Stochastic Local Search for Bayesian Networks

Kalev Kask and Rina Dechter

Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

Abstract

The paper evaluates empirically the suitability of Stochastic Local Search algorithms (SLS) for finding most probable explanations in Bayesian networks. SLS algorithms (e.g., GSAT, WSAT [16]) have recently proven to be highly effective in solving complex constraint-satisfaction and satisfiability problems which cannot be solved by traditional search schemes. Our experiments investigate the applicability of this scheme to probabilistic optimization problems. Specifically, we show that algorithms combining hill-climbing steps with stochastic steps (guided by the network's probability distribution) called *G+StS*, outperform pure hill-climbing search, pure stochastic simulation search, as well as simulated annealing. In addition, variants of G+StS that are augmented on top of alternative approximation methods are shown to be particularly effective.

1 Introduction and Motivation

The *Most Probable Explanation* (MPE) task appears in applications such as diagnosis, abduction, and explanation. For example, given data on clinical findings, MPE can postulate on a patient's probable affliction. In decoding, the task is to identify the most likely input message transmitted over a noisy channel given the observed output. Researchers in natural language consider the understanding of text to consist of finding the most likely facts (in internal representation) that explains the existence of the given text. In computer vision and image understanding, researchers formulate the problem in terms of finding the most likely set of objects that explains the image. Scientific theories are models that attempt to fit the given observations, etc.

Belief networks provide a formalism for reasoning about partial beliefs under conditions of uncertainty. They are defined by a directed acyclic graph over nodes representing random variables of interest (e.g., the temperature of a device, the gender of a patient, a feature of an object, the occurrence of an event.) The arcs signify the existence of direct causal influences between linked variables. The strength of these influences are quantified by conditional probabilities that are attached to each cluster of parents-child nodes in the network.

Definition **1.1 (Belief Networks)** *Given a set, $X = \{X_1, \ldots, X_n\}$ of random variables over multi-valued domains $D_1, \ldots, D_n$, a belief network is a pair (G, P) where G is a directed acyclic graph and $P = \{P_i\}$. $P_i = \{P(X_i \,|\, pa\,(X_i))\}$ are conditional probability matrices associated with X_i. The set $pa\,(X_i)$ is called the parent set of X_i. This network represents the probability $P(x_1, \ldots, x_n) = \prod_{k=0}^{n} P(X_k \,|\, pa\,(X_k))$. The moral graph of a belief network is obtained from G by connecting the parents of each variable to each other and removing the arrows.*

Definition **1.2 (Most Probable Explanation)** *Given a belief network, the Most Probable Explanation (MPE) task is to find a complete assignment $(X_1 = x_1, \ldots, X_n = x_n)$ which agrees with the available evidence, having the highest probability among all such assignments: to find an assignment $(x_1, \ldots, x_n)$ such that*

$$P(x_1, \ldots, x_n) = max_{X_1,\ldots,X_n} \prod_{k=0}^{n} P(X_k \,|\, pa\,(X_k), e)$$

While the MPE task is NP-hard, there are tractable subclasses that have efficient algorithms. For example, when the network is singly connected, there is a linear algorithm for solving the MPE problem [9]. This algorithm can be generalized to multiply connected networks by using either the *cycle cutset* (also called *conditioning*) technique or the *join-tree-clustering* technique [9]. However, these methods work well only if there are small cutsets or clusters in the network. The complexity of algorithms based on the cycle cutset idea is time exponential in the cutset size but requires only linear space. The complexity of join-tree-clustering algorithms is both time and space exponential in the cluster size. It equals the induced width of the network's moral graph.

A variation of the tree-clustering technique is the bucket elimination algorithm [4]. This algorithm partitions the problem's conditional probability tables into buckets relative to the processing order. Each bucket is processed once, computing a new function which is added to the problem.

Approximating the MPE task is NP-hard [1], but there are still cases when approximation algorithms are essential, especially when exact algorithms are not feasible. Approximation algorithms trade completeness for efficiency in the hope that they can find a good solution in a reasonable amount of time for a substantial number of instances.

One can envision two types of approximation algorithms: those approximating the exact elimination type algorithm such as mini-bucket approximation [12], and those approximating search algorithms. This paper focuses on approximating search algorithms known as Stochastic Local Search algorithms (SLS). An example of a SLS algorithm for probabilistic reasoning is Stochastic Simulation [14]. Typically, such algorithms begin with a random complete assignment to all variables. Guided by an objective function, it is improved by changing the values of variables individually. Under certain idealistic conditions these stochastic methods are guaranteed to converge to the optimal solution [2]. However, as we will later show, in practice, the performance of these algorithms may be very poor, unless they are augmented with various heuristics. Unfortunately, once these heuristics are added, the theoretical guarantees no longer hold.

In this paper, we investigate stochastic local search heuristics for the MPE task. These algorithms do not guarantee an optimal solution. They can still be very useful, however, if they - return a close to optimal solution in a large percentage of cases, within a reasonable amount of time and space. In section 3 we will focus on two stochastic search algorithms, Stochastic Simulation and a greedy algorithm. Stochastic Simulation is a popular approximation algorithm for Bayesian networks, and randomized greedy local search algorithms have been very successful in solving constraint satisfaction problems. Whenever possible we compared these algorithms to both the complete bucket elimination algorithm and simulated annealing, which is another popular stochastic local search algorithm used for optimization problems.

In section 4 we will describe the network classes used for testing our algorithms: random uniform networks, random networks having noisy OR gates, and random coding networks. Section 5 presents the results. We show that stochastic simulation is not as successful as the greedy algorithm, although the combined algorithm of greedy and stochastic simulation is better than either greedy or stochastic simulation alone, and superior to simulated annealing. Finally, we demonstrate that our best variation of stochastic local search, $G + StS$ can be improved by augmenting the algorithm with the mini-bucket approximation approach. The algorithm begin with an assignment generated by the mini-bucket algorithm. This version is shown to enhance the approximation quality considerably. For coding problems we commence all local search algorithms with an assignment which equals the channel output. This heuristic greatly improves the performance of all SLS algorithms on this domain.

For reference we also compare the SLS algorithm with the mini-bucket scheme. Although this approach seems to be superior to all SLS algorithms in our experiments, this conclusion is premature. Since, intentionally most problems we experimented with were designed to be good for elimination.

2 Related Work

The problem of finding the most probable explanation (MPE) in belief networks has been studied extensively during the past 15 years. Following Pearl's propagation algorithm for singly connected networks and its extension by conditioning and clustering, and his stochastic simulation proposal [9], researchers have investigated various approaches, especially in the context of medical diagnosis. Our work on greedy algorithms can be viewed as an extension of the line of work presented in [10], [11], ranging from two layered networks to general belief networks. More recently, best first search algorithms were proposed [17] as well as algorithms based on linear programming [15]. Various other authors have worked on extending some of these algorithms to the task of finding the k most-likely explanations [7], [18].

3 Competing Algorithms

3.1 Bucket and Mini-Bucket Elimination

The Bucket Elimination (*Elim-MPE*) is a complete algorithm for solving the MPE problem (for details see [4].) It is described relative to an ordering of the variables. In the experiments reported in this paper we used the min-width variable ordering, which is defined on the moral graph and perceived to be a good heuristic ordering. The algorithm orders the nodes in the graph from last to first. At each step, it chooses the variable having the smallest number of neighbors in the remaining graph restricted to nodes not selected thus far.

Bucket elimination (*Elim-MPE*) has one bucket for each variable. It initially places each probability matrix in the bucket of its variable. The algorithm (see Figure 1) works in two phases. During the first, top-down phase, it processes each variable's bucket individually from the last variable to the first. It takes the product of all probability matrices and creates

Bucket Elimination (*Elim-MPE*) for Finding MPE in Bayesian Networks :

Input: A Belief Network $BN = \{P_1, ..., P_n\}$, an ordering of the variables d, observations e.

Output: The most probable assignment.

1. **Initialize:** Generate an ordered partition of the conditional probability matrices, $bucket_1, \ldots, bucket_n$, where $bucket_i$ contains all matrices whose highest variable is X_i. Put each observed variable in its bucket. Let $S_1, ..., S_j$ be the subset of variables in the processed bucket on which matrices (new or old) are defined.
2. **Backward:** For $p \leftarrow n$ downto 1, for all the matrices $h_1, h_2, ..., h_j$ in $bucket_p$:

   • (bucket with observed variable) **if** $bucket_p$ contains $X_p = x_p$, assign $X_p = x_p$ to each h_i and put each in an appropriate bucket.

   • **else** $U_p \leftarrow \bigcup_{i=1}^{j} S_i - \{X_p\}$. Generate functions $h_p = \max_{X_p} \Pi_{i=1}^{j} h_i$ and $x_p^o = argmax_{X_p} h_p$. Add h_p to bucket of largest-index variable in U_p.

3. **Forward:** Assign values in the ordering d using the recorded functions x^o in each bucket.

Figure 1: Bucket Elimination Algorithm for MPE

Greedy Algorithm for Finding MPE in Bayesian Networks :

Input : A Bayesian Network (G, P), a time bound T and a cutoff factor cf.
Output : An assignment $u = (u_1, \ldots, u_n)$ that is a local minima of $E(X)$.

Perform numerous trial runs of the algorithm. Each trial run consists of the following steps :

1. Check if the time bound T has been exceeded. If yes, quit.
2. Generate a random initial assignment $x = (x_1, \ldots, x_n)$.
3. Perform a trial run of the algorithm :
   (a) Determine if the run should be terminated by seeing if the total time used exceeds $cf \cdot t_{try}$, where t_{try} is the best assignment time in this run. If yes, end the run and go to 1.
   (b) Pick a variable $X_i = u_i$ that has a value $v_i \in D_i$ and maximizes the reduction in the cost function when the value is changed from u_i to v_i. Change the value. This is called a greedy flip. The change in the cost function when we change the value of variable X_i from u_i to v_i is:

$$\Delta E_{X,u_i \to v_i} = -LP(u_i \mid pa(X_i)) + LP(v_i \mid pa(X_i)) -$$

$$- \sum_{A \in Child(X_i)} [-LP(X_A \mid u_i, pa(X_A) - X_i) + LP(X_A \mid v_i, pa(X_A) - X_i)]$$

Figure 2: Greedy Algorithm for MPE

```
Stochastic Simulation for Finding MPE in Bayesian Networks :

Input : A Bayesian Network $(G, P)$, a time bound $T$ and a cutoff factor $cf$.
Output : An assignment $u = (u_1, \ldots, u_n)$.

Perform numerous trial runs of the algorithm. Each trial run consists of the
following steps :

1. Check if the time bound $T$ has been exceeded. If yes, quit.
2. Generate a random initial assignment $x = (x_1, \ldots, x_n)$.
3. Perform a trial run of the algorithm :
   (a) Determine if the run should be terminated by seeing if the total time used
       exceeds $cf \cdot t_{try}$, where $t_{try}$ is the best assignment time in this run.
       If yes, end the run and go to 1.
   (b) Pick a random variable $X_i = u_i$. Compute the probability
       distribution of $X_i$ based on the current values of other variables:
           $P(X_i \mid X - X_i) = P(X_i \mid pa(X_i)) \cdot \prod_{A \in Child(X_i)} P(X_A \mid X_i, pa(X_A))$
       Randomly pick a new value $v_i$ for $X_i$ from this distribution.
       We will call this a stochastic flip.
```

Figure 3: Stochastic Simulation for MPE

```
Greedy Algorithm Combined with Stochastic Simulation for MPE :

Input : A Bayesian Network $(G, P)$, a time bound $T$, a cutoff factor $cf$ and
a probability $P_{sf}$ of stochastic flip.

Output : An assignment $u = (u_1, \ldots, u_n)$ that is a local minima of $E(X)$.

Perform numerous trial runs of the algorithm. Each trial run consists of the
following steps :

1. Check if the time bound $T$ has been exceeded. If yes, quit.
2. Generate a random initial assignment $x = (x_1, \ldots, x_n)$.
3. Perform a trial run of the algorithm :
   (a) Determine if the run should be terminated by seeing if the total time used
       exceeds $cf \cdot t_{try}$, where $t_{try}$ is the best assignment time in this run.
       If yes, end the run and go to 1.
   (b) Pick a variable $X_i = u_i$. With probability $P_{sf}$ execute a stochastic flip,
       with probability $1 - P_{sf}$ execute a greedy flip.
```

Figure 4: Greedy Algorithm Combined with Stochastic Simulation for MPE

a new function. The new function does not contain the bucket's variable that is being eliminated by maximization. During the second, bottom-up phase, the algorithm constructs a solution by assigning a value to each variable along the ordering, consulting the functions created during the top-down phase. The complexity of the algorithm is characterized by the induced width.

Definition **3.1 (Induced Width)** *Given an undirected graph and an ordering $d = X_1, ..., X_n$ of its variables, the width of a node X_i is the number of its 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 the ordering, denoted $w^*(d)$, is the width of the induced graph obtained with respect to ordering d: nodes are processed from last to first; when a node X_i is processed, all its preceding neighbors are connected. The induced width of a graph w^* is the minimal induced width over all orderings. It is also known as the tree-width.*

Theorem [4] : The time and space complexity of the algorithm Elim-MPE are exponential in the induced width w^* of the network's ordered moral graph.

The bucket elimination algorithm can be approximated by a *Mini-Bucket Elimination* algorithm as follows. During processing $bucket_p$, all the functions in this bucket are partitioned into smaller subsets called mini-buckets and then each mini-bucket is processed independently in the same manner. It can be shown that the algorithm produces an upper bound on the MPE value as well as an assignment (and a lower bound) whose quality depends on the coarseness of the partitioning into mini-buckets. The following criteria for partitioning functions into mini-buckets is used. Given a parameter i-bound, partition the functions in $bucket_p$ into mini-buckets so that the number of variables in each mini-bucket does not exceed the i-bound. For more details see [13, 12].

Finally, we would like to mention algorithm *Iterative Belief Propagation* (IBP). The algorithm is identical to Pearl's belief propagation on polytrees [9]. However, it is applied to an arbitrary network in several iterations. The most likely value for each variable is produced as the output assignment.

3.2 Greedy Algorithm

The Greedy Local Search algorithm is a randomized hill-climbing algorithm. It minimizes the cost function $E(X) = -LP(x_1, \ldots, x_n) = -logP(x_1, \ldots, x_n)$. A global minimum of this cost function is a solution to the MPE problem. The greedy algorithm is not a complete algorithm, it is guaranteed to converge only to a local maximum. The algorithm is typical of algorithms used in connectionist networks. Although greedy local search algorithms are not guaranteed to find the optimal solution, in practice they may rapidly solve the constraint satisfaction problem [8, 16, 5, 6].

The algorithm is broken into a series of independent trial runs. Having more than 1 run is not technically necessary, but experiments show that it is beneficial to restart the program every once in awhile. Each run begins with a random initial complete assignment. A basic operation during the execution of the greedy algorithm is picking a variable X_i and changing its value from u_i to v_i. This is called flipping a variable X_i. The variable X_i and its new value v_i are chosen so that the change (reduction) in the cost function $-LP$ is maximized. Greedy search algorithms are also called local search algorithms because the change in the cost function when the value of variable X_i is changed from u_i to v_i can be computed locally based on the current values of X_i and its neighbors (Markov neighborhood.) The greedy algorithm we use for the MPE task is given in figure 2.

One of the issues with using randomized greedy algorithms is that unlike constraint satisfaction problems, in belief networks there is no objective criteria for termination, unless we know the optimal solution in advance. We have designed a heuristic for terminating the algorithm with the idea that if for a long time no new improved assignment was found, search is stopped. We have implemented this heuristic by specifying a cutoff factor cf. When the total time used more than $cf \cdot t$ where t is the best assignment time, the algorithm terminates.

3.3 Stochastic Simulation

Stochastic Simulation, also known as Gibbs Sampling, is a simple randomized algorithm that can be used for solving the MPE problem [9]. Stochastic simulation starts from a random initial assignment. It then generates a random sample of assignments. Each sample is computed from the previous sample by picking a variable X_i and changing its value from u_i to v_i. Variable X_i is picked either randomly or according to a fixed schedule. The new value v_i is picked randomly from the current probability distribution of X_i based on the current value of its neighbors. After computing a sample, stochastic simulation picks the assignment that was either generated most frequently or has the highest probability. Studies show that eventually this process converges to the optimal solution, although convergence may be slow [9].

In our implementation of stochastic simulation, we broke the execution of the algorithm into a number of sets of samples. Instead of computing one set of random samples, we compute a number of independent sets.

Each sample starts with a completely random assignment. When computing a sample we loop through the variables according to an ordering. A value for a variables chosen by inspecting its neighboring variables and computing its probability distribution according to the current values of its neighbors. A formal description of the algorithm is given in Figure 3.

Notice that the structure of this algorithm is very similar to the greedy algorithm. There are two differences:

1. How a variable X_i is picked.
2. How the new value v_i of X_i is computed.

In the greedy algorithm, X_i is picked in a greedy fashion from all variables, while in stochastic simulation it is picked randomly or according to a fixed schedule. Once a variable is selected in the greedy algorithm, its new value v_i is picked in a greedy fashion. In stochastic simulation it is picked randomly from the current probability distribution of X_i.

3.4 A Greedy Algorithm Combined With Stochastic Simulation

Since the greedy algorithm and the stochastic simulation are both local search type algorithms, they can be combined into one algorithm. A formal description of this algorithm is given in Figure 4.

As before, a basic operation is flipping a value of a variable. A flip can be done in two ways

1. A Stochastic Step. A variable is picked randomly and its new value computed stochastically.
2. A Greedy Step. A variable that gives the best improvement is picked and its new value computed greedily.

In order to implement this algorithm, we will use the same basic control structure of trials as the greedy algorithm does. A basic operation of this algorithm is a flip. The rules we used are as follows:

- When a greedy step is not possible (a local minimum), a stochastic step will be executed.
- When a greedy step is possible, it will be executed with probability q, otherwise, a stochastic step will be executed.

3.5 Simulated Annealing

Simulated annealing is a general stochastic algorithm for problem solving that combines both hill-climbing and random selection.

- Let A be a random assignment of values to variables, T be a (high) temperature, and $h = -logP(x_1, \ldots, x_n)$ a cost function.
- Repeat

```
    Select neighbor $A'$ of $A$ at random
    If $h(A') \leq h(A)$
        then $A = A'$
    else $A = A'$ with probability $e^{(h(A)-h(A)')/T}$
    Reduce T
Until temperature $T$ is 0.
```

4 Problem Format

We experimented with three types of networks: random uniform networks, random noisy-OR networks and coding networks. Random uniform networks and noisy-OR networks use the same generator for network's structure. In random uniform networks each probability table is generated according to a uniform random distribution. In random noisy-OR networks, each conditional probability represents a Noisy OR gate that is characterized by two parameters: noise and leak probabilities. Noise probability is the probability of the child node having a value 0 when at least one parent has a value 1. Leak probability is the probability of the child node having a value 1 when all parent nodes have a value 0.

Coding networks are associated with probabilistic decoding problems. These are multi-layered networks whose conditional probabilities are either parity functions or Gaussian distributions. The following subsections provides more details.

4.1 Random Networks

The input parameters to our random problem generator are

- N, the number of variables,
- K, the number of values for each variable,
- C, the number of conditional probability tables,
- P, the number parents in every conditional probability table.

We create a random problem instance by first generating the network structure and then generating probability table. A structure of the problem is created by constructing a complete order of variables (any ordering will do). This order is used to create C conditional probability tables in the following way. A variable X_i that has not been selected yet is randomly chosen. Then we pick P variables that appear after X_i in the ordering as parents of X_i. Notice that when $P = 1$, the network that is a forest, i.e., there are no cycles. For each variable X_i and its parents $pa(X_i)$ we create a conditional probability matrix $P(X_i \mid pa(X_i))$. For uniform networks those tables are generated randomly, for noisy-OR networks by noisy-OR gates. If the variables have parents, we will provide a prior probability. When evidence variables are used, evidence variables are randomly selected and assigned values.

4.2 Random Coding networks

It was recently shown that the task of probabilistic decoding can be formulated as finding the most probable assignment in Bayesian networks. The problem is defined by a pair (N, K), where N is the total number of bits transmitted over a channel, of which K bits are the original desired message for transmission. The rest

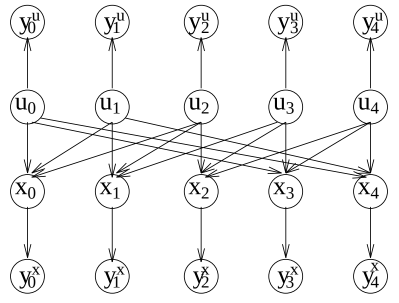

Figure 5: Belief network for structured (10,5) block code with parent set size P=3

of the $N-K$ bits are used for redundancy purposes in order to assist in the decoding task. Frequently, each of these added bits are defined as parity functions over a subset of the original K bits.

We experimented with several types of (N, K) randomly generated linear block codes. We used only the code rate $R = 1/2$, i.e. $N = 2K$. In our random coding networks, each of the $N-K$ bits is an XOR function node has a fixed number of parents, P amongst the original K bits. Therefore, when generating random coding networks, for each parity-check bit x_j, P parents are selected randomly from K information bits.

Random codes for rate R=1/2 are similar to the *low-density generator matrix* codes [3]. Note that the average number of *children* for each information bit is $(N-K)P/K$, which equals P for $N = 2K$. Thus, our random codes are very similar to *low-density generator matrix* codes [3], that have a fixed low number of children per each information bit (but may have huge parent sets if $R > 1/2$.

Our random codes can be represented as four-layer belief networks having K nodes in each layer (see Figure). The two inner layers (channel input nodes) correspond to the input information bits $u_i, 0 \leq i < K$, and to the parity-check bits, $x_i, 0 \leq i < K$. The two outer layers represent the channel output $y = (y^u, y^x)$, where y^u and y^x result from transmitting u and x, respectively and are generated using Gaussian noise functions. The input nodes are binary (0/1), while the output nodes are real-valued.

Given K, P, and the channel noise variance σ^2, a sample coding network is generated as follows. First, the appropriate belief network structure is created as described above. Then, we simulate an input signal assuming uniform random distribution of information bits, and the corresponding values of the parity-check bits are computed. An assignment is generated to the observed vector y by adding Gaussian noise to each information and parity-check bit. Finally, we fix the values of output bits as evidence. The decoding algorithm takes the coding network as an input and the observed channel output y (a real-valued assignment to the y_i^u and y_j nodes) and computes the MPE assignment.

The performance of a decoding algorithm is usually measured by the bit error rate (BER), which is simply the observed fraction of information bit errors, i.e., the ratio of the incorrectly decoded bits to the total number of information bits transmitted through the channel.

One of the most popular coding algorithms is the Iterative Belief Propagation algorithm which uses Pearl's belief propagation algorithm for polytrees iteratively [9]. The algorithm was shown to be extremely effective for coding problems inspite of their cyclic structure. For reference, we are comparing all our algorithms against the IBP algorithm.

5 Experiments

We ran a series of experiments with random uniform networks, random noisy-OR networks and random coding networks. In each experiment, we compared the performance of the following algorithms :

- Stochastic Simulation (StS)
- Greedy (G)
- Greedy combined with Stochastic Simulation (G+StS)
- Simulated Annealing (SA)
- Mini-bucket elimination (MB) with i-bound = 10
- Greedy combined with Stochastic Simulation on mini-bucket assignment (G+StS w/ MB)

We evaluated the algorithms by comparing their solution against the optimal solution whenever possible. We ran Elim-MPE on the same problems and the results were recorded for reference. However, the w^* of the network was often too large and Elim-MPE ran out of memory. So we also ran a combination of a mini-bucket algorithm with a search algorithm that does not have a memory problem, to guarantee an optimal solution. These results are not reported here.

Specifically, we compare the probability of the assignment found by each of the approximation algorithms against the probability of the optimal solution. We divided the results into 5 probability ranges depending on the ratio of probabilities of the solution found by the approximation to the optimal solution. The ranges are: within a factor of 0.95 of the optimal, between 0.95 and 0.5 of the optimal, between 0.5 and 0.2 of the optimal, between 0.2 and 0.01 of the optimal and less than 0.01 times the optimal. Each SLS approximation algorithm had the same amount of CPU time per problem. Once an algorithms used up its time, it outputed the best assignment found.

For random uniform networks and random noisy-OR networks, we chose a set of parameters N, C, P and K,

N C P K	Elim mpe	Sol Prob Range	StS #	G #	G+ StS #	SA #	IBP #	MB #	G+ StS w/ MB
256	97	>0.95	0	0	0	0	0	73	89
95		>0.50	0	5	17	1	0	22	11
2	w^*	>0.20	0	81	81	16	0	5	0
2	12.6	>0.01	0	14	2	38	0	0	0
		<0.01	100	0	0	0	100	0	0
256	81	>0.95	0	0	0	0	0	51	80
100		>0.50	0	7	20	0	0	38	19
2	w^*	>0.20	0	76	78	43	0	11	1
2	14.6	>0.01	0	17	2	57	0	0	0
		<0.01	100	0	0	0	100	0	0
256	69	>0.95	0	0	0	0	0	58	82
105		>0.50	0	1	9	0	0	31	16
2	w^*	>0.20	0	73	84	34	0	10	2
2	15.8	>0.01	0	26	7	65	0	1	0
		<0.01	100	0	0	1	100	0	0
256	35	>0.95	0	0	0	0	0	37	67
110		>0.50	0	1	6	0	0	40	29
2	w^*	>0.20	0	49	79	18	0	23	4
2	17.5	>0.01	0	50	15	82	0	0	0
		<0.01	100	0	0	0	100	0	0

Table 2: Random MPE. SLS time 20 sec. 100 samples.

N C P K	Elim mpe	Sol Prob Range	StS #	G #	G+ StS #	SA #	MB #	G+ StS w/ MB
128	100	>0.95	0	0	2	0	99	100
45		>0.50	0	14	49	0	0	0
2	w^*	>0.20	0	86	49	3	1	0
3	6.5	>0.01	0	0	1	74	0	0
		<0.01	100	0	0	23	0	0
128	97	>0.95	0	0	2	0	96	97
50		>0.50	0	1	30	0	3	3
2	w^*	>0.20	0	87	67	0	1	0
3	7.9	>0.01	0	12	1	42	0	0
		<0.01	100	0	0	58	0	0
128	89	>0.95	0	0	0	0	86	92
55		>0.50	0	4	15	0	12	8
2	w^*	>0.20	0	78	84	0	2	0
3	9.3	>0.01	0	18	1	17	0	0
		<0.01	100	0	0	83	0	0
128	61	>0.95	0	0	0	0	72	79
60		>0.50	0	0	12	0	23	21
2	w^*	>0.20	0	53	81	0	5	0
3	10.8	>0.01	0	47	7	2	0	0
		<0.01	100	0	0	97	0	0

Table 1: Random MPE. SLS time 20 sec. 100 samples.

N C P K	Elim mpe	Sol Prob Range	StS #	G #	G+ StS #	SA #	IBP #
128	100	>0.95	0	0	0	0	0
40		>0.50	0	1	0	0	0
2	w^*	>0.20	0	74	66	1	0
2	5.2	>0.01	0	25	34	90	0
		<0.01	100	0	0	9	100
128	100	>0.95	0	0	0	0	0
45		>0.50	0	0	0	0	0
2	w^*	>0.20	0	28	19	0	0
2	6.4	>0.01	0	72	81	55	0
		<0.01	100	0	0	45	100
128	100	>0.95	0	0	0	0	0
50		>0.50	0	0	0	0	0
2	w^*	>0.20	0	6	13	0	0
2	8.0	>0.01	0	94	87	12	0
		<0.01	100	0	0	88	100
128	100	>0.95	0	0	0	0	0
55		>0.50	0	0	0	0	0
2	w^*	>0.20	0	5	6	0	0
2	9.5	>0.01	0	79	82	2	0
		<0.01	100	16	12	98	100

Table 3: Noisy-OR MPE. Noise 0.2, Leak 0.01. Time 20 sec. 100 samples.

σ	Sol Prob Range	G #	G+ StS #	SA #	IBP #	MB #	G+ StS w/ MB
0.22	>0.95	126	195	56	200	182	193
	>0.50	0	0	0	0	0	0
	>0.20	0	0	0	0	0	0
	>0.01	2	0	0	0	2	0
	<0.01	72	5	144	0	16	7
0.28	>0.95	39	142	5	199	119	152
	>0.50	0	1	0	0	2	1
	>0.20	1	1	0	0	1	1
	>0.01	1	3	0	0	2	2
	<0.01	158	52	194	0	75	43
0.32	>0.95	14	99	1	192	86	114
	>0.50	0	1	0	0	0	0
	>0.20	1	6	0	0	4	2
	>0.01	0	6	0	0	7	5
	<0.01	180	83	194	3	97	74
0.40	>0.95	3	25	0	165	31	54
	>0.50	0	3	0	0	3	7
	>0.20	2	6	0	0	6	4
	>0.01	1	12	0	0	5	8
	<0.01	174	134	182	15	135	107
0.51	>0.95	0	3	0	41	3	5
	>0.50	0	2	0	0	1	2
	>0.20	0	2	0	0	1	3
	>0.01	0	6	0	0	4	16
	<0.01	94	81	94	53	85	68

Table 4: Random coding, N=100 K=50. SLS time 5 sec. 200 samples.

σ	G BER	G+ StS BER	SA BER	IBP BER	MB BER	G+ StS w/ MB BER
0.22	0.003	0.0009	0.015	0.0004	0.003	0.0018
0.28	0.012	0.0108	0.039	0.0006	0.019	0.0135
0.32	0.028	0.0269	0.062	0.0031	0.042	0.0355
0.40	0.090	0.0891	0.107	0.0159	0.100	0.0932
0.51	0.195	0.1976	0.163	0.0875	0.194	0.1921

Table 5: Random coding, N=100 K=50. SLS time 5 sec. 200 samples.

for each we generated a number of instances and ran all the algorithms on each of the problems. For each algorithm, we counted the number of assignments it output that fell in each probability range. For random coding networks, we generated a number of network instances for a fixed number of input bits K, and ran each algorithm a fixed number of times on every network instance, each time on a different random sequence of input bits.

In the following, we present 5 tables with results that are typical of the many experiments we ran. The first two tables contain results of experiments with uniform random networks. Table 3 contains results of experiments with noisy-OR networks. The fourth and fifth tables contain results on random coding networks, where Table 4 compares algorithms in terms of the probability of the solution, while Table 5 compares algorithms in terms of the Bit Error Rate.

In Tables 1 and 2, we report on specific combinations of parameters N, C, P and K, in each the number of conditional probability matrices C varies between sets of problems. Networks with a larger value of C have higher density and higher induced width w^*. We also report the number of problems solved by Elim-MPE, and the average w^* of the network. When successful Elim-MPE solved the problem quickly - often in less than 1 second.

We see that G outperformed StS considerably, and both were outperformed by G+StS. Simulated annealing was better than StS but inferior to G and G+StS. Finally, G+StS with mini-bucket assignment outperformed G+StS. In Table 2 we included the results of Iterative Belief Propagation (IBP.) Unlike in coding networks, the performance of IBP here was very poor.

Table 3 presents results of experiments with random noisy-OR networks. We did not include experiments involving the mini-bucket algorithm here because for these instances having small w^* the mini-bucket algorithm (with i-bound = 10) is complete. Both G and G+StS are superior here.

In Tables 4 and 5 we present results with random coding networks having 50 input bits. Table 4 compares the algorithms using probability ranges. Table 5 compares the algorithms using the Bit Error Rate. Each set of problems has a different level of channel noise. Pure stochastic algorithms (StS, G, G+StS, SA) always start from an assignment obtained by rounding the real-valued output bits to the closest integer. IBP computes a belief for each variable and then picks the most likely value. G+StS with mini-bucket starts from an assignment computed by the mini-bucket algorithm.

Surprisingly, when the channel noise is small, G+StS is able to beat the mini-bucket algorithm. When the noise is large, the mini-bucket algorithms outperform pure stochastic algorithms. When comparing algorithms MB and M+G+StS we see one of the advantages of the stochastic algorithms: they can start from any initial assignment and improve an already good assignment.

We note that the bucket elimination algorithm Elim-MPE failed to solve any of the random coding problems tested. This highlights a disadvantage of the bucket elimination algorithm. Although in the cases of random networks reported, Elim-MPE worked well (ran quickly), as we increased C, we would quickly come to a point where Elim-MPE would always run out of memory.

In coding networks (Tables 4 and 5) we were not always able to find an optimal solution. Therefore we reported only cases where the comparison with an optimal solution was possible (which might be less than the number of problems tried). In Table 5 we reported the average Bit Error Rate for all the problems attempted.

6 Summary and Conclusions

The paper compares several stochastic local search algorithms for solving the Most Probable Explanation in Bayesian networks. We focused on two variations of approximation algorithms, Stochastic Simulation, and Greedy algorithm and compared them to simulated annealing. Stochastic Simulation is a popular approximation algorithm used in Bayesian networks. Randomized greedy algorithms have been very successful recently in solving constraint satisfaction problems. Our test problems included randomly generated networks, noisy-or networks and random coding networks.

Based on these experiments we conclude that, although pure Stochastic Simulation has nice convergence guarantees, it is by far the worst scheme for the MPE task. The pure method that emerged as superior is the Greedy approach. However, the combined algorithm of stochastic simulation and greedy emerged as better than stochastic simulation or greedy alone.

We subsequently examined the power of stochastic local search for improving good solutions that were generated by alternative schemes. We found that when G+StS was applied on top of mini-bucket approximation it improved its solution in almost all cases, even though the mini-bucket solutions were already good.

For coding networks we applied all SLS methods from an initial assignment provided by rounding the output bits to the closest integer. With this heuristic the G+StS outperformed even G+SLS+MB when the noise level was low. Clearly, the superior algorithm for coding networks was IBP, as is known for this domain. Notice, however that IBP was very weak when applied to the random network.

Our current standing is that SLS methods for MPE should not start from a random initial assignment whenever possible. They can be either applied to a good assignment generated by an alternative approximation or use domain-dependent initial assignment.

References

[1] Dagum, P., Luby, M., 1993. Approximating Probabilistic Inference in Bayesian Belief Networks is NP-Hard, In *Proc. of AAAI-93*.

[2] Dagum, P., Luby, M., 1997. An Optimal Approximation Algorithm for Bayesian Inference, *Artificial Intelligence*, 93:1 - 27.

[3] Cheng, J.-F., 1997. Iterative Decoding, Ph.D. Thesis.

[4] Dechter, R., 1996. Bucket Elimination: A Unifying Framework For Probabilistic Inference, In *Proc. of UAI-96*.

[5] Kask, K., Dechter, R., 1995. GSAT and Local Consistency, In *Proc. of IJCAI-95*.

[6] Kask, K., Dechter, R., 1996. A Graph Based Method for Improving GSAT, In *Proc. of AAAI-96*.

[7] Li, Z., D'Ambrosio, B., 1993. An Efficient Approach for Finding the MPE in Belief Networks, In *UAI-93*, pp. 342-349.

[8] Minton, S., Johnson, M.D., Phillips, A.B., 1990. Solving Large Scale Constraint Satisfaction and Scheduling Problems Using a Heuristic Repair Method, In *Proc. of AAAI-90*.

[9] Pearl, J., 1988. Probabilistic Reasoning In Intelligent Systems, Morgan Kaufmann.

[10] Peng, Y., Reggia, J. A., 1986. Plausability of Diagnostic Hypothesis, In *Proc. of AAAI-86*.

[11] Peng, Y., Reggia, J. A., 1989. A Connectionist Model for Diagnostic Problem Solving, In *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 19, No. 2 March/April.

[12] Rish, I., Dechter, R., 1997. A Scheme For Approximating Probabilistic Inference, In *Proc. of UAI-97*.

[13] Dechter, R., 1997. Mini-Buckets: A general scheme for generating approximations in automated reasoning, In *IJCAI-97*, pp. 1297-1302.

[14] Rubinstein, R., 1981. Simulation and the Monte Carlo Method, John Wiley & Sons, New York.

[15] Santos, E., 1991. On the Generation of Alternative Explanations with Implications for Belief Revision, In *UAI-91*, pp. 339-347.

[16] Selman, B., Levesque, H., Mitchell, D., 1992. A New Method for Solving Hard Satisfiability Problems, In *Proc. of AAAI-92*.

[17] Shimony, S. E., Charniack, E., 1900. A New Algorithm for Finding MAP Assignments to Belief Networks, In *P. Bonissone, M. Henrion, L. Kanal, and J. Lemmer (Eds.), Uncertainty in Artificial Intelligence 6*, pp. 185-193. Elsevier Science Publishers B. V. (North Holland).

[18] Sy, B. K., 1992. Reasoning MPE to Multiply Connected Belief Networks Using Message-Passing, In *AAAI-92*, pp. 570-576.