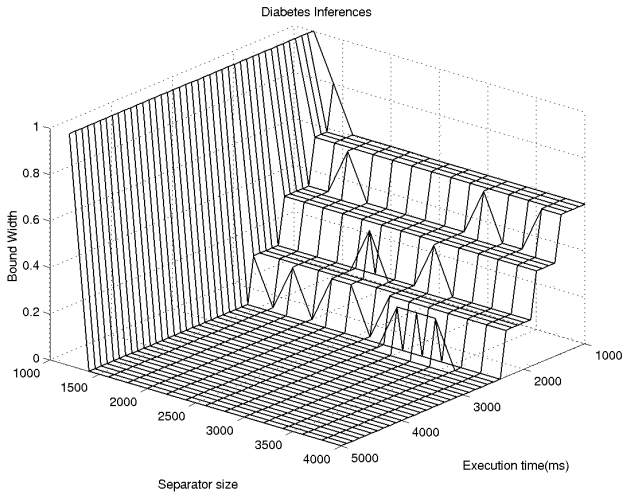



**Figure 11.** Bound width for Diabetes inferences.

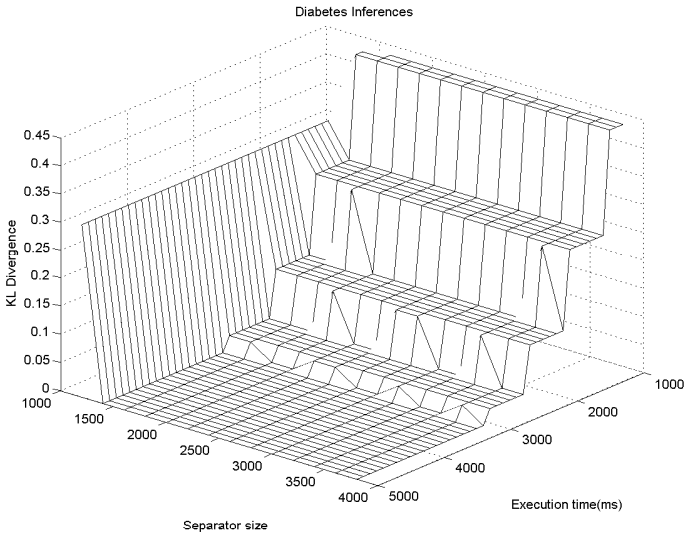



**Figure 12.** Relative Entropy for Diabetes inferences.

and didactic as possible. Our goal was to construct algorithms that can add flexibility to probabilistic reasoning, without explicitly getting into issues of meta-reasoning.

The main contribution of this work is the adaptive conditioning algorithm. We certainly make no claims that adaptive conditioning is the *only* way to attain anytime anyspace behavior in Bayesian network inference. Given the large number of factors involved in such inferences, it is likely that no optimal algorithm exists, whatever is meant by optimal; we should instead focus on algorithms that exercise a balanced combination of trade-offs. We suggest that the adaptive conditioning algorithm provides a sensible balance between the necessary compromises in anytime anyspace probabilistic reasoning; we have tried several other combinations of techniques, only to find that they have marginal gain, if any, while enormously complicating matters. In this context, we feel that adaptive conditioning is an algorithm with clear strenghts, as it:

1. allows simultaneous space and time constraints, and incorporates techniques that allow fine usage of available memory and time.

2. smoothly combines the most effective known techniques for inference (clustering and conditioning).

3. is relatively easy to motivate and to understand; it is not too difficult to implement and does not rely on wildly diverse theoretical facts; it can be taught and appreciated with mild effort.

4. can easily explore three-dimensional trade-offs involving "quality $\times$ space $\times$ time"; we are not aware of previous work that has faced these trade-offs explicitly.

5. is ready for parallel implementation (several techniques for network decomposition in parallel systems are rather close to adaptive conditioning [42, 49, 45]), and can be directly used in "hybrid" implementations that combine exact and approximate algorithms in subnetworks.

The algorithm should be a particularly valuable tool for probabilistic reasoning in embedded systems (for example in robots with limited resources) and in multi-agent communities (for example in sensor networks).

A notable characteristic of adaptive conditioning is that it can handle networks large enough to overwhelm existing exact algorithms. In fact, many of our tests with large networks cannot be reproduced with existing clustering algorithms. Only anyspace algorithms such as recursive conditioning can offer exact solutions to the larger networks, but such algorithms do not have the anytime dimension that adaptive conditioning offers as well.

Overall, we see that the landscape of trade-offs between quality, time and space is rather discontinuous: in some cases, relatively small changes in memory can lead to large differences in running time. Such a behavior suggests that a meta-reasoner could be quite effective in analyzing intermediate steps of the computation and determining that more memory or time would be highly profitable and worth paying for. Such a meta-reasoner would be an interesting piece of work.

Adaptive conditioning can certainly be improved in many ways. There are several possible decomposition and caching strategies, (particularly dynamic caching strategies), and several methods to order variables and instantiations, that could improve the performance of the algorithm. We have not captured and tested the whole spectrum of alternatives in this paper, and we leave many open avenues for future research.

## Acknowledgements

This work has received generous support from HP Labs; we thank Marsha Duro from HP Labs for establishing this support and Edson Nery from HP Brazil for managing it. The work has also been partially supported by CNPq and FAPESP. We thank two reviewers who gave important suggestions, and the editor, who oversaw this long process with great patience — particularly when waiting for us to produce the final version.

## References

1. S. Andreassen, Roman Hovorka, J. Benn, K. G. Olesen, and E. R. Carson. A model-based approach to insulin adjustment. In M. Stefanelli, A. Hasman, M. Fieschi, and J. Talmon, editors, *Proceedings of the Third Conference on Artificial Intelligence in Medicine*, pages 239–248. Springer-Verlag, 1991.

2. I. Beinlich, H. J. Suermondt, R. M. Chavez, and G. F. Cooper. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. *Second European Conference on Artificial Intelligence in Medicine*, pages 247–256, 1989.

3. M. Bloemeke and M. Valtorta. A hybrid algorithm to compute marginal and joint beliefs in Bayesian networks and its complexity. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 16–23, 1998.

4. C. Cannings, E. A. Thompson, and M. H. Skolnick. Probability functions in complex pedigrees. *Advances in Applied Probability*, 10:26–61, 1978.

5. A. Cano and S. Moral. Using probability trees to compute marginals with imprecise probabilities. *International Journal of Approximate Reasoning*, 29:1–46, 2002.

6. J. Cheng and M. J. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13:155–188, 2000.

7. G. F. Cooper. The computacional complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42:393–405, 1990.

8. G. F. Cooper. Bayesian belief-network inference using recursive decomposition. Technical Report KSL-90-05, Knowledge Systems Laboratory, Stanford, CA 94305, 1990.

9. T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc, New York, 1991.

10. R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, New York, 1999.

11. F. G. Cozman. Generalizing variable elimination in Bayesian networks. In *Workshop on Probabilistic Reasoning in Artificial Intelligence*, pages 27–32, São Paulo, Brazil, 2000. Tec Art.

12. P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

13. B. D'Ambrosio. Incremental probabilistic inference. In *Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence*, pages 301–308, Washington, DC, 1993.

14. A. Darviche. Conditioning methods for exact and approximate inference in causal networks. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 99–107, San Francisco, California, 1995. Morgan Kaufmann.

15. A. Darwiche. Any-space probabilistic inference. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 133–142, San Francisco, California, 2000. Morgan Kaufmann.

16. A. Darwiche. Recursive conditioning. *Artificial Intelligence*, 126(1-2):5–41, February 2001.

17. S. Davies. *Fast Factored Density Estimation and Compression with Bayesian Networks*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 2002.

18. T. L. Dean and M. Boddy. An analysis of time-depedent planning. In *Proceedings of Seventh National Conference on Artificial Intelligence*, pages 49–54, Menlo Park, California, 1988. AAAI Press/The MIT Press.

19. R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 211–219, San Francisco, California, 1996. Morgan Kaufmann.

20. R. Dechter. Mini-buckets: A general scheme for generating approximations in automated reasoning in probabilistic inference. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 1297–1302, Nagoya, Japan, 1997.

21. R. Dechter. Topological parameters for time-space tradeoff. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 220–227, San Francisco, California, 1996. Morgan Kaufmann.

22. F. J. Díez. Local conditioning in Bayesian networks. *Artificial Intelligence*, 87:1–20, 1996.

23. A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 176–183, 2000.

24. S. Dwarkadas, A. Schaffer, R. W. Cottingham, A. L. Cox, P. Keleher, and W. Zwaenepoel. Parallelization of general linkage analysis problems. *Human Heredity*, 44:127–141, 1994.

25. G. S. Fishman. *Monte Carlo: concepts, algorithms, and applications.* Springer-Verlag, 1995.

26. A. J. Garvey and V. Lesser. Design-to-time real-time scheduling. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, November/December 1993.

27. D. Geiger, T. Verma, and J. Pearl. Identifying independence in Bayesian networks. *Networks*, 20:507–534, 1990.

28. W. R. Gilks, S. Richardson, and D. J. Spiegelhalter. *Markov Chain Monte Carlo in Practice.* Chapman and Hall, London, England, 1996.

29. H. Guo and W. Hsu. A survey of algorithms for real-time Bayesian network inference. In *AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, pages 1–12, 2002.

30. M. Henrion. Search-based methods to bound diagnostic probabilities in very large belief nets. In *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 142–150. Morgan Kaufmann, 1991.

31. E. Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:159–196, 2001.

32. E. Horvitz, H. J. Suermondt, and G. F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, pages 182–193. Morgan Kaufmann, 1989.

33. E. Horvitz and M. Barry. Display of information for time-critical decision making. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 286–305, Montreal, Canada, 1995. Morgan Kaufmann.

34. E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 256–265. Morgan Kaufmann: San Francisco, 1998.

35. C. S. Jensen and A. Kong. Blocking Gibbs sampling for linkage analysis in large pedigrees with many loops. Research Report R-96-2048, Department of Computer Science, Aalborg University, Denmark, Fredrik Bajers Vej 7, DK-9220 Aalborg Ø, 1996.

36. F. V. Jensen. *An Introduction to Bayesian Networks.* Springer Verlag, New York, 1996.

37. U. Kjaerulff. Triangulation of graphs — algorithms giving small total state space. Technical Report R-90-09, Department of Mathematics and Computer Science, Aalborg University, Denmark, March 1990.

38. U. Kjaerulff. Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, (2):7–17, 1992.

39. U. Kjaerulff. Approximation of Bayesian networks through edge removals. Technical report, Department of Mathematics and Computer Science, Aalborg University, 1993.

40. U. Kjaerulff. Reduction of computational complexity in Bayesian networks through removal of weak dependencies. Technical Report R94-2009, Aalborg University, February 1994.

41. U. Kjaerulff. Combining exact inference and Gibbs sampling in junction trees. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, San Francisco, California, 1995. Morgan Kaufmann.

42. A. V. Kozlov and J. P. Singh. Parallel implementations of probabilistic inference. *Computer*, 29(12):33–40, December 1996.

43. S. L. Lauritzen and D. J. Spiegehalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of Royal Statistics Society*, Series B, 50(2):157–224, 1988.

44. Z. Li and B. D'Ambrosio. Efficient inference in Bayes networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11, 1994.

45. A. L. Madsen and F. V. Jensen. Parallelization of inference in Bayesian networks. Technical Report DK-9220, Department of Computer Science, Aalborg University, Denmark, 1999.

46. S. Monti and G.F. Cooper. Bounded recursive decomposition: a search-based method for belief network inference under limited resources. *International Journal of Approximate Reasoning*, 15(1):49–75, 1996.

47. K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

48. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

49. D. M. Pennock. Logarithmic time parallel Bayesian inference. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 431–438. Morgan Kaufmann, 1998.

50. M. A. Peot and R. D. Shachter. Fusion and propagation with multiple observations in belief networks. *Artificial Intelligence*, 48(3):299–318, 1991.

51. D. Poole. Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks. *Artificial Intelligence*, 88:69–100, 1996.

52. F. T. Ramos, F. G. Cozman, and J. S. Ide. Embedded Bayesian networks: Anytime anyspace inference. In *AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*, pages 13–19, 2002.

53. F. T. Ramos, F. Mikami, and F. G. Cozman. Implementação de redes Bayesianas em sistemas embarcados. In *Proceedings of the IBERAMIA/SBIA 2000 Workshops (Workshop on Probabilistic Reasoning in Artificial Intelligence)*, pages 65–69. Editora Tec Art, 2000 (in Portuguese).

54. S. Russell and E. Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

55. A. Salmerón, A. Cano, and S. Moral. Importance sampling in Bayesian networks using probability trees. *Computational Statistics and Data Analysis*, 34:387–413, 2000.

56. R. Shachter. Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In G. F. Cooper and S. Moral, editors, *In Proceedings of the Fourteenth Conference in Uncertainty in Artificial Intelligence*, pages 480–487, San Francisco, 1998. Morgan Kaufmann.

57. R. D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):873–882, 1986.

58. R. D. Shachter, S. K. Andersen, and P. Szolovits. Global conditioning for probabilistic inference in belief networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 514–522, Seattle, WA, 1994. Morgan Kaufmann.

59. G. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2:327–352, 1990.

60. H. A. Simon. *Models of Bounded Rationality 2.* MIT Press, Cambridge MA, 1982.

61. Y. Weiss and W. T. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. Technical Report CSD-99-1046, CS Department, UC Berkeley, 1999.

62. M. P. Wellman and C. L. Liu. State-space abstration for anytime evaluation of probabilistic networks. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 567–574, 1994.

63. W. X. Wen. Optimal decomposition of belief networks. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, pages 245–256. Morgan Kaufmann, 1990.

64. J. S. Yedidia, W. T. Freeman, and Y. Weiss. Bethe free energies, Kikuchi approximations, and belief propagation algorithms. Technical Report TR 2001-16, 2001.

65. N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *Journal of Artificial Intelligence Research*, pages 301–328, 1996.

66. N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the 10th Canadian Conference on Artificial Intelligence*, pages 16–22, Banff, Alberta, Canada, May 1994.

67. G. Zweig and S. J. Russell. Speech recognition with dynamic Bayesian networks. In *AAAI/IAAI*, pages 173–180, 1998.