An Efficient Exhaustive Anytime Sampling Algorithm for Influence Diagrams

Daniel Garcia-Sanchez and Marek J. Druzdzel

Decision Systems Laboratory, School of Information Sciences and Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA 15260
{dgarcia,marek}@sis.pitt.edu

Summary. We describe an efficient sampling algorithm for solving influence diagrams that achieves its efficiency by testing candidate decision strategies against the same set of samples and, effectively, reusing samples for each of the strategies. We show how by following this procedure we not only save a significant amount of computation but also produce better quality anytime behavior. Our algorithm is exhaustive in the sense of computing the expected utility of each of the possible decision strategies.

Keywords: Bayesian networks, algorithms, stochastic sampling

1 Introduction

Influence diagrams (IDs) [18] are acyclic directed graphs modeling decision problems under uncertainty. An ID encodes three basic elements of a decision: (1) available decision options, (2) factors that are relevant to the decision, including how they interact among each other and how the decisions will impact them, and finally, (3) the decision maker's preferences over the possible outcomes of the decision making process. These three elements are encoded in IDs by means of three types of nodes: decision nodes, typically represented as rectangles, random variables, typically represented as ovals, and value nodes, typically represented as diamonds or hexagons. Most popular type of IDs are those in which both the decision options and the random variables are discrete. A decision node in a discrete ID is essentially a list of labels representing decision options. Each random variable is described by a conditional probability table (CPT) containing the probability distribution over its outcomes conditional on its parents. Each value node encodes a utility function that represents a numerical measure of preference over the outcomes of its immediate predecessors. An ID that contains only random variables is called a Bayesian network (BN) [25].

An ID is amenable to an algorithmic analysis that can yield for each possible strategy (an assignment of values to all decision nodes conditional on

D. Garcia-Sanchez and M.J. Druzdzel: *An Efficient Exhaustive Anytime Sampling Algorithm for Influence Diagrams*, StudFuzz **213**, 255–273 (2007)
www.springerlink.com © Springer-Verlag Berlin Heidelberg 2007

those chance nodes that are observed before the decision nodes, e.g., [5]) its numerical measure of desirability – expected utility. It is a basic premise of decision theory that a decision maker should maximize his or her gain by choosing the strategy that yields the highest expected utility. Even though this analysis is at its foundations NP-hard [7], there exist several ingenious algorithms that make it feasible for practical ID models. Before we review these algorithms, we will focus on four general characteristic of ID algorithms that will be helpful in making the review more systematic.

While having an exact result is always preferable, in most cases precision can be traded off for computation, dividing ID algorithms into *exact* and *approximate*. The latter class does not give guarantees of optimality, even though it will typically reach or approximate the optimal solution. An important subclass of approximate algorithms are *anytime* algorithms, which are algorithms that regardless of the complexity of the problem have an answer available almost immediately. Subsequent computation, which gradually improves the quality of this answer, can be interrupted at any time. An algorithm is *exhaustive* if it computes the expected utility of each of the possible decision strategies or *focused* if its only goal is identification of the optimal strategy. Focused algorithms are in general more efficient than exhaustive algorithms, but this efficiency comes at a price. The output of a focused algorithm identifies the best next step of a strategy but it does not put it in a context of other possible steps or other strategies. In particular, the output of a focused algorithm does not show whether the suggested decision is a clear winner or a close runner with other, alternative decisions. This is not important in some applications, for example in robotics. However, in applications that include interaction with human decision makers, exhaustive algorithms offer a considerable amount of insight that helps in choosing the superior decision strategy or assists in refining the model. Finally, algorithms can be divided into *direct* and *indirect*, i.e., those that work on IDs directly and those that reduce the IDs solution problem into a series of belief updating problems in its underlying BNs.

There is a sizeable body of literature on the topic of solving IDs. Space limitations prevent us from covering this literature with the exception of those algorithms that are most directly relevant to our paper. The oldest among the direct evaluation methods originates from the work of Olmsted [23], whose algorithm performs a series of node removal and arc reversal operations that successively reduce the ID into a single decision node and a single successor utility node that holds the expected utility of each of the decision options. Shachter [30] has later formalized this algorithm and proven its correctness. Shenoy [32] introduced a new approach that transforms the ID into a valuation network that allows for slightly more efficient calculations. Ndilikilikesha [22] modified Olmsted's algorithm making it computationally as efficient as Shenoy's.

The indirect evaluation methods are inspired by a seminal paper by Cooper [6], who proposed transforming each decision node in an ID into a chance node

with uniform distribution and the (single) value node into a chance node whose probability distribution is a linear transformation of the original utility function into the interval [0,1]. Following this transformation, the algorithm instantiates, one by one, each of the strategies, by observing the states of the nodes that represent the original decision nodes, solves the resulting BN using any belief-updating algorithm, and retrieves the expected utility of the current strategy.

Shachter and Peot [28] designed an efficient variation on Cooper's algorithm for IDs that have one decision node and one utility node. After the transformation of the ID into a BN, the algorithm essentially introduces evidence into the value node, solves the BN, and chooses the best decision alternative by comparing the posterior probability of the outcomes of the decision node: the most likely outcome indicates the best decision option. The algorithm is focused, as it identifies only the best decision. A separate, optional step of the algorithm is required to calculate the expected utility of that decision. Zhang [34] describes another focused indirect evaluation algorithm that is an improvement on Shachter and Peot's algorithm. It successively divides the ID into two parts, head and a tail, the tail containing the last decision node and all those other nodes that are relevant to the maximization of that last decision. It then computes the expected utility of the tail and finds the best option for the last decision. The tail, including the last decision node, is then substituted by a single utility node that summarizes all the uncertainty contained in it. The procedure is repeated until no more decision nodes are left. One advantage of this algorithm is that it can be based on any belief-updating algorithm for BNs. Other techniques to solve influence diagrams were offered by Jensen, Jensen and Dittmer [19] and Madsen and Jensen [21].

Another line of research on algorithms for IDs was started by Horvitz [17] who identified a class of inference policies for probabilistic reasoning systems that he called incremental refinement policies. He presented algorithms that improve the accuracy of their solutions as a monotonically increasing function of the allocated resources and the available information. Several algorithms have been developed along these lines. Ramoni [26] proposed an anytime algorithm for solving IDs based on Epistemic Constraint Propagation [27] that incrementally refines the confidence in the expected utility of the decisions. One limitation of this algorithm is that its anytime property holds only for IDs with one decision node. Horsch and Poole [15] developed another anytime algorithm that constructs a tree structure for each decision node in the ID. Each of the constructed tree structures represents a decision function that is improved incrementally. The authors show how improvements to this function lead to the optimal decision function. Similarly to Ramoni's algorithm, the algorithm loses its anytime properties if applied to decision problems with more than one decision node. A refinement of this algorithm, focusing on multi-stage influence diagrams was published later [16]. Charnes and Shenoy [1] propose an approximate sampling algorithm for solving IDs. D'Ambrosio and Burgess

[10] performed experiments comparing a variety of real-time influence diagram algorithms.

This paper shows how sampling algorithms can be utilized very efficiently within an indirect evaluation method. Our approach evaluates all decision strategies on the same set of random samples and allows for saving a significant amount of computation while producing high quality anytime behavior. The algorithm that we propose is suitable for very large decision models for which exact algorithms are not feasible. Recent advances in stochastic sampling algorithms for BNs (e.g., [2; 33]) have made it possible to perform inference in very large models with unlikely evidence. Because our algorithm can be based on any simulation algorithm for BNs, it allows for taking advantage of these developments. The algorithm is general in the sense of admitting full IDs with multiple decision and value nodes and containing observations. As a side-product, the algorithm computes also the posterior probability distribution of every chance node conditional on each of the possible strategies. For example, in a system that helps to decide between medication and surgery, the algorithm will compute not only the expected utility of both choices, but also the probability of the patient dying in each case. We have found that this information is useful in an interactive modeling environment. The algorithm is, to our knowledge, the only algorithm developed for ID that exhibits anytime behavior when the ID has an arbitrary number of decision and utility nodes. Finally, the algorithm is very efficient and this is achieved by reusing random samples. Practically, only one random number per node is generated for each sample of the network. This random number is used to evaluate the probability distribution of the node conditional on each of the strategies.

All random variables used in this paper are multiple-valued, discrete variables. Lower case letters, such as a, b, or c denote random variables. Bold capital letters, such as $\mathbf{A}$, denote sets of variables. Bold capital $\mathbf{E}$ denotes the set of observed nodes and capital $\mathbf{Q}$ denotes the set of other nodes not contained in $\mathbf{E}$. For any node n, $\mathrm{Pa}(n)$ denotes the set of parents of n, $\mathrm{De}(n)$ denotes the descendants of n, and $\mathrm{O}(n)$ denotes the set of outcomes of n.

The remainder of this paper is organized as follows. Section 2 defines some of the terms that we will be using throughout this paper. Section 3 describes the proposed algorithm. Section 4 contains a detailed example that will serve us in explaining the algorithm. Section 5 reports the results of our experiment with the algorithm, and Section 6 contains concluding remarks.

2 Some Definitions

A Bayesian network $\mathcal{B}$ is defined as a pair $\mathcal{B} = (\mathcal{G}, \Pr)$, where $\mathcal{G}(\mathbf{C}, \mathbf{A})$ is an acyclic directed graph consisting of a set of nodes $\mathbf{C}$ and a set of directed arcs $\mathbf{A}$. Each node $n \in \mathbf{C}$ has associated a conditional probability distribution, specifying the probability of each outcome $o \in \mathrm{O}(n)$ conditional on $\mathrm{Pa}(n)$.

Similarly, we denote the joint probability distribution over a set of nodes $\mathbf{J} \subseteq \mathbf{C}$ as $\Pr(\mathbf{X}_J)$.

Given $\mathcal{B} = (\mathbf{C}, \mathbf{A})$ and sets $\mathbf{J}$, $\mathbf{K}$, $\mathbf{L} \subseteq \mathbf{C}$, we say that $\mathbf{X}_L$ is irrelevant [13] to $\mathbf{X}_J$ given $\mathbf{X}_K$ if $\Pr(\mathbf{X}_J|\mathbf{X}_K, \mathbf{X}_L) = \Pr(\mathbf{X}_J|\mathbf{X}_K)$. In other words, $\mathbf{X}_L$ is irrelevant to $\mathbf{X}_J$ given $\mathbf{X}_K$ if, once $\mathbf{X}_K$ has been observed, we cannot learn anything about $\mathbf{X}_J$ by also observing $\mathbf{X}_L$. We extend the notion of irrelevance to decision nodes in the following way. Given an ID model $\mathcal{M} = (\mathbf{C}, \mathbf{A}, \mathbf{D}, \mathbf{U})$ and sets $\mathbf{L} \subseteq \mathbf{D}$ and $\mathbf{J}$, $\mathbf{K} \subseteq \mathbf{C}$, we say that $\mathbf{L}$ is irrelevant to $\mathbf{X}_J$ given $\mathbf{X}_K$ if $\Pr(\mathbf{X}_J|\mathbf{X}_K, \mathbf{L}) = \Pr(\mathbf{X}_J|\mathbf{X}_K)$. In other words, L is irrelevant to $\mathbf{X}_J$ given $\mathbf{X}_K$ if, once $\mathbf{X}_K$ has been observed, making a decision in L does not impact $\mathbf{X}_J$. Irrelevance for utility nodes $\mathbf{U}$ is defined analogously but with respect to expected utility rather than probability.

An ID $\mathcal{M} = (\mathbf{C}, \mathbf{A}, \mathbf{D}, \mathbf{U})$ is a BN extended with $\mathbf{D}$, a set of decision nodes, and $\mathbf{U}$, a set of utility nodes. The decision nodes are ordered in time, $d_1, \ldots, d_n$, and chance nodes are stratified into sets $\mathbf{E}$, $\mathbf{I}_1$, $\ldots$, $\mathbf{I}_{n+1}$, so that chance nodes $\mathbf{I}_i$ are observed before d_i but after d_{i-1}. The nodes in $\mathbf{E}$ have been observed a-priori while the nodes in $\mathbf{I}_{n+1}$ will never be observed. Let $\mathbf{I} = (\mathbf{I}_1, \ldots, \mathbf{I}_n)$ be nodes that are not evidence but that will be observed before the last decision d_n is made.[1] From here on, we assume, without loss of generality, that $\mathbf{E} = \emptyset$. Dealing with evidence in sampling algorithms is quite well understood and is not the focus of this paper. For convenience, we will sometimes say $n \in \mathcal{M}$ if $n \in \{\mathbf{C} \cup \mathbf{D} \cup \mathbf{U}\}$.

Let us define a partial order Ψ_N over a set of nodes $\mathbf{N} \subseteq \mathcal{M}$ such that, given nodes $n, m \in \mathbf{N}$, if $m \in \mathrm{De}(n)$ then n appears in Ψ_N before m. In other words, in Ψ_N a node appears always before any of its descendants.

We say that $d \in (\mathbf{D}, \mathbf{I})$ is an indexing parent of $n \in \mathcal{M}$ if it precedes n in $\Psi_{(\mathbf{D},\mathbf{C},\mathbf{U})}$ and it is relevant to n. We denote by $\mathrm{IP}(n)$ the set of indexing parents of node n. Less formally, the set of indexing parents of a node n is composed of those decision nodes and those chance nodes that will be observed before making a decision, that have an effect on the distribution of the outcomes of n.

A *strategy* is an instantiation of every node $d \in \mathbf{D}$, conditional on $\mathbf{I}$. We will sometimes use the term strategy when making reference to an instantiation of the set of indexing parents of a node.

For each node $n \in \mathcal{M}$ we can define the Cartesian product of the set of outcomes of its indexing parents:

$$\Phi_n = \times_{p \in \mathrm{IP}(n)} \mathrm{O}(p) \ .$$

In other words, Φ_n contains all the possible combinations of the outcomes of n's indexing parents. Similarly, Φ is the set of indexing parents of the global utility node

$$\Phi = \times_{p \in \bigcup_{u_i \in \mathbf{U}} \mathrm{IP}(u_i)} \mathrm{O}(p) \ .$$

[1] We normally indicate this order graphically by having an informational (dashed) arc from every $n \in \mathbf{I}_i$ into d_i.

We will now define the cardinality operator, $\|\Phi_n\|$, in the following way

$$\|\Phi_n\| = \begin{cases} |\Phi_n|, & \Phi_n \neq \emptyset \\ 1, & \Phi_n = \emptyset \end{cases} .$$

This operator, when applied to any set Φ_n, will return the number of elements in Φ_n or 1 if Φ_n is empty. We will now prove a theorem stating that every indexing parent of a node p is also an indexing parent of p's children.

Theorem 1. *Given an ID* $\mathcal{M} = (\mathbf{C}, \mathbf{A}, \mathbf{D}, \mathbf{U})$ *and nodes* $c, p \in \{\mathbf{C}, \mathbf{D}, \mathbf{U}\}$*, if* $p \in Pa(c)$ *then* $IP(p) \subseteq IP(c)$*.*

Proof. By definition, every node $n \in \mathrm{IP}(p)$ is relevant to p. Since $p \in \mathrm{Pa}(c)$, it follows that n must also be relevant to c and, hence, be an indexing parent of c.

3 The Algorithm

This algorithm solves an ID $\mathcal{M} = (\mathbf{C}, \mathbf{A}, \mathbf{D}, \mathbf{U})$, which means that it calculates the expected utility of each possible strategy and also calculates the distribution of each chance node conditional on each of the strategies. As one of its main features, the algorithm is capable of giving an approximate answer at any time. We describe this algorithm in Fig. 1.

Cooper's transformation in Step [1.1] is a two-step operation consisting of replacing all decision nodes and all utility nodes with chance nodes. Transformation of each utility node $u \in \mathbf{U}$ into a chance node, in particular, is performed by transforming the utility function $V(u) = F(p_1, \ldots, p_n)$, $p_1, \ldots, p_n \in \mathrm{Pa}(u)$ into a probability distribution by linearly mapping the range of F onto the interval [0,1].

In order to achieve [1.2], we can use an algorithm like the one described in Druzdzel & Suermondt [11], Lin & Druzdzel [20] or Shachter [29]. Basically, for a given node $n \in \mathcal{M}$, it traverses the ID, starting from n, marking each d-connected node it finds along the way. Once the traversing is done, those nodes marked are added to the set $\mathrm{IP}(n)$.

In Step [3], we iterate only through chance nodes, which means that decision nodes are ignored at this stage. Remember that utility nodes have been transformed into chance nodes in Step [1.1]. The Steps [3.1], [3.2.1], [3.2.2], and [3.2.3] basically follow any forward sampling scheme, such as probabilistic logic sampling [14], likelihood weighting [12; 31], importance sampling [31], Latin hypercube sampling [3], adaptive importance sampling [2], or estimated posterior importance sampling [33]. This produces an unbiased estimator of the distribution of node c. But also note that in [3.2] we are iterating through every $\delta_k \in \Phi_c$ and that in Steps [3.2.1] through [3.2.3] we are assuming 'given δ_k'. The net result of Step [3] is the distribution of each chance node c conditional on each strategy relevant to c:

Input: An ID $\mathcal{M} = (\mathbf{C}, \mathbf{A}, \mathbf{D}, \mathbf{U})$
Output: For every strategy $\delta_k \in \Phi$ expected utility of δ_k.
(1) For each node $n \in \mathcal{M}$
  (1.1) If $n \in \mathbf{U}$, apply Cooper's transformation to n and add n to $\mathbf{C}$
  (1.2) Find IP(n)
  (1.3) Find $\|\Phi_n\|$
(2) Determine Ψ_M, a partial order over $\mathcal{M}$
(3) For each node $c \in \mathbf{C}$ (in Ψ_M order)
  (3.1) Generate a random number
  (3.2) For each strategy $\delta_k \in \Phi_c$ (or once if $\Phi_c = \emptyset$)
    (3.2.1) Look up the outcome of every node $p \in \text{Pa}(c)$ given δ_k
    (3.2.2) Instantiate the outcome of c given δ_k
    (3.2.3) Give a score and annotate the outcome of c given δ_k
(4) If you wish to terminate the algorithm, proceed to Step 5.
  Otherwise jump to Step 3.
(5) Normalize all $\Pr(c), c \in \mathbf{C}$
(6) For each node $u \in \mathbf{U}$
  (6.1) For each $\delta_k \in \Phi_u$ (or once if $\Phi_u = \emptyset$)
    (6.1.1) Compute $EU(u|\delta_k)$, $\delta_k \in \Phi_u$ by reversing Step 1.1
(7) For each $\delta_k \in \Phi$
  (7.1) Compute the global expected utility given δ_k.
(8) Choose the best strategy.

Fig. 1. The basic algorithm.

$$\Pr(c_i|\delta_k), c_i \in \mathbf{C}, \delta_k \in \Phi_c \; . \tag{1}$$

In [3.2.1], we say "Look up the outcome of every node $p \in \text{Pa}(c)$ given δ_k." What this "outcome" means depends on the type of the parent. If $p \in \text{IP}(c)$, "outcome" means state of p in the current δ_k. On the other hand, if $p \notin \text{IP}(c)$, "outcome" means the outcome of p that we computed and annotated in Step [3.2.3]. The fact that we are processing nodes in the order specified by Ψ_M guarantees that p has been dealt with before c, so the outcome of p is available to us. Theorem 1 guarantees that we can always find the state of node p given the current δ_k.

Also, note that we are evaluating in parallel every $\delta_k \in \Phi_c$, reusing the random number generated in [3.1], to compute the distribution of c conditional on every strategy. A complete sample, which is a collection of node outcomes, one for every node, describes a state of the world that we effectively test our decision model on. We use the same state of the world to test each of the possible decision strategies, but we do it in parallel as we continue sampling. This sample reuse achieves two important objectives: (1) it saves a large number of calls to the random number generator, and (2) with every

sample generated, it performs the evaluation of all strategies, which is a key to achieving anytime behavior.

The simulation can be interrupted in Step [4], at which point the algorithm yields its best estimate of the expected utilities of each of the strategies.

If we apply an algorithm with a stopping rule, such as the *bounded variance algorithm* [9], the AA algorithm [8], or the AIS-BN-μ / AIS-BN-σ algorithms [4], we may be able to produce an estimate of the error measure for each of the strategies. If the error bounds around the posterior distributions of the transformed utility nodes are small enough, we may even return an exact ordering of the decision strategies.

Step [6] performs reverse of the transformation applied in [1.1] to utility nodes. Utility nodes, indexed by their indexing parents, contain at this point the expected utility of each possible strategy:

$$EU(u_i|\delta_k),\ u_i \in \mathbf{U},\ \delta_k \in \Phi_u\ . \tag{2}$$

Step [7] is meant to deal with those cases in which $\mathbf{U}$ has more than one utility node. In this case, a global multi-attribute utility function of the form

$$\mathrm{GEU}(u_1, u_2, \ldots, u_n) = f(U_1(u_1), U_2(u_2), \ldots, U_n(u_n)),\quad u_i \in \mathbf{U}$$

should be provided (typically, it is a part of the model). This function is intended to produce a global utility and, for linearly additive utility functions, for example, it is defined as

$$w_1\, U_1(u_1) + w_2\, U_2(u_2) + \ldots + w_n\, U_n(u_n)\ ,\quad u_i \in \mathbf{U}\ .$$

Our algorithm does not depend in any way on the actual form of this multi-attribute utility function. Finally, we identify the strategy $\delta_k \in \Phi$ that maximizes $\mathrm{GEU}(\delta_k)$.

4 An Example

To facilitate understanding of the algorithm, we are going to introduce a simple ID that we will use to illustrate every step of the algorithm described in the previous section. The ID models the following decision problem.

> A group of terrorists have taken control of an airplane and have threatened to kill all passengers and themselves, if their demands are not met. Information is scarce, the nationality of the assailants is unknown and there exists a possibility of them having explosives, which may make any rescue attempt risky. The authorities are facing two decisions. First, they need to determine whether they should negotiate or not. Second, they need to decide whether to deploy a SWAT team to assault the airplane. Although there is a lot of uncertainty, the authorities have reduced the possible nationalities of the terrorists down to two, *CountryA* and *CountryB*.

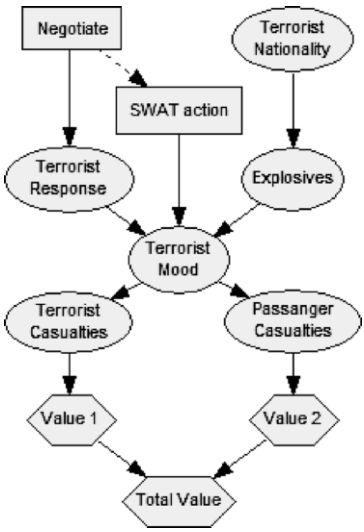

Fig. 2. The hijacked plane ID.

Figure 2 shows the ID for this problem.

Table 1 shows the short labels that we will use in the sequel when referring to the nodes in the model. It also shows the number of outcomes of each of the nodes.

Table 1. The label and the number of outcomes for each node.

Long name	Label	Number of outcomes
Negotiate	*Neg*	2
Terrorists Nationality	*Nat*	2
Swat Action	*Swat*	2
Terrorists Mood	*Mood*	2
Explosives	*Exp*	2
Terrorists Response	*TRes*	3
Terrorist Casualties	*TCas*	2
Passenger Casualties	*PCas*	3
Value 1	*Val1*	–
Value 2	*Val2*	–

We will now walk step-for-step through the algorithm.

Step (1): Initialization

Below we will show, as an example, Cooper's transformation applied to node *Val2*. This node encodes the preferences of the decision-maker with respect to passenger casualties. For simplicity, the decision-maker has considered

only three possible outcomes: *All*, *Half*, or *None* of the passengers die and encoded her preferences in the node *Val2* by means of the following utility function:

$$U(\mathit{Val2}|\mathit{PCas} = \mathit{None}) = 100$$
$$U(\mathit{Val2}|\mathit{PCas} = \mathit{Half}) = 25$$
$$U(\mathit{Val2}|\mathit{PCas} = \mathit{All}) = 0$$

We apply a linear transformation to this utility function (please note that utility is invariant to linear transformations) mapping it to the interval $[0, 1]$:

$$\Pr(\mathit{Val2}|\mathit{PCas} = \mathit{None}) = 1.0$$
$$\Pr(\mathit{Val2}|\mathit{PCas} = \mathit{Half}) = 0.25$$
$$\Pr(\mathit{Val2}|\mathit{PCas} = \mathit{All}) = 0$$

After applying similar transformation to *Val1*, we add both *Val1* and *Val2* to the set of chance nodes. Table 2 summarizes the results of computing the indexing parents (Step [1.2]).

Table 2. Some initial values computed.

Node	IP(node)	$\|\Phi_{node}\|$
Neg	$\emptyset$	–
Nat	$\emptyset$	1
Swat	$\emptyset$	–
Mood	{ *Neg* }	2
Exp	$\emptyset$	1
TRes	{ *Neg*, *Swat* }	4
TCas	{ *Neg*, *Swat* }	4
PCas	{ *Neg*, *Swat* }	4
Val1	{ *Neg*, *Swat* }	4
Val2	{ *Neg*, *Swat* }	4

Step (2): Determine Ψ_M, a partial order over $\mathcal{M}$.
There are many valid partial orders for $\mathcal{M}$. We will use the following

$$\Psi_M = (Neg, Nat, Swat, Exp, Mood, TRes, TCas, PCas, Val1, Val2) \ .$$

Step (3): Take one sample of the network.

Generating a stochastic sample in a BN amounts to randomly instantiating every node to one of its possible states, according to the probability distribution over the node's states, conditional the instantiated states of its parents. In case of forward sampling, this requires every instantiation to be performed in the topological order, i.e., parents are sampled before their children. Supposed we pick the first chance node in Ψ_M, *Nat*, and generate a

random number. Since $\Phi_{Nat} = \emptyset$, we do not have any strategies to iterate through. This means that we proceed as in a plain sampling algorithm for BNs: we generate a random number R and then we pick a state indicated by R, depending on the a-priori probability distribution over *Nat*, encoded in its CPT. Let this state be $Nat = CountryA$. We do the same to the next chance node, *Exp*, and generate $Exp = Yes$.

The next chance node in Ψ_M with a non-empty Φ_{Mood} is *Mood*. [3.1] First we generate a random number, say $R = 0.65$. [3.2] Now we start iterating through all $\delta \in \Phi_{Mood}$. We know that $\Phi_{Mood} = \{(Neg{=}Yes), (Neg{=}No)\}$, so the first δ is $(Neg{=}Yes)$. [3.2.1] We need now to find the outcome of each of *Mood*'s parents, only *Neg* in this case. Since $Neg \in \mathrm{IP}(Mood)$, its outcome is taken from the current δ, i.e., $Neg = Yes$.

Assume that the CPT of *Mood* looks as illustrated in Fig. 3–a. [3.2.2] Looking at the probabilities in the first column, which shows $\Pr(Mood|Neg = Yes)$, we choose $Mood = Good$. [3.2.3] Annotate this result in a table similar to the one shown in Fig. 3–b.

In [3.2], we now pick the next element of Φ_{Mood}, which is $\delta = (Neg = No)$, and repeat the above process. In [3.2.1] the current δ tells us that $Neg = No$, since $Neg \in \mathrm{IP}(Mood)$. [3.2.2] Looking at the probabilities in the second column of Fig. 3–a, which shows $\Pr(Mood|Neg = No)$, we find that $Mood = Bad$. [3.2.3] Annotate this result. Figure 3–b shows the state at this point.

Note in Fig. 3–b that we always assign a score of 1 to the sampled outcome. The reason for this, as noted before, is that we assume that there is no prior evidence in the network. In case there is evidence, we need to weigh these scores.

Neg	Yes	No
Good	0.7	0.1
Bad	0.3	0.9

(a)

Neg	Yes	No
Good	1	0
Bad	0	1
OUTCOME	Good	Bad

(b)

Fig. 3. (a) CPT of node *Mood*. (b) Table to score samples and to annotate current outcome. Note that this table is indexed by the indexing parents of *Mood*.

At this point, we have completed sampling from the node *Mood*. There are two remarks that we would like to make at this point. First, note that counting the samples within individual chance nodes allows us to calculate the posterior probability distribution of that node conditional on the strategy applied. We alluded to this earlier, when we talked about evaluating all strategies in parallel. Second, we are reusing the same random number R for all $\|\Phi_{FMood}\|$ strategies, which saves many calls to the random number generator.

We go back to Step 3 and pick the next chance node in Ψ_M, *TRes*. Something slightly more complex happens with this node. [3.1] We generate a new random number, say $R = 0.32$. [3.2] Again, we know that $\Phi_{TRes} = \{(Neg{=}Yes, Swat{=}None), (Neg{=}Yes, Swat{=}Attack), (Neg{=}No, Swat{=}None), (Neg{=}No, Swat{=}Attack)\}$. This means that our first strategy is $\delta = (Neg = Yes, Swat = None)$. [3.2.1] The parents of *TRes* are *Mood*, *Swat*, and *Exp*. Since $Swat \in \mathrm{IP}(TRes)$, its outcome is taken, as before, from the current δ, i.e., *Swat=None*. Things get just a little more complex when it comes to finding the outcome of *Mood*, since $Mood \notin \mathrm{IP}(TRes)$. This is because *Mood*, as shown in Fig. 3–b, has two sampled outcomes, not just one. We pick the one whose configuration of indexing parents matches the configuration of the current δ.

Swat	None				Attack			
Exp	No		Yes		No		Yes	
Mood	Good	Bad	Good	Bad	Good	Bad	Good	Bad
Liberate	0.8	0.4	0.8	0.4	0.25	0.1	0.25	0.1
Fight	0.2	0.6	0.15	0.5	0.75	0.9	0.65	0.7
Boom	0	0	0.05	0.1	0	0	0.1	0.2

Fig. 4. CPT of node TResponse.

Indeed, what we want to compute now is $\Pr(TRes|Neg = Yes, Swat = None)$, so it follows that we need to consider the distribution $\Pr(Mood|Neg = Yes, Swat = None)$. Since we have determined in Step [1.1] that *Mood* is independent of *Swat*, $\Pr(Mood|Neg = Yes, Swat = None) = \Pr(Mood|Neg = Yes)$. And since *Neg=Yes* in the current δ, we must pick the outcome of *Mood* from the column indexed by *Neg=Yes* in Fig. 3–b. This yields *Mood=Good*.

We take the same approach to find the outcome of the last parent, *Exp*, but in this case everything seems to be easier since *Exp* is independent from both *Neg* and *Swat* and has only one sampled outcome: *Exp=Yes*. Looking at the third column of Fig. 5, which shows $\Pr(TRes|Swat = None, Exp = Yes, Mood = Good)$, we obtain $TRes = Liberate$.

Neg	Yes		No	
Swat	None	Attack	None	Attack
Liberate	1	0	1	1
Fight	0	1	0	0
Boom	0	0	0	0
OUTCOME	Liberate	Fight	Liberate	Liberate

Fig. 5. Table to score samples and to annotate current outcome of node *TRes*.

[3.2] We now pick the next strategy, $\delta = (Neg{=}Yes,\ Swat{=}Attack)$. [3.2.1] From δ, $Swat{=}Attack$. Exp is still Yes. Since $Neg{=}Yes$ again, $Mood{=}Good$ for the same reasons explained above. [3.2.2] Looking at the seventh column of Fig. 5, which shows $\Pr(TRes|Swat = Attack, Exp = Yes, Mood = Good)$, we obtain $TRes{=}Fight$. [3.2.3] Annotate and pick next strategy. [3.2] $\delta{=}$ ($Neg{=}No$, $Swat{=}None$). [3.2.1] From δ, $Swat{=}None$. Exp is still Yes. Since now $Neg{=}No$, Fig. 3–b tells us that $Mood{=}Bad$. [3.2.2] Looking at the fourth column of Fig. 4, which shows $\Pr(TRes|Swat = None, Exp = Yes, Mood = Bad)$, we obtain $TRes$=Liberate. [3.2.3]. Back to [3.2], the last $\delta = (Neg{=}No,\ Swat{=}Attack)$. Note that we are still reusing the same random number $R = 0.32$, which, looking at column 8 in Fig. 4, yields $TRes{=}Fight$. We are done with node $TRes$, and Fig. 5 shows the state at this point.

We repeat this process for all nodes in the network. At this point the reader should have an intuitive idea of how the algorithm works. Basically, we choose a state for every chance node until there are no more nodes left in Ψ_M, at which point we would have completed one sample of the whole network.

Step (4): Decide if we should take another sample.

As said before, we need a means to decide whether to terminate the algorithm. The more samples we take, the more accurate the solution will be. If we decide to terminate, we just go on to Step [5] below to do some final calculations.

Step (5): Normalize distributions of chance nodes.

Normalize each of the distributions $\Pr(c_i|\delta_k)$, $c_i \in \mathbf{C}$, $\delta_k \in \Phi_c$ so that they add up to 1. This corresponds to the columns of the tables containing the scores of the outcomes of each node, like those shown in Figs. 3–b and 5.

Step (6): Compute expected utility of each utility node.

Here we just need to reverse the linear transformations of the utility functions that we performed in [1.1]. This step yields $V(Val1|\delta_k)$, $k \in \Phi_{Val1}$ and $V(Val2|\delta_k)$, $\delta_k \delta_{\in} \Phi_{Val2}$.

Step (7): Compute the global expected utility.

Since we have two utility nodes, $Val1$ and $Val2$, we need a function that combines both and returns a global expected utility for the whole model. Let us assume for simplicity that it is a linear combination of the individual utility functions with unit weights, i.e.,

$$\mathrm{GEU}(Val1, Val2) = U_1(Val1) + U_2(Val2)\ .$$

As before, we iterate through all strategies $\delta \in \Phi$ and compute GEU conditional on every strategy. In other words, we compute:

$$\mathrm{GEU}(\delta_k) = U_1(Val1|\delta_k) + U_2(Val2|\delta_k), \delta_k \in \Phi\ .$$

Step (8): Choose the best strategy.

Choose the strategy $\delta_k \in \Phi$ that maximizes $\mathrm{GEU}(\delta_k)$ calculated in Step [7].

5 Empirical Illustration

It is quite obvious that the proposed algorithm is more efficient in terms of its use of samples and in terms of reducing the variance in results than an algorithm that exhaustively samples all strategies. We illustrate this on HEPAR II [24], a model for diagnosis of liver disorders consisting of 73 nodes and available in Decision Systems Laboratory's model repository at `http://genie.sis.pitt.edu/`. Since HEPAR II is a BN, we first performed a modification that transformed it into an ID. Basically, HEPAR II consists of nodes representing risk factors, liver disorders, medical tests, and symptoms. The structure of the graph is causal: arcs go from risk factors to disorders and from disorders to tests and symptoms. First, we designated four nodes from among the risk factors to act as decision nodes. These decision nodes represented hypothetical choices a person could have about certain risks factors. For example, alcohol abuse can be considered a risk factor for several liver disorders. Converting this chance node into a decision node represents the choice a person has with respect to drinking. Since each of the transformed decision nodes had two states, we ended up with 16 different strategies to evaluate. The second step was to add utility nodes. We decided to add one utility node to each of the disorder nodes such that the utility of not having a particular disorder was 100 while the utility of having the disorder was 0. For simplicity, we assumed that all disorders as equally severe and created an additive linear multi-attribute utility function with weights $w_i = 1.0$ for each of the individual utility nodes. With a total of eight disorders in the model, the multi-attribute utility function ranged from 0, when a patient suffers from all eight disorders, to 800, when a patient suffers from no disorders at all.

We compared our algorithm against indirect exhaustive algorithm, with both algorithms based on likelihood weighting as their underlying BN algorithm. We ran 1,000,000 samples of the network, taking measurements of intermediate values of expected utility for each of the 16 possible strategies.

Figure 6 shows the expected utility of each of the 16 strategies as a function of the number of samples. In the simulation, the range of expected utilities goes from 500 up to 800 (each horizontal line on the graph represents an increment of 50). This wide range is due to the variations in the estimates that we obtained during the first one hundred samples. Crossing of trajectories means essentially a change in the ranking of the strategies. The expected utilities converge to their exact values (verified by means of an exact algorithm) as the number of samples increases. Figure 7 shows the same plot for the algorithm proposed in this paper. In this simulation, the range of expected utilities goes from 550 up to 800 (each horizontal line also represents an increment of 50). The expected utilities converge to exactly the same values (670 for the worst strategy and 715 for the best one).

Both figures show individual trajectories shifting up and down, but while in the crude exhaustive algorithm each trajectory seems to do so independently of the others, in our algorithm they all shift roughly at the same time

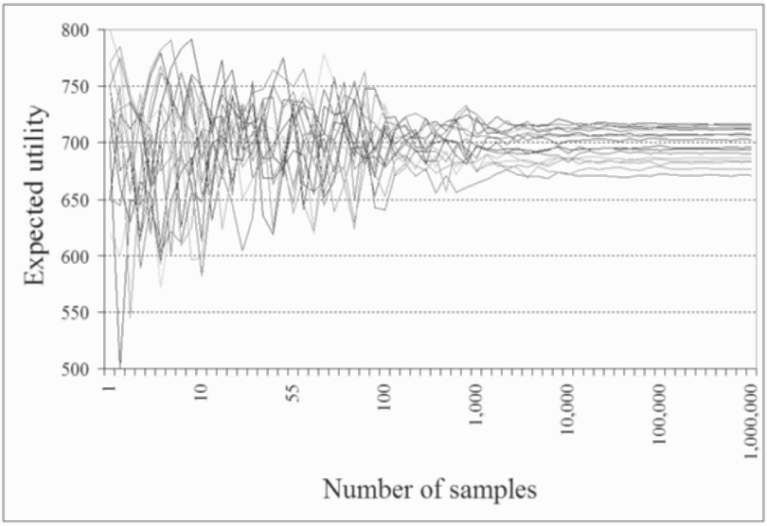

Fig. 6. Expected utility as a function of the number of samples for an indirect exhaustive algorithm based on likelihood weighting.

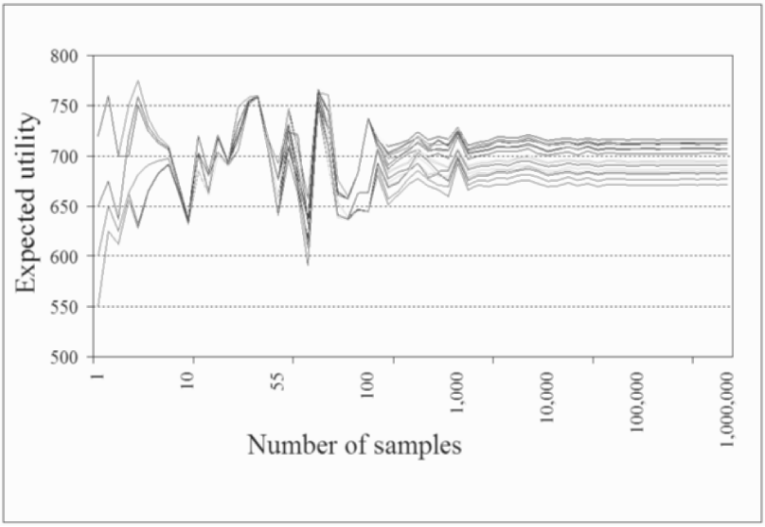

Fig. 7. Expected utility as a function of the number of samples for the exhaustive algorithm based on likelihood weighting proposed in this paper.

and in the same direction. This is a simple consequence of the fact that our simulation tests each strategy against the same sample, i.e., on the same randomly generated state of the world. This greatly reduces variance and our algorithm can rank the strategies after processing a relatively small number

of samples. While in Fig. 6 we can find trajectory lines crossing even after as many as 90,000 samples taken, in Fig. 7 the last crossing of trajectory lines occurs at only 900 samples.

Reuse of samples for each of the 16 strategies reduced the number of calls to the random number generator by 16-fold. There was a 20% to 30% computational time overhead related to reuse, resulting in roughly a 12-fold reduction of the computation time of our algorithm compared to the exhaustive algorithm based on likelihood weighting.

6 Conclusion

We introduced an approximate anytime sampling algorithm for IDs that computes the expected utilities of all decision strategies. The algorithm is indirect, in the sense of reducing the problem of solving an ID by first transforming it into a BN. We have shown how by evaluation of each of the strategies on the same set of samples we not only save much computation but also reduce variance and, hence, produce high quality anytime behavior. The proposed algorithm is, furthermore, amenable to parallelization and a possible further increase in speed.

Our algorithm accommodates any forward sampling scheme. A simple empirical test of the algorithm has shown that it rapidly produces the correct order of strategies. We expect that combining this algorithm with stopping rules (e.g., [4]) will lead to efficient algorithms for IDs that give precision guarantees with respect to both the order and numerical expected utilities of strategies.

Acknowledgments

This research was supported by the National Science Foundation under Faculty Early Career Development (CAREER) Program, grant IRI–9624629, and by the Air Force Office of Scientific Research, grant F49620–03–1–0187. All experimental data have been obtained using SMILE, a Bayesian inference engine developed at the Decision Systems Laboratory and available at `http://genie.sis.pitt.edu/`. While we assume full responsibility for any errors remaining in this paper, we would like to thank our colleagues in the Decision Systems Laboratory for their ideas, comments, and suggestions. Comments from reviewers for the Second European Workshop on Probabilistic Graphical Models (PGM-04), where we presented an earlier version of this paper, helped us to improve the clarity of our presentation.

References

[1] John M. Charnes and Prakash P. Shenoy. Multi-stage Monte Carlo method for solving influence diagrams using local computation. *Management Science*, 50(3):405–418, 2004.

[2] Jian Cheng and Marek J. Druzdzel. BN-AIS: An adaptive importance sampling algorithm for evidential reasoning in large Bayesian networks. *Journal of Artificial Intelligence Research*, 13:155–188, 2000.

[3] Jian Cheng and Marek J. Druzdzel. Latin hypercube sampling in Bayesian networks. In *Proceedings of the 13th International Florida Artificial Intelligence Research Symposium Conference (FLAIRS–2000)*, pages 287–292, Menlo Park, CA, May 2000. AAAI Press/The MIT Press.

[4] Jian Cheng and Marek J. Druzdzel. Confidence inference in Bayesian networks. In *Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–2001)*, pages 75–82, San Francisco, CA, 2001. Morgan Kaufmann Publishers, Inc.

[5] Robert T. Clemen. *Making Hard Decisions: An Introduction to Decision Analysis*. Duxbury Press, An Imprint of Wadsworth Publishing Company, Belmont, California, 1996.

[6] Gregory F. Cooper. A method for using belief network algorithms to solve decision-network problems. In *Proceedings of the 1988 Workshop on Uncertainty in Artificial Intelligence, UAI–88*, Minneapolis, Minnesota, 1988.

[7] Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2–3):393–405, March 1990.

[8] Paul Dagum, Richard Karp, Michael Luby, and Sheldon Ross. An optimal algorithm for Monte Carlo estimation. *SIAM Journal on computing*, 29(5):1481–1496, 2000.

[9] Paul Dagum and Michael Luby. An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence*, 93:1–27, 1997.

[10] Bruce D'Ambrosio and Scott Burgess. Some experiments with real-time decision algorithms. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pages 194–202, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

[11] Marek J. Druzdzel and Henri J. Suermondt. Relevance in probabilistic models: "Backyards" in a "small world". In *Working notes of the AAAI–1994 Fall Symposium Series: Relevance*, pages 60–63, New Orleans, LA (An extended version of this paper is in preparation.), 4–6 November 1994.

[12] Robert Fung and Kuo-Chu Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In M. Henrion, R.D. Shachter, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 209–219. Elsevier Science Publishing Company, Inc., New York, N. Y., 1989.

[13] Dan Geiger, Thomas S. Verma, and Judea Pearl. Identifying independence in Bayesian networks. *Networks*, 20(5):507–534, August 1990.

[14] Max Henrion. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In L.N. Kanal, T.S. Levitt, and J.F. Lemmer, editors, *Uncertainty in Artificial Intellgience 2*, pages 149–163. Elsevier Science Publishing Company, Inc., New York, N. Y., 1988.

[15] Michael C. Horsch and David Poole. Flexible policy construction by information refinement. In *Proceedings of the Twelfth Annual Conference on Uncertainty in Artificial Intelligence (UAI–96)*, pages 315–324, San Francisco, CA, 1996. Morgan Kaufmann Publishers.

[16] Michael C. Horsch and David Poole. An anytime algorithm for decision making under uncertainty. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pages 246–255, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[17] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, IJCAI–89*, pages 1121–1127, San MAteo, CA, 1989. Morgan Kauffman.

[18] Ronald A. Howard and James E. Matheson. Influence diagrams. In Ronald A. Howard and James E. Matheson, editors, *The Principles and Applications of Decision Analysis*, pages 719–762. Strategic Decisions Group, Menlo Park, CA, 1984.

[19] Frank Jensen, Finn V. Jensen, and Søren L. Dittmer. From influence diagrams to junction trees. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–94)*, pages 367–373, San Francisco, CA, 1994. Morgan Kaufmann Publishers.

[20] Yan Lin and Marek J. Druzdzel. Computational advantages of relevance reasoning in Bayesian belief networks. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–97)*, pages 342–350, San Francisco, CA, 1997. Morgan Kaufmann Publishers, Inc.

[21] Anders L. Madsen and Finn V. Jensen. Lazy evaluation of symmetric Bayesian decision problems. In *Proceedings of the Fifteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–99)*, pages 382–390, San Francisco, CA, 1999. Morgan Kaufmann Publishers.

[22] P. Ndilikilikesha. Potential influence diagrams. *International Journal of Approximate Reasoning*, 11:251–285, 1994.

[23] Scott M. Olmsted. *On Representing and Solving Decision Problems*. PhD thesis, Stanford University, Department of Engineering-Economic Systems, Stanford, CA, December 1983.

[24] Agnieszka Oniśko, Marek J. Druzdzel, and Hanna Wasyluk. Learning Bayesian network parameters from small data sets: Application of Noisy-OR gates. *International Journal of Approximate Reasoning*, 27(2):165–182, 2001.

[25] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1988.

[26] Marco Ramoni. Anytime influence diagrams. In *Working notes of the IJCAI–95 workshop on "Anytime Algorithms and Deliberation Scheduling"*, pages 55–62, Montreal, Canada, 1995.

[27] Marco Ramoni and A. Riva. Belief maintenance with probabilistic logic. In *Working notes of the AAAI–93 Fall Symposium on "Automated Deduction in Non-Standard Logics"*, Raleigh, NC, 1993.

[28] R. D. Shachter and M. A. Peot. Decision making using probabilistic inference methods. In *Proceedings of the Eighth Annual Conference on Uncertainty in Artificial Intelligence (UAI–92)*, pages 276–283, San Francisco, CA, 1992. Morgan Kaufmann Publishers.

[29] Ross Shachter. Bayes-Ball: The rational pasttime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pages 48–487, San Francisco, CA, 1998. Morgan Kaufmann Publishers.

[30] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34(6):871–882, November–December 1986.

[31] Ross D. Shachter and Mark A. Peot. Simulation approaches to general probabilistic inference on belief networks. In M. Henrion, R.D. Shachter, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence 5*, pages 221–231. Elsevier Science Publishing Company, Inc., New York, N. Y., 1989.

[32] Prakash P. Shenoy. Valuation-based systems for Bayesian decision analysis. *Operations Research*, 40(3):463–484, 1992.

[33] Changhe Yuan and Marek Druzdzel. An importance sampling algorithm based on evidence pre-propagation. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 624–631, San Francisco, CA, 2003. Morgan Kaufmann Publishers.

[34] N. L. Zhang. Probabilistic inference in influence diagrams. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI–98)*, pages 514–522, San Francisco, CA, 1998. Morgan Kaufmann Publishers.