Case-Based Probability Factoring in Bayesian Belief Networks

Luigi Portinale
Dipartimento di Informatica - Universita' di Torino
C.so Svizzera 185 - 10149 Torino (Italy)

Abstract

Bayesian network inference can be formulated as a combinatorial optimization problem, concerning in the computation of an optimal factoring for the distribution represented in the net. Since the determination of an optimal factoring is a computationally hard problem, heuristic greedy strategies able to find approximations of the optimal factoring are usually adopted. In the present paper we investigate an alternative approach based on a combination of genetic algorithms (GA) and case-based reasoning (CBR). We show how the use of genetic algorithms can improve the quality of the computed factoring in case a static strategy is used (as for the MPE computation), while the combination of GA and CBR can still provide advantages in the case of dynamic strategies. Some preliminary results on different kinds of nets are then reported.

1 Introduction

The goal of inference on Bayesian belief networks [10] is in general to compute the posterior probability of some nodes in the network, given that a set of nodes has been observed as evidence; this might be either the computation of some marginal posteriors or the determination of the *most probable explanation* (MPE) or the computation of specific *joint-conditional queries*. While in the first case the task is to determine the posterior probability of specific instances of single nodes (marginals), the second and third case concern the computation of joint posterior probabilities and in particular, in the case of MPE, the computation of the scenario (instantiations of all variables of the net) having higher probability (this generalizes to the computation of the the first k scenarios having higher probability). Recent works [8, 9] have shown that general inference on Bayesian networks can be considered as a *factoring problem* on the probability distributions explicitly represented in the net. In particular, for arbitrary conjunctive queries (i.e. queries involving an arbitrary subset of net's variables), the use of an optimal factoring of such distributions leads to an optimal performance of the inference algorithms, in terms of the number of multiplications needed to obtain the desired posterior probability distribution; this may result in a considerable improvement of the efficiency of inference.

Factoring strategies can be either *static* or *dynamic*, meaning that factorings are computed off-line (before any query) or on-line (every time a new query is made) respectively. Dynamic factoring strategies have the advantage of being more suitable to the query at hand, but they have to be implemented for performing in reasonable time. There is then a trade-off between computational effort and quality of the result that is very important for dynamic factoring strategies. Since determining an optimal factoring appears to be a hard problem[1], in [9] a particular *greedy algorithm* for dynamic factoring called *set factoring* has been proposed. Experimental results have shown a

[1]To the best of our knowledge, there is evidence that the problem could be NP-hard [9], but there is no proof yet.

good performance of the set-factoring heuristic and its computational complexity has been proved to be $O(m^3)$, where m is the number of nodes of the subnet $S_N(Q, E)$ determined on the original belief network N by the set of evidence and queried variables E and Q.

In the present paper we investigate an alternative way of addressing the trade-off mentioned above; we consider a search factoring strategy based on a genetic algorithm (GA) [4], combined with a case-based reasoning methodology [7] which is able to exploit factorings computed by the genetic algorithm for cases similar to the current one. The task of interest is then the computation of joint-conditional probability for conjunctive queries, having the MPE computation task as a special case. The approach is based on the following considerations: genetic algorithms represent a good class of algorithms for solving a wide range of optimization problems; they are usually better than local heuristics such as greedy algorithms essentially because they can stochastically explore a wider search space. On the other hand they work well on off-line computations, since they may require a lot of time for convergence to a result near to the optimum to be found. In terms of factoring, this means that a genetic approach could be reasonable in a context of static factorization, as for instance for the computation of the MPE of a belief net, while for queries involving a large number of variables and in a context of dynamic factoring strategies, it could not be reasonable to directly use a genetic approach.

Given a particular belief network, there are two parameters influencing the factoring: the set of evidence variables and the set of queried variables. Since the sets of evidence and queried variables (E and Q) determine a subnet $S_N(Q, E)$ on the original belief network N, we can consider the input to a factoring problem as being characterized by such a subnet and by the set of queried variables. The subnet identifies the probability distributions that must be factorized, while the set of queried variables is used to determine which variables must be summed over during the application of the factoring (see the algorithm in [8]). If we are able to provide a good factoring for a particular case, then there should be a good chance that the same factoring will provide a good result also for cases that are similar to the given one, where the similarity is intended to be in terms of subnet $S_N(Q, E)$ and queried variables in Q. Case-Based Reasoning (CBR) [7] deals exactly with this kind of problem: exploiting past problem-solving experiences for solving new cases. We then investigated the possibility of a case-based approach to probability factoring in belief networks, where the factoring to be used in the case to be solved is the factoring associated with the most similar case(s) retrieved from a case memory. For cases to be stored for future retrieval, the genetic approach can be adopted in order to (off-line) compute the corresponding factoring, by taking advantage of the search power of such a class of algorithms. The key point of the approach is that we have implemented a retrieval strategy of stored cases whose complexity is $O(m)$, where m in the number of variable nodes in the subnet corresponding to the current case. By comparing this with the $O(m^3)$ complexity of set factoring, we can notice that, even if the application of the retrieved factoring to the current case will lead to a higher number of multiplications than the application of set factoring, there is still the possibility of having a more efficient approach, agreeing that the difference in the required multiplications is somewhat limited. Having a precise evaluation of such a limitation is quite hard; however, the set of data we have collected from some experiments support the claim that, if some methodology is used to store cases, the above difference can be not very significant.

In the following, we will discuss how genetic algorithms can be employed in the search of a near-to optimum factoring; we then show some results in applying such a genetic approach to static factoring as for MPE computation and finally how the use of a GA for factoring computation can be integrated in a CBR perspective, for the implementation of dynamic factoring strategies.

2 A Genetic Approach to Factoring

The genetic approach we have investigated in the current stage of the work is essentially a plain application of standard genetic methodologies (see [4]). This means that we did not developed specialized genetic operators for this task, leaving this possibility open to further research. First of all we have chosen an integer-based representation of chromosomes. Each *chromosome* is then a string of integers of length equal to the number of variable nodes (and also to the number of probability distributions explicitly represented) in the subnet $S_N(Q, E)$ determined by evidence and queried nodes (E and Q) on the original net N. Each *gene* corresponds to a particular distribution of the subnet and the *allele* (i.e. the value of the gene) represents an index for the corresponding distribution. As an example, consider the net of figure 1; we may associate an index with each

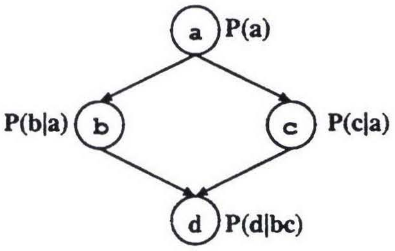

Figure 1: A Simple Belief Net

distribution of the net. For instance if $P(a), P(b|a), P(c|a)$ and $P(d|bc)$ have indices $1, 2, 3$ and 4 respectively, then the chromosome $\alpha = \langle 2143 \rangle$ represents the sequence $\langle P(b|a)P(a)P(d|bc)P(c|a) \rangle$.

The *fitness function* f is represented by the number of multiplications determined by the optimal parenthesization associated with the order of distributions as implied by the chromosome. Such a function will obviously depend also from the set of queried variables Q. For instance, in the example above, by considering $Q = \{a\}$ and if the algorithm described in [8] is applied and we assume all nodes to represent binary variables, the optimal parenthesization for the chromosome α is $(P(b|a)(P(a)(P(d|bc)P(c|a))))$, resulting in 16 multiplications (i.e. $f(\alpha) = 16$). This fitness function can be easily implemented by a classical dynamic programming algorithm for determining minimal cost alphabetic trees [6]. A chromosome together with the optimal parenthesization represents a *factoring*. The GA's task is then to determine a near-to-optimum factoring by repeatedly applying a set of genetic operators (classically *selection*, *crossover* and *mutation*) to a population of chromosomes, in order to generate a new population. The process of generating a new population through GA operators is said to be a *generation*. Of course, a GA is a kind of *anytime algorithm* in which at each generation we can stop by providing in output the best chromosomes (in terms of the fitness function) of the current population; more generations are allowed, better is the result we get.

To implement our genetic algorithm, we have exploited the *LibGA* library of genetic operators [2]. In the current set of experiments, we used a *roulette selection* and a *pmx crossover*. A roulette selection means that chromosomes are picked from the population using a roulette wheel having slots sized according to fitness (i.e. chromosomes with higher fitness have higher probability of selection), while the *pmx* is one of the best crossover methods on problems that, like the factoring problem, can be formulated as *sequencing problems* (see [3] for more details). We also assume a fixed rate of mutation of 0.6 with a *swap type mutation* (i.e. two alleles are swapped with probability 0.6)[2].

[2]More accurate experiments varying both mutation and cross-over rate are planned in order to determine how

Net	N. Nodes	N. Arcs	GA	Set Factoring
ADDER	49	60	558	780
CAR	45	45	288	240
PRINTER	26	25	238	292
CHILD	20	25	813	894
RANDOM1	50	125	622114	1013770
RANDOM2	44	112	338908	866816
RANDOM3	46	121	110728	1123660
RANDOM4	45	151	987067	4308836

Table 1: Comparison of GA with Set Factoring for MPE

The genetic framework described above has been verified by comparing it with an algorithm finding optimal factoring; due to the exponential complexity of the latter such a comparison has been performed only for very small nets (the algorithm for optimal factoring is simply an algorithm generating all the $m!$ possible permutations of m elements for nets having m variable nodes and computing the fitness function for each of such permutations). Different experiments have been performed on nets having a number of nodes ranging from 5 to 12 and they showed that the GA was always able to find the optimal factoring. The criterion to stop the GA was to run it until convergence, meaning that the all fitness values in the population pool were identical.

Having evidence that the GA approach is suitable to find good factorings, we investigated the possibility of exploiting the approach for static factoring of belief networks; in particular, we concentrate on the problem of finding the MPE of a given net. This is clearly a problem where a static factoring strategy can be applied since the set of queried variables is fixed (i.e. all the net's variables). We considered the MPE computation for a set of nets partially taken from real-world and partially obtained with a random generation[3]. Results are summarized in table 1; in this batch of experiments we considered 4 real-world nets namely ADDER (a four-component full adder), CAR (a net for car engine fault detection), PRINTER (a net for printing problems troubleshooting developed at Microsoft Corp. and illustrated in [5]) and CHILD (a net for congenital condition leading to a "blue baby" and described in [12]), together with 4 randomly generated nets (RANDOM1 through RANDOM4). In table 1 we indicated both the number of nodes and arcs of each net and the number of multiplications needed for computing the MPE using the GA and the set factoring heuristic respectively; table 1 shows that the genetic approach can produce results that are considerably better than those obtained by means of set factoring. In particular, more sparse in the space of solutions (meaning that values for fitness function are not concentrated on a particular region), better is the performance of the GA. This can be noticed by looking on table 1 at the good results obtained by the GA on the nets having a high degree of connection between nodes. Indeed, in this kind of networks there may be huge differences in the fitness values of different chromosomes and the search power of the GA is well exploited, while the set factoring heuristic can be trapped on sub-optimal regions that may be very far from the real optimum.

However, as mentioned above, giving the same time resources to the genetic algorithm and to the set factoring will in general result in a better performance for set factoring. Genetic algorithms need time to produce generations corresponding to search regions near to the optimum and so are not very adequate in a context of dynamic factoring. The next section illustrates how by combining

these parameters can influence the overall performance of the algorithm.

[3]The random net generator of the IDEAL system [13] has been used for the second class of nets.

the genetic approach in a case-based framework we can address the trade-off underlying static vs dynamic factoring strategies.

3 Case-Based Factoring

To obviate the problem of slow convergence of the genetic approach with respect to set factoring, we have chosen to exploit the search power of the genetic algorithm in a undirected way by means of a case-based reasoning approach; given a belief network N, when a new case $C = \langle S_N(Q,E), Q\rangle$ (characterized by a subnet of the original belief net and by a set of queried variables) has to be solved, we retrieve from a case memory one or more cases similar to the current one. Let us define the notion of *covering* among subnets.

Definition 1 *A belief network N* covers *another belief network N' if and only if the set of distributions represented by N' is a subset of the set of distributions represented by N.*

This essentially means that for every node X of N', X is in N and every parent of X in N is also in N'.

Among all the stored cases, we retrieve those cases having the minimal (with respect to the set of variables) subnet $S_N(Q',E')$ *covering* the subnet $S_N(Q,E)$ of the current case and such that the set of queried variable Q' has maximal intersection with the queried variables of the current case Q. We have devised a very simple organization of the case memory allowing us a fast retrieval of cases. In particular, we associate an index with every variable of the belief network, pointing to the list of cases containing that variable in their subnet. Every stored case will then be represented as a triple $C' = \langle S_N(Q',E'), Q', F\rangle$ where F is the genetic computed factoring for the case. This means that we can exploit the power of the GA off-line, in order to compute a significant near-to-optimum factoring for cases to be stored for future retrieval.

Given the case $C = \langle S_N(Q,E), Q\rangle$ to be solved, we just have to follow indices for nodes contained in $S_N(Q,E)$ and so we can retrieve cases in time $O(m)$ where m is the number of variables in $S_N(Q,E)$. Once we have filtered cases with minimal covering subnet and maximal intersection of queried variables, we get a set of factorings that can be reasonably applied to the current case. In general, the retrieved factorings contain distributions that are not relevant to the current case however, the fact that those factorings are produced by a genetic algorithm widely exploring the solution space, could assure good results also for the case to be solved. Concerning this point, is clear that higher is the difference in the number of distributions contained in the current subnet and in the retrieved one, worse is the application of the retrieved factorings to the current case. One possible mechanism is to prevent the application of a given retrieved factoring (and so to consider the corresponding case as not retrieved) in case the above difference is too high.

Definition 2 *Given two belief networks N and N' with m and m' variable nodes respectively and such that N covers N', the* relative distance *between N and N' is $\delta(N,N') = \frac{|m-m'|}{m'}$.*

We can then avoid to consider the retrieval of cases whose corresponding subnet has a relative distance with the current case greater than a given threshold; we call such a threshold *maximum admissible relative distance* (MARD).

Two basic issues characterize the case-based approach: the choice of cases to be stored and the adaptation of retrieved factorings to the current case. Concerning the first point, a *training phase* is responsible for the construction of a significant initial case memory. Learning may occur every time a new case not supported in a relevant way by stored cases is presented. This may happen both when it is not possible to retrieve a subnet covering the current one and when this is possible, but the relative distance is greater than MARD. We developed a strategy that allows us to store, especially during the training phase, more than one case from a particular situation. Indeed, storing the case exactly corresponding to the subnet induced by evidence and queried variables may result to be too specific. It is possible to devise methods allowing us to "enlarge" the subnet to be considered, producing then more cases to be stored. This can be done by considering the set of nodes relevant to a given query and evidence (through the computation of *d-connecting paths* [10]) and by enlarging such a set in ways similar to the one described in [1].

Regarding the issue of factoring adaptation, in the present work we assume to directly apply the retrieved factorings to the current case. The problem of adaptation is left to future works and will probably involve the study of more specific genetic operators. In the next section we will report on the experimental results we have obtained by using the approach described.

4 Experimental Results and Future Works

We have performed a set of experiments with three of the networks used for the analysis of the MPE computation: the CAR net (a quite sparse network), the ADDER net (a medium sparse network) and the RANDOM1 net (a highly connected network). We essentially concentrate on

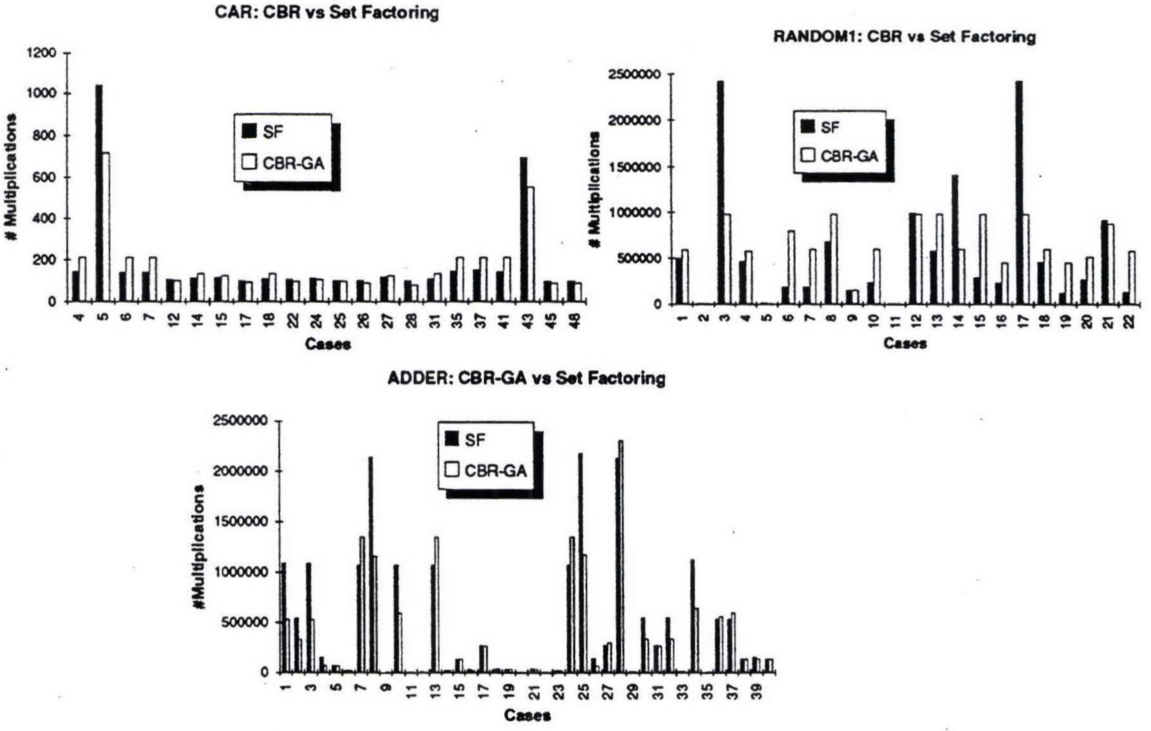

Figure 2: Comparison between Set Factoring and CBR-GA

diagnostic problems in which evidence was put on nodes having no descendant and queries were posed at root nodes. Concerning the first two nets, we stored a set of cases from a training set of basic cases, constructed by hand directly analyzing the corresponding net. In particular, for CAR we determined 48 basic cases (set of evidence nodes) among which we used 26 cases for training (resulting in 37 cases effectively stored into memory) and 22 for test. For ADDER we identified 30 basic cases resulting in 19 cases to be stored in memory (some cases have resulted into the same subnet and set of queried variables even if they had different evidence); we then randomly generated evidence for 40 cases to be used for test. For RANDOM1 we randomly generated 50 cases for training and 40 for test. This determined a case memory containing 64 cases, but that was not *stable* enough for the testing cases. In fact, by considering a MARD=35%, the experiments resulted in 17 cases having no retrieval from the case memory. Since the goal of the present paper was to test the quality of the approach without considering learning aspects in depth, we also stored the 17 cases in memory performing the experiment with 84 cases in memory (the initial 64 with such additional 17 cases and those resulted in their "enlargement") and 23 for testing. Figure 2 shows a direct comparison (in terms of number of multiplications) between the application of set factoring (SF) and of a retrieved genetic computed factoring (CBR-GA) on a given case for the three nets above. Figure 3 shows, for each of the above nets, a cumulative graph on the number of multiplications for the same set of experiments of figure 2; From this set of very preliminary experiments we can

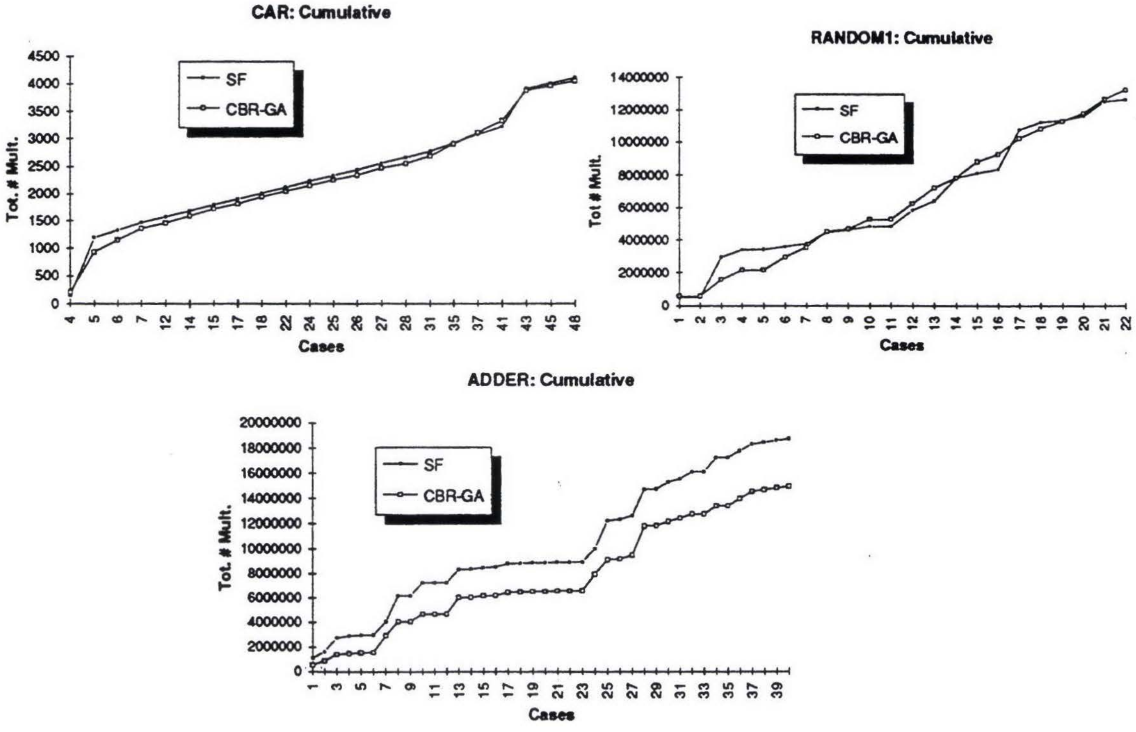

Figure 3: Cumulative Comparison between Set Factoring and CBR-GA

notice that the two approaches are essentially comparable in terms of number of multiplications (for the ADDER network the CBR-GA approach performed definitively better). The basic advantage of the CBR-GA approach is the fact that it only require $O(m)$ time to determine the factoring opposite to the $O(m^3)$ of the set factoring. The disadvantage is that it require the management of a memory of *good* past cases and the related aspects (essentially concerning learning aspects) have still to be studied in depth.

As mentioned in the previous section, open issues are essentially related to the role of the genetic algorithm and to the definition of more formal learning strategies. In the first case, specific genetic operators should be studied, in order to take advantage of specific knowledge on the factoring problem; this could also influence the convergence rate of the algorithm by possibly changing its off-line role and the definition of adaptation strategies for the retrieved factorings. Learning problems concern the definition of specific strategies for adding (and/or deleting) cases to memory; solutions to such problems should involve the *utility* and the *competence* of cases and are still an open problem [11].

Acknowledgments

I would like to thank Filippo Neri for his help in the preliminary stage of this research. This work has been supported by grant MURST 60%.

References

[1] J.S. Breese and D. Heckermann. Decision-theoretic case-based reasoning. In *Proc. 5th International Workshop on AI and Statistics*, Fort Lauderdale, FL, 1995.

[2] A.L. Corcoran and R.L. Wainwright. LibGA: A user-friendly workbench for order-based genetic algorithm research. In *Proc. 1993 ACM/SIGAPP Symposium on Applied Computing" (SAC 93)*, Indianapolis, 1993.

[3] B.R. Fox and M.B. McMahon. Genetic operators for sequencing problems. In G.J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 284–300. Morgan Kaufmann, 1991.

[4] D.E. Goldberg. *Genetic Algorithms in Search Optimisation and Machine Learning.* Addison-Wesley, 1989.

[5] D. Heckermann and M.P. Wellman. Bayesian networks. *Communications of the ACM*, 38(3):27–30, 1995.

[6] T.C. Hu. *Combinatorial Algorithms.* Addison Wesley, 1982.

[7] J.L. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, 1993.

[8] Z. Li and B. D'Ambrosio. An efficient approach for finding the MPE in belief networks. In *Proc. 9 Conference on Uncertainty in Artificial Intelligence*, pages 342–349, Washington, 1993.

[9] Z. Li and B. D'Ambrosio. Efficient inference in bayesian networks as a combinatorial optimization problem. *International Journal of Approximate Reasoning*, 11(1):55–81, 1994.

[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1989.

[11] B. Smyth and M.T. Keane. Remembering to forget. In *Proc. 14th IJCAI*, pages 377–382, Montreal, 1995.

[12] D.J. Spiegelhalter, A.P. Dawid, S.L. Lauritzen, and R.G. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8(3):153–219, 1993.

[13] S. Srinivas and J.S. Breese. IDEAL: a software package for analysis of influence diagrams. In *Proc. 6th Conf. on Uncertainty in Artificial Intelligence*, Cambridge, MA, 1990.