Using Weighted MAX-SAT Engines to Solve MPE

James D. Park
Computer Science Department
University of California
Los Angeles, CA 90095
jd@cs.ucla.edu

Abstract

Logical and probabilistic reasoning are closely related. Many examples in each group have natural analogs in the other. One example is the strong relationship between weighted MAX-SAT and MPE. This paper presents a simple reduction of MPE to weighted MAX-SAT. It also investigates approximating MPE by converting it to a weighted MAX-SAT problem, then using the incomplete methods for solving weighted MAX-SAT to generate a solution. We show that converting MPE problems to MAX-SAT problems and using a method designed for MAX-SAT to solve them often produces solutions that are vastly superior to the previous local search methods designed directly for the MPE problem.

Introduction

Probabilistic reasoning has a strong relation to logical reasoning. Many problems from one class have natural analogs in the other. The similarities between the problems from probabilistic reasoning and logical reasoning sometimes allows solution techniques to be transfered from one domain to the other. For example, previous results for the Most Probable Explanation (MPE) (Kask & Dechter 1999) have been obtained by noticing the similarity between MPE and satisfiability, and leveraging incomplete methods designed for satisfiability in order to approximately solve the MPE problem. In the other direction, methods used in probabilistic problems have been applied in order to improve the theoretical analysis of logical problems (Rish & Dechter 2000).

The *Most Probable Explanation* (MPE) is the problem of finding the variable instantiation of a Bayesian network that has the highest probability given some evidence. Essentially it provides the most likely state of the system given what has been observed. MPE has a number of applications including diagnosis and explanation. Unfortunately, MPE is an NP-complete problem (Littman 1999), so approximation techniques are necessary. As mentioned above, local search techniques for MPE have been inspired by the relation between satisfiability and MPE. MPE enjoys even closer ties with weighted maximum satisfiability (weighted MAX-SAT), another problem from logic which shares MPE's optimization characteristics. *Weighted MAX-SAT* is the problem of taking a set of clauses with associated weights, and finding the instantiation that produces the largest sum of the weights of satisfied clauses. Weighted MAX-SAT is used for example to resolve conflicts in a knowledge base. Finding approximate solutions to weighted MAX-SAT has received significant research attention, and novel algorithms have been developed that have proved to be very successful.

This paper investigates using local search algorithms developed for weighted MAX-SAT and applying them to approximately solve MPE. Local search is a general optimization technique which can be used alone or as a method for improving solutions found by other approximation methods. We compare two successful local search algorithms in the MAX-SAT domain (Discrete Lagrangian Multipliers (Wah & Shang 1997), and Guided Local Search (Mills & Tsang 2000)) to the local search method proposed for MPE (Kask & Dechter 1999). For large problems, the MAX-SAT algorithms proved to be significantly more powerful, typically providing instantiations that are orders of magnitude more probable.

The paper is organized as follows: First, we formally introduce the MPE and MAX-SAT problems. Then we present the reduction of MPE to MAX-SAT. We then introduce the MAX-SAT algorithms that will be evaluated. Finally, we provide experimental results comparing the solution quality of MPE approximations using the MAX-SAT methods to the previously proposed local search method developed for MPE.

MPE and MAX-SAT

The variables that appear in the Bayesian networks we will consider are over a finite domain of possible values. Each variable can take on a single value from a finite set of mutually exclusive possibilities.

Definition 1 *An instantiation of a set of variables is a function that assigns a value to each variable in the set. Two instantiations are compatible if they agree on the assignment of all of the variables that they have in common.*

We denote instantiations by the values of each variable to simplify the notation since the variable it is associated with will be clear from the context. Consider, for example, a variable A that can take on values in $\{a_1, a_2, a_3\}$, and variable

A	$Pr(A)$
a_1	.3
a_2	.5
a_3	.2

A	B	$Pr(B\|A)$
a_1	b_1	.2
a_1	b_2	.8
a_2	b_1	1
a_2	b_2	0
a_3	b_1	.6
a_3	b_2	.4

Figure 1: CPTs for a Bayesian network $A \rightarrow B$.

B that can take on values in $\{b_1, b_2\}$. Then instantiation (a_1) is compatible with instantiations (a_1, b_2) and (b_3), but not with (a_2, b_2). For boolean variables, we denote the values by the lowercase variable name or the negated variable name. For example for boolean variables C and D, the instantiation $(c, \overline{d})$ represents the assignment of C to true and D to false.

Definition 2 *A conditional probability table (CPT) T for a variable V with a set of parent variables $\mathbf{P}$ is a function that maps each instantiation of $V \cup \mathbf{P}$ to a real value in $[0, 1]$ such that for any instantiation $\mathbf{p}$ of $\mathbf{P}$, $\sum_v T(\{v\} \cup \mathbf{p}) = 1$ where v ranges over the values of V.*

A CPT provides the probability for each possible value of V given a particular instantiation of the parents. It is called a table since it is often represented in tabular form, enumerating the conditional probability associated with each instantiation. Figure 1 contains example CPTs corresponding to $Pr(A)$ and $Pr(B|A)$

Bayesian networks represent a probability distribution over a set of variables, factored into a specific form. It consists of a directed graph which specifies specific independence relationships between the variables, together with a conditional probability table for each variable. Formally,

Definition 3 *A Bayesian network is a pair $(\mathcal{G}, \mathcal{P})$ where $\mathcal{G}$ is a directed acyclic graph whose nodes are variables, and $\mathcal{P}$ is a set which consists of the CPT of each variable in $\mathcal{G}$, where the parents of each CPT correspond to the parents of the corresponding variable in the graph.*

Because of the way Bayesian networks are factored, computing the probability of a complete instantiation of its variables is very simple. The probability of a complete instantiation is the product of the entry of each CPT that is compatible with the instantiation. For example, in the network in Figure 1 the instantiation (a_3, b_1) has probability $.2 * .6 = .12$

The *MPE* problem is defined as follows: Given a Bayesian Network and an instantiation of a subset of the variables (the *evidence*), find the (not necessarily unique) complete instantiation with the highest probability that is compatible with the evidence.

Now we will consider some related concepts in logical reasoning. Unlike in probabilistic reasoning, logical reasoning deals with propositional variables. We will use the following standard terminology:

Definition 4 *A literal is a variable or its negation. A clause is a disjunction of literals and a weighted clause is a clause, together with a non-negative weight. A weighted CNF formula is a set of weighted clauses.*

We denote clauses using standard CNF formula conventions. Weights are denoted as superscripts. We denote a weighted CNF formula by the weighted clauses conjoined together. For example, the following formula consists of three weighted clauses : $(x \vee \overline{y} \vee \overline{z})^3 \wedge (\overline{x})^{10.1} \wedge (y)^{.5}$. The weight of a complete instantiation of a weighted CNF formula is the sum of the weight of the satisfied clauses. So, for the previous example, the instantiation $(x, y, \overline{z})$ has weight $3 + .5 = 3.5$ since it satisfies the first and third clauses, but not the second.

The *MAX-SAT* problem is defined as follows: Given a weighted CNF formula, find the (not necessarily unique) instantiation which has the highest weight.

Reducing MPE to MAX-SAT

An MPE problem can be converted into a weighted CNF expression whose MAX-SAT solution immediately produces the solution to the corresponding MPE problem. We begin by showing how to reduce a Bayesian network with only binary variables and positive CPT entries, and later show how to extend it to handle zeros, non-binary variables, and evidence.

The basic idea is that each CPT entry induces a weighted clause in the induced MAX-SAT problem. The only instantiations that do not satisfy the clause $l_1 \vee l_2 \vee ... \vee l_n$ are the instantiations in which each literal in the clause evaluates to false. We use this fact as the basis of the conversion. Each row in the CPT generates a weighted clause which contains the negation of each of the variables in the row and is weighted with the negative log of the conditional probability. For example, the network $C \rightarrow D$ with CPTs :

C	$Pr(C)$
c	.3
$\overline{c}$	.7

C	D	$Pr(D\|C)$
c	d	.2
c	$\overline{d}$	.8
$\overline{c}$	d	.1
$\overline{c}$	$\overline{d}$	.9

induces the weighted CNF expression

$$(\overline{c})^{-\log .3} \wedge (c)^{-\log .7} \wedge (\overline{c} \vee \overline{d})^{-\log .2}$$

$$\wedge (\overline{c} \vee d)^{-\log .8} \wedge (c \vee \overline{d})^{-\log .1} \wedge (c \vee d)^{-\log .9}$$

.

Consider, for example, the instantiation $c, \overline{d}$. It satisfies all of the clauses except $(\overline{c})^{-\log .3}$ and $(\overline{c} \vee d)^{-\log .8}$. So the sum of weights of the unsatisfied clauses is $-\log .24$ which is the negative log of the probability of that instantiation. Notice that for any instantiation of the variables, the sum of the weights of the unsatisfied clauses equals the negative log of the probability of the instantiation in the Bayesian network. This is true in general and forms the basis for the reduction.

Theorem 1 *For any instantiation I of a positive Bayesian Network which contains only binary variables, the sum of the weights of the clauses that I leaves unsatisfied in the induced weighted CNF expression equals* $-\log \Pr(I)$.

Proof: The unsatisfied clauses are those in which every literal evaluates to false. A clause is not satisfied if and only if the corresponding CPT entry is compatible with the instantiation. Thus the sum of the weights of the unsatisfied clauses is the sum of the negative logs of the compatible CPT entries. Since the probability of a complete instantiation I is the product of the CPT entries compatible with I, the sum of the weights of the unsatisfied clauses is $-\log \Pr(I)$.

Maximizing the weight of the satisfied clauses minimizes the sum of the excluded clauses which is equivalent to maximizing the probability in the original network. Thus solving the MAX-SAT problem also solves the MPE problem.

Handling Zeros

A CPT entry which has zero probability can not be transformed by the previous transformation because the log of zero is undefined. This problem can be circumvented by replacing the clause weight $-\log 0$ with a sufficiently large value w_0. The value must be large enough that it preserves the ordering of the solution qualities. In other words, if one instantiation of the Bayesian network has a higher probability than another, the corresponding MAX-SAT instantiations should be ordered in the same way. Letting w_0 be larger than the sum of all of the weights of the clauses associated with non-zero CPT entries is sufficient.

Theorem 2 *Let* w_0 *be a greater than the sum of the negative logs of the non-zero CPT entries of a Bayesian network with binary variables. Let the weight of the clauses whose associated CPT entries are zero have weight* w_0*, with the other weights as before. Then, any positive probability instantiation I has a higher score for the weighted CNF expression than any zero probability instantiation. Additionally, the sum of the weights of the unsatisfied clauses for I remains* $-\log \Pr(I)$.

Proof: Because w_0 is larger than the sum of all of the non w_0 weights, any instantiation that satisfies the w_0 clauses (corresponding to a positive Bayesian network instantiation) has a higher score than any instantiation that does not satisfy one of them (corresponding to a zero probability network instantiation). For those that satisfy the w_0 clauses, the sum of the unsatisfied clauses remains $-\log \Pr(I)$, so the solution ordering is preserved.

Replacing $-\log 0$ with a constant has the added benefit that hill climbing can be performed in the MAX-SAT space, even when in the MPE space all neighbors of the state have zero probability.

Beyond Binary Variables

When there are more than 2 possible values for a variable, we can not interpret it as a single boolean variable. Instead, we introduce a variable (the *indicator variable*) for each value the network variable can take on, with additional clauses to enforce the constraint that exactly one of them must be true. The clauses for the CPT entries are created as before except the indicator variables appear in the clause instead. An additional weighted clause of the form $(v_1 \vee .... \vee v_n)$ with a large weight is introduced to force one of the values to be true. By choosing a large enough weight, any locally maximal MAX-SAT instantiation will include at least one positive assignment of an indicator corresponding to each network variable. One possible choice for the weight of a constraint clause for a variable is double the sum of the weights of the other clauses that contain the corresponding indicators.

We must also ensure that only one indicator corresponding to a particular network variable is set in the resulting solution. An obvious way to ensure that is to add clauses with large weight of the form $\overline{v_i} \vee \overline{v_j}$ for each pair of indicators associated with a network variable. The problem with this scheme is that it adds a lot of clauses, which complicates the problem. There is a simpler alternative. Note that all of the indicators appear negatively except in the constraint clause. Thus having multiple indicators instantiated can not improve the score. In fact, setting more than one indicator for a particular network variable will decrease the score unless all of the non-constraint clauses that contain the extra variable have weight zero. All we need to ensure is that each clause contributes some positive weight. This is satisfied automatically for sensible networks since a sufficient condition is that the network variable does not necessarily attain a particular value. In other words, for at least one parent instantiation, it has a probability of less than one of achieving that value. Still, for completeness such a perverse network can be handled by adding a small constant to each of the weights associated with the CPT entry clauses before computing the weights for the constraint clauses, or by treating that variable as if it were set to the necessary value by some evidence. Thus we have

Theorem 3 *Any locally maximal instantiation of the induced weighted CNF expression satisfies the constraint that exactly one of the indicator variables is true for each variable. Additionally, the weight of the unsatisfied clauses for positive probability instantiation I remains* $-\log \Pr(I)$.

To illustrate, we return to our original example. The network from Figure 1 induces the weighted CNF formula

$$(\overline{a_1})^{-\log(.3)} \wedge (\overline{a_2})^{-\log(.5)} \wedge (\overline{a_3})^{-\log(.2)} \wedge$$
$$(a_1 \vee a_2 \vee a_3)^{\omega_a} \wedge (\overline{a_1} \vee \overline{b_1})^{-\log(.2)} \wedge (\overline{a_1} \vee \overline{b_2})^{-\log(.8)} \wedge$$
$$(\overline{a_2} \vee \overline{b_1})^{-\log(1)} \wedge (a_2 \vee b_2)^{w_0} \wedge (\overline{a_3} \vee \overline{b_1})^{-\log(.6)} \wedge$$
$$(\overline{a_3} \vee \overline{b_2})^{-\log(.4)} \wedge (b_1 \vee b_2)^{\omega_b}$$

where w_0,ω_a, and ω_b are chosen as described above.

Entering Evidence

Evidence for a Bayesian network is an instantiation of a subset of its variables. Evidence is entered simply by replacing the propositional variables that correspond to the evidence to their appropriate values then simplifying by dropping any clause that contains true, and removing false from all clauses. Any clause that would not be satisfied by an instantiation compatible with the evidence would still not be satisfied for a compatible instantiation over the new problem. Thus the sum of the unsatisfied clauses remains the same. Continuing with the example, entering evidence (a_1) replaces a_1 with true, and a_2 and a_3 with false, resulting in the weighted CNF formula

$$()^{-\log(.3)} \wedge (\overline{b_1})^{-\log(.2)} \wedge (\overline{b_2})^{-\log(.8)}$$

.

MAX-SAT Algorithms

There are a variety of algorithms in the literature for approximating MAX-SAT. We consider two methods that have been shown to be particularly effective in solving MAX-SAT problems. They are the method of Discrete Lagrangian Multipliers (DLM) and Guided Local Search (GLS).

Both of the algorithms that we consider are local search techniques. Local search is a general optimization technique. The most basic local search method is hill climbing. Hill climbing works by repeatedly improving the current solution by moving to a better neighboring solution. This continues until no neighbor is a better solution. For our purposes, we define the neighbors of an instantiation to be those instantiations produced by changing which indicator corresponding to a particular network variable is set. For example, for the MAX-SAT problem induced by the network in Figure 1, the neighbors of the instantiation $(a_1, \overline{a_2}, \overline{a_3}, b_1, \overline{b_2})$ are $(\overline{a_1}, a_2, \overline{a_3}, b_1, \overline{b_2})$, $(\overline{a_1}, \overline{a_2}, a_3, b_1, \overline{b_2})$, and $(a_1, \overline{a_2}, \overline{a_3}, \overline{b_1}, b_2)$.

Both DLM and GLS use hill climbing as a subroutine. They work by modifying the cost function used to search each time a local optimum is found. Instead of simply using the sum of the weights of the unsatisfied clauses as the cost function, another criteria, which changes throughout execution is chosen. This allows the algorithms to naturally explore the search space even after a local minimum has been obtained. Empirically this has been shown to be much better than randomly choosing a new instantiation each time a local minimum is encountered. At a high level they both work as follows:

```
do
   current=hillClimb(p,current)
   augmentCostFunction(p,current)
while(termination condition not met)
```

Here p is the problem instance. The hillClimb routine iterates through the variables, selecting the best change for each (as measured by the current cost function) until a local minimum is reached. It also computes the true score (as opposed to the score for the modified cost function) at each step and remembers the best state encountered.

The MAX-SAT algorithms we consider differ only in the way in which they generate the objective function.

Discrete Lagrangian Multipliers

Discrete Lagrangian Multipliers (Wah & Shang 1997) is a framework for combinatorial optimization subject to constraints. It has a solid theoretical foundation which is based on an extension of constrained optimization using Lagrange multipliers for continuous variables. In the weighted MAX-SAT domain, the clauses are the constraints, and the sum of the weights of the unsatisfied clauses is the cost function. In addition to the weight w_C, a Lagrangian multiplier λ_C is associated with each clause C. The cost function function for DLM is of the form

$$\sum_C w_C + \sum_C \lambda_C$$

where C ranges over the unsatisfied clauses. Initially, each λ_C is zero. Each time a local minimum is encountered, the λs corresponding to the unsatisfied clauses are increased by adding a constant.

Guided Local Search

As opposed to the theoretically based DLM, Guided Local Search (Mills & Tsang 2000) is a heuristically developed method for solving combinatorial optimization problems. It has been shown to be extremely successful in solving general weighted MAX-SAT problems. The initial cost function is the number of unsatisfied clauses. Like DLM, GLS associates an additional weight with each clause. Since the number of unsatisfied clauses is constant for MPE based formulas with the neighbor definition we use, the objective function essentially becomes $\sum_C \lambda_C$ where C ranges over the unsatisfied clauses. When a local minimum is reached, the weights of some of the unsatisfied clauses are increased. Specifically, the λs of the unsatisfied clauses with the maximum utility are incremented by 1 where the utility of a clause C is given by $w_C/(1+\lambda_C)$. Notice that unlike DLM which increments the λs of all of the unsatisfied clauses, GLS modifies only a few (often just one) of them.

Additionally, the authors of both algorithms note that it is advantageous to periodically scale the weights of the the λs when augmenting the objective function. This prevents the search space from becoming too rugged. We implemented the rescaling by multiplying each λ by .8 every 200 times a local minimum is found.

Experimental Results

We implemented DLM, GLS as well as a previous stochastic local search (SLS) based MPE approximation (Kask & Dechter 1999) that performed well when compared to other MPE approximation schemes. SLS is a local search algorithm that at each step performs either a hill climbing or a stochastic variable change. Periodically the search is restarted in order to escape local maxima.

For SLS, we used the parameters specified in (Kask & Dechter 1999). We found no mention of the preferred probability for taking a stochastic flip, so tuned it experimentally. We used networks drawn from the same distribution as those in the first data set. The value that worked best was 0.2. For DLM, the increment used to modify the weights was chosen in the same manner. The value chosen was 0.02. GLS had no parameters to tune.

We experimented with three synthetic sets of 100 Bayesian networks consisting of binary variables, and ten

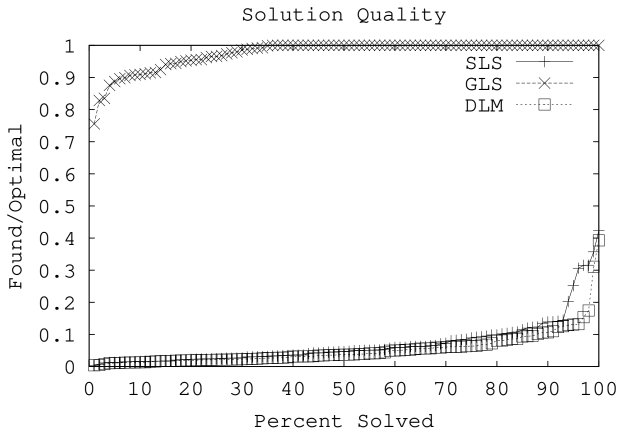

Figure 2: A plot of the ratios of the approximate solution to the exact MPE probability. The values are sorted, so that for a particular point on the X axis it means that X percentage of the instances had solutions that were at least as far off as the Y value for that point.

	SLS	GLS	DLM
exact	0	100	0
non-zero	0	100	7

Table 1: Only GLS was able to perform well on the deterministic data sets.

real world networks. In the first and second data sets, we generated networks with 100 variables, and widths[1] around 16 in order to be large enough to be difficult for the local search methods, but small enough to be able to use an exact algorithm for comparison. We used the method described in (Darwiche 2001) to generate the network topology. In the third set each network had 500 variables.We generated the network topology so that the probability of having an edge between any two nodes was a constant. This has the effect of producing very connected networks that are beyond the reach of structure based methods (they typically have widths of over 100). The CPTs for the first and third sets were instantiated randomly. In the second, each non-root CPT was instantiated as a deterministic function of its parents, while the root CPTs were instantiated randomly. For the first and second sets, we computed the exact MPE probability in order to be able to compare solutions. We then gave each algorithm 30 seconds per network to generate the best solution it could. Each algorithm was initialized to start at the same location.

In all three synthetic experiments, GLS clearly dominated the other methods. GLS was able to produce the exact answer for most of the problems in the first two data sets. In the third set, we were unable to compute the exact MPE values so we can not tell how close to the exact value each algorithm achieved, but GLS produced significantly better solutions than the other methods.

[1]The width of a Bayesian network is a property of the network topology that is related to how connected the network is. The state of the art algorithms that perform inference on Bayesian networks typically have complexity that is linear in the number of variables and exponential in the width.

	Worst	Median	Best
SLS/GLS	$7\text{x}10^{-14}$	$1\text{x}10^{-10}$	$2\text{x}10^{-7}$
DLM/GLS	$3\text{x}10^{-11}$	$6\text{x}10^{-7}$	$1\text{x}10^{-3}$

Table 2: GLS dominated the other methods. We show some statistics of the ratio of the other methods to GLS. The GLS solution was several orders of magnitude more probable, even for the closest solution.

Figure 2 shows the results for the first data set. The ratio of the probability of the found solution to the probability of the true solution is plotted for each algorithm. The ratio values are sorted from worst to best solution quality for each method. Only GLS was able to compute the exact value of any problem, and it did so in 65 of the 100 instances. The lowest score it produced was 0.76 of the optimal value. SLS and DLM had very similar performance on this data set.

Table 1 shows the result for the second data set. For the deterministic problems, the vast majority of the instantiations have probability 0. The table shows how many of the 100 problems that each method was able to solve exactly, as well as how many were able to find a solution with a non-zero probability. In the deterministic set GLS again dominated, solving all problems exactly. DLM performed slightly better than SLS in that it was able to find non zero solutions for seven of the networks.

In the third set, we again gave each algorithm 30 seconds per network to produce a solution. These networks were too connected to produce exact solutions, so we compare only the relative performance between the algorithms.

GLS significantly outperformed the other methods on the third data set as well. Table 2 provides some statistics on the solution quality produced by SLS and DLM relative to GLS. The worst median, and best columns show the worst median, and best ratios of the found value to the GLS for the 100 networks. So for example, even at its best, SLS produced a solution that was only $2\text{x}10^{-7}$ as large as the GLS solution for that network.

In each MPE problem in all three synthetic sets, GLS produced a better solution than the other two methods, and in many cases, produced solutions that were many times better.

We also experimented with 10 real world networks obtained from the Bayesian network repository(Bayesian network repository). For each network, we ran 20 test cases. Each test case had 10 randomly selected evidence variables. Care was taken to ensure that the probability of evidence was positive. The parameters and time limit were the same as the ones used for the synthetic network experiments.

The results on the real world networks showed significant differences in the performance characteristics of the algorithms. Table 3 summarizes the results. For each network we list the number of variables and the average and maximum number of possible values for its variables. We also

		var insts		SLS/GLS			DLM/GLS			SLS		DLM		GLS	
Network	# vars	avg	max	min	med	max	min	med	max	best	> 0	best	> 0	best	> 0
Barley	48	8	67	9e-2	1e2	9e9	7e−2	1e2	3e10	11	20	6	20	3	20
Diabetes	413	11	21	1	1	1	1	1	∞	14	0	20	6	14	0
Link	724	2	4	0	0	0	0	0	0	0	0	0	0	20	20
Mildew	35	17	100	0	1.4	3e1	0	1.2	2e1	15	19	4	19	7	20
Munin1	189	5	21	0	0	0	0	0	5e3	0	0	1	7	19	20
Munin2	1003	5	21	0	0	0	0	0	3e76	0	0	9	9	11	20
Munin3	1044	5	21	0	0	0	0	0	1e101	0	0	8	8	12	20
Munin4	1041	5	21	0	0	1	0	0	5e111	2	0	8	6	14	18
Pigs	441	3	3	0	0	1	1e-19	1e-2	1	1	1	6	20	20	20
Water	32	3	4	1	1	1	1	1	1	20	20	20	20	20	20

Table 3: Statistics for experiments on 10 real world networks.

provide minimum, median and maximum statistics for the ratio of SLS and DLM to GLS over the 20 runs. We also count the number of times the solution was the best of the three algorithms, as well as the number of positive probability solutions found.

No method dominated the others for all networks. For networks with a relatively small number of variables with many values per variable (ex. Barley, Mildew), SLS slightly outperformed DLM, and significantly outperformed GLS. For the networks with many variables, GLS and DLM performed significantly better. GLS excelled at finding positive instantiations, finding a positive instantiation in nearly every case for every network except Diabetes. It also produced the best solutions most often for the large networks. DLM was not able to find as many positive solutions, but for some of the large networks (ex. Munin2, Munin3, Munin4) generated solutions vastly superior to GLS on those in which it was able to find a positive solution. We thought that this may be a result of too little search time for such large problems, but we found increasing the time limit from 30 seconds to 2 minutes did not qualitatively alter this relationship. It appears from the data we generated, that DLM performance relative to GLS increases as the number of states per variable increases, although more study would be needed to verify this.

All of the methods have their merits. SLS seems to do well for networks with few variables and many values per variable, though it performs poorly on problems in which the vast majority of instantiations have 0 probability. DLM performs better than SLS when the number of variables is increased, and can sometimes outperform GLS. GLS provided the best overall performance, providing the best answer of the three algorithms in the majority of cases. Still, in a significant minority of the cases for the large networks, DLM was able to outperform GLS.

Conclusion

MPE and MAX-SAT problems are strongly related. The reduction from MPE to MAX-SAT is easy to perform. Using this conversion, methods for approximating weighted MAX-SAT can be leveraged to approximate MPE. Both DLM and GLS can often provides answers that are far superior to those produced by the local search method designed directly for MPE.

Acknowledgement

This work has been partially supported by MURI grant N00014-00-1-0617.

References

Bayesian network repository. www.cs.huji.ac.il/labs/compbio/Repository.

Darwiche, A. 2001. Recursive conditioning. *Artifical Intelligence Journal* 125(1-2):5–41.

Kask, K., and Dechter, R. 1999. Stochastic local search for Bayesian networks. In *Workshop on AI and Statistics 99*, 113–122.

Littman, M. 1999. Initial experiments in stochastic satisfiability. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, 667–672.

Mills, P., and Tsang, E. 2000. Guided local search for solving SAT and weighted MAX-SAT problems. *Journal of Automated Reasoning* 24:205–233.

Rish, I., and Dechter, R. 2000. Resolution versus search: Two strategies for SAT. *Journal of Automated Reasoning* 24(1/2):225–275.

Wah, B., and Shang, Y. 1997. Discrete Lagrangian-based search for solving MAX-SAT problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 378–383.