To appear, *Workshop on Probabilistic Reasoning in Artificial Intelligence*, Atibaia, Brazil, November 20, 2000

Generalizing Variable Elimination in Bayesian Networks

FABIO GAGLIARDI COZMAN*

Escola Politécnica, University of São Paulo
Av. Prof. Mello Moraes, 2231, 05508-900, São Paulo, SP - Brazil
fgcozman@usp.br

Abstract. This paper describes a generalized version of the variable elimination algorithm for Bayesian networks. Variable elimination computes the marginal probability for some specified set of variables in a network. The algorithm consists of a single pass through a list of data structures called buckets. The generalization presented here adds a second pass to the algorithm and produces the marginal probability density for *every* variable in the buckets. The algorithm and the presentation focus on algebraic operations, striving for simplicity and easy of understanding. The algorithm has been implemented in the JavaBayes system, a freely distributed system for the construction and manipulation of Bayesian networks.

1 Introduction

Bayesian networks occupy a prominent position as a a model for uncertainy in decision making and statistical inference [6]. Several algorithms manipulate Bayesian networks to produce posterior values [8, 11, 13]. We can identify two types of algorithms. There are algorithms that focus on algebraic operations, such as the SPI algorithm [9], the variable elimination algorithm [14] and the bucket elimination algorithm [3]. And there are algorithms that focus on graphical properties, mostly dealing with junction trees [6]. One of the advantages of junction tree algorithms is that it is possible to efficiently compute marginal probability values for every variable in a Bayesian network. Algebraic schemes like variable and bucket elimination compute marginal probability values only for a given set of variables.

An attractive property of approaches based on variable elimination is that they are relatively easy to understand and to implement. Junction tree algorithms are quite complex in comparison, demanding long digressions on graph theoretic concepts. There has been an effort to derive junction tree algorithms without resort to graphical concepts [2, 4], but these efforts have not produced a variable-elimination-like scheme for inference. This paper presents such a scheme, describing a generalized variable elimination algorithm that produces marginal probability values for certain collections of queries. The goal here is to describe an algorithm that is simple to understand and to implement, while keeping all the characteristics of more sophisticated algorithms.

2 Bayesian networks

This section summarizes the theory of Bayesian networks and introduces notational conventions used throughout the paper. All random variables are assumed to have a finite number of possible values. Sets of variables are denoted in bold; for instance, $\mathbf{X}$. The set of all variables that belong to $\mathbf{X}$ but do not belong to $\mathbf{Y}$ is indicated by $\mathbf{X}\backslash\mathbf{Y}$. The expression $\sum_{\mathbf{X}} f(\mathbf{X}, \mathbf{Y})$ indicates that all variables in $\mathbf{X}$ are summed out from the function $f(\mathbf{X}, \mathbf{Y})$. Denote by $p(X)$ the *probability density* of X: $p(x)$ is the probability measure of the event $\{X = x\}$. Denote by $p(X|Y)$ the probability density of X conditional on values of Y.

A Bayesian network represents a joint probability density over a set of variables $\mathbf{X}$ [6]. The joint density is specified through a directed acyclic graph. Each node of this graph represents a random variable X_i in $\mathbf{X}$. The *parents* of X_i are denoted by $\mathrm{pa}(X_i)$; the *children* of X_i are denoted by $\mathrm{ch}(X_i)$. And the parents of children of X_i that are not children themselves are denoted by $\mathrm{spo}(X_i)$ — these are the “spouses” of X_i in the polygamic society of Bayesian networks.

Figure 1 shows a network and some graphical relationships among variables.

The semantics of the Bayesian network model is determined by the *Markov condition*: Every variable is independent of its nondescendants nonparents given its parents. This condition leads to a unique joint probability density [11]:

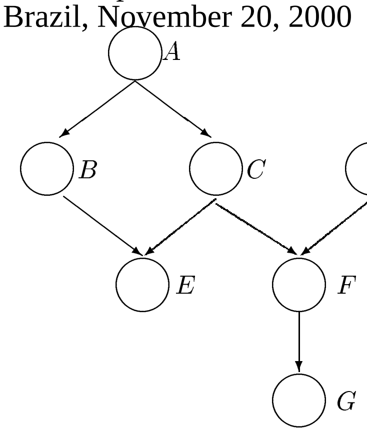
$$p(\mathbf{X}) = \prod_i p(X_i|\mathrm{pa}(X_i))\,. \tag{1}$$

Every random variable X_i is associated with a conditional probability density $p(X_i|\mathrm{pa}(X_i))$, as indicated in Figure 1. The independence relations in the Markov condition imply several other independence relations among variables in a network. The complete relationship between probabilistic independence and the graphical structure of the network is given by the concept of d-separation [5].

Definition 1 *Given three sets of variables* $\mathbf{X}$, $\mathbf{Y}$ *and* $\mathbf{Z}$,

*The author was partially supported by CNPq through grant 300183/98-4.

To appear, *Workshop on Probabilistic Reasoning in Artificial Intelligence,*
Atibaia, Brazil, November 20, 2000



Probability densities in the network:
$p(A), p(B|A), p(C|A), p(D),$
$p(E|B, C), p(F|C, D), p(G|F).$

A	parent of	B and C
B and C	parents of	E
F	child of	C and D
E and F	children of	C
E, F and G	descendants of	C
A, B and D	nondescendants of	C
B, C, E, F and G	descendants of	A
D	nondescendant of	A
B and D	parents of children of	C

Figure 1: An example network.

suppose that along every path between a variable in $\mathbf{X}$ *and a variable in* $\mathbf{Y}$ *there is a variable* W *such that: either* W *has converging arrows and is not in* $\mathbf{Z}$ *and none of its descendants are in* $\mathbf{Z}$*, or* W *has no converging arrows and is in* $\mathbf{Z}$*. Then* $\mathbf{Y}$ *and* $\mathbf{X}$ *are* d-separated *by* $\mathbf{Z}$*.*

A key property of Bayesian networks is that graphical d-separation and probabilistic independence are equivalent [11, page 117].

Suppose we want to find the minimal set of variables that make variable X independent from all other variables in a Bayesian network. Call this minimal set of variables the *Markov blanket* of X. Using d-separation, it is easy to see that the Markov blanket of a variable X is the union of three sets: the parents of X, the children of X, and the parents of the children of X.

Given a Bayesian network, the event E denotes the *evidence*, or the *observations*, in the network. For example, $E = \{X_1 = x_{12}, X_3 = x_{31}\}$ fixes the values of variables X_1 and X_3. Denote by $\mathbf{X}_E$ the set of observed variables; in the previous sentence, $\mathbf{X}_E = \{X_1, X_3\}$.

Inferences with Bayesian networks usually involve the calculation of the posterior marginal for a set of *query variables* $\mathbf{X}_q$ [3, 10]. The algorithm presented in this paper performs exactly this computation. Note that there is no restriction on the number of variables in $\mathbf{X}_q$. To simplify the presentation, the symbol $\mathbf{X}_q$ is used to denote both the query variables and the event that the query variables have a specified value. The posterior probability of $\mathbf{X}_q$ given E is:

$$P(\mathbf{X}_q|E) = \frac{P(\mathbf{X}_q, E)}{P(E)} = \frac{\sum_{\mathbf{X}\setminus\{\mathbf{X}_q,\mathbf{X}_E\}} P(\mathbf{X})}{\sum_{\mathbf{X}\setminus\mathbf{X}_E} P(\mathbf{X})}. \quad (2)$$

3 Representing probability densities and evidence

From a computational point of view, the important entity in (2) is the numerator:

$$p(\mathbf{X}_q, E) = \sum_{\mathbf{X}\setminus\{\mathbf{X}_q,\mathbf{X}_E\}} P(\mathbf{X}),$$

because once we have the numerator, we can easily obtain the denominator. The denominator $p(E)$ is simply a normalization constant as the evidence E is fixed. So, a basic rule when deriving algorithms for Bayesian networks is: Compute the numerator $p(\mathbf{X}_q, E)$, and obtain the denominator $p(E)$ in the very last stage.

Note that a variable must appear as a conditioning variable once for each of its children. That is, if X_1 is a child of X_2, then X_2 must appear as a conditioning variable in the density $p(X_1|X_2, \ldots)$. Note also that any variable X_i appears *only once* as a non-conditioning variable in the density $p(X_i|\mathrm{pa}(X_i))$.

When manipulating probability densities in inference algorithms, it is often necessary to multiply densities defined over different sets of variables. For example, we may have that X_1 and X_3 are independent given X_2, and then we obtain $p(X_1, X_2|X_3)$ by multiplying densities $p(X_1|X_2)$ and $p(X_2|X_3)$. Suppose the value of X_2 is fixed by the evidence; consequently, $p(X_1, X_2 = x_2|X_3)$ is equal to the product $p(X_1|X_2 = x_2) \times p(X_2 = x_2|X_3)$. In general, it is cumbersome to represent explicitly every observation in a Bayesian network. Instead, we lump all observations into E, and adopt a second basic rule: Any inference algorithm begins by fixing the observed variables at their observed values. For example, if X_2 is observed, then the algorithm does not deal with X_2 at all; computations proceed as if we had unnormalized functions:

$$p(X_1|X_3) = p(X_1)\,p(X_3)\,.$$

These functions are not probability densities in general. We can even multiply them by positive constants arbitrarily, for example to avoid numeric underflow.

4 Standard variable elimination

Variable elimination is an algebraic scheme for inferences in Bayesian networks. Variable and bucket elimination are essentially identical; we use the former term to name the algorithm but adopt the term "bucket" to name the storage units in the algorithm, as this term is quite appropriate.[1] The basic principles of bucket manipulation have been studied extensively by Dechter, and have led to a vast array of results and applications [3].

Given a Bayesian network over variables $\mathbf{X}$, evidence E and a query $\mathbf{X}_q$, computation of $p(\mathbf{X}_q|E)$ typically involves only a subset of the densities associated with the network. If density $p(X_i|\mathrm{pa}(X_i))$ is necessary for answering a query, then X_i is a *requisite variable* [12]. There are polynomial algorithms to obtain requisite variables based on graph separation properties [5, 12]. Denote by $\mathbf{X}_R$ the set of requisite variables. Variables in $\mathbf{X}_q$ necessarily belong to $\mathbf{X}_R$, but not all observed variables belong to $\mathbf{X}_R$ (only observed variables that have non-observed parents in $\mathbf{X}_R$ belong to $\mathbf{X}_R$).

We are interested in computing the following expression:

$$p(X_q, E) = \sum_{\mathbf{X}_R \backslash \{\mathbf{X}_q, \mathbf{X}_E\}} \left(\prod_{X_i \in \mathbf{X}_R} p(X_i|\mathrm{pa}(X_i)) \right),$$

where probability densities must be restricted to domains containing no evidence.

Denote by N the number of requisite variables that are not observed and are not in $\mathbf{X}_q$. Now, suppose that these variables are ordered in some special way, so we have an ordering $\{X_1, X_2, X_3, \ldots, X_N\}$. The quality of orderings and the construction of good orderings are discussed later. We have:

$$p(X_q, E) \;=\; \sum_{X_N} \cdots \sum_{X_1} p(X_N|\mathrm{pa}(X_N)) \times \ldots \times p(X_1|\mathrm{pa}(X_1))\,. \tag{3}$$

Because X_1 can only appear in densities $p(X_j|\mathrm{pa}(X_j))$ for $X_j \in \{X_1, \mathrm{ch}(X_1)\}$, we can move the summation for X_1:

$$\sum_{X_N} \cdots \sum_{X_2} \left(\prod_{\substack{X_i \in \\ \mathbf{X}_R \backslash \{X_1, \mathrm{ch}(X_1)\}}} p(X_i|\mathrm{pa}(X_i)) \right) \times \left(\sum_{X_1} \prod_{\substack{X_j \in \\ \{X_1, \mathrm{ch}(X_1)\}}} p(X_j|\mathrm{pa}(X_j)) \right).$$

At this point, we have "used" the densities for $X_j \in \{X_1, \mathrm{ch}(X_1)\}$. To visualize operations more clearly, we can define the following unnormalized density:

$$p(\mathrm{ch}(X_1)|\mathrm{pa}(X_1), \mathrm{spo}(X_1)) \;=\; \sum_{X_1} \left(\prod_{X_1, \mathrm{ch}(X_1)} p(X_j|\mathrm{pa}(X_j)) \right).$$

(Remember that $\mathrm{spo}(X_i)$ denotes the parents of children of X_i that are not children of X_i.)

Think of the various densities as living in a "pool" of densities. We collect the densities that contain X_1, take them off of the pool, construct the a new (unnormalized) density $p(\mathrm{ch}(X_1)|\mathrm{pa}(X_1), \mathrm{spo}(X_1))$ and add this density to the pool.

The result of these operations is that X_1 was "eliminated" from the problem.

Now we move to X_2. We collect all densities that contain X_2 from the pool, take them off of the pool, multiply them together, and sum out X_2. The result of these operations must again be a density. We place this density into the pool, and move to X_3.

At the end, we have some (at least one) densities for $\mathbf{X}_q$. By multiplying these densities together and normalizing the result appropriately, we obtain $p(\mathbf{X}_q|E)$.

In short, the idea of the algorithm is to multiply the members of the density pool in the sequence given by the ordering. The algorithm attempts to eliminate variables as soon as possible, to keep intermediate products at a manageable size.

The ordering of variables is arbitrary, but different orderings lead to different computational loads. Unfortunately, the construction of an optimal ordering is known to be a NP-complete problem, so we should settle for a good, not necessarily optimal, ordering. Several heuristics are known to produce efficient orderings in practice [7, 14].

The variable algorithm can also be used to compute maximum a posteriori configurations, essentially by turning some of the summations into maximizations [3]. There is

[1] It should be mentioned that many ideas developed in the SPI, variable elimination and bucket elimination algorithms were first developed for the *peeling* algorithm in genetics [1].

no need to compute sequences of queries for maximum a posteriori calculations, so the generalization presented in this paper does not apply to maximum a posteriori values.

5 Buckets and bucket trees

The variable elimination algorithm can be described as follows.

1. Generate an ordering for the N requisite, non-observed, non-query variables.
2. Place all network densities in a pool of densities.
3. For i from 1 to N:
   (a) Create a data structure B_i, called a *bucket*, containing:
   - the variable X_i, called the *bucket variable*;
   - all densities that contain the bucket variable, called the *bucket densities*;

   (b) Multiply the densities in B_i. Store the resulting unnormalized density in B_i; the density is called B_i's *cluster*.

   (c) Sum out X_i from B_i's cluster. Store the resulting unnormalized density in B_i's; the density is called B_i's *separator*.

   (d) Place the bucket separator in the density pool.
4. At the end of the process, collect the densities that contain the query variables in a bucket B_q. Multiply the densities in B_q together and normalize the result.

The sequence of buckets created by the algorithm can be represented linearly as a vector. The first bucket is processed by collecting densities from the Bayesian network, computing the bucket separator and "sending" the separator to a bucket down the vector of buckets. The second bucket is then processed, again by collecting densities from the Bayesian network, and possibly by taking the separator from the first bucket. The third bucket is then processed, and so on. Figure 2 shows a linear arrangement of buckets, with the separator flow indicated by arrows.

As the separator for bucket B_i is not necessarily sent to bucket B_{i+1}, the set of buckets can be drawn as a tree stemming from the query variables. Actually, this is a much more interesting representation, shown in Figure 3.

6 Generalizing variable elimination

Suppose that we need to compute the marginal probability density for every variable in a Bayesian network. This type of result is produced by junction tree algorithms, somewhat complex procedures heavily based on graph theory. Here we seek a simpler scheme that adapts variable elimination to the task.

The following conventions are adopted in this section. The bucket variable for bucket B_i is X_i. Denote the variables in B_i's separator by $\mathbf{S}_i$, and denote the variables in B_i's cluster by $\mathbf{C}_i$. Denote by E_i the evidence contained in the sub-tree above and including bucket B_i, and denote by E_i^c the evidence outside of E_i.

6.1 Updating buckets right above the root

Suppose we have a bucket tree and variable elimination has been applied to the tree. Consequently, the normalized density $p(\mathbf{X}_q|E)$ is stored at the root.

Now look at one of the buckets right above the root, the bucket B_a with bucket variable X_a. Note that B_a's cluster is an unnormalized density containing X_a and some of the variables in $\mathbf{X}_q$; no other variable can be in B_a's cluster.

We discard the possibility that B_a's cluster does not contain any of the query variables in $\mathbf{X}_q$ — if this happens, the sub-tree of buckets above and including B_a is disconnected from the rest of the bucket tree. This disconnected sub-tree should then be processed as a separate Bayesian network. In fact, we *always* assume that the outgoing separator of any bucket is nonempty: A bucket with an empty separator can be processed in an appropriate sub-network.

Whatever the composition of B_a's cluster, we can always generate the normalized density $p(X_a|\mathbf{S}_a, E_a)$ from it simply by normalizing the cluster with respect to X_a. The reason for this is that X_a cannot be a conditioning variable in B_a's cluster, because we must have used the density $p(X_a|\mathrm{pa}(X_a))$ before or at the bucket.

Section 6.4 demonstrates the following important fact: $p(X_a|\mathbf{S}_a, E_a)$ is actually equal to $p(X_a|\mathbf{X}_q, E)$. As a result, we can compute

$$p(X_a, \mathbf{X}_q|E) = p(X_a|\mathbf{S}_a, E_a) \times p(\mathbf{X}_q|E) .$$

The updated density for X_a is then obtained by summing out $\mathbf{X}_q$ from the density $p(X_a, \mathbf{X}_q|E)$. So, we have a method to update the clusters for all buckets right above the root.

6.2 Updating buckets away from the root

Now take a generic bucket B_b at some distance from the root. Assume that variable elimination has been executed and B_b's cluster has already been normalized with respect to X_b; consequently, B_b's cluster contains the normalized density $p(X_b|\mathbf{S}_b, E_b)$.

To create a finite induction argument, suppose that the bucket right below B_b, denoted by B_c, has been updated. That is, B_c's cluster contains the density $p(X_c, \mathbf{S}_c|E)$.

Note that $\mathbf{S}_b$ must be contained in $\mathbf{C}_c$, because B_b's separator was sent to B_c during variable elimination. Con-

To appear, *Workshop on Probabilistic Reasoning in Artificial Intelligence,*
Atibaia, Brazil, November 20, 2000

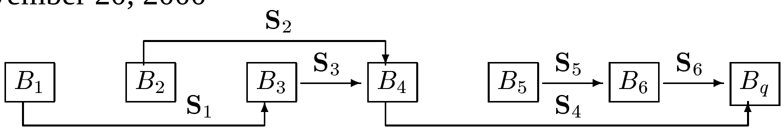


Figure 2: A sequence of buckets and their separators.

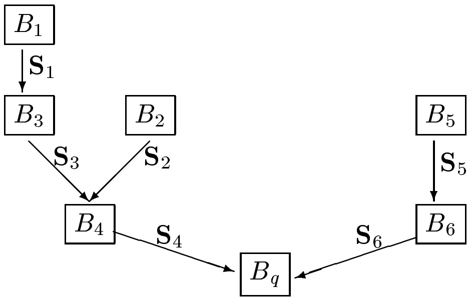


Figure 3: A tree of buckets and their separators.

sequently, $p(\mathbf{S}_b|E_b, E_b^c)$ can be computed in B_c:

$$p(\mathbf{S}_b|E) = \sum_{\mathbf{C}_c \setminus \mathbf{S}_b} p(\mathbf{C}_c|E)\,.$$

Once again, the important thing is that $\{X_b, E_b\}$ and E_b^c are independent given $\mathbf{S}_b$ (Section 6.4). So, we can compute

$$p(X_b, \mathbf{S}_b|E) = p(X_b|\mathbf{S}_b, E_b) \times p(\mathbf{S}_b|E)\,.$$

The updated density for X_b is then obtained by summing out $\mathbf{S}_b$ from $p(X_b, \mathbf{S}_b|E)$.

6.3 A summary of the updating procedure

The previous sections constructed an induction argument. If the child of bucket B_i has been updated, then the following steps update B_i itself:

1. Normalize B_i's cluster with respect to the bucket variable, obtaining the normalized density $p(X_i|\mathbf{S}_i, E_i)$.
2. Ask the child of B_i to provide $p(\mathbf{S}_i|E)$. This probability density replaces B_i's separator.
3. Multiply $p(\mathbf{S}_i|E)$ and the normalized bucket cluster $p(X_i|\mathbf{S}_i, E_i)$, obtaining $p(X_i, \mathbf{S}_i|E)$. This probability density replaces B_i's cluster.

After updating all buckets, the probability density for any variable can be easily computed: just find a bucket cluster that contains the variable and compute the appropriate marginal probability density.

Note that variable elimination proceeds from the top of the tree down to the root, and updating proceeds from the bottom up.

6.4 The separation properties of bucket separators

A bucket tree may contain several branches; every branch contains disjoint sets of variables. Denote by T_i the branch starting at bucket B_i — that is, the sub-tree above and including B_i. Denote by $\mathrm{var}(T_i)$ the set of variables in T_i.

In the previous sections, the following result was used without proof: the variables $\mathrm{var}(T_i)\setminus\mathbf{S}_i$ are independent of all other variables in the bucket tree, conditional on $\mathbf{S}_i$. We can draw a small diagram:

$$\boxed{T_i} \to \mathbf{S}_i \to \boxed{T_i^c},$$

where T_i^c denotes the bucket tree without T_i. We want to establish that $\mathbf{S}_i$ in fact separates, by independence, T_i and T_i^c.

Note first that all variables in the Markov blanket of variable X_i *must* be present in $\mathrm{var}(T_i)$. This happens because the densities $p(X_j|\mathrm{pa}(X_j))$, for $X_j \in \{X_j, \mathrm{ch}(X_j)\}$, must have been used during variable elimination in branch T_i. Thinking recursively, we conclude that only two types of variables may be present in $\mathrm{var}(T_i)$:

- the bucket variables for the buckets in T_i; and
- the Markov blanket for each one of these bucket variables.

The bucket variables in T_i are eliminated before or at B_i, so they cannot be present in B_i's separator. Consequently, B_i's separator contains the union of Markov blankets for all bucket variables in T_i, *excluding* the bucket variables in T_i. Given this construction, we see that B_i's separator is a "Markov blanket" for all bucket variables in T_i. The joint distribution conditional on $\mathbf{S}_i$ factorizes into two pieces, one containing $\mathrm{var}(T_i)$, and the other containing $\mathrm{var}(T_i^c)$. So, variables in T_i are independent of variables in T_i^c given B_i's separator.

7 Conclusion

The algorithm presented in this paper aims at computing efficiently the probability densities for several queries in a Bayesian network. The construction of bucket trees simplifies the presentation, producing an algorithm that is easy to grasp, to teach and to implement.

The algorithm in this paper works by adding a second pass to variable elimination. One of the characteristics of the algorithm is that it relies only on independence relations and probability manipulations. The algorithm does not use graphical concepts, such as triangulations and cliques, focusing solely on probability densities and avoiding complex digressions on graph theory. But the generalized variable elimination algorithm can be also interpreted using graph theoretic tools, as the bucket tree is actually a tree of cliques (the cliques of the triangulated tree induced by the ordering of variables).

All algorithms discussed in this paper have been implemented in the JavaBayes system, a freely distributed package for the construction and manipulation of Bayesian networks (available at http://www.cs.cmu.edu/javabayes).

References

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

[2] A. Darwiche. Dynamic jointrees. In G. F. Cooper and S. Moral, editors, *Proceedings of the XIV Conference on Uncertainty in Artificial Intelligence*, pages 97–194, San Francisco, California, 1998. Morgan Kaufmann.

[3] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In M. I. Jordan, editor, *Learning in Graphical Models*, pages 75–104. MIT Press, Dordecht, The Netherlands, 1999.

[4] D. Draper. Clustering without (thinking about) triangulation. *Proceedings of the XI Conference on Uncertainty in Artificial Intelligence*, 1995.

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

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

[7] 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.

[8] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50(2):157–224, 1988.

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

[10] J. Pearl. On probability intervals. *International Journal of Approximate Reasoning*, 2:211–216, 1988.

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

[12] Ross D. 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, *Proceedings of the XIV Conference on Uncertainty in Artificial Intelligence*, San Francisco, California, United States, 1998. Morgan Kaufmann.

[13] P. P. Shenoy and G. Shafer. Axioms for probability and belief-function propagation. In R. D. Shachter, T. S. Levitt, L. N. Kanal, and J. F. Lemmer, editors, *Uncertainty in Artificial Intelligence 4*, pages 169–198. Elsevier Science Publishers, North-Holland, 1990.

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