Processing Boolean queries over Belief networks

Rina Dechter and Padhraic Smyth

Department of Information and Computer Science
University of California, Irvine, CA 92697-3425
{dechter,smyth}@ics.uci.edu

Abstract

The paper presents a variable elimination method for computing the probability of a cnf query over a belief network. We present a bucket-elimination algorithm whose complexity is controlled by the induced-width of the moral graph combined with the interaction graph of the cnf. We show that the algorithm can be easily extended to answer a host of additional cnf-related queries such as finding the most probable model of the cnf theory, or finding the most probable tuple satisfying the cnf theory, as well as belief assessment conditioned on disjunctive type observations.

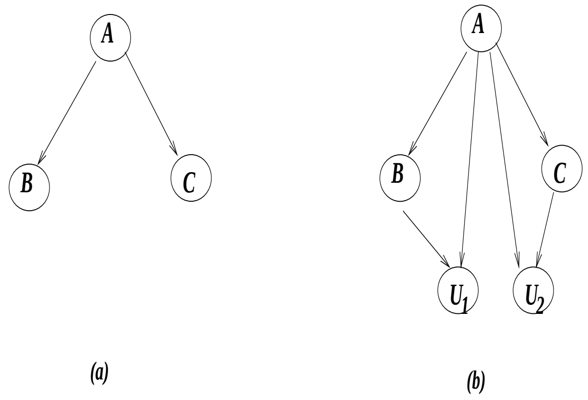

Figure 1: A three variable belief network (a), augmenting the query with a query subnetwork (b)

1 Introduction and related work

The paper presents an algorithm for computing the probability of a Boolean query in conjunctive normal form (cnf) over a probability distribution expressed by a belief network. The algorithm belongs to the class of bucket-elimination algorithms. Although such queries can be handled by modeling the cnf query as part of the belief network, the proposed method can benefit from keeping the query distinct from the knowledge-base (the belief network) in several ways. Computationally, by facilitating *constraint propagation* (e.g., arc-consistency or unit resolution) that proved essential for efficient processing of Boolean and constraint expressions [Dechter, 1999b]. Furthermore, belief networks can stand for physical causal mechanisms having semantical significance beyond the pure numerical function they express [Pearl, 2000].

Consider the simple belief network over three propositional variables A,B,C, defined by the graph in Figure 1a. The joint probability associated with this network is:

$$P(A,B,C) = P(C|A)P(B|A)P(A)$$

Let's consider the query $\varphi = (\neg B \vee A) \wedge (\neg A \vee C) \wedge B$. The task is to determine $P(\varphi)$.

One approach for handling such queries described in [Pearl, 1988], is the *modeling approach*, namely expressing each clause as a conditional probability table (CPT) between the clause's variables and a new hidden variable. In our example this requires adding two hidden variables $U_1 = \neg B \vee A$ and $U_2 = \neg A \vee C$, expressed explicitly in the network (see Figure 1b). Subsequently, a standard inference algorithm for conjunctive queries (e.g., variable elimination) can be applied to the augmented network (Figure 1b) to evaluate $P(U_1, U_2, B)$. The alternative approach we propose here maintains a distinction between the deterministic cnf query and the CPTs of the belief network, and applies a specialized bucket-elimination algorithm for this task.

Assume we order the variables $d = (C, A, B)$. Our algorithm partitions both the CPTs and the clauses into buckets (the rationale is given later) and process the buckets from last to first. In each bucket, instead of computing a simple sum over the product of the CPTs (as in Elim-Bel ??), we compute

restricted sums. For example, in bucket B we compute $\lambda^B(A) = \sum_{\{B|(\neg B \vee A) \wedge B=true\}} P(B|A)$. The resulting function is placed in bucket A. Next we compute the sum over tuples satisfying the clauses appearing in bucket A, namely $\lambda^A(C) = \sum_{\{A|(\neg A \vee C)=true\}} P(C|A)P(A)\lambda^B(A)$ and so on. The resulting bucket configuration is:
bucket(B): $P(B|A)$, $(\neg B \vee A)$, B
bucket(A): $P(C|A)P(A)$, $(\neg A \vee C)$ $||\lambda^B(A)$
bucket(C): $||\lambda^A(C)$
The λ functions are the new functions, and they are denoted to the right of the bar in each bucket.

Going back to the modeling approach, there exists an ordering of the variables (e.g., C, A, B, U_1, U_2) that, when applying Elim-Bel, yields a similar performance. However, aside from the annoying aspect of having to introduce hidden variables for each clause, the modeling approach hides the prospect of exploiting constraint propagation algorithms. For example, using our algorithm, instead of proceeding mechanically as above, we can apply unit resolution in bucket B and infer the unit clause A. This will be placed in the bucket of A and will *trigger* unit resolution in bucket A as well, inferring the unit clause C, to be placed in bucket C. Since each bucket contains a unit clause, computing the probability of this tuple (A, B, C) can be accomplished in linear time.

Extending the bucket-elimination approach to answering cnf queries over probabilistic database, has the additional advantage of sharing a variety of improvements that are applicable to this class of algorithms, such as trading space for time [El-Fattah and Dechter, 1996], as well as the mini-bucket approximation [Dechter and Rish, 1997]. Finally, we show that simple modifications elegantly yield methods for processing a variety of additional cnf-related queries.

An alternative approach for answering cnf queries is by stochastic simulation [Pearl, 1988]. These algorithms use the belief network to generate tuples from the distribution, and then answer any query by treating the colletion of tuples produced as the probability distribution. These methods, however, are incomplete and is likely to be very ineffective for formulas having a small number of models.

Following Motivations (Section 2) and Background (Section 3), Section 4 presents the general bucket-elimination algorithm for assessing the probability of cnf queries and analyzes its performance. In particular, we show that when the query's interaction graph is subsumed by the belief network's moral graph, the complexity of evaluating the probability of a cnf in the same as belief assessment of singleton propositions. Section 5 discusses additional cnf-related queries and section 6 concludes.

2 Motivation

The problem of computing cnf-queries using belief networks has a direct application to query answering in massive databases. Relational database technology has been very successful commercially now for a number of years, particularly for efficient storage, access, and management of large-volumes of operational transaction data (e.g., in the banking, airline, and retail industries). A more recent trend is leveraging this operational data for decision-support purposes (e.g., data-mining or exploratory data analysis). Typically this requires a somewhat different style of human-database interface than traditional batch query processing such as SQL. In particular, for massive data archives, the generation of approximate answers to queries in real-time is of significant practical interest [Vitter and Wang, 1999, Pavlov *et al.*,]. An approximate solution which is obtained quickly may be far more useful to the data analyst than an exact solution which requires a time-consuming complete linear scan of the entire data archive.

A general solution in this context is to construct an approximate model of the data *offline* using a belief network, and then to answer real-time queries using the approximate model (without recourse to the original data) [Pavlov *et al.*,]. An important open question in this context is how to answer arbitrary Boolean queries from a probabilistic model such as a belief network. In traditional query optimization work in databases there has been significant amount of research on answering arbitrary Boolean queries exactly (based on the data) in as efficient a manner as possible. Here we are focusing on equivalent research questions where the queries are being posed to an approximate model (the belief network) rather than to the data directly.

As an example, one of the authors (PS) is working with a retail data analysis company on techniques for analyzing large retail transaction data sets. One such data set contains over a million transactions involving several hundred product categories. Each attribute indicates whether a customer purchased a particular product category or not. Examples of these product attributes are sports-coat, rain-coat, dress-shirt, tie, etc. Marketing analysts are interested in posing queries such as "how many customers purchased a coat and a shirt and a tie?" In Boolean terms this can be expressed (for example) as the cnf-query (sports-coat $\vee$ rain-coat) $\wedge$ (dress-shirt $\vee$ casual-shirt) $\wedge$ tie. A query expressed as a conjunction of such clauses represents a particular type of prototypical transaction (particular combination of items) and the focus is on discovering more information about customers who had such a combination of transactions. In [Pavlov *et al.*,] the use of belief network models to support fast conjunctive queries for this and other similar types of transaction data sets, is described. One immediate application of the techniques proposed here is to support general cnf queries for such models.

3 Preliminaries and background

3.1 Belief networks

Belief networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty.

Let $X = \{X_1, ..., X_n\}$ be a set of random variables over multi-valued domains, $D_1, ..., D_n$, respectively. A *belief network* is a pair (G, P) where $G = (X, E)$ is a directed acyclic graph over the variables, and $P = \{P_i\}$, where P_i denotes conditional probability matrices $P_i = \{P(X_i|pa_i)\}$, where pa_i is the set of *parents* nodes pointing to X_i in the graph. The family of X_i, F_i, includes X_i and its parent variables. 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 restriction of a tuple x over a subset of variables S. An evidence set e is an instantiated subset of variables. We use upper case letters for variables and nodes in a graph and lower case letters for values in a variable's domain. We also call $X_i \cup pa_i$ the *scope* of P_i. The *moral graph* of a directed graph is the undirected graph obtained by connecting the parent nodes of each variable and eliminating direction. We define $\bar{x}_i = (x_1, ..., x_i)$ and $\bar{x}_i^j = (x_i, x_{i+1}, ..., x_j)$.

Example 3.1 *The network in Figure 2a can express the causal relationship between 'Season' (A), 'The configuration of an automatic sprinkler system' (B), 'The amount of rain expected' (C), 'The wetness of the pavement' (F), 'Whether or not the pavement is slippery' (G) and Tthe amount of manual watering necessary' (D). The belief-network is defined by*

$$\forall a, b, c, d, f, g, \quad P(g, f, d, c, b, a) =$$

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

In this case, $pa(F) = \{B, C\}$. *The moral graph is given in Figure 2b.*

Propositional variables, which take only two values $\{true, false\}$ or "1" and "0" are denoted by uppercase letters P,Q, R. Propositional literals (i.e., $P, \neg P$) stand for $P =$ "$true$" or $P =$ "$false$," and disjunctions of literals, or *clauses*, are denoted by $\alpha, \beta, \ldots$. For instance, $\alpha = (P \vee Q \vee R)$ is a clause. A *unit clause* is a clause of size 1. The *resolution* operation over two clauses $(\alpha \vee Q)$ and $(\beta \vee \neg Q)$ results in a clause $(\alpha \vee \beta)$, thus eliminating Q. A formula φ in conjunctive normal form (*cnf*) is a set of clauses $\varphi = \{\alpha_1, \ldots, \alpha_t\}$ that denotes their conjunction. The set of *models* or *solutions* of a formula φ, denoted *models*(φ) is the set of all truth assignments to all its symbols that do not violate any clause.

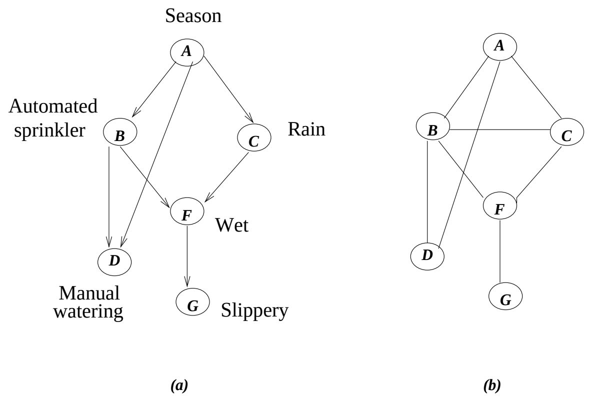

Figure 2: Belief network $P(g, f, d, c, b, a) = P(g|f)P(f|c, b)P(d|b, a)P(b|a)P(c|a)P(a)$

Queries. The primary query over belief networks is *belief updating* Given a set of observations, computes the posterior probability of each proposition. In this paper we address queries that use Boolean formulas to express deterministic restrictions on the set of propositions of interest. Given a belief network defined over a set of propositional variables $X = \{X_1, ..., X_n\}$ and given a cnf query φ over a subset $Q \subseteq X$, we define the following queries:

- *Cnf Probability Evaluation (CPE)*: finding the probability $P(\varphi)$. (generalizing Belief assessment).
- *Cnf Max Probability Evaluation (CMPE)*: finding the most likely complete tuple that satisfies φ (generalizing MPE task [Pearl, 1988]). for example, given the set of customers who have satisfy the predicate ϕ, what is their most likely "complete profile" in terms of the their purchasing patterns, i.e., what is the most likely assignment of the other product attributes for customers who satisfy ϕ.
- *Finding the most likely model (CMAP)*: assessing the most likely assignment to the models of φ, (generalizing MAP).

- *Belief assessment condition on cnf*: updating the belief in a proposition conditioned on knowing φ, (generalizes the notion of observation.) For example, given that a customer purchased a coat, a shirt, but did not buy a tie, what is the probability that they will also purchase shoes? This type of query is very valuable for predictive modeling, e.g., "cross-sell" applications where we determine which other products a customer is likely to purchase.

3.2 Bucket-elimination

Bucket elimination is a unifying algorithmic framework for dynamic programming algorithms applicable to probabilistic and deterministic reasoning [Bertele and Brioschi, 1972, Dechter, 1996, Dechter, 1999a].

The input to a bucket-elimination algorithm consists of a collection of functions or relations (e.g., clauses for propositional satisfiability, constraints, or conditional probability matrices for belief networks). Given a variable ordering, the algorithm partitions the functions into buckets, where a function is placed in the bucket of its latest argument in the ordering. The algorithm processes each bucket, from the last variable to the first, by a variable elimination procedure that computes a new function that is placed in an earlier (lower) bucket. For belief assessment, the bucket procedure generates the product of all probability matrices and sums over the values of the bucket's variable. Figure 3 shows *Elim-Bel*, the bucket-elimination algorithm for belief assessment. [Dechter, 1996].

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 [Arnborg, 1985]. It was shown that,

THEOREM **3.2** *[Dechter, 1996] The time and space complexity of the algorithm Elim-Bel is exponential in the induced width $w^*(d)$ of the network's ordered moral graph along the ordering d.* □

Algorithm Elim-Bel
Input: A belief network $BN = \{P_1, ..., P_n\}$; an ordering of the variables, d; observations e.
Output: The belief $P(X_1|e)$.
1. **Initialize:** Partition BN into $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all matrices whose latest (highest) variable is X_i. Put each observed variable into its appropriate bucket. Let $S_1, ..., S_j$ be the subset of variables in the processed bucket on which matrices (new or old) are defined.
Call all the functions n a bucket by h_i.
2. **Backward:** For $p \leftarrow n$ downto 1, do
for $\lambda_1, \lambda_2, ..., \lambda_j$ in $bucket_p$, do
• **If** $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each λ_i and put each resulting function into its appropriate bucket.
• **Else,** generate the functions λ^p:
$\lambda^p = \sum_{X_p} \Pi_{i=1}^{j} \lambda_i$.
Add λ^p to the bucket of the latest variable in $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$.
3. **Return** $P(X_1|e)$ by normalizing the product in the first bucket.

Figure 3: Algorithm *Elim-Bel*

4 Bucket-elimination for CPE

There are two primary approaches for the CPE task. One is based on search, namely, on enumerating all the models of the cnf formula, then assessing the belief of each model (by evaluating conjunctive queries over the BN and accumulating the sum.). The second approach is the variable elimination approach on which we focus.

The following paragraph derive the algorithm. Given a belief network and a cnf formula φ, the CPE task is to compute the sum:

$$P(\varphi) = \sum_{\bar{x}_Q \in models(\varphi)} P(\bar{x}_Q).$$

Using the belief-network product form we get:

$$P(\varphi) = \sum_{\{\bar{x}|\bar{x}_Q \in models(\varphi)\}} \prod_{i=1}^{n} P(x_i|x_{pa_i})$$

For derivation purpose, we next assume that X_n is one of the query variables, and we separate the summation over X_n and $X - \{X_n\}$. We denote by γ_n the set of all clauses that are defined on X_n and by β_n all the rest of the clauses. The scope of γ_n is denoted Q_n, $S_n = X - Q_n$ and U_n is the set of all variables in the scopes of CPTs and clauses that are defined over X_n. We get:

$$P(\varphi) =$$

$$\sum_{\{\bar{x}_{n-1}|\bar{x}_{S_n} \in models(\beta_n)\}}, \sum_{\{x_n|\bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{i=1}^{n} P(x_i|x_{pa_i})$$

Denoting by t_n the indices of functions in the product that *do not* mention X_n and by $l_n = \{1, ...n\} - t_n$ we get:

$$P(\varphi) = \sum_{\{\bar{x}_{n-1}|\bar{x}_{S_n} \in models(\beta_n)\}} \prod_{j \in t_n} P_j \cdot$$
$$\sum_{\{x_n|\bar{x}_{Q_n} \in models(\gamma_n)\}} \prod_{j \in l_n} P_j$$

Therefore:

$$P(\varphi) = \sum_{\{\bar{x}_{n-1}|\bar{x}_{S_n} \in models(\beta_n)\}} (\prod_{j \in t_n} P_j) \cdot \lambda^{X_n}$$

where λ^{X_n} is defined over $U_n - \{X_n\}$, by

$$\lambda^{X_n} = \sum_{\{x_n|\bar{x}_{Q_n} \in models(\alpha_n)\}} \prod_{j \in l_n} P_j \qquad (1)$$

Therefore, in the bucket of X_n we should compute λ^{X_n}. We need to place all CPTs and clauses mentioning X_n and then compute the function in EQ. (1). The computation of the rest of the expression proceeds with X_{n-1} in the same manner. This yields algorithm Elim-CPE, described in Figure 4. The elimination operation is denoted by the general operator symbol $\otimes$ that instantiate to summation for the current query. Thus, for every ordering of the propositions, once all the CPTs and clauses are partitioned (each clause and CPT is placed in the latest bucket of their scope), we process the buckets from last to first, in each applying the following operation. Let $\lambda_1, ...\lambda_t$ be the probabilistic functions in bucket P over scopes $S_1, ..., S_t$ and $\alpha_1, ...\alpha_r$ be the clauses over scopes $Q_1, ..., Q_r$. The algorithm computes a new function λ^P over $U_p = S \cup Q - \{X_p\}$ where $S = \cup_i S_i$, and $Q = \cup_j Q_j$, defined by:

$$\lambda^P = \sum_{\{x_p|\bar{x}_Q \in models(\alpha_1,...,\alpha_r)\}} \prod_j \lambda_j$$

)

Example 4.1 *Consider the belief network in Figure 2 and the query $\varphi = (B \vee C) \wedge (G \vee D) \wedge (\neg D \vee \neg B)$. The initial partitioning into buckets along the ordering $d = A, C, B, D, F, G$, as well as the output buckets are given in Figure 6. In bucket G we compute:*
$\lambda^G(f, d) = \sum_{\{g|g \vee d = true\}} P(g|f)$
In bucket F:
$\lambda^F(b, c, d) = \sum_f P(f|b, c)\lambda^G(f, d)$
In bucket D:
$\lambda^D(a, b, c) = \sum_{\{d|\neg d \vee \neg b = true\}} P(d|a, b)\lambda^F(b, c, d)$
In bucket B:
$\lambda^B(a, c) = \sum_{\{b|b \vee c = true\}} P(b|a)\lambda^D(a, b, c)\lambda^F(b, c)$
In bucket C:
$\lambda^C(a) = \sum_c P(c|a)\lambda^B(a, c)$

Algorithm Elim-CPE
Input: A belief network $BN = \{P_1, ..., P_n\}$; A cnf formula on k propositions $\varphi = \{\alpha_1, ...\alpha_m\}$ defined over k propositions, an ordering of the variables, d
Output: The belief $P(\varphi)$.
1. **Initialize:** Place buckets with unit clauses last in the ordering (to be processed first). Partition the BN and φ into $bucket_1$, ..., $bucket_n$, where $bucket_i$ contains all matrices and clauses whose highest variable is X_i. Put each observed variable into its appropriate bucket. Let $S_1, ..., S_j$ be the scopes of CPTs, and $Q_1, ...Q_r$ be the scopes of clauses. (We denote probabilistic functions as λs and clauses by αs).
2. **Backward:** Process from last to first.
Let P be the current bucket.
For $\lambda_1, ..., \lambda_j$, $\alpha_1, ..., \alpha_r$ in $bucket_p$, do
• **Process-bucket**$_p(\sum, (\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r))$
3. **Return** $P(\varphi)$ as the results of the elimination function in the first bucket.

Figure 4: Algorithm *Elim-CPE*

Process-bucket$_P(\otimes, (\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r))$
• **If** $bucket_p$ contains $X_p = x_p$,
1. Assign $X_p = x_p$ to each λ_i and put each resulting function into its appropriate earlier bucket.
2. Resolve each α_i with the unit clause, put non-tautology resolvents in lower buckets and **move any bucket with unit clause to top of processing.**

• **Else,** generate λ^P:

$$\lambda^P = \times_{\{x_p|\bar{x}_{U_p} \in models(\alpha_1,...,\alpha_r)\}} \prod_{i=1}^{j} \lambda_i$$

Add λ^p to the bucket of the largest-index variable in $U_p \leftarrow \bigcup_{i=1}^{j} S_i \bigcup_{i=1}^{r} Q_i - \{X_p\}$.

Figure 5: Process-bucket procedure

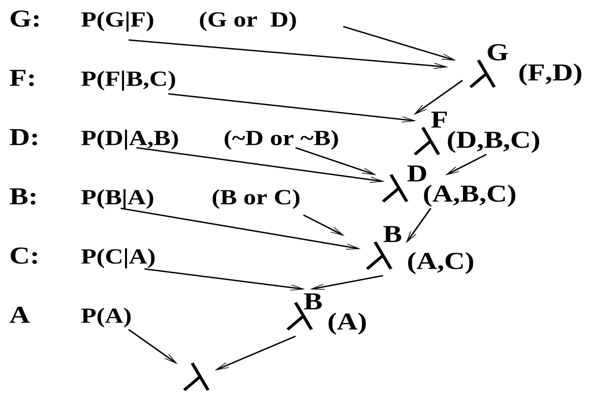

Figure 6: Bucket elimination execution of elim-CPE

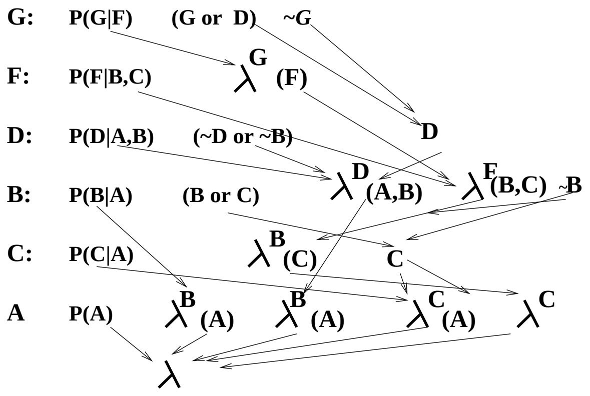

Figure 7: bucket elimination execution of elim-bel-cnf

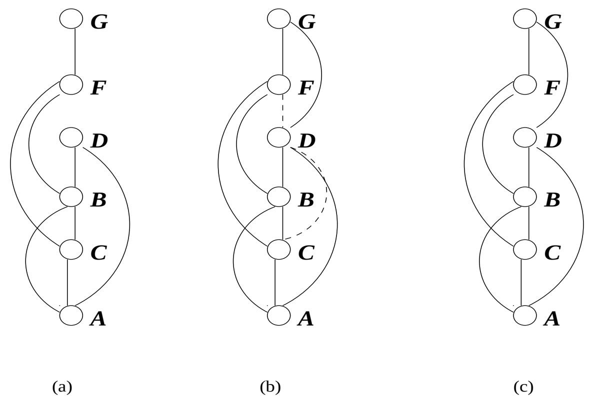

Figure 8: The induced augmented graph

In bucket A:
$\lambda^A = \sum_a P(a)\lambda^C(a)$
$P(\varphi) = \lambda^A$.

Notice that algorithm Elim-CPE also includes a unit resolution step whenever possible (in step 2) and a dynamic reordering of the buckets that prefer processing buckets that include unit clauses. This may have a significant impact on efficiency because treating observations (namely unit clauses) specially can avoid creating new dependencies. In fact, any level of pairwise resolution in each bucket would be valid (but not necessarily more efficient).

Example 4.2 *Let's now extend the example by adding $\neg G$ to the query. This will place $\neg G$ in the bucket of G. When processing bucket G, unit resolution creates the unit clause D, whichis then placed in bucket D. Next, processing bucket F creates a probabilistic function on the two variables B and C. Processing bucket D that now contains a unit clause will assign the value D to the CPT in that bucket and apply unit resolution, generating the unit clause $\neg B$ that is placed in bucket B. Subsequently, in bucket B we can apply unit resolution again, generating C placed in bucket C, and so on. In other words, aside from bucket F, we were able to process all buckets as observed buckets, by propagating the observations. (See Figure 7.)*

The algorithm in Figure 4 exploits unit resolution even further by allowing dynamic ordering always processing unit buckets (those containing unit clauses) first. (This same heuristic was proposed in the original Davis-Putnam algorithm for satisfiability). To incorporate this feature, after processing bucket G, we should move bucket D to the top (since it has a unit clause). Then, following its processing, we should process bucket B and then bucket C, then F and finally A.

4.1 Complexity

As usual, the complexity of bucket elimination algorithms is related to the number of variables appearing in each bucket, both in the scope of probability functions as well as in the scopes of clauses. The worst-case complexity is time and space exponential in the size of the maximal bucket, which is captured by the induced-width of the relevant graph. For the task at hand, the relevant graph is the belief network's moral graph combined with the cnf interaction graph. In the interaction graph of a cnf, every two nodes appearing in the same clause are connected.

Definition **4.1 (augmented graph)** *Given a belief network and a cnf query, the augmented graph of the network is the moral graph with additional arcs between each two variables appearing in the same clause of the cnf.*

Consider now the computation inside a bucket. In general, if γ_P is the cnf theory in bucket P defined over Q and $\lambda_1, \ldots.\lambda_j$ are the probability functions whose union of scopes is S, we compute:

$$\lambda^P = \sum_{\{x_p|\bar{x}_Q \in models(\gamma_P)\}} \prod_i \lambda_i \qquad (2)$$

defined over $U = Q \cup S - \{X_p\}$. A brute force computation of this expression enumerates each tuple $\bar{x}$ of U, tests if it satisfies γ_P, computes the product and accumulates the sum. This approach is time and space $exp(|U|+1)$. (Some special improvements are feasible using search methods for finding the models of a cnf formula.)

Clearly, the complexity of Elim-CPE is $O(n \cdot exp(w^*))$, where w^* is the induced width of the augmented ordered graph. In Figure 8 we see that while the induced width of the moral graph is just 2 (Figure 8a), the induced width of the augmented graph is 3 (Figure 8b).

To capture the simplification associated with observed variables or unit clauses, we use the notion of an *adjusted induced graph.* The adjusted induced graph is created by processing the variables from last to first in the given ordering. Only parents of each non-observed variable are connected. The adjusted induced width is the width of the adjusted induced-graph. Figure 8c shows the adjusted induced-graph relative to the evidence in $\neg G$. We see that the induced width, adjusted for the observation, is just 2 (Figure 8c).

In summary,

THEOREM **4.3** *Given a belief network over n variables, a cnf and an ordering o, algorithm Elim-CPE is time and space $O(n \cdot exp(w^*(o))$, where $w^*(o)$ is the width along o of the adjusted augmented induced graph.*

Since unit resolution increases the number of buckets having unit clauses, and since those are processed in linear time, it can improve performance substantially. Such buckets can be identified a priori by applying unit resolution on the cnf formula.

Corollary: When the the augmented graph is identical to the moral graph, the complexity of Elim-CPE does not increase beyond the complexity of standard inference. Moreover, unit resolution may sometime cause exponential speedup.

4.2 Conditional belief

Frequently we want to evaluate the probability of a cnf formula given a set of observations e, namely to compute $P(\varphi|e)$. This can be done by computing $P(\varphi \wedge e)$ and subsequently computing $P(e)$ using two executions of the bucket-elimination algorithm.

5 Related queries

Algorithm Elim-CPE can be easily extended to compute $CMPE$ and $CMAP$ as well as belief conditioned on φ. Furthermore the algorithm can be generalized for relational constraints in a simple manner.

5.1 Computing CMPE

Given a belief network and a cnf formula over a subset of the propositions, Q, the $CMPE$ task is to compute a tuple $\bar{x}^o$ s.t.

$$P(\bar{x}^o) = \max_{\{\bar{x}|\bar{x}_Q \in models(\varphi)\}} \prod_{i=1}^{n} P(x_i|x_{pa_i}).$$

Clearly, following the derivation for Elim-CPE, we can observe that the only change is that the summation operation is replaced by maximization, as is shown in the process-bucket procedure that uses $\otimes = max$. Accordingly, algorithm *Elim-MCPE* is identical to Elim-CPE, except that summation is replaced by maximization. The algorithm is given in Figure 9.

Algorithm Elim-CMPE
Input: A belief network $BN = \{P_1, ..., P_n\}$; a cnf formula $\varphi = \{\alpha_1, ... \alpha_m\}$ defined over k propositions, an ordering of the variables, d
Output: A CMPE tuple.

1. **Initialize:** (same as Elim-CPE).
2. **Backward:** Process from last to first. P is the current bucket. For $\lambda_1, \lambda_2, ..., \lambda_j$, $\alpha_1, ..., \alpha_r$ in $bucket_p$, do
Process-bucket$_p$(max, $(\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r)$)
3. The $CMPE$ is obtained by the product in $bucket_1$. A maximizing tuple is obtained by assigning values in the ordering d consulting recorded functions in each bucket: given the assignment $x = (x_1, ..., x_{i-1})$ choose x_i s.t $x_i = argmax_{\{x_i|\alpha_1 \wedge, ..., \wedge \alpha_r = true\}} \prod_{\{\lambda_j \in bucket_i | x=(x_1,...,x_{i-1})\}} \lambda_j$

Figure 9: Algorithm *Elim-CMPE*

5.2 Computing CMAP

The task of the CMAP is to compute:

$$P(\bar{x_Q}^o) = \max_{\bar{x}_Q \in models(\varphi)} P(\bar{x}_Q)$$

where the formula φ is defined over a subset of propositions, Q. Let $Q = X_1, ..., X_k$. Then,

$$P(\bar{x}^o) = \max_{\{\bar{x}_Q|\bar{x}_Q \in models(\varphi)\}} \sum_{\bar{x}_{k+1}^n} \prod_{i=1}^{n} P(x_i|x_{pa_i})$$

Using the usual derivation for eliminating the last variable X_n, there is a need to distinguish the two cases of summation and maximization. Variables that appear in φ should be processed by maximization (They correspond to hypothesis variables in the regular MAP task.) Furthermore, the normal restriction on variable orderings in MAP tasks applies here, too. Variables in φ should initiate the ordering. This yields algorithm $Elim - CMAP$ that is identical to algorithm *Elim-CMPE* or *Elim-CPE* except that the bucket operation is summation or maximization (see Figure 10).

5.3 Belief updating conditioning on a cnf

It is possible to assume that sometimes evidence may be uncertain and will come in the form of disjunctive information(for example, a ppatient has measles or chickenpox, and). Thus, we can can generalize the form of evidence to a cnf formula φ and compute

Algorithm Elim-CMAP
Input: A belief network $BN = \{P_1, ..., P_n\}$; A cnf formula $\varphi = \{\alpha_1, ...\alpha_m\}$ defined over $Q = X_1, ..., X_k$, an ordering d that starts with Q.
Output: A CMAP tuple over Q.
1. **Initialize:** (same as Elim-Bel-Cnf).
2. **Backward:** Process from last to first. Let P be the current bucket. For $\lambda_1, \lambda_2, ..., \lambda_j, \alpha_1, ..., \alpha_r$ in $bucket_p$, do
If X_p not in Q then,
Process-bucket$_p(\sum, (\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r))$
Else
Process-bucket$_p(\max, (\lambda_1, ..., \lambda_j), (\alpha_1, ..., \alpha_r))$
3. Assign values to Q in the ordering $d = X_1, ..., X_k$, using the information recorded in each bucket.

Figure 10: Algorithm *Elim-CMAP*

$P(X_1|\varphi)$. Since,

$$P(X_1|\varphi) = \frac{P(X_1, \varphi)}{P(\varphi)}$$

we can compute both numinator and denominator by one run of Elim-CPE as follows: if we place X_1 as the first in the ordering (it will be processed last) and now apply Elim-CPE as usual, the algorithm computes the numinator, and then arrives at the conditioned formula by normalization in the first bucket (as in Elim-Bel).

6 Discussion and conclusions

The nice property of the bucket-elimination algorithms is that their complexity is not dependent on the number of models in the cnf formula. Clearly, all the tasks addressed here could be also solved by conditioning search or by some combination of search and inference. However, the analysis of all these algorithm would have to be related to the number of models or solutions of the formula in question. There is no good way to evaluate the number of models in advance, and frequently there are many models. Nevertheless, such search methods would avoid the space complexity of bucket elimination and may work well in practice. A brute-force search approach can generate each model, one by one, and for each compute the probability of the resulting model using belief network algorithms for conjunctive queries. Clearly, a variety of improvements can be utilized using general constraint satisfaction methods [Dechter, 1999b]. However, the benefits of each proposal should be evaluated empirically. This is outside the scope of the current paper.

The bucket-elimination methods presented here show the power of this scheme in solving a variety of queries, thus making the study of this algorithmic framework even more significant. In particular, approximation schemes and methods that combine inference with search for exact and anytime computations are immediately applicable. This would be very useful in practice in real-time approximate query answering.

References

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

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

[Dechter and Rish, 1997] R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI'97)*, pages 132–141, 1997.

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

[Dechter, 1999a] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[Dechter, 1999b] R. Dechter. Constraint satisfaction. In *The MIT Encyclopedia of Cognitive Sciences (MITECS)*, pages 195–197, 1999.

[El-Fattah and Dechter, 1996] 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.

[Pavlov *et al.*,] D. Pavlov, H. Mannila, and P. Smyth. Probabilistic models for query approximation with 20 large sparse binary data sets. In *Submitted to UAI2000.*

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

[Pearl, 2000] J. Pearl. *Ausality: Models, Reasoning and Inference.* Cambridge Press, 2000.

[Vitter and Wang, 1999] J. Vitter and W. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of SIGMOD*, pages 193–204, 1999.