A New Stochastic Algorithm for Strategy Optimisation in Bayesian Influence Diagrams

Michal Matuszak & Tomasz Schreiber

Faculty of Mathematics and Computer Science,
Nicolaus Copernicus University,
Toruń, Poland
{gruby, tomeks}@mat.umk.pl

Abstract. The problem of solving general Bayesian influence diagrams is well known to be NP-complete, whence looking for efficient approximate stochastic techniques yielding suboptimal solutions in reasonable time is well justified. The purpose of this paper is to propose a new stochastic algorithm for strategy optimisation in Bayesian influence diagrams. The underlying idea is an extension of that presented in [2] by Chen who developed a self-annealing algorithm for optimal tour generation in traveling salesman problems (TSP). Our algorithm generates optimal decision strategies by iterative self-annealing reinforced search procedure, gradually acquiring new information while driven by information already acquired. The effectiveness of our method has been tested on computer-generated examples.

1 Introduction

Influence diagrams[4–6] are widely acknowledged as an important probabilistically oriented graphical representation paradigm for decision problems. An influence diagram is built on a directed acyclic graph (DAG) whose nodes and arcs admit standard interpretations steming from and extending those used for Bayesian (belief) networks. Three principal types of nodes are considered: chance nodes standing for random variables (represented as ovals in our figures below), decision nodes corresponding to available decisions (rectangles in our figures) and utility nodes (rhombi) specifying the utilities to be maximized by suitable choices of decision policies. The arcs leading to chance nodes indicate direct causal relationships (at least if the network is well designed) not necessarily corresponding to any temporal ordering. On the other hand, the arcs leading to decision nodes specify the information available at the moment of decision making, thus feeding input to decision policies. Some arcs between decision nodes may also be of informative nature as determining the order of decision making. The influence diagrams can be considered as a generalization of (symmetric) decision trees, see [4].

In Fig. 1, generated by Hugin Lite package [3], a simple example of an influence diagram is shown. The decision node *treatment* represents the choice whether or not to visit a doctor. A visit to a doctor does increase the chance of

no cough, yet it also causes negative effects, such as the need to pay the fee for visit. Further, wearing a *scarf* decreases the chance of getting sore throat but also negatively affects our appearence. All these effects are jointly taken into account in the utility node *happiness*.

A number of algorithms for solving influence diagrams have been developed, falling beyond the scope of the present article. We refer the reader to Chapter 10 in [4] for a detailed discussion, see also the references in Subsection 5.2.2 of [5].

In general, finding an optimal decision strategy for an influence diagram is an NP-hard task. This is easily shown by reducing an NP-complete problem to the considered task. A natural choice is the traveling salesman problem (TSP) known as a classical NP-complete task. To each city a decision node is ascribed with the remaining cities as admissible states. The decision taken coincides with the next city to visit. Further, a utility node is created with incoming arcs from all decision nodes. The utility function is defined by summing up the negative distances between cities and their successors in case the decision sequence yields a valid Hamilton tour, and is set to $-\infty$ otherwise. Clearly, solving the TSP problem is, in this set-up, equivalent to finding the optimal decision strategy for the so-constructed influence diagram.

In his work [2] (see also the discussion in [7]) Chen proposed an appealing and simple stochastic optimisation algorithm for the TSP problem, quite original in its design, highly effective and yet apparently somewhat underestimated in the literature. In the course of an iterative procedure subsequent TSP tours are randomly generated: each city is assigned a table of weights for connections to all remaining cities and each time the choice of the next city to visit is made by random among cities not yet visited, with probabilities proportional to the corresponding connection weights. This way all cities get visited and eventually we get back to the starting point. Next, the so generated tour is compared with the one obtained in the previous iteration. Depending on whether the new cycle is longer or shorter than the previous one, the connection weights between cities neighbouring in both tours are correspondingly reinforced or faded. The algorithm stops when the re-normalised weights are close enough to zeros and ones, which corresponds to a deterministic tour choice, converging to the optimal one under suitable reinforcement/fading protocols.

With the TSP problem regarded as a particular case of decision strategy optimisation, the purpose of the present paper is to is to extend Chen's approach to general influence diagrams. As we will see, this can be done neatly and effectively, although not without substantial extensions of Chen's idea.

2 The algorithm

To give a formal description of the proposed decision strategy optimisation algorithm, assume an influence diagram $(\mathcal{S}, \mathcal{P}, \mathcal{U})$ is given, built on a connected DAG $\mathcal{S}$, with CPTs $\mathcal{P}$ and utility functions $\mathcal{U}$. The set of nodes in $\mathcal{S}$ splits into chance nodes $C_{\mathcal{S}}$, decision nodes $D_{\mathcal{S}}$ and utility nodes $U_{\mathcal{S}}$. All these objects are

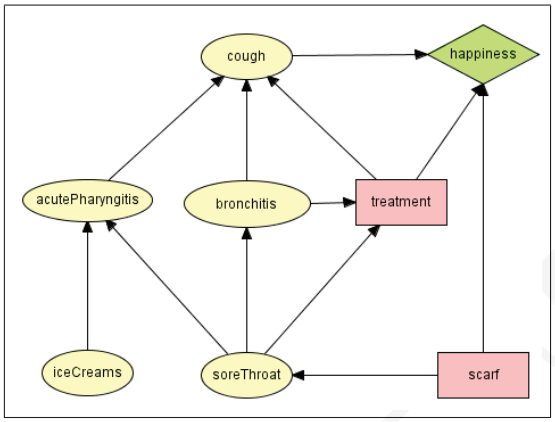

Fig. 1. Sample influence diagram.

assumed fixed in the course of strategy optimisation. In addition, for optmisation purposes we attach to each decision node $D \in D_{\mathcal{S}}$ the *randomised policy* δ_D which assigns to each configuration $\bar{w}$ of $\mathrm{pa}(D)$ a probability distribution on possible decisions to be taken, that is to say $\delta_D(d|\bar{w})$ stands for the probability of choosing decision d given that $\mathrm{pa}(D) = \bar{w}$. These randomised policies will evolve in the course of the optimisation process, eventually to become (sub)optimal deterministic policies collectively determining the utility maximizing strategy for the influence diagram considered. The initial choice of $\delta_D,\ D \in D_{\mathcal{C}}$ can be either *uniform*, with all decisions equiprobable, or *heuristic* provided some additional knowledge is available on our system allowing us to make a good *first guess* about the optimal strategy.

Our algorithm relies on an iterative procedure carried out in *epochs* of fixed length N. At the beginning of each epoch t, $t = 1, 2, \ldots$ we divide the collection $D_{\mathcal{C}}$ of decision nodes into the set of *trainable decision nodes* $TD^t_{\mathcal{C}}$ whose randomised decision policies are to undergo updates, and *frozen decision nodes* $FD^t_{\mathcal{C}}$ whose status is to remain unaltered throughout the epoch. Roughly speaking, the purpose of this splitting is to ensure that only a modest fraction of system parameters are updated at a time, which is indispensible for the stability of algorithm, see below for a more detailed discussion. In addition, we impose the following requirement, whose relevance is discussed in the sequel.

[**Forbidden path condition**] No two trainable nodes are connected by a directed path in $\mathcal{S}$.

With $\mathcal{D} \subseteq D_{\mathcal{S}}$ write $A[\mathcal{D}]$ for the ancestry of $\mathcal{D}$ in $\mathcal{S}$ that is to say $A[\mathcal{D}]$ is the collection of nodes in $\mathcal{S}$ from which a decision node from $\mathcal{D}$ can be reached along a directed path in $\mathcal{S}$. Clearly, the forbidden path condition is equivalent to requiring that $A[TD^t_{\mathcal{S}}] \cap TD^t_{\mathcal{S}} = \emptyset$ during tth epoch. Moreover, since the utility nodes have no progeny, we readily conclude that $A[TD^t_{\mathcal{S}}] \subseteq C_{\mathcal{S}} \cup FD^t_{\mathcal{S}}$.

A standard way of selecting the trainable collection, as implemented in our software, goes as follows.

– Whenever passing to a new epoch, do sequentially for all decision nodes $D \in D_{\mathcal{C}}$
  - If D is a trainable node then freeze it with some probability p_F,
  - If D is a frozen node, make it trainable with some probability $p_T < p_F$ unless doing so violates the forbidden path condition and unless the fraction of time during which D was trainable exceeds maximal admissible value (specified as algorithm parameter).

Note that the *quota* imposed on the fraction of time a given node is trainable is aimed at preventing the situation where a decision node with numerous progeny in $D_{\mathcal{C}}$ receives only a very poor training time fraction as predominantly blocked by its progeny.

The following iterative procedure, repeated a fixed number N of times during each optimisation epoch, say tth epoch, and directly motivated by the ideas developed in [2], lies at the very heart of our algorithm.

1. Set the iteration counter $i := 0$.
2. Generate an instance $\bar{w}_A^{(i)}$ of the *ancestral variable configuration* for $A[TD_{\mathcal{S}}^t] \subseteq C_{\mathcal{S}} \cup FD_{\mathcal{S}}^t$ according to the CPTs of chance nodes and using the randomised decision policies of frozen decision nodes in $FD_{\mathcal{S}}^t \cap A[TD_{\mathcal{S}}^t]$ as CPTs. This is carried out in the standard way with nodes handled recursively proceeding from causes to effects in the policy subnetwork $A[TD_{\mathcal{S}}^t]$. This is where the forbidden path condition is of use as ensuring that no trainable node falls into $A[TD_{\mathcal{S}}^t]$.
3. Repeat a fixed number M of times
   (a) Sample the decisions taken, $d_1, \ldots, d_k$, independently for all trainable decision nodes $D_1, \ldots, D_k \in TD_{\mathcal{S}}^t$ according to their respective current randomised policies $\delta_{D_j},\ j = 1, \ldots, k$ given $\bar{w}_A^{(i)}$. Note that $\bigcup_{j=1}^k \mathrm{pa}(D_j) \subseteq A[TD_{\mathcal{S}}^t]$ and thus the knowledge of $\bar{w}_A^{(i)}$ is sufficient for this sampling.
   (b) Evaluate the expected total utility

$$u^{(i)} := \mathbb{E}[U_{\mathrm{total}} | \bar{w}_A^{(i)}; d_1, \ldots, d_k]$$

   given $\bar{w}_A^{(i)}$ and $d_1, \ldots, d_k$, under the randomised policies of frozen nodes used as respective CPTs. This is easily done by standard Monte-Carlo network instance generating scheme, recursively proceeding from causes to effects. This is possible because the non-instantiated part of the network $\mathcal{S} \setminus [A[TD_{\mathcal{S}}^t] \cup TD_{\mathcal{S}}^t]$ contains no trainable decision nodes and it has the upward cone property – whenever it contains a node X it also contains all its children and, inductively, its whole progeny.
   (c) If $i \geq 1$, set

$$\Delta := u^{(i)} - u^{(i-1)}$$

and update the policies $\delta_{D_j},\ j = 1, \ldots, k$ for trainable nodes by putting

$$\delta_{D_j}(d_j|\bar{w}_A^{(i)} \cap \mathrm{pa}(D_j)) := \exp(\beta\Delta)\delta_{D_j}(d_j|\bar{w}_A^{(i)} \cap \mathrm{pa}(D_j))$$

and, thereupon, re-normalising $\delta_{D_j}(\cdot|\bar{w}_A^{(i)} \cap \mathrm{pa}(D_j))$ so that it remain a probability distribution. The positive constant β here is a parameter of the algorithm, the larger it is the faster the system learns but the less stable the optimisation is.

4. Set $i := i + 1$ and, if $i < N$, return to 1. Otherwise terminate the current epoch.

The intuitive meaning of the above procedure is that the network is presented with a configuration sampled according to the CPTs and current randomised policies of the frozen nodes, whereupon the randomised policies of the trainable nodes are used for decision sampling, with succesful choices (positive Δ) leading to reinforcement of the corresponding probability entries and, on the other hand, with choices deteriorating the performance resulting in fading of the corresponding probability entries (negative Δ). The reinforcement/fading strength depends on the value of the utility gain/loss compared to the previous run. In analogy to Chen's work [2], also here after a large enough number of epochs we eventually end up with the situation where all randomised policies become nearly deterministic in that, for each $D \in D_{\mathcal{C}}$ and parent configuration $\bar{w}$ for $\mathrm{pa}(D)$ the value of $\delta_D(d|\bar{w})$ is close to one for a unique d and close to zero otherwise. This determinism can be easily quantified by looking at the maximal value of $\min(\delta_D(d|\bar{w}), 1 - \delta_D(d|\bar{w}))$ and declaring a policy *nearly deterministic* when this falls below, say, 0.01. To sum up, our algorithm carries out subsequent optimisation epochs until all policies become nearly deterministic. Note in this context that is is crucial to ensure that each node is trainable during a sufficiently large fraction of time, for otherwise it might long remain untrained slowing down the entire process. As already mentioned, this is handled by our training selection scheme discussed above.

3 Implementation and examples

The programme has been implemented in language `D` [8], currently gaining popularity as a natural successor to `C++`, and uses the Tango library [1]. The implementation, aimed so far mainly at algorithm evaluation purposes, can be described as careful but not fully performance-optimised, with the total utility evaluation under frozen decision nodes in 3.(b) performed using the standard Monte-Carlo rather than a more refined and effective scheme. The graph of the diagram was represented using neighbourhood lists. The utility functions were always fit to $[0, 1]$. All test runs were executed on a machine with Intel Core 2 Q9300 2.50 GHz CPU and 2GB RAM.

A sequential version of our algorithm run for a randomly generated influence diagram with 10 chance nodes, 10 decision nodes, 10 utility nodes and 40 arcs, see Fig. 3 generated from Hugin Lite [3], required 1.1ms time per epoch. The

algorithm parameters were set as follows: $p_T = 0.01$, $p_F = 0.04$ and $\beta = 0.01$. After 10000 epochs (11 seconds) the strategy output by our algorithm achieved utility only 4% inferior to the optimal one (as determined using the Hugin package). For a different randomly generated network (doubled number of nodes) execution time per one epoch was 1.8ms, with 10000 epochs. Again, the utility of the output strategy was only 4% inferior to the optimal one.

Getting back to the network depicted in Fig. 1, we performed 10000 iterations of our algorithm, with parameters $p_T = 0.01$, $p_F = 0.04$ and $\beta = 0.03$. The convergence of decision policies for node *treatment* in the course of our algorithm is shown in Fig. 3. The table 3 represents the policy obtained upon convergence of the algorithm. It can be concluded that the sore throat is of crucial importance for our decision whether or not to visit a doctor. On the other hand, the bronchitis appears to be ignored. This does coincide with the optimal strategy as determined by the Hugin Lite package.

Table 1. Decision policy for node *treatment*.

Y	N	
1.00	0.00	sore throat = Y, bronchitis = Y
0.99	0.01	sore throat = Y, bronchitis = N
0.03	0.97	sore throat = N, bronchitis = Y
0.01	0.99	sore throat = N, bronchitis = N

For the diagram given in Fig. 3 (12 chance nodes, 6 decision nodes, one utility node and 42 arcs) the mean execution time per epoch was 0.8ms. The decision strategy after 10000 iterations with parameters $p_T = 0.01$, $p_F = 0.04$ and $\beta = 0.01$ was inferior by 5 % to the optimal strategy.

References

1. Bell, K., Igesund, L.I., Kelly, S., Parker, M. Learn to Tango with D, *Apress*, 2008.
2. Chen, K. Simple learning algorithm for the traveling salesman problem, *Phys. Rev. E* **55**, 7809-7812 (1997).
3. `http://www.hugin.com`
4. Jensen, F.V., Nielsen, T.D. Bayesian Networks and Decision Graphs, 2nd Ed., *Springer*, 2007.
5. Neapolitan, R. E. Learning Bayesian Networks, *Prentice Hall Series in Artificial Intelligence*, *Pearson Prentice Hall*, 2004.
6. Pearl, J. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, *Morgan Kaufmann Publishers Inc.*, 1988.
7. Peretto, P. An Introduction to the Modeling of Neural Networks, *Collection Aléa-Saclay*, *Cambridge University Press*, 1992.
8. `http://www.digitalmars.com/d/`

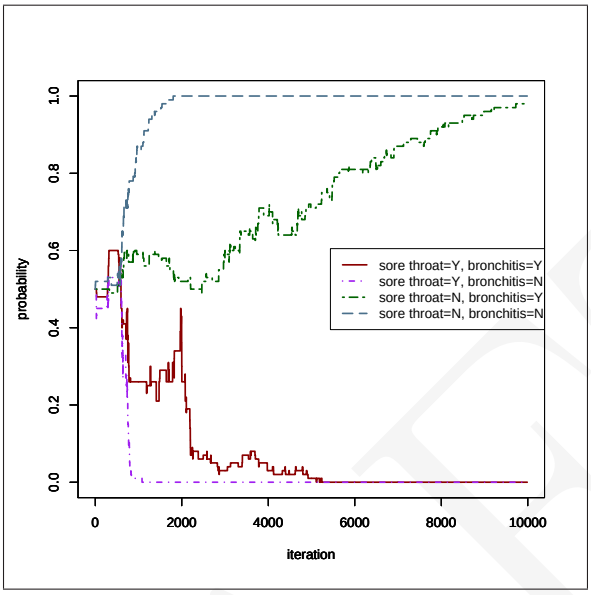

Fig. 2. Convergence of decision policies.

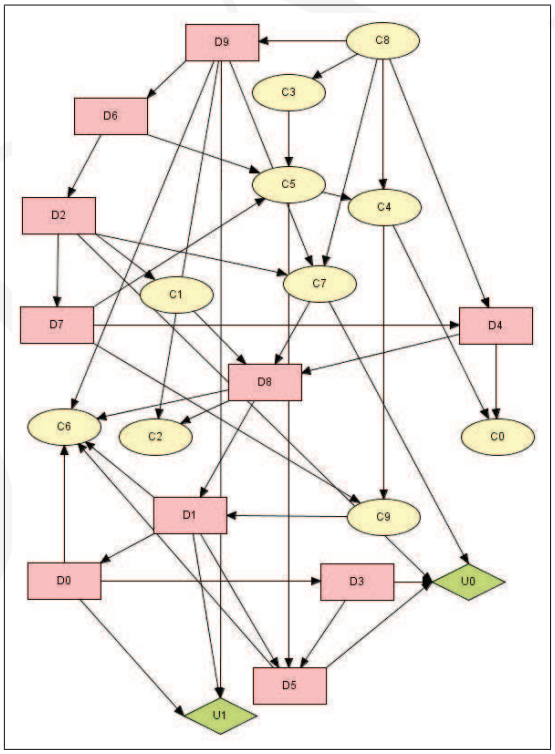

Fig. 3. Randomly generated influence diagram.

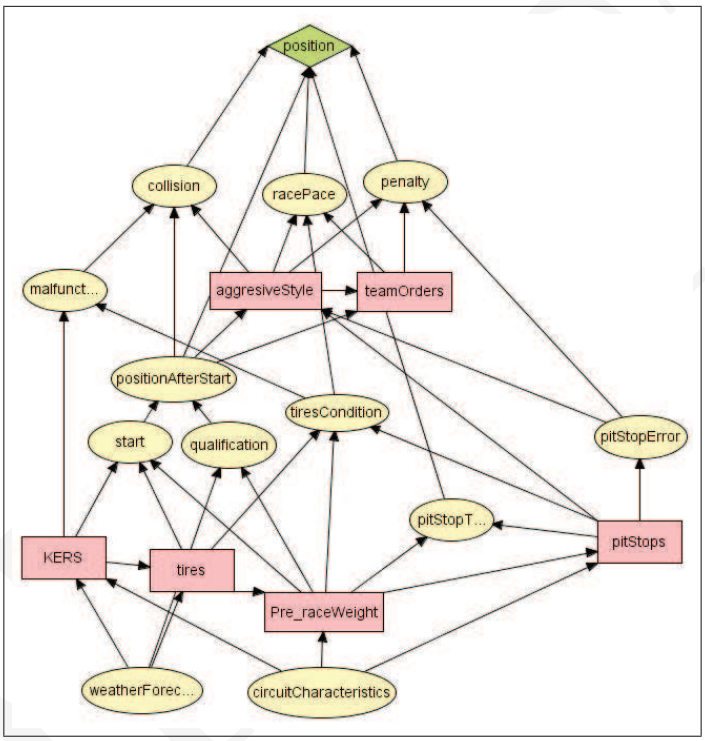

Fig. 4. Race results.