Bayesian Inference in the Presence of Determinism

David Larkin and Rina Dechter
Department of Information and Computer Science
University of California, Irvine, CA 92697-3425
{dlarkin,dechter}@ics.uci.edu

Abstract

In this paper, we consider the problem of performing inference on Bayesian networks which exhibit a substantial degree of determinism. We improve upon the determinism-exploiting inference algorithm presented in [4], showing that the information brought to light by constraint propagation may be exploited to a much greater extent than has been previously possible. This is confirmed with theoretical and empirical studies.

1 Introduction

Belief networks [9] are a popular model for reasoning with uncertainty in Artificial Intelligence. In general, performing inference on a belief network is NP-hard. However, tractable subclasses have been identified. The most important of these is the case when the network graph can be embedded in a triangulated (chordal) graph of bounded treewidth [8, 13, 3]. It is also possible to efficiently process networks with a special structure in their quantitative component, such as the common case of Noisy-OR gates [9], or context-specific independence [1]. In this paper, building upon research reported in [4], we identify another tractable class, the case when the network exhibits a high degree of determinism. With a deterministic relation, such as $x + y = z$, it is possible to perform exact logical deductions, such as determining the value of z given x and y. In a probabilistic model, however, knowledge of x and y might tell us that z assumes a certain value with a certain probability, but we cannot say anything for sure. It is clear that in general deterministic relations are more informative, and it would seem likely that special purpose algorithms can speed inference when they are present. In [4] such an algorithm, Elim-CPE, was indeed proposed and empirically shown to be effective. It relied upon constraint propagation to elucidate the determinism present in the network. In this paper we seek to show that this information can be exploited much more effectively by more sophisticated algorithms. This can result in greatly improved running times for inference when determinism is present, as we will demonstrate empirically and theoretically.

The main motivation for our study is the similarity that exists between belief networks that exhibit many functional relationships and deterministic networks, such as constraint satisfaction problems [7, 2]. Since these belief networks have a substantial deterministic substructure beneath their probabilistic facade, it would seem logical that the sophisticated techniques developed by the constraint satisfaction community might also be leveraged to speed Bayesian inference. If the deterministic information present in the belief network were represented explicitly as a set of constraints, then it would certainly be possible to run a backtracking-style CSP algorithm to enumerate all of the consistent variable assignments (i.e., with non-zero probability) and to add up the probabilities of each. Constraint propagation (i.e., arc-consistency) would greatly speed this task. However, belief networks are in general less deterministic than constraint networks. Except in the case of an exceptional query or observation, they do not exhibit inconsistency, and moreover they typically have a very large number of solutions. Even the most efficient CSP solver could take an unacceptably long time to enumerate the possibly exponentially many solutions. Therefore we would like to find a way to marry the techniques of CSP solving, mainly constraint propagation, with standard, efficient algorithms for general-purpose Bayesian inference to handle this special class of networks.

Constraint propagation is essentially the search for deterministic information that is implied but not stated explicitly in the input. In this paper we will consider a hierarchy of four algorithms that exploit the available deterministic information to an increasing extent. This hierarchy is represented schematically

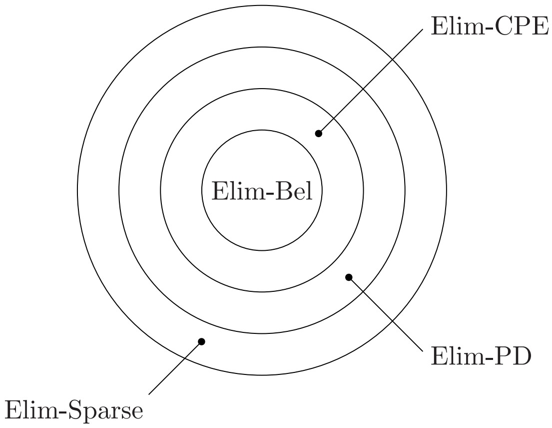

Figure 1: A Hierarchy of Determinism-Exploiting Algorithms

in Figure 1. The simplest method is the standard variable-elimination algorithm, Elim-Bel. It accepts in the input a list of observations, or variables which are known to be fixed at a certain value. All CPTs mentioning these variables can be instantiated with their values, thus reducing their size and complexity. In effect the network graph is simplified by removing all of the observed variables. The next three methods go beyond Elim-Bel by actively searching for helpful deterministic information, mainly by constraint propagation. Elim-CPE, introduced in [4], attempts to infer from the functional structure of the network and the known variable values the fact that certain other variables, not observed directly, must also be fixed at certain values. These "hidden" observations can be exploited exactly as in Elim-Bel. The next two algorithms, contributed by this paper, also perform constraint-propagation, but a much larger proportion of the available information is effectively utilized to speed inference. Elim-PD uses partial information about variables to simplify the functions that mention them without eliminating them entirely. If a variable is known *not* to assume a certain value, its domain can effectively be treated as if it were smaller and the effective table size of all the functions that mention it can be correspondingly reduced. Finally, Elim-Sparse, which is the main focus of this paper, is the most aggressive in its attempt to exploit the available determinism. Instead of using information about single variables only, it exploits the knowledge that an assignment to a *set* of variables is known not to have taken place. Functions are stored in a list representation that depends only on the number of non-zero elements of the domain. If an assignment is discovered to be inconsistent, all tuples mentioning it can be removed from all functions. When the functions have small lists (though individual variables may have unrestricted domains), this can be very effective, but when this is not true there is a substantial overhead since a nearly full list of tuples is much more difficult to search and maintain than a table.

This paper is divided into several parts. Following this introduction, we outline basic definitions and concepts in Section 2. Then in Sections 3, 4, 5, and 6, we define Elim-Bel, Elim-CPE, Elim-PD, and Elim-Sparse, respectively. In Section 7 we theoretically analyze their expected behavior in a set of simple uniform random scenarios, and in Section 8 we provide preliminary empirical results on real and random networks. Finally in Section 9 we conclude.

2 Preliminaries

A belief network is a tuple (X, D, G, P) where $X = \{X_1, X_2, ..., X_n\}$ is a set of variables, $D = \{D_i\}$ is a set of variable domains, G is a directed acyclic graph in which the nodes correspond to variables, and P is a set of conditional probability tables. The set of variables attached to arcs pointing into a variable X_i is called the parent set of X_i, pa_i. The conditional probability of X_i assuming a certain value given an assignment to its parents is given by the CPT $P(X_i|pa_i)$. The whole network defines a joint probability distribution $\prod_i P(X_i|pa_i)$.

The Bayesian inference problem is to calculate the probability $P(X_i|e)$ of a variable X_i assuming a certain value given some observations e. The observations are most commonly given as a set of variables which are known to have certain values, but it is also possible to include more complex observations, such as the knowledge that a certain assignment to a subset of variables is known not to have happened. In general we will represent e as a set of constraints $\{C_1, ..., C_k\}$ on variables representing the available knowledge. The task then is to infer the effect this knowledge has on our beliefs in the unknown variables.

3 Elim-Bel

Elim-Bel, also known as variable elimination [13, 3], is a standard belief-inference algorithm. It is well known that its complexity is exponential in the treewidth of the triangulated, moralized network graph, also known as the induced width [3]. It is also known that it can be simplified considerably by the presence of observations of fixed variables. These variables can be deleted from the graph before inference begins. Elim-Bel does not, however, make any effort to discover such fixed variables, beyond those which are given directly in the input.

Algorithm Elim-Bel
Input: `Belief net` $B = (X, D, G, P)$, `evidence` e.
Output: $P(X_i|e)$.

`Remove all observed variables by instantiation.`
`Set` $F = \{C_1, ..., C_k, P(X_1|pa_1), ..., P(X_n|pa_n)\}$.
`For each variable` $X_j \neq X_i$, `in some ordering,`
  `Let` Θ `be the functions in` F `defined on` X_j.
  `Let` $F := \{F - \Theta\} \cup \{\sum_{X_j} \prod_{f_r \in \Theta} f_r\}$.
`Return` F.

Figure 2: The Elim-Bel Algorithm

The basic inference problem is to find $P(X_i|e)$. By the use of a normalization constant, this can be reduced to finding $P(X_i \wedge e)$. Initially, the input should be simplified by treating observed variables as constants and deleting all explicit references to them. Then the main inference algorithm can begin. During each step, we will have a set of functions $F = \{f_1, f_2, ..., f_m\}$ and a set of variables $Y \subseteq X$ such that $\prod_j f_j = P(Y \wedge e)$, as we will show by induction. Initially, Y is simply the set of variables remaining after instantiating the observations, and $F = \{C_1, ..., C_k, P(X_1|pa_1), ..., P(X_n|pa_n)\}$ is just the simplified input. Each C_j is taken as a boolean 0-1 function, their product being the global observed constraint. During the induction step, we want to calculate $P(\{Y - X_j\} \wedge e)$ for some variable X_j. This is $\sum_{X_j} P(Y \wedge e) = \sum_{X_j} \prod_l f_l$. If θ is the set of indices of functions in F that mention X_j, and γ is the set of all other indices, then this reduces to $\prod_{l \in \gamma} f_l \sum_{X_j} \prod_{h \in \theta} f_h = \prod_{l \in \gamma} f_l f^j$, where $f^j = \sum_{X_j} \prod_{h \in \theta} f_h$ is calculated and stored directly. This completes the inductive proof.

After all variables but X_i have been eliminated, the induction hypothesis tells us that the product of the remaining functions will be the desired quantity $P(X_i \wedge e)$. Pseudo-code for Elim-Bel is given in Figure 2.

4 Elim-CPE

Elim-CPE, first presented in [4], goes beyond Elim-Bel by actively looking for deterministic information that is implied, but not stated directly, by the input. It maintains a representation of the deterministic information in the network, which initially includes the evidence and a constraint corresponding to each CPT, forbidding the tuples which have zero probability under it. This deterministic "skeleton" of the original network is then processed by constraint propagation. In its original form [4], Elim-CPE used directional unit resolution on CNF clauses. Here we modify it to use generalized arc consistency, in order to make it more comparable with the following algorithms. A constraint is generalized arc consistent if no variable domain contains a value that cannot be extended consistently with it. If this is not the case, it can be enforced by suitably pruning the domains. This may cause other constraints to lose their arc consistency. Therefore the GAC subroutine enforces consistency on every constraint from first to last to the first again, stopping when a complete pass is made without changing any domains. Pseudo-code for this procedure is given in Figure 3. It takes a set of constraints as input and returns a set of pruned domains. These are equivalent to unary constraints. Derived singleton constraints which fix a variable's value can be considered the same as direct observations. By appending them to the original evidence, they are made explicit and available to Elim-Bel, which can be called as a subroutine, with reduced complexity. Pseudo-code for Elim-CPE can be found in Figure 4.

Algorithm Enforce-GAC
Input: `Constraints` $C = \{C_1, ..., C_k\}$.
Output: `Set of pruned variable domains.`

`For each constraint` C_j,
  `Prune domains to make` C_j `consistent.`
`If nothing was pruned, return final domains.`
`Otherwise execute the main loop again.`

Figure 3: The Enforce-GAC Algorithm

Algorithm Elim-CPE
Input: `Belief net` $B = (X, D, G, P)$, `evidence` e.
Output: $P(X_i|e)$.

`Let constraint` K_j `represent` $P(X_j|pa_j)$.
`Let` $C := \{K_1, ..., K_n\} \cup e$.
`Let` D' `:= Enforce-GAC(`C`).`
`Return Elim-Bel(`B`,`$e \wedge D'$`).`

Figure 4: The Elim-CPE Algorithm

5 Elim-PD

Enforcing generalized arc consistency can reveal a substantial amount of deterministic information. Elim-CPE is capable of using the fact that certain variables are known to be fixed at certain values, but it cannot do anything if some variable domains are pruned only a little bit. This deficiency is remedied by Elim-PD.

If a variable is observed, Elim-Bel can delete every mention of it from the network. In fact the same thing can be done by Elim-PD with variable values. Every entry in an input function's table that mentions the value can be deleted, effectively rewriting the problem so that variables have smaller domains. Since the size

Algorithm Elim-PD
Input: `Belief net` $B = (X, D, G, P)$`, evidence` e.
Output: $P(X_i|e)$.

```
Let constraint K_j represent P(X_j|pa_j).
Let C := {K_1, ..., K_n} ∪ e.
Let D' := Enforce-GAC(C).
Rewrite B and e with smaller domains D'.
Let B' and e' be the simplified problem.
Return Elim-Bel(B', e' ∧ D').
```

Figure 5: The Elim-PD Algorithm

of a function's table is the product of the domain sizes of its constituent variables, this can lead to substantial savings. When a domain is reduced to a singleton, this process is exactly the same as the instantiation done by Elim-Bel. Pseudo code for this algorithm is given in Figure 5.

6 Elim-Sparse

There may be substantial determinism present in a network, even when the projection of the set of solutions on any particular variable may not reveal that any values should be pruned. For example, consider the case when the network is completely connected with equality constraints. Any domain value for any single variable is possible, but there are still only d solutions, where d is the domain size. To exploit the determinism present in this case, a more sophisticated algorithm is needed. Elim-Sparse is intended to meet that need.

Elim-Sparse relies upon a sparse function representation. Instead of being recorded on a table as large as the product of the domain sizes of all variables, a function is maintained as a list of tuples with non-zero probability. In the above example, with the equality constraints, defining the set of solutions as a single function would require a table of size d^n for Elim-CPE or Elim-PD, where n is the number of variables, but only nd for Elim-Sparse (d tuples of size n each). Efficient operations to work with these functions are also available. These are mainly based on the Hash-Join procedure which is well-known in database theory [6]. The product of two functions is computed by hashing every tuple in the smaller function into a hash table on the basis of its assignments to the common variables. Then every tuple in the larger function is consulted, checking the appropriate hash table entry to find all consistent smaller tuples. Each consistent pair generates a tuple in the list of the output function, which is associated with the product of their values. The operation of summing a variable out of a function can

Algorithm Elim-Sparse
Input: `Belief net` $B = (X, D, G, P)$`, evidence` e.
Output: $P(X_i|e)$.

```
Let constraint K_j represent P(X_j|pa_j).
Let D' := Enforce-GAC({K_1, ..., K_n} ∪ e).
Del.  values ∉ D' from B, e, compressing tables.
Cast B, e into sparse lists B', e'.
Let F = {C'_1, ..., C'_k, P'(X_1|pa_1), ..., P'(X_n|pa_n)}.
For every variable X_j ≠ X_i, in some ordering,
   Let Θ be the functions in F defined on X_j.
   Find f^j := Σ_{X_j} Π_{f_r ∈ Θ} f_r w/ sparse operations.
   Let F := {F − Θ} ∪ {f^j}.
Return F.
```

Figure 6: The Elim-Sparse Algorithm

also be accomplished efficiently with a hash table. The average-case complexity of these operations is optimal, being the sum of the sizes of the operands and the output. The complete algorithm, making use of the sparse data structure and the associated operations, is given in Figure 6. Its overall complexity depends on the amount of determinism in the problem. If enough is present for the largest function lists to be comparatively short, it can be fairly efficient, but if it is not present, the overhead of manipulating nearly full tuple lists can be much larger than dealing with a table.

Other structured function representations, such as decision trees [1] or rule-based systems [10] might seem appropriate in this case. These systems partition the domain space of functions into regions of equal value, which only requires one function definition per region. The regions are defined to be all tuples consistent with some partial assignment. However, in general, it is desirable to be able to remove zero-valued tuples arbitrarily from the explicit representation, without being constrained to respect the structure of partial-assignment space. Therefore we employed our simpler method for the purposes of this study. More sophisticated sparse table methods [11] are also available. Investigating the impact these might have is an open question.

In previous work [5], sparse representations have been used to speed Bayesian inference. The sum and product operations have also been previously expressed as relational database operators [12], which coincides with our own interpretation. Our main contribution here is to show how the sparse representation can amplify the gain that results from constraint propagation. This can make higher levels of constraint propagation worthwhile. We will make this clearer in the theoretical analysis of the following Section.

7 Theoretical Analysis

In this Section we will compare the expected performance of the four algorithms on some uniform random cases. We assume that there are n variables, each with domain size d. The deterministic part of the network is represented by m random binary constraints. An assignment to a constraint's variables is consistent with uniform probability q. When a variable is eliminated, a new function is created which is defined on k variables.

We will use a version of Enforce-GAC that does only one pass through the constraints. When a constraint is processed, every variable value that cannot be extended consistently with it is flagged for deletion, but not removed from the domain. The flagged domain values are pruned after all constraints have been processed once, then the algorithm stops. The actual Enforce-GAC algorithm is harder to analyze, but since it does more pruning, it is likely that the predicted advantage of the determinism-exploiting algorithms would increase under it.

We can expect $n-k$ variable elimination steps to occur before we are left with a set of functions defined on no more than k variables. The effort required to compute the desired probability at this point is no greater than that needed to perform one more elimination step, so we can consider the problem solved here. Elim-Bel will produce a function of size d^k for each elimination step, for a total complexity of $(n-k)d^k$. To calculate the expected running times of the other algorithms, we need three quantities. For Elim-CPE, we need to know the probability p_s that a variable will be reduced to a singleton domain by the constraint propagation step. These variables can be removed in the preprocessing step. For Elim-PD, we also need the expected domain size after propagation, e_d. This will determine the average complexity of a function over the pruned domains. Finally Elim-Sparse's complexity depends on the probability that an assignment of unpruned values will be listed in the function produced by eliminating variable i. Since this depends on i, we will call it $S(i)$.

When a constraint is processed, a variable value is flagged for deletion if it cannot be extended to any of the d possibilities for the other variable. This occurs with probability $(1-q)^d$. So the value remains unflagged with probability $1-(1-q)^d$. There are m constraints, for a total of $2m$ domain-pruning checks, $2m/n$ of which will involve any particular variable. For a domain value to remain unpruned, it must survive each check, which will happen with probability $p_v = (1-(1-q)^d)^{\frac{2m}{n}}$. It will be the only survivor if the $d-1$ other values are pruned, which occurs with probability $p_v(1-p_v)^{d-1}$. There are d possible sole survivors, so the total chance that a domain will be reduced to a singleton is $p_s = dp_v(1-p_v)^{d-1}$. This is the first quantity we were looking for. The expected domain size after propagation e_d is $d \cdot p_v$. This was the second requirement.

It is now necessary to find $S(i)$. As stated before, $2m/n$ constraints can be expected to involve variable i. We call a variable a neighbor of i if it is connected to it by a constraint. The variables are eliminated from last to first, so when i is removed only variables 1 through $i-1$ will be left. So $\frac{2m}{n} \cdot \frac{i-1}{n-1}$ predecessors of i can be expected to be neighbors, and $\frac{2m}{n} \cdot \frac{n-i}{n-1}$ successors. We call these two quantities Before[i] and After[i], respectively. Looking at the internal structure of the constraints, we want to find the expected number of unpruned values a given unpruned value can be extended to. Before constraint propagation, this is qd. From the point of view of a given constraint, almost all of the domain pruning is done by other (independent) constraints, so the expected probability that a given pair of unpruned values is consistent under it after constraint propagation is still (approximately) q. Now, suppose Enforce-GAC has finished deleting all domain values, leaving the expected domain size at e_d, and a variable j has h neighbors. An assignment to all of the neighbors can be extended consistently to a particular one of j's values with probability q^h. A particular value of j cannot be part of a consistent extension with probability $1-q^h$, and no extension to j is possible with probability $(1-q^h)^{e_d}$. So then the assignment to the h neighbors can be extended consistently with j with probability $PC[h] = 1-(1-q^h)^{e_d}$. Now, suppose we are eliminating variable i by creating a new function on k variables. Any of i's succeeding neighbors was also a neighbor of $\frac{2m}{n}\frac{k}{n-1}$ of these k variables, in the average case. An assignment to the k variables will appear as a tuple in the function being created only if it can be extended consistently to variable i and all of its After[i] succeeding neighbors. Before[i] of the k variables will be neighbors of i. So a tuple will appear with probability $S(i) = PC[\text{Before}[i]] \cdot PC[\frac{2m}{n}\frac{k}{n-1}]^{\text{After}[i]}$. This is the last quantity we were looking for.

So, Elim-CPE will remove a fraction p_s of the variables by instantiation before the main inference step, which will be carried out by Elim-Bel. So its expected cost is $(1-p_s)(n-k)d^k$. Elim-PD will also instantiate these variables, and moreover when a variable is eliminated during the main inference the function created has size e_d^k. So its expected cost is $(1-p_s)(n-k)e_d^k$. Finally, when Elim-Sparse generates a function when eliminating variable i, it will have $S(i)e_d^k$ tuples. Elim-Sparse is somewhat less efficient than the other algorithms in generating the new function, because the complexity is dependent not only on the size of the output but

m	Elim-Bel	Elim-CPE	Elim-PD	Elim-Sparse
350	1.831e12	1.801e12	2.115e10	2.768e10
400	1.831e12	1.787e12	1.111e10	2.374e9
450	1.831e12	1.768e12	5.829e9	3.670e8

Figure 7: Expected Costs Under Three Scenarios

m	Elim-Bel	Elim-CPE	Elim-PD	Elim-Sparse
350	66.6	65.1	0.76	1.0
400	578	564	3.5	1.0
450	6209	5995	20	1.0

Figure 8: Normalized Expected Costs

also the intermediate functions created by the pairwise product operations. The total size of the list of the function generated by eliminating i is $kS(i)e_d^k$, and since this is much larger than the intermediate functions calculated to produce it, it dominates the overall cost. However to take into account the overhead from these intermediate operations, we will assume Elim-Sparse takes $ckS(i)e_d^k$ time to generate a new function, for $c > 1$. Its total time then is $(1-p_s)\sum_{i=k}^{n} ckS(i)e_d^k$.

These quantities were calculated by computer for some sample scenarios. The number of variables n was 75, the function arity k was 15, the domain size d was 5, the constraint looseness q was 1/2, and the Elim-Sparse overhead factor c was 4. The number of constraints varied from 350 to 400 to 450. The results are listed in Figure 7. The normalized complexities, where the cost of Elim-Sparse is treated as 1, are given in Figure 8.

It can be seen from this example that Elim-CPE gains a fairly mild advantage over Elim-Bel as the determinism present in the network, measured by the number of constraints, goes up. Elim-PD gets a much more significant improvement in efficiency, becoming orders of magnitude faster as the determinism increases. Elim-Sparse at the lowest level of determinism is somewhat less efficient than Elim-PD, due mainly to the overhead of the tuple lists. But as constraints are added it clearly makes more effective use of the deterministic information, becoming three times faster at 400 constraints and 20 times faster at 450.

It should be noted that if we used the sparse function representation alone, without performing constraint propagation first, the efficiency gains would have been considerably less. $S(i)$, the chance of a tuple appearing in a newly generated function, would be higher in this case, since inconsistencies previously uncovered by constraint propagation would go undetected. Moreover the total cost would be approximately $\sum_{j=k}^{n} ckS(i)d^k$ instead of $(1-p_s)\sum_{j=k}^{n} ckS(i)e_d^k$, which is the complexity of Elim-Sparse. Calculations with this formula reveal that in the above scenarios, Elim-Sparse can be expected to be orders of magnitude faster than sparse inference without constraint propagation.

From this we may conclude that performing constraint propagation may be expected to be very profitable under certain uniform random conditions, provided that a suitably aggressive determinism-exploiting algorithm (like Elim-Sparse) is used.

8 Empirical Results

In this Section we discuss the results of preliminary empirical tests of Elim-Bel, Elim-CPE, and Elim-Sparse on a variety of real and random networks. Because of time constraints we were unable to implement Elim-PD.

We tested six real life networks: Diabetes, Water, Mildew, Munin1, Hailfinder, and Insurance. These can be found at `http://www.cs.huji.ac.il/labs/compbio/Repository/networks.html`. We also tested some randomly generated networks. They had 30 variables, with domain sizes ranging from 2 to 4. The network DAG was constructed randomly, with 25 variables getting four random parents and the other 5 getting none. The CPTs were generated randomly. The level of determinism present was controlled by a parameter d. Each assignment to a variable's parents had a d percent chance of being deterministic, meaning that the value of the child would be functionally determined in this case. We tested three classes of random networks with varying determinism: Random1 with $d = 0.25$, Random2 with $d = 0.50$, and Random3 with $d = 0.75$.

The properties of the networks we used, and the queries we submitted to them, are listed in Figure 9. The first column V is the number of variables. D is the average domain size, w^* is the induced width or treewidth of the network graph under a heuristically chosen ordering, and R was the average ratio of forbidden to allowed entries in the CPTs and the evidence. The next three columns list the query properties. O is the number of observations in the query, or variables declared to be fixed at some value. C is the number of random binary constraints observed, each allowing every assignment with probability L.

We generated 50 random queries for each network, testing the three algorithms on each. The random networks were generated anew for each query. The average performance of the algorithms is given in Figure 10. EB is the time taken by Elim-Bel, EC is Elim-CPE, and ES is Elim-Sparse. B/S is the ratio of Elim-

Network	V.	D.	w^*	R.	O.	C.	L.
Insurance	27	3	10	0.697	0	20	0.5
Random1	30	3	14	0.740	5	9	0.5
Random2	30	3	14	0.644	5	9	0.5
Random3	30	3	14	0.539	5	9	0.5
Water	32	3	15	0.615	1	10	0.75
Mildew	35	17	7	0.568	5	0	N/A
Hailfinder	56	3	7	0.743	2	20	0.5
Munin1	189	5	13	0.457	30	0	N/A
Diabetes	413	11	11	0.400	80	0	N/A

Figure 9: Network Statistics for Empirical Tests

Network	EB	EC	ES	B/S	C/S
Insurance	3.56	1.05	0.24	14.83	4.38
Random1	4.10	2.00	2.89	1.42	0.69
Random2	2.66	1.99	1.17	2.27	1.70
Random3	2.15	1.49	0.21	10.24	7.10
Water	4.65	3.22	0.29	16.03	11.10
Mildew	7.64	4.51	0.94	8.15	4.81
Hailfinder	1.99	1.14	0.99	2.01	1.15
Munin1	15.92	3.58	0.84	18.95	4.26
Diabetes	18.77	12.20	9.67	1.94	1.26

Figure 10: Average Times

Bel's time to Elim-Sparse's, and C/S is the ratio of Elim-CPE to Elim-Sparse.

The relative performance of Elim-Sparse to Elim-CPE was subject to some variation. On the random networks, Elim-CPE was slightly faster when the determinism was at its lowest, but Elim-Sparse rapidly improved as the determinism increased, to the point of being 7 times faster on Random3. This is roughly in line with our theoretical results, which predict that Elim-Sparse should broaden its advantage over the other algorithms by orders of magnitude with increasing determinism. It appears, however, that the high constant factor that accrues with the manipulation of tuple lists was enough to destroy its advantage when the determinism was relatively low.

On the real-life networks the results are also somewhat mixed. Elim-Sparse was considerably faster than Elim-CPE on Insurance, Water, Mildew, and Munin1 (by a factor of 4 or more), but less so on Hailfinder and Diabetes (less than twice as fast).

We can conclude that Elim-Sparse is generally more efficient than Elim-CPE on a significant class of networks.

9 Conclusion

In this paper, we presented new methods for exploiting the deterministic information in a belief network that is brought to light by constraint propagation. They are able to exploit a considerably greater proportion of the information than previous methods. We believe this shows that the techniques applicable to deterministic networks, such as constraint satisfaction problems, show some promise in making belief networks with many functional relationships and complex evidence more tractable for inference. In further research we hope to continue to investigate the connections between belief networks and CSPs and to apply more classically deterministic techniques to belief inference.

References

[1] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in bayesian networks. In *Proc. Conference on Uncertainty in Artificial Intelligence*, 1996.

[2] R. Dechter and F. Rossi. Constraint satisfaction. *Survey ECS*, 2000.

[3] Rina Dechter. Bucket elimination: a unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, September 1999.

[4] Rina Dechter and David Larkin. Hybrid processing of beliefs and constraints. In *Proceedings of UAI '01*, 2001.

[5] F. Jensen and S. Andersen. Approximations in bayesian belief universes for knowledge-based systems. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 162–169, 1990.

[6] Henry Korth and Abraham Silberschatz. *Database System Concepts*. McGraw Hill, 1991.

[7] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *A. I. Magazine*, 13(1):32–44, 1992.

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

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

[10] D. Poole. Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference. In *Proc. Fifteenth International Joint Conference in Artificual Intelligence*, 1997.

[11] R. E. Tarjan and A. Yao. Storing a sparse table. *Communications of the ACM*, 22(11), 1979.

[12] S. K. M. Wong. An extended relational data model for probabilistic reasoning. *Journal of Intelligent Information Systems*, 9(2):181–202, 1997.

[13] N.L. Zhang and D. Poole. A simple algorithm for bayesian network computations. In *Proc of the tenth Canadian Conference on Artificial Intelligence*, pages 171–178, 1994.