BUCKET ELIMINATION: A UNIFYING FRAMEWORK FOR PROBABILISTIC INFERENCE

R. DECHTER
Department of Information and Computer Science
University of California, Irvine
dechter@ics.uci.edu

Abstract.

Probabilistic inference algorithms for belief updating, finding the most probable explanation, the maximum a posteriori hypothesis, and the maximum expected utility are reformulated within the *bucket elimination* framework. This emphasizes the principles common to many of the algorithms appearing in the probabilistic inference literature and clarifies the relationship of such algorithms to nonserial dynamic programming algorithms. A general method for combining conditioning and bucket elimination is also presented. For all the algorithms, bounds on complexity are given as a function of the problem's structure.

1. Overview

Bucket elimination is a unifying algorithmic framework that generalizes dynamic programming to accommodate algorithms for many complex problem-solving and reasoning activities, including directional resolution for propositional satisfiability (Davis and Putnam, 1960), adaptive consistency for constraint satisfaction (Dechter and Pearl, 1987), Fourier and Gaussian elimination for linear equalities and inequalities, and dynamic programming for combinatorial optimization (Bertele and Brioschi, 1972). Here, after presenting the framework, we demonstrate that a number of algorithms for probabilistic inference can also be expressed as bucket-elimination algorithms.

The main virtues of the bucket-elimination framework are *simplicity* and *generality*. By simplicity, we mean that a complete specification of bucket-elimination algorithms is feasible without introducing extensive terminology (e.g., graph concepts such as triangulation and arc-reversal), thus making the algorithms accessible to researchers in diverse areas. More important, the uniformity of the algorithms facilitates understanding, which encourages cross-fertilization and technology transfer between disciplines. Indeed, all bucket-elimination algorithms are similar enough for any improvement to a single algorithm to be applicable to all others expressed in this framework. For example, expressing probabilistic inference algorithms as bucket-elimination methods clarifies the former's relationship to dynamic programming and to constraint satisfaction such that the knowledge accumulated in those areas may be utilized in the probabilistic framework.

The generality of bucket elimination can be illustrated with an algorithm in the area of deterministic reasoning. Consider the following algorithm for deciding satisfiability. Given a set of clauses (a clause is a disjunction of propositional variables or their negations) and an ordering of the propositional variables, $d = Q_1, ..., Q_n$, algorithm *directional resolution* (DR) (Dechter and Rish, 1994), is the core of the well-known Davis-Putnam algorithm for satisfiability (Davis and Putnam, 1960). The algorithm is described using *buckets* partitioning the given set of clauses such that all the clauses containing Q_i that do not contain any symbol higher in the ordering are placed in the bucket of Q_i, denoted $bucket_i$.

The algorithm (see Figure 1) processes the buckets in the reverse order of d. When processing $bucket_i$, it resolves over Q_i all possible pairs of clauses in the bucket and inserts the resolvents into the appropriate lower buckets. It was shown that if the empty clause is not generated in this process then the theory is satisfiable and a satisfying truth assignment can be generated in time linear in the size of the resulting theory. The complexity of the algorithm is exponentially bounded (time and space) in a graph parameter called *induced width* (also called *tree-width*) of the *interaction graph* of the theory, where a node is associated with a proposition and an arc connects any two nodes appearing in the same clause (Dechter and Rish, 1994).

The belief-network algorithms we present in this paper have much in common with the resolution procedure above. They all possess the property of compiling a theory into one from which answers can be extracted easily and their complexity is dependent on the same induced width graph parameter. The algorithms are variations on known algorithms and, for the most part, are not new, in the sense that the basic ideas have existed for some time (Cannings *et al.*, 1978; Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Tatman and Shachter, 1990; Jensen *et al.*, 1990; R.D. Shachter and Favro, 1990; Bacchus and van Run, 1995; Shachter, 1986; Shachter, 1988; Shimony and Charniack, 1991; Shenoy, 1992). What we are presenting here is a syntactic and uniform exposition emphasizing these algorithms' form

Algorithm directional resolution
Input: A set of clauses φ, an ordering $d = Q_1, ..., Q_n$.
Output: A decision of whether φ is satisfiable. If it is, an equivalent output theory; else, an empty output theory.
1. **Initialize:** Generate an ordered partition of the clauses, $bucket_1, ..., bucket_n$, where $bucket_i$ contains all the clauses whose highest literal is Q_i.
2. For $p = n$ to 1, do
• **if** $bucket_p$ contains a unit clause, perform only unit resolution. Put each resolvent in the appropriate bucket.
• **else,** resolve each pair $\{(\alpha \vee Q_p), (\beta \vee \neg Q_p)\} \subseteq bucket_p$. If $\gamma = \alpha \vee \beta$ is empty, φ is not satisfiable, else, add γ to the appropriate bucket.
3. **Return:** $\bigcup_i bucket_i$.

Figure 1. Algorithm *directional resolution*

as a straightforward elimination algorithm. The main virtue of this presentation, beyond uniformity, is that it allows ideas and techniques to flow across the boundaries between areas of research. In particular, having noted that elimination algorithms and clustering algorithms are very similar in the context of constraint processing (Dechter and Pearl, 1989), we find that this similarity carries over to all other tasks. We also show that the idea of *conditioning*, which is as universal as that of elimination, can be incorporated and exploited naturally and uniformly within the elimination framework.

Conditioning is a generic name for algorithms that search the space of partial value assignments, or partial conditionings. Conditioning means splitting a problem into subproblems based on a certain condition. Algorithms such as *backtracking* and *branch and bound* may be viewed as conditioning algorithms. The complexity of conditioning algorithms is exponential in the conditioning set, however, their space complexity is only linear. Our resulting hybrid of conditioning with elimination which trade off time for space (see also (Dechter, 1996b; R. D. Shachter and Solovitz, 1991)), are applicable to all algorithms expressed within this framework.

The work we present here also fits into the framework developed by Arnborg and Proskourowski (Arnborg, 1985; Arnborg and Proskourowski, 1989). They present table-based reductions for various NP-hard graph problems such as the independent-set problem, network reliability, vertex cover, graph k-colorability, and Hamilton circuits. Here and elsewhere (Dechter and van Beek, 1995; Dechter, 1997) we extend the approach to a different set of problems.

Following preliminaries (section 2), we present the bucket-elimination algorithm for belief updating and analyze its performance (section 3). Then, we extend the algorithm to the tasks of finding the most probable explanation (section 4), and extend it to the tasks of finding the maximum a posteriori hypothesis (section 5) and for finding the maximum expected utility (section 6). Section 7 relates the algorithms to Pearl's poly-tree algorithms and to join-tree clustering. We then describe schemes for combining the conditioning method with elimination (section 8). Conclusions are given in section 9.

2. Preliminaries

Belief networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. It is defined by a directed acyclic graph over nodes representing random variables that takes value from given domains. The arcs signify the existance of direct causal influences between the linked variables, and the strength of these influences are quantified by conditional probabilities. A belief network relies on the notion of a directed graph.

A *directed graph* is a pair, $G = \{V, E\}$, where $V = \{X_1, ..., X_n\}$ is a set of elements and $E = \{(X_i, X_j)|X_i, X_j \in V, i \neq j\}$ is the set of edges. If $(X_i, X_j) \in E$, we say that X_i points to X_j. For each variable X_i, the set of parent nodes of X_i, denoted $pa(X_i)$, comprises the variables pointing to X_i in G, while the set of child nodes of X_i, denoted $ch(X_i)$, comprises the variables that X_i points to. Whenever no confusion can arise, we abbreviate $pa(X_i)$ by pa_i and $ch(X_i)$ by ch_i. The family of X_i, F_i, includes X_i and its child variables. A directed graph is *acyclic* if it has no directed cycles. In an *undirected graph*, the directions of the arcs are ignored: (X_i, X_j) and (X_j, X_i) are identical.

Let $X = \{X_1, ..., X_n\}$ be a set of random variables over multivalued domains, $D_1, ..., D_n$. A *belief network* is a pair (G, P) where G is a directed acyclic graph and $P = \{P_i\}$, where P_i denotes probabilistic relationships between X_i and its parents, namely conditional probability matrices $P_i = \{P(X_i|pa_i)\}$. The belief network represents a probability distribution over X having the product form

$$P(x_1, ..., x_n) = \Pi_{i=1}^{n} P(x_i|x_{pa_i})$$

where an assignment $(X_1 = x_1, ..., X_n = x_n)$ is abbreviated to $x = (x_1, ..., x_n)$ and where x_S denotes the projection of a tuple x over a subset of variables S. An evidence set e is an instantiated subset of variables. $A = a$ denotes a partial assignment to a subset of variables A from their respective domains. We use upper case letter for variables and nodes in a graph and lower case letters for values in variable's domains.

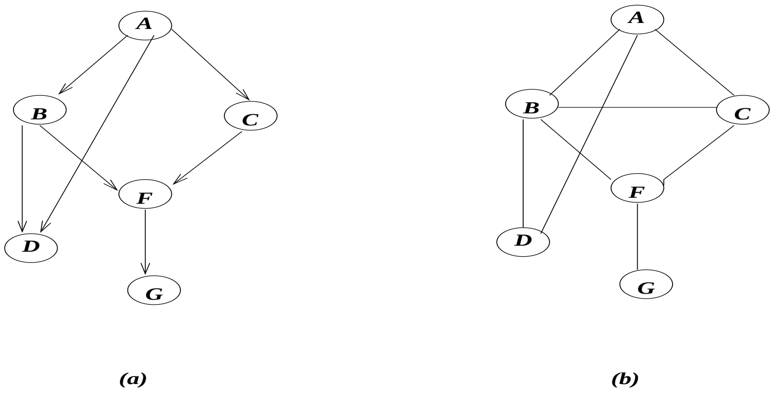

Figure 2. belief network $P(g, f, d, c, b, a) = P(g|f)P(f|c, b)P(d|b, a)P(b|a)P(c|a)$

Example 2.1 *Consider the belief network defined by*

$$P(g, f, d, c, b, a) = P(g|f)P(f|c, b)P(d|b, a)P(b|a)P(c|a).$$

Its acyclic graph is given in Figure 2a. In this case, $pa(F) = \{B, C\}$.

The following queries are defined over belief networks: 1. *belief updating* namely, given a set observations, computing the posterior probability of each proposition, 2. *Finding the most probable explanation (mpe)*, or, given some observed variables, finding a maximum probability assignment to the rest of the variables 3. *Finding the maximum aposteriori hypothesis (map)*, or, given some evidence, finding an assignment to a subset of hypothesis variables that maximizes their probability, and finally, 4. given also a utility function, finding an assignment to a subset of decision variables that *maximizes the expected utility of the problem (meu)*.

It is known that these tasks are NP-hard. Nevertheless, they all permit a polynomial propagation algorithm for singly-connected networks (Pearl, 1988). The two main approaches to extending this propagation algorithm to multiply-connected networks are the *cycle-cutset* approach, also called *conditioning*, and *tree-clustering* (Pearl, 1988; Lauritzen and Spiegelhalter, 1988; Shachter, 1986). These methods work well for sparse networks with small cycle-cutsets or small clusters. In subsequent sections bucket-elimination algorithms for each of the above tasks will be presented and relationship with existing methods will be discussed.

We conclude this section with some notational conventions. Let u be a partial tuple, S a subset of variables, and X_p a variable not in S. We use (u_S, x_p) to denote the tuple u_S appended by a value x_p of X_p.

Definition 2.2 (elimination functions) *Given a function h defined over subset of variables S, where $X \in S$, the functions* $(min_X h)$, $(max_X h)$,

$(mean_X h)$, and $(\sum_X h)$ are defined over $U = S - \{X\}$ as follows. For every $U = u$, $(min_X h)(u) = \min_x h(u,x)$, $(max_X h)(u) = \max_x h(u,x)$, $(\sum_X h)(u) = \sum_x h(u,x)$, and $(mean_X h)(u) = \sum_x \frac{h(u,x)}{|X|}$, where $|X|$ is the cardinality of X's domain. Given a set of functions $h_1, ..., h_j$ defined over the subsets $S_1, ..., S_j$, the product function $(\Pi_j h_j)$ and $\sum_J h_j$ are defined over $U = \cup_j S_j$. For every $U = u$, $(\Pi_j h_j)(u) = \Pi_j h_j(u_{S_j})$, and $(\sum_j h_j)(u) = \sum_j h_j(u_{S_j})$.

3. An Elimination Algorithm for Belief Assessment

Belief updating is the primary inference task over belief networks. The task is to maintain the probability of singleton propositions once new evidence arrives. Following Pearl's propagation algorithm for singly-connected networks (Pearl, 1988), researchers have investigated various approaches to belief updating. We will now present a step by step derivation of a general variable-elimination algorithm for belief updating. This process is typical for any derivation of elimination algorithms.

Let $X_1 = x_1$ be an atomic proposition. The problem is to assess and update the belief in x_1 given some evidence e. Namely, we wish to compute $P(X_1 = x_1|e) = \alpha \cdot P(X_1 = x_1, e)$, where α is a normalization constant. We will develop the algorithm using example 2.1 (Figure 2). Assume we have the evidence $g = 1$. Consider the variables in the order $d_1 = A, C, B, F, D, G$. By definition we need to compute

$$P(a, g = 1) = \sum_{c,b,f,d,g=1} P(g|f)P(f|b,c)P(d|a,b)P(c|a)P(b|a)P(a)$$

We can now apply some simple symbolic manipulation, migrating each conditional probability table to the left of summation variables which it does not reference, we get

$$= P(a) \sum_c P(c|a) \sum_b P(b|a) \sum_f P(f|b,c) \sum_d P(d|b,a) \sum_{g=1} P(g|f) \qquad (1)$$

Carrying the computation from right to left (from G to A), we first compute the rightmost summation which generates a function over f, $\lambda_G(f)$ defined by: $\lambda_G(f) = \sum_{g=1} P(g|f)$ and place it as far to the left as possible, yielding

$$P(a) \sum_c P(c|a) \sum_b P(b|a) \sum_f P(f|b,c)\lambda_G(f) \sum_d P(d|b,a) \qquad (2)$$

$$
\begin{aligned}
bucket_G = &\quad P(g|f), g = 1\\
bucket_D = &\quad P(d|b,a)\\
bucket_F = &\quad P(f|b,c)\\
bucket_B = &\quad P(b|a)\\
bucket_C = &\quad P(c|a)\\
bucket_A = &\quad P(a)
\end{aligned}
$$

Figure 3. Initial partitioning into buckets using $d_1 = A, C, B, F, D, G$

Summing next over d (generating a function denoted $\lambda_D(a,b)$, defined by $\lambda_D(a,b) = \sum_d P(d|a,b)$), we get

$$P(a) \sum_c P(c|a) \sum_b P(b|a) \lambda_D(a,b) \sum_f P(f|b,c) \lambda_G(f) \tag{3}$$

Next, summing over f (generating $\lambda_F(b,c) = \sum_f P(f|b,c)\lambda_G(f)$), we get,

$$P(a) \sum_c P(c|a) \sum_b P(b|a) \lambda_D(a,b) \lambda_F(b,c) \tag{4}$$

Summing over b (generating $\lambda_B(a,c)$), we get

$$P(a) \sum_c P(c|a) \lambda_B(a,c) \tag{5}$$

Finally, summing over c (generating $\lambda_C(a)$), we get

$$P(a)\lambda_C(a) \tag{6}$$

The answer to the query $P(a|g = 1)$ can be computed by evaluating the last product and then normalizing.

The bucket-elimination algorithm mimics the above algebraic manipulation using a simple organizational devise we call *buckets*, as follows. First, the conditional probability tables ($CPTs$, for short) are partitioned into buckets, relative to the order used $d_1 = A, C, B, F, D, G$, as follows (going from last variable to first varaible): in the bucket of G we place all functions mentioning G. From the remaining CPTs we place all those mentioning D in the bucket of D, and so on. The partitioning rule can be alternatively stated as follows. In the bucket of variable X_i we put all functions that mention X_i but do not mention any variable having a higher index. The resulting initial partitioning for our example is given in Figure 3. Note that observed variables are also placed in their corresponding bucket.

This initialization step corresponds to deriving the expression in Eq. (1). Now we process the buckets from top to bottom, implementing the

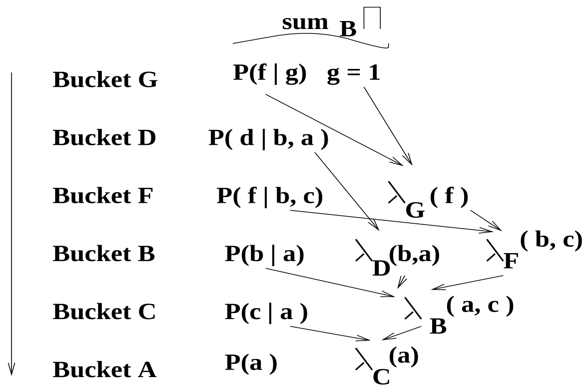

Figure 4. Bucket elimination along ordering $d_1 = A, C, B, F, D, G$.

right to left computation of Eq. (1). $Bucket_G$ is processed first. Processing a bucket amounts to eliminating the variable in the bucket from subsequent computation. To eliminate G, we sum over all values of g. Since, in this case we have an observed value $g = 1$ the summation is over a singleton value. Namely, $\lambda_G(f) = \sum_{g=1} P(g|f)$, is computed and placed in $bucket_F$ (this corresponds to deriving Eq. (2) from Eq. (1)). New functions are placed in lower buckets using the same placement rule.

$Bucket_D$ is processed next. We sum-out D getting $\lambda_D(b, a) = \sum_d P(d|b, a)$, that is computed and placed in $bucket_B$, (which corresponds to deriving Eq. (3) from Eq. (2)). The next variable is F. $Bucket_F$ contains two functions $P(f|b, c)$ and $\lambda_G(f)$, and thus, following Eq. (4) we generate the function $\lambda_F(b, c) = \sum_f P(f|b, c) \cdot \lambda_G(f)$ which is placed in $bucket_B$ (this corresponds to deriving Eq. (4) from Eq. (3)). In processing the next $bucket_B$, the function $\lambda_B(a, c) = \sum_b (P(b|a) \cdot \lambda_D(b, a) \cdot \lambda_F(b, c))$ is computed and placed in $bucket_C$ (deriving Eq. (5) from Eq. (4)). In processing the next $bucket_C$, $\lambda_C(a) = \sum_{c \in C} P(c|a) \cdot \lambda_B(a, c)$ is computed (which corresponds to deriving Eq. (6) from Eq. (5)). Finally, the belief in a can be computed in $bucket_A$, $P(a|g = 1) = P(a) \cdot \lambda_C(a)$. Figure 4 summarizes the flow of computation of the bucket elimination algorithm for our example. Note that since throughout this process we recorded two-dimensional functions at the most, the complexity the algorithm using ordering d_1 is (roughly) time and space quadratic in the domain sizes.

What will occur if we use a different variable ordering? For example, lets apply the algorithm using $d_2 = A, F, D, C, B, G$. Applying algebraic manipulation from right to left along d_2 yields the following sequence of derivations:

$P(a, g = 1) = P(a) \sum_f \sum_d \sum_c P(c|a) \sum_b P(b|a)\ P(d|a, b) P(f|b, c) \sum_{g=1} P(g|f) =$

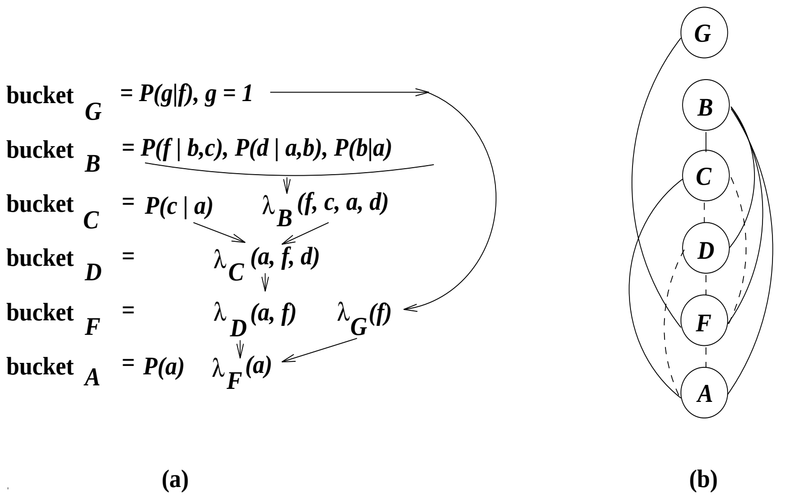

Figure 5. The buckets output when processing along $d_2 = A, F, D, C, B, G$

$P(a)\sum_f \lambda_G(f)\sum_d \sum_c P(c|a)\sum_b P(b|a)\ P(d|a,b)P(f|b,c)=$
$P(a)\sum_f \lambda_G(f)\sum_d \sum_c P(c|a)\lambda_B(a,d,c,f) =$
$P(a)\sum_f \lambda_g(f)\sum_d \lambda_C(a,d,f) =$
$P(a)\sum_f \lambda_G(f)\lambda_D(a,f) =$
$P(a)\lambda_F(a)$

The bucket elimination process for ordering d_2 is summarized in Figure 5a. Each bucket contains the initial $CPTs$ denoted by P's, and the functions generated throughout the process, denoted by λs.

We summarize with a general derivation of the bucket elimination algorithm, called *elim-bel*. Consider an ordering of the variables $X = (X_1, ..., X_n)$. Using the notation $\bar{x}_i = (x_1, ..., x_i)$ and $\bar{x}_i^j = (x_i, x_{i+1}, ..., x_j)$, where F_i is the family of variable X_i, we want to compute:

$$P(x_1, e) = \sum_{x=\bar{x}_2^n} P(\bar{x}_n, e) = \sum_{\bar{x}_2^{(n-1)}} \sum_{x_n} \Pi_i P(x_i, e|x_{pa_i}) =$$

Seperating X_n from the rest of the variables we get:

$$\sum_{x=\bar{x}_2^{(n-1)}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}) \cdot \sum_{x_n} P(x_n, e|x_{pa_n}) \Pi_{X_i \in ch_n} P(x_i, e|x_{pa_i}) =$$

$$\sum_{x=\bar{x}_2^{(n-1)}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}) \cdot \lambda_n(x_{U_n})$$

where

$$\lambda_n(x_{U_n}) = \sum_{x_n} P(x_n, e|x_{pa_n}) \Pi_{X_i \in ch_n} P(x_i, e|x_{pa_i}) \tag{7}$$

Algorithm elim-bel
Input: A belief network $BN = \{P_1, ..., P_n\}$; an ordering of the variables, $d = X_1, ..., X_n$; evidence e.
Output: The belief in $X_1 = x_1$.
1. **Initialize:** Generate an ordered partition of the conditional probability matrices, $bucket_1, ..., bucket_n$, where $bucket_i$ contains all matrices whose highest variable is X_i. Put each observed variable in its bucket. Let $S_1, ..., S_j$ be the subset of variables in the processed bucket on which matrices (new or old) are defined.
2. **Backward:** For $p \leftarrow n$ downto 1, do
for all the matrices $\lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$, do
• (bucket with observed variable) **if** $X_p = x_p$ appears in $bucket_p$, assign $X_p = x_p$ to each λ_i and then put each resulting function in appropriate bucket.
• **else**, $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$. Generate $\lambda_p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i$ and add λ_p to the largest-index variable in U_p.
3. **Return:** $Bel(x_1) = \alpha P(x_1) \cdot \Pi_i \lambda_i(x_1)$(where the λ_i are in $bucket_1$, α is a normalizing constant).

Figure 6. Algorithm *elim-bel*

Where U_n denoted the variables appearing with X_n in a probability component, excluding X_n. The process continues recursively with X_{n-1}.

Thus, the computation performed in the bucket of X_n is captured by Eq. (7). Given ordering $X_1, ..., X_n$, where the queried variable appears first, the CPTs are partitioned using the rule described earlier. To process each bucket, all the bucket's functions, denoted $\lambda_1, ..., \lambda_j$ and defined over subsets $S_1, ..., S_j$ are multiplied, and then the bucket's variable is eliminated by summation. The computed function is $\lambda_p : U_p \rightarrow R$, $\lambda_p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i$, where $U_p = \cup_i S_i - X_p$. This function is placed in the bucket of its largest-index variable in U_p. The procedure continues recursively with the bucket of the next variable going from last variable to first variable. Once all the buckets are processed, the answer is available in the first bucket. Algorithm elim-bel is described in Figure 6.

Theorem 3.1 *Algorithm elim-bel compute the posterior belief $P(x_1|e)$ for any given ordering of the variables.* □

Both the peeling algorithm for genetic trees (Cannings *et al.*, 1978), and Zhang and Poole's recent algorithm (Zhang and Poole, 1996) are variations of elim-bel.

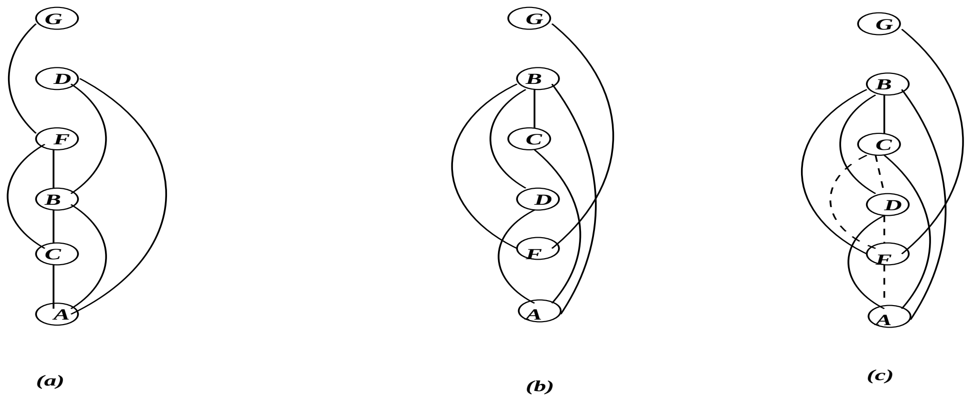

Figure 7. Two ordering of the moral graph of our example problem

3.1. COMPLEXITY

We see that although elim-bel can be applied using any ordering, its complexity varies considerably. Using ordering d_1 we recorded functions on pairs of variables only, while using d_2 we had to record functions on four variables (see $Bucket_C$ in Figure 5a). The arity of the function recorded in a bucket equals the number of variables appearing in that processed bucket, excluding the bucket's variable. Since recording a function of arity r is time and space exponential in r we conclude that the complexity of the algorithm is exponential in the size of the largest bucket which depends on the order of processing.

Fortunately, for any variable ordering bucket sizes can be easily read in advance from an ordered associated with the elimination process. Consider the *moral graph* of a given belief network. This graph has a node for each propositional variable, and any two variables appearing in the same CPT are connected in the graph. The moral graph of the network in Figure 2a is given in Figure 2b. Let us take this moral graph and impose an ordering on its nodes. Figures 7a and 7b depict the ordered moral graph using the two orderings $d_1 = A, C, B, F, D, G$ and $d_2 = A, F, D, C, B, G$. The ordering is pictured from bottom up.

The *width* of each variable in the ordered graph is the number of its *earlier* neighbors in the ordering. Thus the width of G in the ordered graph along d_1 is 1 and the width of F is 2. Notice now that using ordering d_1, the number of variables in the initial buckets of G and F, are also 1, and 2 respectively. Indeed, in the initial partitioning the number of variables mentioned in a bucket (excluding the bucket's variable) is always identical to the width of that node in the corresponding ordered moral graph.

During processing we wish to maintain the correspondance that any two nodes in the graph are connected iff there is function (new or old) deefined on both. Since, during processing, a function is recorded on all the variables apearing in a bucket, we should connect the corresponding nodes in the graph, namely we should connect all the earlier neighbors of a processed variable. If we perform this graph operation recursively from last node to first, (for each node connecting its earliest neighbors) we get the *the induced graph.* The width of each node in this induced graph is identical to the bucket's sizes generated during the elimination process (see Figure 5b).

Example 3.2 *The induced moral graph of Figure 2b, relative to ordering $d_1 = A, C, B, F, D, G$ is depicted in Figure 7a. In this case the ordered graph and its induced ordered graph are identical since all earlier neighbors of each node are already connected. The maximum induced width is 2. Indeed, in this case, the maximum arity of functions recorded by the elimination algorithms is 2. For $d_2 = A, F, D, C, B, G$ the induced graph is depicted in Figure 7c. The width of C is initially 1 (see Figure 7b) while its induced width is* 3. *The maximum induced width over all variables for d_2 is 4, and so is the recorded function's dimensionality.*

A formal definition of all the above graph concepts is given next.

Definition 3.3 *An* ordered graph *is a pair (G, d) where G is an undirected graph and $d = X_1, ..., X_n$ is an ordering of the nodes. The* width of a node *in an ordered graph is the number of the node's neighbors that precede it in the ordering. The* width of an ordering *d, denoted $w(d)$, is the maximum width over all nodes. The* induced width of an ordered graph*, $w^*(d)$, is the width of the induced ordered graph obtained as follows: nodes are processed from last to first; when node X is processed, all its preceding neighbors are connected. The* induced width of a graph*, $w*$, is the minimal induced width over all its orderings. The* tree-width *of a graph is the minimal induced width plus one (*Arnborg, 1985*).*

The established connection between buckets' sizes and induced width motivates finding an ordering with a smallest induced width. While it is known that finding an ordering with the smallest induced width is hard (Arnborg, 1985), usefull greedy heuristics as well as approximation algorithms are available (Dechter, 1992; Becker and Geiger, 1996).

In summary, the complexity of algorithm elim-bel is dominated by the time and space needed to process a bucket. Recording a function on all the bucket's variables is time and space exponential in the number of variables mentioned in the bucket. As we have seen the induced width bounds the arity of the functions recorded; variables appearing in a bucket coincide with the earlier neighbors of the corresponding node in the ordered induced moral graph. In conclusion,

Theorem 3.4 *Given an ordering d the complexity of elim-bel is (time and space) exponential in the induced width $w^*(d)$ of the network's ordered moral graph.* $\square$

3.2. HANDLING OBSERVATIONS

Evidence should be handled in a special way during the processing of buckets. Continuing with our example using elimination on order d_1, suppose we wish to compute the belief in $A = a$ having observed $b = 1$. This observation is relevant only when processing $bucket_B$. When the algorithm arrives at that bucket, the bucket contains the three functions $P(b|a)$, $\lambda_D(b,a)$, and $\lambda_F(b,c)$, as well as the observation $b = 1$ (see Figure 4).

The processing rule dictates computing $\lambda_B(a,c) = P(b = 1|a)\lambda_D(b = 1, a)\lambda_F(b = 1, c)$. Namely, we will generate and record a two-dimensioned function. It would be more effective however, to apply the assignment $b = 1$ to each function in a bucket separately and then put the resulting functions into lower buckets. In other words, we can generate $P(b = 1|a)$ and $\lambda_D(b = 1, a)$, each of which will be placed in the bucket of A, and $\lambda_F(b = 1, c)$, which will be placed in the bucket of C. By so doing, we avoid increasing the dimensionality of the recorded functions. Processing buckets containing observations in this manner automatically exploits the cutset conditioning effect (Pearl, 1988). Therefore, the algorithm has a special rule for processing buckets with observations: the observed value is assigned to each function in the bucket, and each resulting function is moved individually to a lower bucket.

Note that, if the bucket of B had been at the top of our ordering, as in d_2, the virtue of conditioning on B could have been exploited earlier. When processing $bucket_B$ it contains $P(b|a), P(d|b,a), P(f|c,b)$, and $b = 1$ (see Figure 5a). The special rule for processing buckets holding observations will place $P(b = 1|a)$ in $bucket_A$, $P(d|b = 1, a)$ in $bucket_D$, and $P(f|c, b = 1)$ in $bucket_F$. In subsequent processing, only one-dimensional functions will be recorded.

We see that the presence of observations reduces complexity. Since the buckets of observed variables are processed in linear time, and the recorded functions do not create functions on new subsets of variables, the corresponding new arcs should not be added when computing the induced graph. Namely, earlier neighbors of observed variables should *not* be connected.

To capture this refinement we use the notion of *adjusted induced graph* which is defined recursively as follows. Given an ordering and given a set of observed nodes, the adjusted induced graph is generated by processing from top to bottom, connecting the earlier neighbors of unobserved nodes only. The *adjusted induced width* is the width of the adjusted induced graph.

Theorem 3.5 *Given a belief network having n variables, algorithm elim-bel when using ordering d and evidence e, is (time and space) exponential in the adjusted induced width $w^*(d, e)$ of the network's ordered moral graph.* □

3.3. FOCUSING ON RELEVANT SUBNETWORKS

We will now present an improvement to elim-bel whose essence is restricting the computation to relevant portions of the belief network. Such restrictions are already available in the literature in the context of existing algorithms (Geiger *et al.*, 1990; Shachter, 1990).

Since summation over all values of a probability function is 1, the recorded functions of some buckets will degenerate to the constant 1. If we could recognize such cases in advance, we could avoid needless computation by skipping some buckets. If we use a topological ordering of the belief network's acyclic graph (where parents precede their child nodes), and assuming that the queried variable starts the ordering[1], we can recognize skipable buckets dynamically, during the elimination process.

Proposition 3.6 *Given a belief network and a topological ordering $X_1, ..., X_n$, algorithm elim-bel can skip a bucket if at the time of processing, the bucket contains no evidence variable, no query variable and no newly computed function.* □

Proof: If topological ordering is used, each bucket (that does not contain the queried variable) contains initially at most one function describing its probability conditioned on all its parents. Clearly if there is no evidence, summation will yield the constant 1. □

Example 3.7 *Consider again the belief network whose acyclic graph is given in Figure 2a and the ordering $d_1 = A, C, B, F, D, G$, and assume we want to update the belief in variable A given evidence on F. Clearly the buckets of G and D can be skipped and processing should start with $bucket_F$. Once the bucket of F is processed, all the rest of the buckets are not skipable.*

Alternatively, the relevant portion of the network can be precomputed by using a recursive marking procedure applied to the ordered moral graph. (see also (Zhang and Poole, 1996)).

Definition 3.8 *Given an acyclic graph and a topological ordering that starts with the queried variable, and given evidence e, the marking process works as follows. An evidence node is marked, a neighbor of the query variable is marked, and then any earlier neighbor of a marked node is marked.*

[1] otherwise, the queried variable can be moved to the top of the ordering

Algorithm elim-bel

. . .

2. **Backward:** For $p \leftarrow n$ downto 1, do
for all the matrices $\lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$, do
• (bucket with observed variable) **if** $X_p = x_p$ appears in $bucket_p$, then substitute $X_p = x_p$ in each matrix λ_i and put each in appropriate bucket.
• **else**, *if* $bucket_p$ *is NOT skipable*, then
$U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$ $\lambda_p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i$. Add λ_p to the largest-index variable in U_p.

. . .

Figure 8. Improved algorithm elim-bel

The marked belief subnetwork, obtained by deleting all *unmarked* nodes, can be processed now by elim-bel to answer the belief-updating query. It is easy to see that

Theorem 3.9 *The complexity of algorithm elim-bel given evidence e is exponential in the adjusted induced width of the marked ordered moral subgraph.*

Proof: Deleting the unmarked nodes from the belief network results in a belief subnetwork whose distribution is identical to the marginal distribution over the marked variables. □.

4. An Elimination Algorithm for mpe

In this section we focus on the task of finding the most probable explanation. This task appears in applications such as diagnosis and abduction. For example, it can suggest the disease from which a patient suffers given data on clinical findings. Researchers have investigated various approaches to finding the *mpe* in a belief network. (See, e.g., (Pearl, 1988; Cooper, 1984; Peng and Reggia, 1986; Peng and Reggia, 1989)). Recent proposals include best first-search algorithms (Shimony and Charniack, 1991) and algorithms based on linear programming (Santos, 1991).

The problem is to find x^0 such that $P(x^0) = \max_x \Pi_i P(x_i, e|x_{pa_i})$ where $x = (x_1, ..., x_n)$ and e is a set of observations. Namely, computing for a given ordering $X_1, ..., X_n$,

$$M = \max_{\bar{x}_n} P(x) = \max_{\bar{x}_{n-1}} \max_{x_n} \Pi_{i=1}^{n} P(x_i, e|x_{pa_i}) \tag{8}$$

This can be accomplished as before by performing the mximization operation along the ordering from right to left, while migrating to the left, at each step, all components that do not mention the maximizing variable. We get,

$$M = \max_{x=\bar{x}_n} P(\bar{x}_n, e) = \max_{\bar{x}_{(n-1)}} \max_{x_n} \Pi_i P(x_i, e|x_{pa_i}) =$$

$$\max_{x=\bar{x}_{n-1}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}) \cdot \max_{x_n} P(x_n, e|x_{pa_n}) \Pi_{X_i \in ch_n} P(x_i, e|x_{pa_i}) =$$

$$\max_{x=\bar{x}_{n-1}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}) \cdot h_n(x_{U_n})$$

where

$$h_n(x_{U_n}) = \max_{x_n} P(x_n, e|x_{pa_n}) \Pi_{X_i \in ch_n} P(x_i, e|x_{pa_i})$$

Where U_n are the variables appearing in components defined over X_n. Clearly, the algebraic manipulation of the above expressions is the same as the algebraic manipulation for belief assessment where summation is replaced by maximization. Consequently, the bucket-elimination procedure *elim-mpe* is identical to elim-bel except for this change. Given ordering $X_1, ..., X_n$, the conditional probability tables are partitioned as before. To process each bucket, we multiply all the bucket's matrices, which in this case are denoted $h_1, ..., h_j$ and defined over subsets $S_1, ..., S_j$, and then eliminate the bucket's variable by maximization. The computed function in this case is $h_p : U_p \rightarrow R$, $h_p = \max_{X_p} \Pi_{i=1}^{j} h_i$, where $U_p = \cup_i S_i - X_p$. The function obtained by processing a bucket is placed in the bucket of its largest-index variable in U_p. In addition, a function $x_p^o(u) = argmax_{X_p} h_p(u)$, which relates an optimizing value of X_p with each tuple of U_p, is recorded and placed in the bucket of X_p.

The procedure continues recursively, processing the bucket of the next variable while going from last variable to first variable. Once all buckets are processed, the *mpe* value can be extracted in the first bucket. When this *backwards* phase terminates the algorithm initiates a *forwards phase* to compute an *mpe* tuple.

Forward phase: Once all the variables are processed, an *mpe* tuple is computed by assigning values along the ordering from X_1 to X_n, consulting the information recorded in each bucket. Specifically, once the partial assignment $x = (x_1, ..., x_{i-1})$ is selected, the value of X_i appended to this tuple is $x_i^o(x)$, where x^o is the function recorded in the backward phase. The algorithm is presented in Figure 9. Observed variables are handled as in elim-bel.

Example 4.1 *Consider again the belief network of Figure 2. Given the ordering $d = A, C, B, F, D, G$ and the evidence $g = 1$, process variables from last to the first after partitioning the conditional probability matrices into buckets, such that $bucket_G = \{P(g|f), g = 1\}$, $bucket_D = \{P(d|b, a)\}$,*

Algorithm elim-mpe
Input: A belief network $BN = \{P_1, ..., P_n\}$; an ordering of the variables, d; observations e.
Output: The most probable assignment.
1. **Initialize:** Generate an ordered partition of the conditional probability matrices, $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all matrices whose highest variable is X_i. Put each observed variable in its bucket. Let $S_1, ..., S_j$ be the subset of variables in the processed bucket on which matrices (new or old) are defined.
2. **Backward:** For $p \leftarrow n$ downto 1, do
for all the matrices $h_1, h_2, ..., h_j$ in $bucket_p$, do
• (bucket with observed variable) **if** $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each h_i and put each in appropriate bucket.
• **else**, $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$. Generate functions $h_p = \max_{X_p} \Pi_{i=1}^{j} h_i$ and $x_p^o = argmax_{X_p} h_p$. Add h_p to bucket of largest-index variable in U_p.
3. **Forward:** Assign values in the ordering d using the recorded functions x^o in each bucket.

Figure 9. Algorithm *elim-mpe*

$bucket_F = \{P(f|b,c)\}$, $bucket_B = \{P(b|a)\}$, $bucket_C = \{P(c|a)\}$, and $bucket_A = \{P(a)\}$. To process G, assign $g = 1$, get $h_G(f) = P(g = 1|f)$, and place the result in $bucket_F$. The function $G^o(f) = argmax h_G(f)$ is placed in $bucket_G$ as well. Process $bucket_D$ by computing $h_D(b,a) = \max_d P(d|b,a)$ and putting the result in $bucket_B$. Record also $D^o(b,a) = argmax_D P(d|b,a)$. The bucket of F, to be processed next, now contains two matrices: $P(f|b,c)$ and $h_G(f)$. Compute $h_F(b,c) = \max_f p(f|b,c) \cdot h_G(f)$, and place the resulting function in $bucket_B$. To eliminate B, we record the function $h_B(a,c) = \max_b P(b|a) \cdot h_D(b,a) \cdot h_F(b,c)$ and place it in $bucket_C$. To eliminate C, we compute $h_C(a) = \max_c P(c|a) \cdot h_B(a,c)$ and place it in $bucket_A$. Finally, the mpe value is given in $bucket_A$, $M = \max_a P(a) \cdot h_C(a)$, is determined, along with the mpe tuple, by going forward through the buckets.

The backward process can be viewed as a compilation (or learning) phase, in which we compile information regarding the most probable extension of partial tuples to variables higher in the ordering (see also section 7.2).

Similarly to the case of belief updating, the complexity of elim-mpe is bounded exponentially in the dimension of the recorded matrices, and those are bounded by the induced width $w^*(d,e)$ of the ordered moral graph. In summary,

Theorem 4.2 *Algorithm elim-mpe is complete for the mpe task. Its complexity (time and space) is $O(n \cdot exp(w^*(d,e)))$, where n is the number of variables and $w^*(d,e)$ is the e-adjusted induced width of the ordered moral graph.* $\square$

5. An Elimination Algorithm for MAP

We next present an elimination algorithm for the map task. By its definition, the task is a mixture of the previous two, and thus in the algorithm some of the variables are eliminated by summation, others by maximization.

Given a belief network, a subset of hypothesized variables $A = \{A_1, ..., A_k\}$, and some evidence e, the problem is to find an assignment to the hypothesized variables that maximizes their probability given the evidence. Formally, we wish to compute $\max_{\bar{a}_k} P(x,e) = \max_{\bar{a}_k} \sum_{\bar{x}^n_{k+1}} \Pi^n_{i=1} P(x_i, e|x_{pa_i})$ where $x = (a_1, ..., a_k, x_{k+1}, ..., x_n)$. In the algebraic manipulation of this expression, we push the maximization to the left of the summation. This means that in the elimination algorithm, the maximized variables should initiate the ordering (and therefore will be processed last). Algorithm *elim-map* in Figure 10 considers only orderings in which the hypothesized variables start the ordering. The algorithm has a backward phase and a forward phase, but the forward phase is relative to the hypothesized variables only. Maximization and summation may be somewhat interleaved to allow more effective orderings; however, we do not incorporate this option here. Note that the relevant graph for this task can be restricted by marking in a very similar manner to belief updating case. In this case the initial marking includes all the hypothesized variables, while otherwise, the marking procedure is applied recursively to the summation variables only.

Theorem 5.1 *Algorithm elim-map is complete for the map task. Its complexity is $O(n \cdot exp(w^*(d,e))$, where n is the number of variables in the relevant marked graph and $w^*(d,e)$ is the e-adjusted induced width of its marked moral graph.* $\square$

6. An Elimination Algorithm for MEU

The last and somewhat more complicated task we address is that of finding the maximum expected utility. Given a belief network, evidence e, a real-valued utility function $u(x)$ additively decomposable relative to functions $f_1, ..., f_j$ defined over $Q = \{Q_1, ..., Q_j\}$, $Q_i \subseteq X$, such that $u(x) = \sum_{Q_j \in Q} f_j(x_{Q_j})$, and a subset of decision variables $D = \{D_1, ...D_k\}$ that are assumed to be root nodes, the meu task is to find a set of decisions

Algorithm elim-map
Input: A belief network $BN = \{P_1, ..., P_n\}$; a subset of variables $A = \{A_1, ..., A_k\}$; an ordering of the variables, d, in which the A's are first in the ordering; observations e.
Output: A most probable assignment $A = a$.
1. **Initialize:** Generate an ordered partition of the conditional probability matrices, $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all matrices whose highest variable is X_i.
2. **Backwards** For $p \leftarrow n$ downto 1, do
for all the matrices $\beta_1, \beta_2, ..., \beta_j$ in $bucket_p$, do
• (bucket with observed variable) **if** $bucket_p$ contains the observation $X_p = x_p$, assign $X_p = x_p$ to each β_i and put each in appropriate bucket.
• **else**, $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$. If X_p not in A and if $bucket_p$ contains new functions, then $\beta_p = \sum_{X_p} \Pi_{i=1}^{j} \beta_i$; else, $X_p \in A$, and $\beta_p = \max_{X_p} \Pi_{i=1}^{j} \beta_i$ and $a^0 = argmax_{X_p} \beta_p$. Add β_p to the bucket of the largest-index variable in U_p.
3. **Forward:** Assign values, in the ordering $d = A_1, ..., A_k$, using the information recorded in each bucket.

Figure 10. Algorithm *elim-map*

$d^o = (d^o{}_1, ..., d^o{}_k)$ that maximizes the expected utility. We assume that the variables *not* appearing in D are indexed $X_{k+1}, ..., X_n$. Formally, we want to compute

$$E = \max_{d_1,\ldots,d_k} \sum_{x_{k+1},\ldots x_n} \Pi_{i=1}^{n} P(x_i, e | x_{pa_i}, d_1, ..., d_k) u(x),$$

and

$$d^0 = argmax_D E$$

As in the previous tasks, we will begin by identifying the computation associated with X_n from which we will extract the computation in each bucket. We denote an assignment to the decision variables by $d = (d_1, ..., d_k)$ and $\bar{x}_k^j = (x_k, ..., x_j)$. Algebraic manipulation yields

$$E = \max_{d} \sum_{\bar{x}_{k+1}^{n-1}} \sum_{x_n} \Pi_{i=1}^{n} P(x_i, e | x_{pa_i}, d) \sum_{Q_j \in Q} f_j(x_{Q_j})$$

We can now separate the components in the utility functions into those mentioning X_n, denoted by the index set t_n, and those not mentioning X_n, labeled with indexes $l_n = \{1, ..., n\} - t_n$. Accordingly we get

$$E = \max_{d} \sum_{\bar{x}_{k+1}^{(n-1)}} \sum_{x_n} \Pi_{i=1}^{n} P(x_i, e | x_{pa_i}, d) \cdot (\sum_{j \in l_n} f_j(x_{Q_j}) + \sum_{j \in t_n} f_j(x_{Q_j}))$$

$$E = \max_d [\sum_{\bar{x}_{k+1}^{(n-1)}} \sum_{x_n} \Pi_{i=1}^n P(x_i, e|x_{pa_i}, d) \sum_{j \in l_n} f_j(x_{Q_j})$$
$$+ \sum_{\bar{x}_{k+1}^{(n-1)}} \sum_{x_n} \Pi_{i=1}^n P(x_i, e|x_{pa_i}, d) \sum_{j \in t_n} f_j(x_{Q_j})]$$

By migrating to the left of X_n all of the elements that are not a function of X_n, we get

$$\max_d [\sum_{\bar{x}_{k+1}^{n-1}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}, d) \cdot (\sum_{j \in l_n} f_j(x_{Q_j})) \sum_{x_n} \Pi_{X_i \in F_n} P(x_i, e|x_{pa_i}, d)$$
$$+ \sum_{\bar{x}_{k+1}^{n-1}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}, d) \cdot \sum_{x_n} \Pi_{X_i \in F_n} P(x_i, e|x_{pa_i}, d) \sum_{j \in t_n} f_j(x_{Q_j})] \tag{9}$$

We denote by U_n the subset of variables that appear with X_n in a probabilistic component, excluding X_n itself, and by W_n the union of variables that appear in probabilistic and utility components with X_n, excluding X_n itself. We define λ_n over U_n as (x is a tuple over $U_n \cup X_n$)

$$\lambda_n(x_{U_n}|d) = \sum_{x_n} \Pi_{X_i \in F_i} P(x_i, e|x_{pa_i}, d) \tag{10}$$

We define θ_n over W_n as

$$\theta_n(x_{W_n}|d) = \sum_{x_n} \Pi_{X_i \in F_n} P(x_i, e|x_{pa_i}, d) \sum_{j \in t_n} f_j(x_{Q_j})) \tag{11}$$

After substituting Eqs. (10) and (11) into Eq. (9), we get

$$E = \max_d \sum_{\bar{x}_{k+1}^{n-1}} \Pi_{X_i \in X - F_n} P(x_i, e|x_{pa_i}, d) \cdot \lambda_n(x_{U_n}|d) [\sum_{j \in l_n} f_j(x_{Q_j}) + \frac{\theta_n(x_{W_n}|d)}{\lambda_n(x_{U_n}|d)}] \tag{12}$$

The functions θ_n and λ_n compute the effect of eliminating X_n. The result (Eq. (12)) is an expression, which does not include X_n, where the product has one more matrix λ_n and the utility components have one more element $\gamma_n = \frac{\theta_n}{\lambda_n}$. Applying such algebraic manipulation to the rest of the variables in order yields the elimination algorithm *elim-meu* in Figure 11. We assume here that decision variables are processed last by elim-meu. Each bucket contains utility components θ_i and probability components, λ_i. When there is no evidence, λ_n is a constant and we can incorporate the marking modification we presented for elim-bel. Otherwise, during processing, the algorithm generates the λ_i of a bucket by multiplying all its

Algorithm elim-meu
Input: A belief network $BN = \{P_1, ..., P_n\}$; a subset of variables $D_1, ..., D_k$ that are decision variables which are root nodes; a utility function over X, $u(x) = \sum_j f_j(x_{Q_j})$; an ordering of the variables, o, in which the D's appear first; observations e.
Output: An assignment $d_1, ..., d_k$ that maximizes the expected utility.
1. **Initialize:** Partition components into buckets, where $bucket_i$ contains all matrices whose highest variable is X_i. Call probability matrices $\lambda_1, ..., \lambda_j$ and utility matrices $\theta_1, ..., \theta_l$. Let $S_1, ..., S_j$ be the probability variable subsets and $Q_1, ..., Q_l$ be the utility variable subsets.
2. **Backward:** For $p \leftarrow n$ downto $k+1$, do
for all matrices $\lambda_1, ..., \lambda_j, \theta_1, ..., \theta_l$ in $bucket_p$, do
• (bucket with observed variable) **if** $bucket_p$ contains the observation $X_p = x_p$, then assign $X_p = x_p$ to each λ_i, θ_i and put each resulting matrix in appropriate bucket.
• **else**, $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$ and $W_p \leftarrow U_p \cup (\bigcup_{i=1}^{l} Q_i - \{X_p)\}$. If bucket contains an observation or new λ's, then $\lambda_p = \sum_{X_p} \Pi_i \lambda_i$ and $\theta_p = \frac{1}{\lambda_p} \sum_{X_p} \Pi_{i=1}^{j} \lambda_i \sum_{j=1}^{l} \theta_j$; else, $\theta_p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i \sum_{j=1}^{l} \theta_j$. Add θ_p and λ_p to the bucket of the largest-index variable in W_p and U_p, respectively.
3. **Forward:** Assign values in the ordering $o = D_1, ..., D_k$ using the information recorded in each bucket of the decision variable.

Figure 11. Algorithm *elim-meu*

probability components and summing over X_i. The θ of bucket X_i is computed as the average utility of the bucket; if the bucket is marked, the average utility of the bucket is normalized by its λ. The resulting θ and λ are placed into the appropriate buckets.

The maximization over the decision variables can now be accomplished using maximization as the elimination operator. We do not include this step explicitly, since, given our simplifying assumption that all decisions are root nodes, this step is straightforward. Clearly, maximization and summation can be interleaved to some degree, thus allowing more efficient orderings.

As before, the algorithm's performance can be bounded as a function of the structure of its *augmented graph*. The augmented graph is the moral graph augmented with arcs connecting any two variables appearing in the same utility component f_i, for some i.

Theorem 6.1 *Algorithm elim-meu computes the meu of a belief network augmented with utility components (i.e., an influence diagram) in* $O(n \cdot$

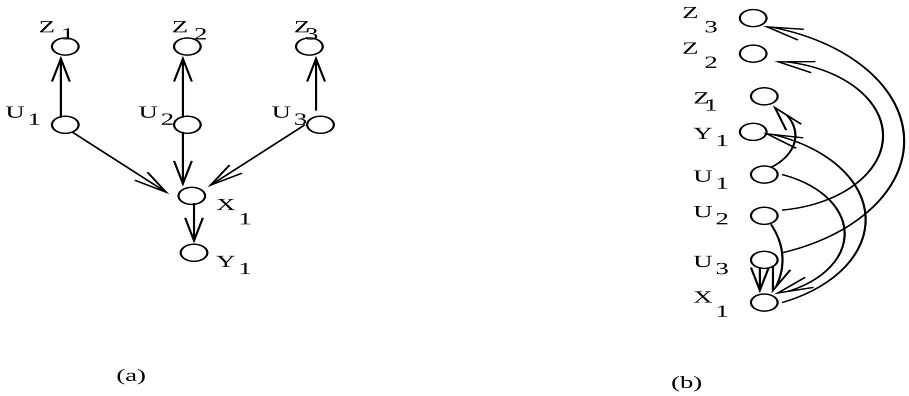

Figure 12. (a) A poly-tree and (b) a legal processing ordering

$exp(w^*(d,e))$, where $w^*(d,e)$ is the induced width along d of the augmented moral graph. □

Tatman and Schachter (Tatman and Shachter, 1990) have published an algorithm that is a variation of elim-meu, and Kjaerulff's algorithm (Kjæaerulff, 1993) can be viewed as a variation of elim-meu tailored to dynamic probabilistic networks.

7. Relation of Bucket Elimination to Other Methods

7.1. POLY-TREE ALGORITHM

When the belief network is a poly-tree, both belief assessment, the mpe task and map task can be accomplished efficiently using Pearl's poly-tree algorithm (Pearl, 1988). As well, when the augmented graph is a tree, the *meu* can be computed efficiently. A poly-tree is a directed acyclic graph whose underlying undirected graph has no cycles.

We claim that if a bucket elimination algorithm process variables in a topological ordering (parents precede their child nodes), then the algorithm coincides (with some minor modifications) with the poly-tree algorithm. We will demonstrate the main idea using bucket elimination for the mpe task. The arguments are applicable for the rest of the tasks.

Example 7.1 *Consider the ordering $X_1, U_3, U_2, U_1, Y_1, Z_1, Z_2, Z_3$ of the poly-tree in Figure 12a, and assume that the last four variables are observed (here we denote an observed value by using primed lowercase letter and leave other variables in lowercase). Processing the buckets from last to first, after the first four buckets have been processed as observation buckets, we get*
$bucket(U_3) = P(u_3), P(x_1|u_1,u_2,u_3), P(z\prime_3|u_3)$
$bucket(U_2) = P(u_2), P(z\prime_2|u_2)$

$bucket(U_1) = P(u_1), P(z\prime_1|u_1)$
$bucket(X_1) = P(y\prime_1|x_1)$
When processing bucket(U_3) by elim-mpe, we get $h_{U_3}(u_1, u_2, u_3)$, which is placed in bucket(U_2). The final resulting buckets are
$bucket(U_3) = P(u_3), P(x_1|u_1, u_2, u_3), P(z\prime_3|u_3)$
$bucket(U_2) = P(u_2), P(z\prime_2|u_2),\ h_{U_3}(x_1, u_2, u_1)$
$bucket(U_1) = P(u_1), P(z\prime_1|u_1),\ h_{U_2}(x_1, u_1)$
$bucket(X_1) = P(y\prime_1|x_1),\ h_{U_1}(x_1)$
We can now choose a value x_1 that maximizes the product in X_1's bucket, then choose a value u_1 that maximizes the product in U_1's bucket given the selected value of X_1, and so on.

It is easy to see that if elim-mpe uses a topological ordering of the poly-tree, it is time and space $O(exp(|F|))$, where $|F|$ is the cardinality of the maximum family size. For instance, in Example 7.1, elim-mpe records the intermediate function $h_{U_3}(X_1, U_2, U_1)$ requiring $O(k^3)$ space, where k bounds the domain size for each variable. Note, however, that Pearl's algorithm (which is also time exponential in the family size) is better, as it records functions on single variables only.

In order to restrict space needs, we modify elim-mpe in two ways. First, we restrict processing to a subset of the topological orderings in which sibling nodes and their parent appear consecutively *as much as possible.* Second, whenever the algorithm reaches a set of consecutive buckets from the same family, all such buckets are combined and processed as one *super-bucket.* With this change, elim-mpe is similar to Pearl's propagation algorithm on poly-trees.[2] Processing a super-bucket amounts to eliminating all the super-bucket's variables without recording intermediate results.

Example 7.2 *Consider Example 7.1. Here, instead of processing each bucket of U_i separately, we compute by a brute-force algorithm the function $h_{U_1,U_2,U_3}(x_1)$ in the super-bucket of U_1, U_2, U_3 and place the function in the bucket of X_1. We get the unary function*
$h_{U_1,U_2,U_3}(x_1) = \max_{u_1,u_2,u_3} P(u_3)P(x_1|u_1, u_2, u_3)P(z\prime_3|u_3)P(u_2)P(z\prime_2|u_2)P(u_1)P(z\prime_1|u_1).$

The details for obtaining an ordering such that all families in a poly-tree can be processed as super-buckets can be worked out, but are beyond the scope of this paper. In summary,

Proposition 7.3 *There exist an ordering of a poly-tree, such that bucket-elimination algorithms (elim-bel, elim-mpe, etc.) with the super-bucket modification have the same time and space complexity as Pearl's poly-tree algorithm for the corresponding tasks. The modified algorithm's time complexity is exponential in the family size, and it requires only linear space.* □

[2] Actually, Pearl's algorithm should be restricted to message passing relative to one rooted tree in order to be identical with ours.

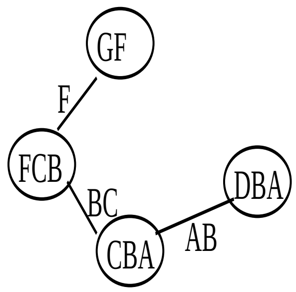

Figure 13. Clique-tree associated with the induced graph of Figure 7a

7.2. JOIN-TREE CLUSTERING

Join-tree clustering (Lauritzen and Spiegelhalter, 1988) and bucket elimination are closely related and their worst-case complexity (time and space) is essentially the same. The sizes of the cliques in tree-clustering is identical to the induced-width plus one of the corresponding ordered graph. In fact, elimination may be viewed as a directional (i.e., goal- or query-oriented) version of join-tree clustering. The close relationship between join-tree clustering and bucket elimination can be used to attribute meaning to the intermediate functions computed by elimination.

Given an elimination ordering, we can generate the ordered moral induced graph whose maximal cliques (namely, a maximal fully-connected subgraph) can be enumerated as follows. Each variable and its earlier neighbors are a clique, and each clique is connected to a parent clique with whom it shares the largest subset of variables (Dechter and Pearl, 1989). For example, the induced graph in Figure 7a yields the clique-tree in Figure 13, If this ordering is used by tree-clustering,the same tree may be generated.

The functions recorded by bucket elimination can be given the following meaning (details and proofs of these claims are beyond the scope of this paper). The function $h_p(u)$ recorded in $bucket_p$ by elim-mpe and defined over $\cup_i S_i - \{X_p\}$, is the maximum probability extension of u, to variables appearing later in the ordering and which are also mentioned in the clique-subtree rooted at a clique containing U_p. For instance, $h_F(b,c)$ recorded by elim-mpe using d_1 (see Example 3.1) equals $\max_{f,g} P(b,c,f,g)$, since F and G appear in the clique-tree rooted at (FCB). For belief assessment, the function $\lambda_p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i$, defined over $U_p = \cup_i S_i - X_p$, denotes the probability of all the evidence e^{+p} observed in the clique subtree rooted at a clique containing U_p, conjoined with u. Namely, $\lambda_p(u) = P(e^{+p}, u)$.

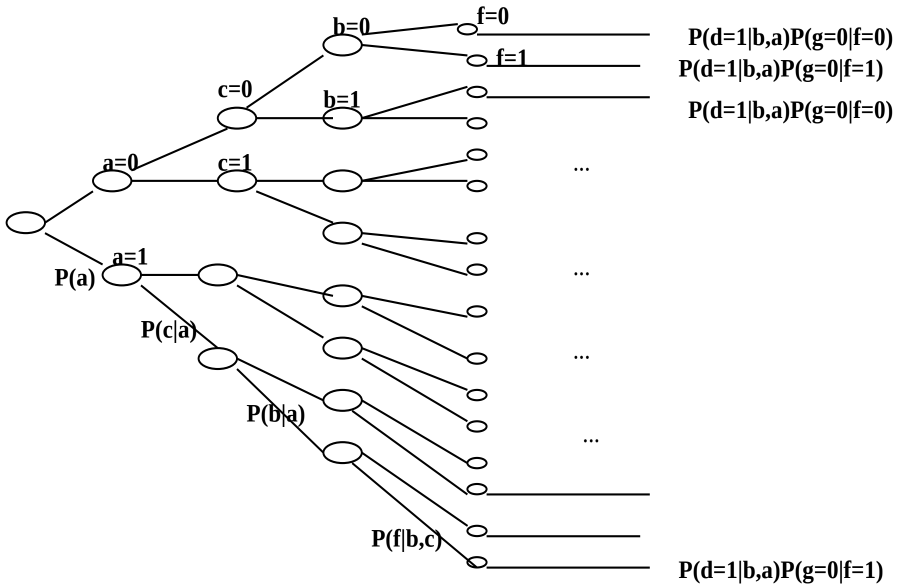

Figure 14. probability tree

8. Combining Elimination and Conditioning

A serious drawback of elimination algorithms is that they require considerable memory for recording the intermediate functions. Conditioning, on the other hand, requires only linear space. By combining conditioning and elimination, we may be able to reduce the amount of memory needed yet still have performance guarantee.

Conditioning can be viewed as an algorithm for processing the algebraic expressions defined for the task, from left to right. In this case, partial results cannot be assembled; rather, partial value assignments (conditioning on subset of variables) unfold a tree of subproblems, each associated with an assignment to some variables. Say, for example, that we want to compute the expression for mpe in the network of Figure 2:

$$M = \max_{a,c,b,f,d,g} P(g|f)P(f|b,c)P(d|a,b)P(c|a)P(b|a)P(a)$$

$$= \max_a P(a) \max_c P(c|a) \max_b P(b|a) \max_f P(f|b,c) \max_d P(d|b,a) \max_g P(g|f). \tag{13}$$

We can compute the expression by traversing the tree in Figure 14, going along the ordering from first variable to last variable. The tree can be traversed either breadth-first or depth-first and will result in known search algorithms such as best-first search and branch and bound.

Algorithm elim-cond-mpe
Input: A belief network $BN = \{P_1, ..., P_n\}$; an ordering of the variables, d; a subset C of conditioned variables; observations e.
Output: The most probable assignment.
Initialize: $p = 0$.

1. For every assignment $C = c$, do
   - $p_1 \leftarrow$ The output of elim-mpe with $c \cup e$ as observations.
   - $p \leftarrow \max\{p, p_1\}$ (update the maximum probability).
2. **Return** p and a maximizing tuple.

Figure 15. Algorithm *elim-cond-mpe*

We will demonstrate the idea of combining conditioning with elimination on the mpe task. Let C be a subset of conditioned variables, $C \subseteq X$, and $V = X - C$. We denote by v an assignment to V and by c an assignment to C. Clearly,

$$\max_x P(x, e) = \max_c \max_v P(c, v, e) = max_{c,v} \Pi_i P(x_i, c, v, e | x_{pa_i})$$

Therefore, for every partial tuple c, we compute $\max_v P(v, c, e)$ and a corresponding maximizing tuple

$$(x_V^o)(c) = argmax_V \Pi_{i=1}^n P(x_i, e, c | x_{pa_i})$$

by using the elimination algorithm while treating the conditioned variables as observed variables. This basic computation will be enumerated for all value combinations of the conditioned variables, and the tuple retaining the maximum probability will be kept. The algorithm is presented in Figure 15.

Given a particular value assignment c, the time and space complexity of computing the maximum probability over the rest of the variables is bounded exponentially by the induced width $w^*(d, e \cup c)$ of the ordered moral graph adjusted for both observed and conditioned nodes. In this case the induced graph is generated without connecting earlier neighbors of both evidence and conditioned variables.

Theorem 8.1 *Given a set of conditioning variables, C, the space complexity of algorithm elim-cond-mpe is $O(n \cdot exp(w^*(d, c \cup e))$, while its time complexity is $O(n \cdot exp(w^*(d, e \cup c) + |C|))$, where the induced width $w^*(d, c \cup e)$, is computed on the ordered moral graph that was adjusted relative to e and c.* $\Box$

When the variables in $e \cup c$ constitute a cycle-cutset of the graph, the graph can be ordered such that its adjusted induced width equals 1. In this case elim-cond-mpe reduces to the known loop-cutset algorithms (Pearl, 1988; Dechter, 1990).

Clearly, algorithm elim-cond-mpe can be implemented more effectively if we take advantage of shared partial assignments to the conditioned variables. There is a variety of possible hybrids between conditioning and elimination that can refine the basic procedure in elim-cond-mpe. One method imposes an upper bound on the arity of functions recorded and decides dynamically, during processing, whether to process a bucket by elimination or by conditioning (see (Dechter and Rish, 1996)). Another method which uses the super-bucket approach, collects a set of consecutive buckets into one super-bucket that it processes by conditioning, thus avoiding recording some intermediate results (Dechter, 1996b; El-Fattah and Dechter, 1996).

9. Related work

We had mentioned throughout this paper many algorithms in probabilistic and deterministic reasoning that can be viewed as similar to bucket-elimination algorithms. In addition, unifying frameworks observing the common features between various algorithms had also appeared both in the past (Shenoy, 1992) and more recently in (Bistarelli *et al.*, 1997).

10. Summary and Conclusion

Using the bucket-elimination framework, which generalizes dynamic programming, we have presented a concise and uniform way of expressing algorithms for probabilistic reasoning. In this framework, algorithms exploit the topological properties of the network without conscience effort on the part of the designer. We have shown, for example, that if algorithms elim-mpe and elim-bel are given a singly-connected network then the algorithm reduces to Pearl's algorithms (Pearl, 1988) for some orderings (always possible on trees). The same applies to elim-map and elim-meu, for which tree-propagation algorithms were not explicitly derived.

The simplicity and elegance of the proposed framework highlights the features common to bucket-elimination and join-tree clustering, and allows for focusing belief-assessment procedures toward the relevant portions of the network. Such enhancements were accompanied by graph-based complexity bounds which are more refined than the standard induced-width bound.

The performance of bucket-elimination and tree-clustering algorithms is likely to suffer from the usual difficulty associated with dynamic programming: exponential space and exponential time in the worst case. Such performance deficiencies also plague resolution and constraint-satisfaction algorithms (Dechter and Rish, 1994; Dechter, 1997). Space complexity can be reduced using conditioning. We have shown that conditioning can be implemented naturally on top of elimination, thus reducing the space requirement while still exploiting topological features. Combining conditioning and elimination can be viewed as combining the virtues of forward and backward search.

Finally, no attempt was made in this paper to optimize the algorithms for distributed computation, nor to exploit compilation vs run-time resources. These issues can and should be addressed within the bucket-elimination framework. In particular, improvements exploiting the structure of the conditional probability matrices, as presented recently in (Santos *et al.*, in press; Boutilier, 1996; Poole, 1997) can be incorporated on top of bucket-elimination.

In summary, what we provide here is a uniform exposition across several tasks, applicable to both probabilistic and deterministic reasoning, which facilitates the transfer of ideas between several areas of research. More importantly, the organizational benefit associated with the use of buckets should allow all the bucket-elimination algorithms to be improved uniformly. This can be done either by combining conditioning with elimination, as we have shown, or via approximation algorithms as is shown in (Dechter, 1997).

11. Acknowledgment

A preliminary version of this paper appeared in (Dechter, 1996a). I would like to thank Irina Rish and Nir Freidman for their useful comments on different versions of this paper. This work was partially supported by NSF grant IRI-9157636, Air Force Office of Scientific Research grant AFOSR F49620-96-1-0224, Rockwell MICRO grants ACM-20775 and 95-043, Amada of America and Electrical Power Research Institute grant RP8014-06.

References

S. Arnborg and A. Proskourowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete and Applied Mathematics*, 23:11–24, 1989.

S.A. Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability - a survey. *BIT*, 25:2–23, 1985.

F. Bacchus and P. van Run. Dynamic variable ordering in csps. In *Principles and Practice of Constraints Programming (CP-95)*, Cassis, France, 1995.

A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal jnmction trees. In *Uncertainty in AI (UAI-96)*, pages 81–89, 1996.

U. Bertele and F. Brioschi. *Nonserial Dynamic Programming.* Academic Press, 1972.

S. Bistarelli, U. Montanari, and F Rossi. Semiring-based constraint satisfaction and optimization. *Journal of the Association of Computing Machinery (JACM)*, to appear, 1997.

C. Boutilier. Context-specific independence in bayesian networks. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 115–123, 1996.

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

G.F. Cooper. Nestor: A computer-based medical diagnosis aid that integrates causal and probabilistic knowledge. Technical report, Computer Science department, Stanford University, Palo-Alto, California, 1984.

M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association of Computing Machinery*, 7(3), 1960.

R. Dechter and J. Pearl. Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, pages 353–366, 1989.

R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Principles of Knowledge Representation and Reasoning (KR-94)*, pages 134–145, 1994.

R. Dechter and I. Rish. To guess or to think? hybrid algorithms for sat. In *Principles of Constraint Programming (CP-96)*, 1996.

R. Dechter and P. van Beek. Local and global relational consistency. In *Principles and Practice of Constraint programming (CP-95)*, pages 240–257, 1995.

R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

R. Dechter. Constraint networks. *Encyclopedia of Artificial Intelligence*, pages 276–285, 1992.

R. Dechter. Bucket elimination: A unifying framework for probabilistic inference algorithms. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 211–219, 1996.

R. Dechter. Topological parameters for time-space tradeoffs. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 220–227, 1996.

R. Dechter. Mini-buckets: A general scheme of generating approximations in automated reasoning. In *Ijcai-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

Y. El-Fattah and R. Dechter. An evaluation of structural parameters for probabilistic reasoning: results on benchmark circuits. In *Uncertainty in Artificial Intelligence (UAI-96)*, pages 244–251, 1996.

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

F.V. Jensen, S.L Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computation. *Computational Statistics Quarterly*, 4, 1990.

U. Kjæaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Uncertainty in Artificial Intelligence (UAI-93)*, pages 121–149, 1993.

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

J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

Y. Peng and J.A. Reggia. Plausability of diagnostic hypothesis. In *National Conference on Artificial Intelligence (AAAI86)*, pages 140–145, 1986.

Y. Peng and J.A. Reggia. A connectionist model for diagnostic problem solving, 1989.

D. Poole. Probabilistic partial evaluation: Exploiting structure in probabilistic inference. In *Ijcai-97: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1997.

S.K. Anderson R. D. Shachter and P. Solovitz. Global conditioning for probabilistic inference in belief networks. In *Uncertainty in Artificial Intelligence (UAI-91)*, pages 514–522, 1991.

B. D'Ambrosio R.D. Shachter and B.A. Del Favro. Symbolic probabilistic inference in belief networks. *Automated Reasoning*, pages 126–131, 1990.

E. Santos, S.E. Shimony, and E. Williams. Hybrid algorithms for approximate belief updating in bayes nets. *International Journal of Approximate Reasoning*, in press.

E. Santos. On the generation of alternative explanations with implications for belief revision. In *Uncertainty in Artificial Intelligence (UAI-91)*, pages 339–347, 1991.

R.D. Shachter. Evaluating influence diagrams. *Operations Research*, 34, 1986.

R.D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36, 1988.

R. D. Shachter. An ordered examination of influence diagrams. *Networks*, 20:535–563, 1990.

P.P. Shenoy. Valuation-based systems for bayesian decision analysis. *Operations Research*, 40:463–484, 1992.

S.E. Shimony and E. Charniack. A new algorithm for finding map assignments to belief networks. In *P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer ed., Uncertainty in Artificial Intelligence*, volume 6, pages 185–193, 1991.

J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 1990.

N.L. Zhang and D. Poole. Exploiting causal independence in bayesian network inference. *Journal of Artificial Intelligence Research (JAIR)*, 1996.