ELSEVIER International Journal of Approximate Reasoning 27 (2001) 263–283

INTERNATIONAL JOURNAL OF APPROXIMATE REASONING

www.elsevier.com/locate/ijar

Partial abductive inference in Bayesian belief networks by simulated annealing

Luis M. de Campos [a], José A. Gámez [b,*], Serafín Moral [a]

[a] *Departamento de Ciencias de la Computación e I.A., Universidad de Granada, 18071 Granada, Spain*

[b] *Departamento de Informática, Escuela Politécnica Superior, Universidad de Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain*

Received 1 March 2000; accepted 1 April 2001

Abstract

Abductive inference in Bayesian belief networks (BBN) is intended as the process of generating the K most probable configurations given observed evidence. When we are only interested in a subset of the network variables, this problem is called partial abductive inference. Due to the noncommutative behaviour of the two operators (summation and maximum) involved in the computational process of solving partial abductive inference in BBNs, the process can be unfeasible by exact computation even for networks in which other types of probabilistic reasoning are not very complicated. This paper describes an approximate method to perform partial abductive inference in BBNs based on the simulated annealing (SA) algorithm. The algorithm can be applied to multiple connected networks and for any value of K. The evaluation function is based on clique tree propagation, and allow to evaluate neighbour configurations by means of local computations, in this way the efficiency with respect to previous algorithms (based on the use of genetic algorithms (GAs)) is improved.

Keywords: Simulated annealing; Bayesian networks; Abductive reasoning; Maximum a posteriori hypothesis; Probabilistic reasoning

* Corresponding author.

E-mail addresses: lci@decsai.ugr.es (L.M. de Campos), jgamez@info-ab.uclm.es (J.A. Gámez), smc@decsai.ugr.es (S. Moral).

0888-613X/01/$ - see front matter
PII: S0888-613X(01)00043-3

1. Introduction

Bayesian belief networks (BBNs) [23,15] are frequently used as the kernel of *Probabilistic Expert Systems*, because they provide an efficient representation of the joint probability distribution, and allow us to calculate probabilities by means of *local* computation, i.e., only relevant information is considered when a probability has to be calculated.

Although the most commonly used type of inference in BBNs is *probabilities* (*or evidence*) *propagation*, in this paper we are interested in another type of inference, known as *abductive reasoning* (also known as *diagnostic reasoning* because it is in the field of diagnostic where abductive reasoning has its most clear application [12,24,25]).

Abduction is defined as the process of generating a plausible explanation for a given set of observations or facts [26], and in the context of probabilistic reasoning, abductive inference corresponds to finding the maximum a posteriori (MAP) probability state of the system's variables, given some *evidence* (observed variables) [22]. The increasing attention received by abductive inference in BBNs over the last decade has yielded as a result the development of both exact [4,9,21,22,28,31] and approximate solution methods [10,11,19,27,34]. However, only some of these algorithms [11,21,27,28,34] do not have any limitation on the structure of the network and are able of searching for the K most probable configurations instead of only the best one.

When we are interested in obtaining the K most probable configurations only for a subset of the network variables called *explanation set* [20], the problem is known as the *Partial Abductive Inference* or *MAP hypothesis*. Although this problem seems to be more useful in practical applications (because we can select the relevant [1] variables as the explanation set) than *total* abductive inference it has received much less attention.

Our goal in this paper is to approach the *partial abductive inference problem* in BBNs by means of a search algorithm based on the well known *simulated annealing* (SA) scheme. The rest of the paper is organised as follows: In Section 2 we briefly recall some concepts about BBNs and abductive inference. In Section 3, the SA technique is described. In Section 4 we present our algorithm. In Section 5 the experiments carried out are described, and finally, in Section 6, we consider the conclusions.

[1] In this paper we suppose that the explanation set is known (provided by the user, by another algorithm, etc.) Determining which variables must be included in an explanation is a related problem [3,29,30].

2. Bayesian belief networks and abductive inference

A *Bayesian belief network* [15,23] is a directed acyclic graph (DAG) where each node represents a random variable, and the topology of the graph shows the (in)dependence relations among the variables. The quantitative part of the model is given by attaching to each variable X_i a probability distribution $P(X_i|pa(X_i))$, where $pa(X_i)$ contains the parents of X_i in the graph. If $X_{\mathscr{U}} = \{X_1, \ldots, X_n\}$ is the set of variables in the network, then the joint probability can be calculated as:

$$P(X_{\mathscr{U}}) = \prod_{X_i \in X_{\mathscr{U}}} P(X_i|pa(X_i)). \tag{1}$$

Eq. (1) is known as the *chain-rule*.

Abductive inference in BBNs, also known as *the most probable explanation problem* (MPE), corresponds to finding the MAP probability state of the network, given the observed variables (the evidence). In a more formal way: if X_{O} is the set of observed variables and X_{U} is the set of unobserved variables, we aim to obtain the configuration x_{U}^* of X_{U} such that:

$$x_{\mathrm{U}}^* = \arg\max_{x_{\mathrm{U}}} P(x_{\mathrm{U}}|x_{\mathrm{O}}), \tag{2}$$

where $X_{\mathrm{O}} = x_{\mathrm{O}}$ is the observed evidence. Usually, x_{U}^* is known as the MPE. It is well known that the MPE can be found using probabilities propagation methods but replacing summation by maximum in the marginalisation operator (due to the distributive property of maximum with respect to multiplication) [4]. Therefore, the process of searching for the MPE has the same complexity as probabilities propagation. However, in order to search for the K MPEs more complex methods have to be used; for example, in [21] clique tree propagation is combined with a divide and conquer algorithm that iteratively identifies the K MPEs.

In *partial* abductive inference, if we denote the explanation set by $X_{\mathrm{E}} \subset X_{\mathrm{U}}$, then the goal is to obtain the configuration x_{E}^* of X_{E} such that:

$$x_{\mathrm{E}}^* = \arg\max_{x_{\mathrm{E}}} P(x_{\mathrm{E}}|x_{\mathrm{O}}) = \arg\max_{x_{\mathrm{E}}} \sum_{x_{\mathrm{R}}} P(x_{\mathrm{E}}, x_{\mathrm{R}}|x_{\mathrm{O}}), \tag{3}$$

where $X_{\mathrm{R}} = X_{\mathrm{U}} \setminus X_{\mathrm{E}}$. In general, x_{E}^* is not equal to the projection of the configuration x_{U}^* over X_{E}, so we need to obtain x_{E}^* directly (Eq. (3)).

The process of finding the configuration x_{E}^* is more complex than that of finding x_{U}^*, because not all clique tree obtained from the original BBN is valid. In fact, because summation and maximum have to be used simultaneously and these operations do not show a commutative behaviour, the variables of X_{E} must form a sub-tree of the complete tree (we will refer to this kind of trees as *valid* clique trees). Xu [33] gives a method for transforming the initial tree into another one containing a clique in which the variables of X_{E} are included. The

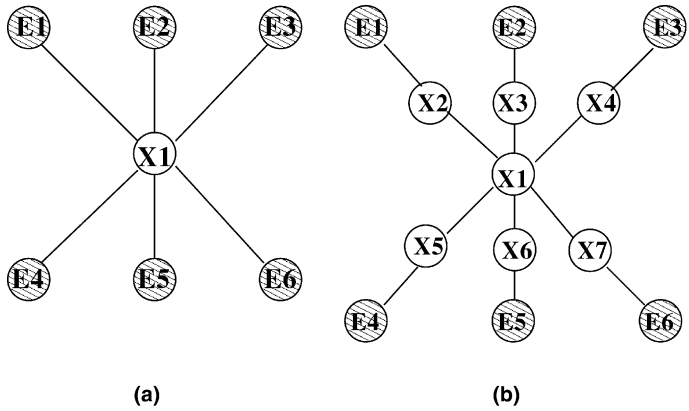

Fig. 1. Two moral graphs.

problem of this method is that if X_E contains many variables, then the problem could be intractable due to the size of the probability table (potential) associated with the clique containing X_E. Nilsson [21] outlines how to slightly modify Xu's algorithm in order to allow (when possible) the variables of X_E to constitute a sub-tree and not a single clique. An alternative approach [6] is to obtain a valid clique tree directly, by using constrained deletion sequences in the triangulation step. However, trying to solve the problem of partial abductive inference by using these methods can be impossible (or too expensive in time and space) even for networks in which probabilities propagation or the *total* abductive inference problem can be solved efficiently. To illustrate this situation we give the following example:

Example 2.1. Let us consider the moral graph (or interpret it as a fragment of a bigger one) shown in Fig. 1(a). This graph is already triangulated and we can see that it contains six cliques: [2] $\{X_1, E_1\}, \{X_1, E_2\}, \{X_1, E_3\}, \{X_1, E_4\}, \{X_1, E_5\}$, and $\{X_1, E_6\}$.

This set of cliques can be structured in several ways. Thus, Figs. 2(a) and (b) show two possibilities. If we consider that all the variables can take five different states, then the probability table (potential) associated with every clique has 25 entries. We will refer to this data as the clique *size*, that is, for us the size of a clique is the size of its associate potential (size of state space representation). The size of a clique tree is obtained by summing the size of all the cliques it contains. In this case, the size of the clique tree is 150. If the number of cases for each variable were 10, then the size of each clique would be 100 and the size

[2] A *clique* is a maximal complete sub-graph.

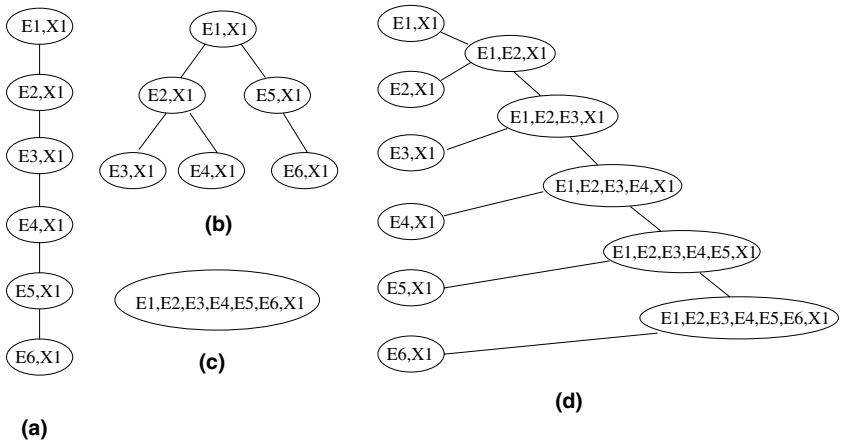

Fig. 2. Different clique trees.

of the whole tree would be 600. Therefore, the propagation over these clique trees will be very efficient, because they have a small size (150 or 600).

Now, let us consider a partial abductive inference problem over the same graph by taking $X_E = \{E_1, \ldots, E_6\}$ as the explanation set. Below we examine the effect of applying the exact techniques cited above to this problem:

- Methods based on the variable elimination algorithm [9,18] or methods based on a direct search of a *valid* clique tree [6]. In both methods the ordering in which the variables are going to be eliminated must contain the variables not in X_E in the first positions, so, in this case the first variable to remove is X_1. To eliminate variable X_1 the first step is the combination of all the probability tables containing X_1. Thus, a probability table over $\{X_1, E_1, E_2, E_3, E_4, E_5, E_6\}$ is built. The size of this probability table will be 78,125 if 5 states per variable are considered or 10,000,000 if 10 states per variable are considered (Fig. 2(c)).
- Methods based on the modification of the clique tree [21,33]. Fig. 2(d) shows the effect yielded by the application of Xu's method. As we can see, a clique containing the variables $\{X_1, E_1, E_2, E_3, E_4, E_5, E_6\}$ is included in the tree. The size of this clique is 78,125 or 10,000,000 depending on the number of different states per variable (5 or 10).

In this case the modification introduced by Nilsson does not have any effect over the obtained tree. To give an idea of the difference, let us suppose that a new variable E_7 is introduced in the explanation set, and that variable is connected only with E_6 in the moral graph. Then, the method proposed by Xu will introduce a clique $\{X_1, E_1, E_2, E_3, E_4, E_5, E_6, E_7\}$ into the clique tree, while the modification proposed by Nilsson will introduce the following sub-tree:

$$(X_1, E_1, E_2, E_3, E_4, E_5, E_6) - (E_6, E_7),$$

which clearly has a smaller size.

The reader can check that the same situation is obtained when the moral graph in Fig. 1(b) is considered.

With the previous example we have shown that the problem of partial abductive inference can be intractable even for networks in which probabilities propagation or total abductive inference can be solved efficiently. The following example tries to deepen in this situation by showing the effect of the explanation set cardinality in the size of the obtained *valid* clique tree.

Example 2.2. To illustrate the increment of the clique tree size with respect to the number of variables in the explanation set ($|X_E|$), we have carried out the following experiment: we randomly generated explanation sets containing 1, 7, 13, 19, 25, 31, and 37 variables for the *Alarm* network [1] (which has 37 variables). Then, valid clique trees were obtained for every generated explanation set. The results are shown in Table 1, where:

- the first row (#nodes) shows the number of nodes in the explanation set,
- the second row (CT size) shows the size of the obtained *valid* clique tree (averaged over 100 explanation sets), and
- the third row (LC size) shows the size of the largest clique in the tree (also by average).

Therefore, for this network exact partial abductive inference may require to deal with a clique tree of size 10e5 or 10e6 approximately, while exact total abductive inference or probabilities propagation requires to deal with a clique tree of size 10e3 approximately. If the size of the largest clique in the tree is considered, then it passes from 10e2 to values close to 10e6.

We can see, then, that given the increment in the clique tree size, the use of approximate methods for partial abductive inference in BBNs is even more necessary than for total abductive inference.

In the literature we can find approximate algorithms based on the reduction of the probability tables size involved in computations, as in [10], where it is shown how the mini-buckets scheme can be applied to the search of the best explanation, although this method only search for the best explanation and not

Table 1
Sizes respect to $|X_E|$

# Nodes	1	7	13	19	25	31	37
CT size	1.044e3	8.094e3	2.404e5	8.531e5	1.282e6	5.816e4	1.014e3
LC size	1.094e2	4.478e3	2.273e5	8.303e5	9.065e5	5.376e4	1.08e2

for the K best. Another approach is based on the use of optimisation meta-heuristics, as the genetic algorithm (GA) designed for searching the best K configurations presented in [5].

The exercise of dealing with partial abductive inference using optimisation methods could seem easier than that of dealing with total abductive inference, because the size of the search space in the partial case is considerably smaller than in the total abductive inference problem. However, this is not the case because of the increasing complexity of the evaluation (*fitness*) function. In fact, as $P(x_U|x_O)$ is proportional to $P(x_U, x_O)$, this value can be used to rank the different configurations, and so in the total case the *chain rule* can be applied in order to evaluate a configuration [11,27,34]. Thus, in the total case, to evaluate a configuration $|X_{\mathscr{U}}|$ multiplications are carried out.

Nevertheless, in the partial case, as we must remove (by addition) the variables in X_R, the chain rule cannot be used to evaluate a configuration x_E because of the large number of necessary operations (additions and multiplications). For example, if we have a network with $|X_{\mathscr{U}}| = 50, |X_E| = 15$, $|X_R| = 30$ and $|X_O| = 5$, the number of operations is bounded by 2^{30} additions and 50×2^{30} multiplications. Clearly, this is computationally intractable given the large number of evaluated individuals in this kind of algorithms. Because of this, in [5] the fitness $P(x_E, x_O)$ of a configuration x_E is computed by the process described in Algorithm 1, where $\mathscr{T} = \{C_0, C_1, \ldots, C_t\}$ is a rooted clique tree, being C_0 the root.

Algorithm 1 (*Evaluation function*).
1. Enter the evidence x_O in $\mathscr{T}$,
2. Enter (as evidence) the configuration x_E in $\mathscr{T}$,
3. Perform *CollectEvidence* from the root (C_0) (i.e., an upward propagation), and
4. $P(x_E, x_O)$ is equal to the sum of the potential stored in the root (C_0).

Therefore, to evaluate a configuration an exact propagation is carried out, or more correctly half propagation, because only the *upward* phase is performed and not the *downward* one (see [15] for details about clique tree propagation). Furthermore, for this propagation we can use a clique tree obtained without constraints and so its size is much smaller than the clique tree used for exact partial abductive inference. In addition, in [5] it is shown how the tree can be pruned with respect to the explanation set, reducing (on the average) its size about 20% in our experiments. The following paragraph outlines the prunning process.

Because the propagation is performed in a bottom-up way, it can be observed that independently of the configuration being evaluated some computations are always the same. Concretely, if a leaf node in the tree does not contain any variable belonging to X_E in its residual set (variables to be removed

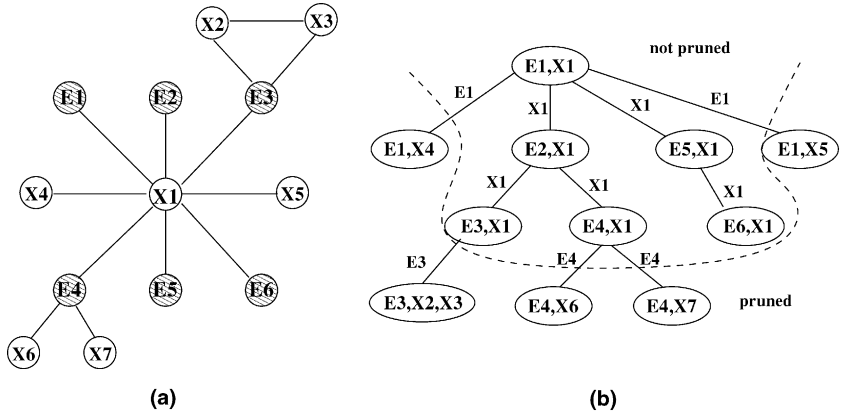

Fig. 3. An example of the prune operator.

by addition), then the message sent from this node is the same for every configuration x_E of X_E. This kind of message is calculated only the first time, then it is combined with the potential stored in its parent clique and the leaf node is deleted. The process is repeated (from leaves to root) until no more "leaf" nodes can be deleted. So, the complexity of the precomputation process is equivalent to perform an upward propagation.

The following example shows the effect of the prune operation.

Example 2.3. Let us consider the moral graph in Fig. 3(a), which is obtained by adding $\{X_2, \ldots, X_7\}$ to the moral graph considered in Fig. 1(a). A clique tree for this graph is shown in Fig. 3(b), where the links are labelled with the separator set.

Now, if we consider $\{E_1, \ldots, E_6\}$ as explanation set, then we can prune all the cliques below the dashed line. Therefore, the tree used to evaluate configurations has a smaller size.

Given that propagation is a time-expensive process, it would be desirable to avoid the need of performing a whole upward propagation each time a distinct [3] individual has to be evaluated. Following this idea, in [7] the authors developed specific genetic operators that allow us to take advantage of the calculations previously carried out when a new individual is being evaluated. Thus, in the mutation operator only the computations involving ancestor cliques of the one containing the mutated variable(s) have to be carried out, while the rest of computations can be retrieved from previous evaluations. In the crossover operator the situation is similar: as the parents interchange the genes contained in a sub-tree of the clique tree, then if C is the root of the inter-

[3] A hash table is used in order to store the fitness of individuals previously evaluated, so if an individual is re-sampled by the GA its fitness can be retrieved efficiently.

changed sub-tree, only the computations involving ancestor cliques of C have to be carried out. By using these specific operators the need of performing a whole upward propagation is avoided, although more memory is needed because the messages sent during the propagation of the population individuals are stored (details can be found in [7]).

In this paper we will try to show that if we change the metaheuristic from GAs to SA [16,32], then it is possible to evaluate a new configuration by carrying out only the computations that involve one clique and its neighbours. Therefore, by using this way to evaluate an individual the calculations are performed more locally than in the improved GA [7].

3. Simulated annealing

Simulated annealing [16,32] is an optimisation technique to solve (NP-hard) combinatorial optimisation problems, inspired in the physical annealing process of solids, and has been successfully applied to solve other NP-hard problems in Bayesian networks, as decomposition of BBN [17], convex set of probabilities propagation [2], or causal orderings approximation [8].

The problem to solve is the optimisation of a cost function $\mathrm{cost}(x)$ in a search space W defined for a set of random variables N. The way to proceed is similar to a *hill climbing* algorithm, but some times the algorithm accepts cost-function increases, in order to avoid to be trapped at local optima. The probability of accepting cost-function increases is controlled by a parameter t, called *temperature*. Initially, the temperature is *high* and cost-increases are accepted easily, but in successive iterations the temperature is decreased according to a cooling procedure, and the probability of accepting cost-increases also decreases. Concretely, if we are minimising $\mathrm{cost}(x)$ the algorithm changes from configuration x to another configuration x' with probability $\mathrm{e}^{-(\mathrm{cost}(x')-\mathrm{cost}(x))/t}$. If $f(t)$ is the cooling function and $N(x)$ represents the neighbourhood of x, i.e., configurations obtained by applying a small perturbation to x, then the structure of the simulated annealing algorithm is shown below.

Algorithm 2 (*Structure of SA*).

1. Set the initial temperature t.
2. Set the number of iterations n for each value of t.
3. While not (stopping condition) do
   (a) For $i = 1$ to n do
      (i) Select $x' \in N(x)$
      (ii) if $\mathrm{cost}(x') \leqslant \mathrm{cost}(x)$, then accept x'
         else accept x' with prob. $\mathrm{e}^{-(\mathrm{cost}(x')-\mathrm{cost}(x))/t}$
   (b) $t = f(t)$
4. Return x as solution.

Under theoretical conditions the algorithm converges to the global optimum [13], but this *ideal annealing* has an excessive time complexity. Therefore, in order to get more efficient implementations some convergence conditions are relaxed. Thus, our cooling procedure will be based on the simple scheme introduced in [16], that is, a geometric decreasing of the temperature according to the formula: $t_{i+1} = \alpha \cdot t_i$, with $\alpha \in [0.8, 1)$.

4. The proposed simulated annealing algorithm

It is clear that we can implement a SA program for solving our problem by using the structure described in Algorithm 2 (modifying the acceptance criterion to $e^{-(\mathrm{cost}(x)-\mathrm{cost}(x'))/t}$ given that we want to maximise $P(x_E, x_O)$), and the evaluation function in Algorithm 1. However, our goal is to take advantage of the SA structure in order to evaluate the configurations by means of local computations.

4.1. Local computation of neighbour configurations

Consider the rooted clique tree in Fig. 4(a), and the explanation set $X_E = \{A, C, E, G, H\}$. Variables inside brackets represent the *separator* of a clique with its clique parent (intersection), and variables outside brackets are the *residual* set. Variables in the explanation set have been underlined for a clearer identification. Fig. 4 shows the state (messages) of the tree after the evaluation

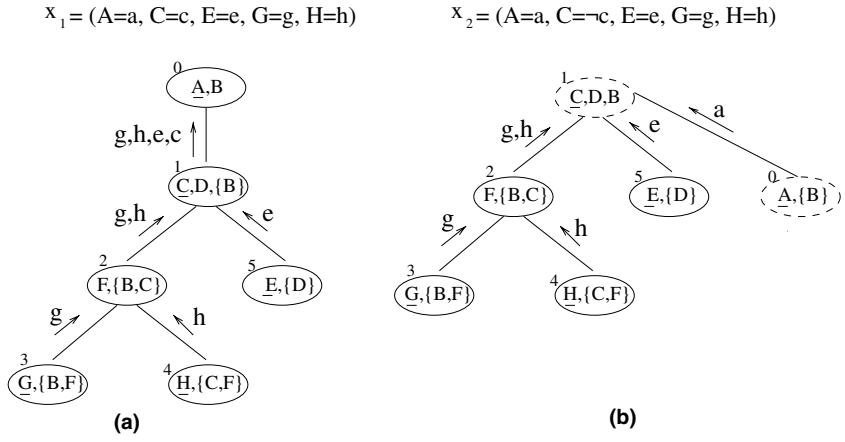

Fig. 4. Local computation of a neighbour configuration.

of configuration $x_1 = (\mathtt{a}, \mathtt{c}, \mathtt{e}, \mathtt{g}, \mathtt{h})$ using the procedure described in Algorithm 1, where the message labels represent the information contained in the message related to the configuration being evaluated.

As we know, $N(x)$ is the set of configurations that can be obtained from x by means of a small perturbation. To make possible the evaluation of x neighbours by local computation, we define $N(x)$ as the set of configurations that can be obtained from x by modifying the state of one variable contained in an adjacent clique to the one actually being considered as root (or more precisely in a clique such that there is no clique containing an explanation variable in the path connecting it to the root). In the example, the actual root is C_0 and the only clique connected with it is C_1, so $N(x_1)$ contains only the configurations that differ from x_1 only in the state taken by $\mathtt{C}$. If we suppose that all the variables can take two states, then $N(x_1) = \{(\mathtt{a}, \neg\mathtt{c}, \mathtt{e}, \mathtt{g}, \mathtt{h})\}$. Fig. 4(b) shows that considering C_1 as the new root, then to evaluate this new configuration only the message from the old root to the new one has to be calculated, because the rest of the messages can be reused. So, only two cliques (the new and the old root) are involved in new computations. The process is:

1. Compute the message $M^{C_{\mathrm{or}} \to C_{\mathrm{nr}}}$, where C_{or} denotes the *old root* and C_{nr} denotes the *new root*.
2. Combine the potential stored in C_{nr} with $M^{C_{\mathrm{or}} \to C_{\mathrm{nr}}}$.
3. Choose a configuration of $N(x)$ and evaluate it by addition in C_{nr}.
4. Select a neighbour of the actual root as new root and continue the process.

Some important remarks have to be done at this point:

R1. To avoid the need of reloading (or recalculating) the initial clique potentials, the computations are performed without modifying them, that is:

$$M^{C_{\mathrm{or}} \to C_{\mathrm{nr}}} = \left(\psi(C_{\mathrm{or}}) \otimes \left\{\bigotimes_{C_k \in \mathrm{adj}(C_{\mathrm{or}}) \setminus C_{\mathrm{nr}}} M^{C_k \to C_{\mathrm{or}}}\right\}\right)^{\downarrow C_{\mathrm{or}} \cap C_{\mathrm{nr}}},$$

$$P(x_{\mathrm{E}}, x_{\mathrm{O}}) = \left(\psi(C_{\mathrm{nr}}) \otimes \left\{\bigotimes_{C_k \in \mathrm{adj}(C_{\mathrm{nr}})} M^{C_k \to C_{\mathrm{nr}}}\right\}\right)^{\downarrow \emptyset},$$

where $\mathrm{adj}(C_i)$ is the set of adjacent cliques to C_i, $\otimes$ denotes *combination* (pointwise multiplication), and $\downarrow A$ denotes marginalisation (summation over variables not in A).

R2. For the same reason as in the previous point, the configuration being evaluated cannot be entered in the clique potentials. To avoid this difficulty we modify the marginalisation operator in the following way:

$$\psi(X,Y)^{\downarrow_{\mathtt{c}\{Z=z\}} Y} = \begin{cases} \sum_X \psi(X,Y) & \text{if } X \cap Z = \emptyset, \\ \sum_{x : x^{\downarrow(X \cap Z)} = z^{\downarrow(X \cap Z)}} \psi(X,Y), & \text{if } X \cap Z \neq \emptyset, \end{cases}$$

where $x^{\downarrow A}$ denotes the configuration obtained from x by removing the literals not in A (notice the double use of the $\downarrow$ operator, projection for configurations and marginalisation for potentials). That is, if we use $\downarrow_{c\{X_E=x_E\}}$ with x_E the configuration actually being evaluated, we are working *as if the configuration x had been entered in the clique tree*. Keeping this in mind and for the sake of simplicity we will abbreviate $\downarrow_{c\{X_E=x_E\}}$ by $\downarrow_c$.

R3. If the next root is previously known, then the computations in R1 can be simplified. Note that a great deal of the computations carried out during the calculation of $P(x_E, x_O)$ in C_{nr}, are repeated in the calculation of the message sent from C_{nr} to C_{fr} (the future root), as we can see in the following expression:

$$M^{C_{nr}\to C_{fr}} = \left(\psi(C_{nr}) \otimes \left\{\bigotimes_{C_k\in \mathrm{adj}(C_{nr})\setminus C_{fr}} M^{C_k\to C_{nr}}\right\}\right)^{\downarrow_c(C_{nr}\cap C_{fr})}.$$

However, if we know the future root when computing $P(x_E, x_O)$, then these repetitions can be avoided by using a temporal potential $\psi(T)$, and structuring the computations in the following way:

$$\begin{aligned}
&\psi(T) = \left(\psi(C_{nr}) \otimes \left\{\bigotimes_{C_k\in \mathrm{adj}(C_{nr})\setminus C_{fr}} M^{C_k\to C_{nr}}\right\}\right),\\
&P(x_E, x_O) = \left(\psi(T) \otimes M^{C_{fr}\to C_{nr}}\right)^{\downarrow_c C_{nr}},\\
&M^{C_{nr}\to C_{fr}} = \left(\psi(T)\right)^{\downarrow_c C_{nr}\cap C_{fr}}.
\end{aligned}$$

In order to know the next root in advance, the solution adopted has been the use of a sequence of cliques (σ) to drive the iterative process. The sequence must verify the following conditions:

(i) $\sigma(0)$ is the clique used as root in the initial topology of the rooted clique tree (C_0). This is necessary because of the (bottom-up) evaluation of the first configuration.

(ii) $\forall j, 0 \leqslant j < |\sigma|$, it holds that $\sigma(j)$ and $\sigma((j+1) \bmod |\sigma|)$ are adjacent cliques in the tree.

(iii) All the cliques in the tree are in σ (some of them more than once in order to guarantee the previous condition).

Therefore, the algorithm uses an index that iteratively goes through this sequence to select the root cliques. So, the next root is always known. This sequence is computed in runtime just before starting the SA iterations.

An additional consequence of using a sequence σ to drive the iterative process is that it avoids the possibility that the process gets stuck in a clique. In fact, by using σ we are sure that all the cliques will be visited in a near future, and so the probability of every configuration to be visited is greater than zero, which is a required condition by SA convergence properties.

Example 4.1. Let us consider the rooted clique tree shown in Fig. 3(b) in its pruned version. Let us also identify each clique by the index of the explanation variable contained by it (that is, C_1 for $\{E_1, X_1\}$, C_2 for $\{E_1, X_2\}$, etc.). Then, σ can be computed as $(C_1, C_2, C_3, C_2, C_4, C_2, C_1, C_5, C_6, C_5)$.

R4. After precomputing the clique tree, we are sure that all the leaves contain at least one variable belonging to the explanation set. However, this may not be the case for all the inner cliques. When one of these cliques is selected as root, the process is exactly the same as for the rest, except that the current configuration does not change.

R5. As we are interested in the K best configurations and not only in the best one, when a new configuration is evaluated, it is studied whether it must be included in K *best* (an array containing the K best configurations found until this moment). When the algorithm finishes, if $X_O \neq \emptyset$ the probability associated to the configurations in K best is divided by $P(x_O)$ in order to get $P(x_E|x_O)$.

R6. A modification that can improve the efficiency of the SA algorithm was proposed by Greene and Supowit [14]. According to it, when $|N(x)|$ is *small*, and so we can calculate the cost of all its members, then instead of selecting a configuration of $N(x)$ randomly, it is better to choose a configuration, $x' \in \{N(x) \cup \{x\}\}$, with probability proportional to $e^{-(\text{cost}(x'))/t}$. In our case, if we restrict $N(x)$ to $N'(x)$, with $N'(x)$ being the set of configurations x', such that, only the state of variables in the actual root changes with respect to x, $|N'(x)|$ is small. Furthermore, given that $\forall x' \in N'(x), \text{cost}(x')$ can be computed from the potential stored in the actual root by using $\downarrow_c$, this modification can be used in our algorithm.

As we search for the K MPEs and not only for the best one, the implementation of the modification proposed by Greene and Supowit could be specially interesting for us, because it allows us to explore the neighbourhood of good configurations.

On the other hand, this modification makes more probable the re-sampling of some configurations. In order to avoid re-evaluating the same configurations over and over again, we use a hash table in which pairs ⟨configuration, probability⟩ are stored, so, when a previously visited configuration has to be evaluated its probability is (efficiently) retrieved from the hash table.

4.2. About the cooling procedure

As we have said in Section 3 our cooling procedure will be based on the expression $t_i = \alpha \cdot t_{i-1}$. So, two parameters have to be selected: the cooling factor (α), and the initial temperature (t_0). In general, the selection of α does not represent a difficult problem, because it is known that values in the interval [0.9, 0.99] are good choices. However, we have detected some problems related with t_0.

Theoretically, the initial temperature, t_0 should be set to some large value, such that the probability to accept a worse configuration is close to 1. However, very often it is enough to take an initial temperature such that worse configurations are accepted with high probability. In related problems [2,8]

usual values for t_0 are 0.5, 1.0, 2.0, ..., and so values in these ranges were our first choice. However, the results were not good and after a preliminary study we discovered that under these conditions the algorithm needs too many iterations before the search is focused (that is, before the probability of accepting *bad* moves is small).

In fact, as we use $P(x_E, x_O)$ as cost function, $P(x_O)$ can be used as an upper bound for the value taken by a configuration x_E. In general, this upper bound can be a very small value, because very often the evidence represents states rarely taken by the observed variables. So if we set t_0 to values as 1.0, 2.0, ..., the number of iterations to carry out until t is close to the values taken by the cost function is too high. To dismiss these large number of iterations we set t_0 to $P(x_O)$, but the result did not improve very much because the large distance between $P(x_O)$ and $P(x_E, x_O)$ – for the majority of configurations x_E – made the number of iterations to perform very large. Because of this we normalised the cost function using the actual configuration, that is, if x is the actual configuration then we use $e^{(C(x')/C(x))/t}$ instead of $e^{C(x')/t}$ to calculate the probability of accepting x' as the new configuration. With this change and using an initial temperature t_0 that allows us to accept worse configurations (than the actual) with a large probability ($t_0 = 1.0, 2.0, \ldots$), the accuracy and efficiency of the algorithm has been improved significantly.

5. Experimental evaluation

In order to evaluate our algorithm we have performed seven experiments. For the first six experiments we have used the Alarm belief network [1], and two artificially generated networks (*artificial3* and *artificial5*). Some of these networks have been used to test the genetic algorithms proposed in [5,7]. The Alarm network has 37 variables which can take 2, 3 or 4 different states. Both artificial networks have 25 variables, each one taking three different states in *artificial3* and 5 in *artificial5*.

We have carried out two experiments over each network. In the two experiments over the Alarm network, four variables were selected (randomly) as evidence. In the experiments over the networks *artificial3* and *artificial5* three variables ($\{X_{22}, X_{23}, X_{24}\}$) were selected as evidence. In all the experiments the observed states were selected in a random way. The $K = 50$ MPEs were calculated using (1) an exact algorithm – giving it enough time and memory –, (2) the improved GA presented in [7], and (3) our SA algorithm. Table 2 shows some data about the experiments.

At the end of each experiment several statistics were obtained.

- top1. % of runs in which the best MPE was obtained.
- top10. % (in average) of MPEs found between the second and the tenth. For example, top10 = 33.3% means that three explanations in the range 2–10

Table 2
Description of the experiments

# Exp.	$\lvert X_E \rvert$	Network	X_E	$\lvert \Omega_{X_E} \rvert$
1	12	alarm	Roots	9216
2	12	alarm	Randomly	248,832
3	11	artificial3	Even index	177,147
4	11	artificial3	Odd index	177,147
5	11	artificial5	Even index	48,828,125
6	11	artificial5	Odd index	48,828,125

were found. However, the positions are not considered, so (2,3,6) or (7,8,9) yield the same value for top10.

- top25. Similar to top10 but with the range 11–25.
- top50. Similar to top10 but with the range 26–50.
- mass1. If p_1 is the probability of the real true MPE, and $\hat{p}_1$ is the probability of the best MPE found by the algorithm, then mass1 $= \hat{p}_1 \cdot 100/p_1$.
- massT. Similar to mass1, but taking

$$p_k = \sum_{i=1,\ldots,K} p_i \quad \text{and} \quad \hat{p}_k = \sum_{i=1,\ldots,K} \hat{p}_i.$$

Respect to the stopping condition, the algorithm is run $N = 50$ iterations in the outer cycle and $n \cdot |\sigma|$ in the inner one. In the experiments, n was set to 2 (exp. 1–4), and 8 (exp. 5 and 6). The value of α depends on the number of iterations in order to ensure that the algorithm finishes with a small value of t. In our case we have selected $t_f = 0.1$. When $t = t_f$ the probability of accepting a configuration 10 times worse than the actual is lower than $5e-5$. In order to be sure that the algorithm ends with $t = t_f$, we have calculated α as $(t_f/t_0)^{1/N}$.

Tables 3–8 show the results obtained by the SA algorithm averaged over 100 runs. Table 9 shows the results obtained by the GA.

From these results we can see that the algorithm has a high accuracy, obtaining similar results to the GA. In fact, some improvement can be observed over the GAs, but not by an overwhelming margin. From Tables 3–8 we can see that the algorithm finds the MPE in almost all the runs, finding

Table 3
Results for the experiment 1

N	top1	top10	top25	top50	mass1	massT
10	50.00	37.11	41.73	25.24	79.89	43.62
20	79.00	63.89	64.93	45.44	96.46	70.42
30	93.00	83.00	85.07	62.64	99.72	87.90
40	100.00	92.33	94.67	75.96	100.00	95.44
50	100.00	97.78	98.53	84.88	100.00	98.47

Table 4
Results for the experiment 2

N	top1	top10	top25	top50	mass1	massT
10	66.00	50.67	39.40	34.68	86.75	55.17
20	98.00	79.78	68.67	60.04	99.30	82.77
30	100.00	94.78	86.60	79.84	100.00	94.54
40	100.00	98.78	92.53	87.44	100.00	97.57
50	100.00	99.33	93.27	88.68	100.00	97.97

Table 5
Results for the experiment 3

N	top1	top10	top25	top50	mass1	massT
10	13.00	18.22	16.87	18.88	74.94	53.01
20	29.00	35.44	32.07	35.16	87.27	72.10
30	59.00	56.11	49.07	53.72	94.33	84.43
40	81.00	74.56	66.27	68.08	97.60	91.64
50	83.00	79.33	71.13	72.40	97.62	93.18

Table 6
Results for the experiment 4

N	top1	top10	top25	top50	mass1	massT
10	29.00	27.22	28.73	27.00	82.68	61.93
20	64.00	52.44	50.27	49.96	94.30	79.57
30	84.00	76.22	74.80	70.24	98.61	91.29
40	97.00	93.33	90.53	83.36	99.80	97.31
50	98.00	95.67	92.93	87.12	99.87	98.18

Table 7
Results for the experiment 5

N	top1	top10	top25	top50	mass1	massT
10	14.00	9.89	9.67	6.44	47.87	26.43
20	43.00	25.11	23.07	17.12	73.29	47.40
30	81.00	56.56	51.20	37.16	92.93	73.54
40	95.00	85.44	80.40	62.96	98.37	90.72
50	96.00	89.33	85.53	68.24	98.76	92.94

Table 8
Results for the experiment 6

N	top1	top10	top25	top50	mass1	massT
10	14.00	12.56	6.20	6.48	64.64	22.42
20	32.00	29.78	15.87	15.16	81.41	41.49
30	75.00	64.67	38.93	37.24	94.65	69.74
40	98.00	94.00	77.93	72.40	99.59	93.09
50	99.00	96.00	83.80	78.72	99.80	95.13

Table 9
Results obtained by genetic algorithms

# Exp.	top1	top10	top25	top50	mass1	massT
1	100.00	99.78	88.80	81.48	100.00	97.32
2	98.00	98.33	93.07	92.64	99.30	97.21
3	95.00	85.78	73.73	76.24	98.78	95.04
4	95.00	90.33	78.67	75.80	99.08	95.02
5	100.00	97.78	93.53	76.96	100.00	97.13
6	100.00	100.00	99.93	98.76	100.00	99.95

another high quality solution in the rest (see column mass1). Furthermore, the algorithm is able to find the majority of the K best explanations (columns top10, top25, top50 and massT). With respect to the efficiency of the proposed algorithm, it is more efficient in memory than the GA presented in [5] and much more efficient than the GA presented in [7] (because this algorithm needs to store the messages – potentials – sent during the evaluation of each individual actually included in the population). Time requirements are heavily dependent on the parameters N and n for SA, and number of generations and population size for GAs. For the results presented here the proposed SA algorithm needs (rougly speaking) between the 25% and the 35% less time than the optimised GA proposed in [7], and less than half the time than the GA presented in [5].

Once we have analysed the results and the accuracy of the SA algorithm has been tested in problems for which exact results were also calculated (requiring lot of time and space in some of them), we have carried out another experiment. An artificial network that we have called *extreme100* has been artificially generated. The network has 100 variables which can take two different states. The probabilities have been generated as follows: two random uniform numbers, x and y are generated, and the probability of the two values (marginals for root nodes and conditionals for the rest) of a variable are determined by normalising x^5 and y^5, which give rise to extreme probabilities.

Table 10 shows some data about the experiment performed over the network extreme100. As we can see 30 variables were selected as the explanation set, in the following way: several sets containing 30 variables were randomly generated and then that being more difficult to be solved by exact computation was choosen. Six variables were randomly selected as evidence, fixing them to their less probable state (the probability of the evidence was 7.41e−12).

Table 10
Description of experiment 7

# Exp.	$\lvert X_E \rvert$	Network	X_E	$\lvert \Omega_{X_E} \rvert$
7	30	extreme100	Pseudo-random	1,073,741,824

We have not been able to solve this problem in our computer (an Intel Pentium III (500 MHz) with 384 MB of RAM), because of memory requirements. Exact computation was feasible with subsets of the whole explanation set containing the first 3, 6, 9, 12, 15, 18 and 24 variables. To give you an idea of the problem complexity, the time employed by the exact algorithm (implemented in Java, virtual machine jdk 1.2) searching for the best explanation when all the unobserved variables are selected as the explanation set was 9 s, while the time employed when the first 24 variables of the explanation set are selected as the explanation set was 4420 s.

We have run the SA algorithm presented here and the GA presented in [7] 100 times over the problem, using the same parameters as for the *artificial5* network. As we do not know the exact results the statistics calculated for the six first experiments cannot be calculated now. Figs. 5 and 6 show the histogram of the 5000 (50×100) configurations found by both algorithms, where the height of each bar corresponds to the number of times of the 100 repetitions in which the configuration with that probability was found.

Although we cannot be sure if we have really found the best 50 MPEs, given the obtained results and the accuracy exhibited by both algorithms (SA and GA) in the six first experiments, we can be almost sure that the best explanations have been found. Respect to the obtained results, the SA algorithm (clearly) shows a better performance than the the method based on GAs, finding the three most probable configurations in all the runs. With respect to the computation time required by both algorithms (implementation in *C*), was

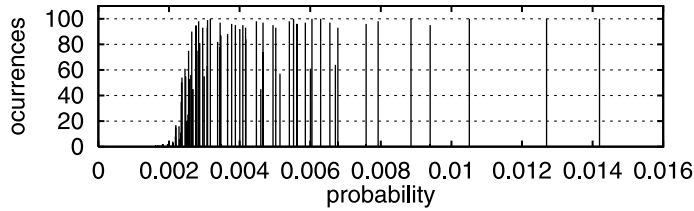

Fig. 5. Histogram for the extreme100 network using SA.

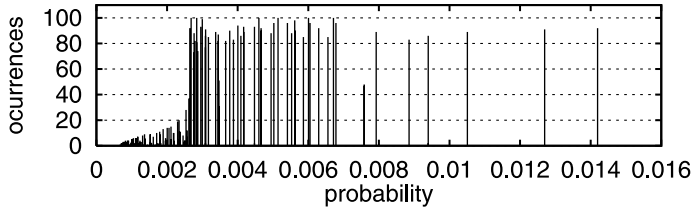

Fig. 6. Histogram for the extreme100 network using GA.

152 s for SA and 250 s for GAs. Of course, with respect to memory requirements the comparison is clearly advantageous for the SA approach.

6. Concluding remarks

The problem of performing partial abductive inference in BBNs has been studied. This type of probabilistic reasoning has the disadvantage that depending on the selected explanation set, exact inference can be unfeasible even for networks in which exact inference is feasible for others types of probabilistic reasoning (like probabilities propagation or *total* abductive inference).

We have presented a SA algorithm for partial abductive reasoning in BBNs. The algorithm has an *anytime* performance, i.e., it can continue the search while time is available or can be stopped at any time to yield the K MPEs found until this moment. The experiments have revealed that the algorithm has a high accuracy, improving in some cases the results obtained by previous methods based on GAs.

With respect to the efficiency of the proposed algorithm, it is more efficient in memory than the GA presented in [5] and much more efficient than the GA presented in [7]. With respect to time requirements, the SA approach is clearly faster than the previous approach based on GAs, even for the optimised GA presented in [7] which take advantage of the individuals previously evaluated by means of specific genetic operators.

The algorithm is *general* in the sense that as it is based on clique tree propagation, it can be applied to multiple connected networks, and it does not impose any constraint about the number of MPEs to be found.

The algorithm is approximate in the sense that we cannot be sure that the optimum has been really found, but as the evaluation function used here is based on exact propagation, we can be sure of the probability assigned to each explanation, so if explanation e_1 is ranked before explanation e_2, then e_1 is (really) more probable than e_2.

The two previous remarks are *good* consequences of using an evaluation function based on clique tree propagation. However, this implies to assume that the network is such that an exact probabilistic propagation is feasible. In the future we plan to relax this assumption by evaluating configurations using approximate techniques.

Acknowledgements

This work has been supported by the Spanish Comisión Interministerial de Ciencia y Tecnología (CICYT) under Project TIC97-1135-CO4-01.

References

[1] I.A. Beinlich, H.J. Suermondt, R.M. Chavez, G.F. Cooper, The ALARM monitoring system: a case study with two probabilistic inference techniques for belief networks, in: Proceedings of the Second European Conference on Artificial Intelligence in Medicine, Springer, Berlin, 1989, pp. 247–256.

[2] A. Cano, J.E. Cano, S. Moral, Convex sets of probabilities propagation by simulated annealing, in: Proceedings of the Fifth International Conference on Information Processing Management of Uncertainty (IPMU), Paris, France, 1994.

[3] U. Chajewska, J.Y. Halpern, Defining explanation in probabilistic systems, in: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI–97), Morgan Kaufmann Publishers, San Francisco, CA, 1997, pp. 62–71.

[4] A.P. Dawid, Applications of a general propagation algorithm for probabilistic expert systems, Statistics and Computing 2 (1992) 25–36.

[5] L.M. de Campos, J.A. Gámez, S. Moral, Partial abductive inference in Bayesian belief networks using a genetic algorithm, Pattern Recognition Letters 20 (11–13) (1999) 1211–1217.

[6] L.M. de Campos, J.A. Gámez, S. Moral, On the problem of performing exact partial abductive inference in Bayesian belief networks using junction trees, in: Proceedings of the Eighth International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'00), vol. III, 2000, pp. 1270–1277.

[7] L.M. de Campos, J.A. Gámez, S. Moral, Partial abductive inference in Bayesian belief networks: an evolutionary computation approach by using problem specific genetic operators, IEEE Transactions on Evolutionary Computation, to appear.

[8] L.M. de Campos, J.F. Huete, Approximating causal orderings for Bayesian networks using genetic algorithms and simulated annealing, in: Proceedings of the Eighth International Conference on Information Processing and Management of Uncertainty in Knowledge-based Systems (IPMU'00), vol. I, 2000, pp. 333–340.

[9] R. Dechter, Bucket elimination: a unifying framework for probabilistic inference, in: Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96), Portland, Oregon, 1996, pp. 211–219.

[10] R. Dechter, I. Rish, A scheme for approximating probabilistic inference, in: Proceedings of the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI–97), Morgan Kaufmann Publishers, San Francisco, CA, 1997, pp. 132–141.

[11] E.S. Gelsema, Abductive reasoning in Bayesian belief networks using a genetic algorithm, Pattern Recognition Letters 16 (1995) 865–871.

[12] E.S. Gelsema, Diagnostic reasoning based on a genetic algorithm operating in a Bayesian belief network, Pattern Recognition Letters 17 (1996) 1047–1055.

[13] S. Geman, D. Geman, Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images, IEEE Transaction on Pattern Analysis and Machine Intelligence 6 (1984) 721–741.

[14] J.W. Greene, K.J. Supowit, Simulated annealing without rejected moves, in: Proceedings of the IEEE International Conference on Computer Design, Port Chester, 1984, pp. 658–663.

[15] F.V. Jensen, An Introduction to Bayesian Networks, UCL Press, 1996.

[16] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.

[17] U. Kjærulff, Optimal decomposition of probabilistic networks by simulated annealing, Statistic and Computing 2 (1992) 7–17.

[18] Z. Li, B. D'Ambrosio, An efficient approach for finding the mpe in belief networks, in: Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence, Morgan and Kaufmann, San Mateo, 1993, pp. 342–349.

[19] R. Lin, A. Galper, R. Shachter, Abductive inference using probabilistic networks: randomized search techniques, Technical Report KSL-90-73, Knowledge Systems Laboratory, Stanford University, California, 1990.
[20] R.E. Neapolitan, Probabilistic Reasoning in Expert Systems. Theory and Algorithms, Wiley/Interscience, New York, 1990.
[21] D. Nilsson, An efficient algorithm for finding the M most probable configurations in Bayesian networks, Statistics and Computing 8 (1998) 159–173.
[22] J. Pearl, Distributed revision of composite beliefs, Artificial Intelligence 33 (1987) 173–215.
[23] J. Pearl, Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, San Mateo, 1988.
[24] Y. Peng, J.A. Reggia, A probabilistic causal model for diagnostic problem solving – Part One, IEEE Transactions on Systems, Man, and Cybernetics 17 (2) (1987) 146–162.
[25] Y. Peng, J.A. Reggia, A probabilistic causal model for diagnostic problem solving – Part Two, IEEE Transactions on Systems, Man, and Cybernetics 17 (3) (1987) 395–406.
[26] H.E. Pople, On the mechanization of abductive logic, in: Proceedings of the Third International Joint Conference on Artificial Intelligence, 1973, pp. 147–152.
[27] C. Rojas-Guzman, M.A. Kramer, Galgo: A genetic algorithm decision support tool for complex uncertain systems modeled with Bayesian belief networks, in: Proceedings of the Ninth Conference on Uncertainty in Artificial Intelligence, Morgan Kauffman, San Mateo, 1993, pp. 368–375.
[28] B. Seroussi, J.L. Goldmard, An algorithm directly finding the K most probable configurations in Bayesian networks, International Journal of Approximate Reasoning 11 (1994) 205–233.
[29] S.E. Shimony, The role of relevance in explanation I: irrelevance as statistical independence, International Journal of Approximate Reasoning 8 (1993) 281–324.
[30] S.E. Shimony, The role of relevance in explanation II: disjunctive assignments and approximate independence, International Journal of Approximate Reasoning 13 (1995) 27–60.
[31] B.K. Sy, Reasoning MPE to multiply connected belief networks using message passing, in: Proceedings of the 11th National Conference on AI, AAAI, 1993, pp. 570–576.
[32] P.J.M. Van Laarhoven, E.H.L. Aarts, Simulated Annealing, Reidel Publishers, 1988.
[33] H. Xu, Computing marginals for arbitrary subsets from marginal representation in Markov trees, Artificial Intelligence 74 (1995) 177–189.
[34] X. Zhong, E. Santos Jr., Directing genetic algorithms for probabilistic reasoning through reinforcement learning, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 8 (2000) 167–186.