Simulation-based inference for plan monitoring

Neal Lesh[1], **James Allen**[2]
lesh@merl.com, james@cs.rochester.edu

[1] MERL - A Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139
[2] University of Rochester, Rochester, NY 14627

Abstract

The dynamic execution of plans in uncertain domains requires the ability to infer likely current and future world states from past observations. We cast this task as inference on Dynamic Belief Networks (DBNs) but the resulting networks are difficult to solve with exact methods. We investigate and extend simulation algorithms for approximate inference on Bayesian networks and propose a new algorithm, called Rewind/Replay, for generating a set of simulations weighted by their likelihood given past observations. We validate our algorithm on a DBN containing thousands of variables, which models the spread of wildfire.

Introduction

In domains that contain uncertainty, evidential reasoning can play an important role in plan execution. For example, suppose we are executing an evacuation plan and have received the following messages:

Bus_1: Arrived in Abyss, loading 5 passengers.
Exodus weather: Storm clouds are forming to the east.
Bus_2: Engine overheated on the way to Delta.
Bus_1: Got a flat tire on the way to Barnacle.
Bus_2: Loading 9 passengers.
Bus_1: It is starting to snow in Barnacle.

Given the messages received so far, we might ask questions such as what is the probability that a severe storm will hit Barnacle? Or what is the probability that Bus_1 will get another flat tire? Answers to these questions can be used to improve the plan. We might send storm supplies to Barnacle or send Bus_1 on a longer but better paved road. We might also ask about the likely outcomes of various actions. For example, given the evidence received so far, will the plan be more likely to succeed if we send a helicopter to evacuate the people in Exodus or use Bus_3, as originally planned?

We cast plan monitoring as inference on Bayesian networks, but are interested in planning domains for which the resulting networks are difficult to solve with exact methods. In this paper, we explore the use of stochastic sampling methods for performing plan monitoring. Our objective is an algorithm for quickly generating a set of weighted simulations, where the weight indicates the probability of the simulation given observations made during the partial execution of a plan. We show that a set of weighted simulations can be used to estimate the probability of events and also as a basis for planning future actions based on past evidence.

There are several simulation-based inference algorithms for Bayesian networks, including: likelihood weighting (Shachter and Peot 1989; Fung and Chang 1989), arc reversal (Shachter 1986; Fung and Chang 1989), and Survival Of The Fittest (SOF) (Kanazawa *et al.* 1995). Below, we consider these algorithms and find that none are especially well suited for, or were designed for, plan monitoring. We describe a modification to SOF that improves its performance and introduce a new algorithm, Rewind/Replay (RR). We show that RR performs significantly better than the other algorithms on a large DBN which models the spread of wildfire.

The rest of this paper is organized as follows. We first formulate our task and then discuss previous algorithms and analyze them on two example networks. We then present RR and a generalization of SOF. Finally, we describe our experiments and then discuss related and future work.

Problem formulation

Previous work has shown how to encode probabilistic processes and plans as Bayesian networks (e.g., (Dean and Kanazawa 1989; Hanks *et al.* 1995)). Here, we formulate our task as inference on Bayesian networks.

A Bayesian network describes the joint distribution over a finite set of discrete random variables $\mathcal{X}$. Each variable $X_i \in \mathcal{X}$ can take on any value from a finite domain $val(X_i)$. A *variable assignment* is an expression of the form $X_i = x_j$ indicating that X_i has value x_j. We use capital letters (e.g., X, Y, Z) for variables and lowercase letters (e.g., x, y, z) for values of variables.

A Bayesian network is a directed, acyclic graph in which nodes correspond to variables and arcs to direct probabilistic dependence relations among variables. A network is defined by a variable set $\mathcal{X}$, a parent func-

tion Π, and a conditional probability table CPT. The Π function maps X_i to X_i's parents. The CPT function maps X_i and a variable assignment for each parent of X_i to a probability distribution over $val(X_i)$. Variable X_i can be *sampled* by randomly drawing one value from the probability distribution returned by CPT. An entire network can be sampled by sampling the variables in an order such that each variable is sampled after its parents are sampled. See (Pearl 1988) for a more thorough description.

We are especially interested in *Dynamic belief networks* (DBNs) also called temporal belief networks (e.g., (Kjaerulff 1992)). DBNs are Bayesian networks used to model temporal processes. A DBN can be divided into subsections, called *time slices*, that correspond to snapshots of the state of the world. Typically, certain variables are designated as observation variables.

Figure 1 shows a simple DBN. The state variables Pos_i and Dir_i represent a person's position and direction, respectively, at time i. The person's direction influences their position and direction in the following time slice. The variable Obs_i represents some possibly noisy observation on Dir_i, such as the output of a compass or the observed person announcing their direction. Note that we can infer a probability distribution over the position of the person given a set of observations, even though the position is never directly observed.

We now define the *simulated inference task*. The inputs are a Bayesian network B and a set of assignments $\mathcal{O} = \{O_1 = o_1, ..., O_n = o_n\}$, which give the observed values for some of the observation variables in B, such as $\{Obs_1 = west, Obs_2 = east\}$. The output is a set of positively weighted samples of B, where each sample s_i is weighted by an estimate of the probability $\mathbf{P}(s_i|\mathcal{O})$.

Inference on Bayesian networks is more typically formulated as determining the probability of a query expression given the evidence. We chose our formulation because it offers a wide range of reasoning for plan monitoring. Suppose we are projecting from past observations to find the safest escape route in some hazardous environment. We might enumerate each possible path and query a probabilistic reasoner to infer the probability that it is safe. A more tractable alternative, however, is to apply path-following algorithms to a set of weighted simulations and let each simulation "vote" its weight as to what is the safest path. We give an example of this approach in our experiments.

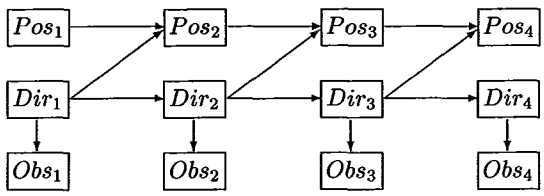

Figure 1: Example DBN

Previous algorithms

We now discuss previous simulation methods.

Logical sampling

A simple approach, called *logical sampling* (LS) (Henrion 1988), is to repeatedly sample the network and discard samples in which $O_i \neq o_i$ for any $O_i = o_i \in \mathcal{O}$. LS assigns the same weight to each retained simulation. Given a query Ω, LS estimates the probability of Ω as the percentage of retained samples in which Ω holds. Logical sampling is an *unbiased* technique: as the number of simulations approaches infinity, the estimate of Ω approaches the true probability of Ω.

The probability of retaining a simulation, however, is exponentially low in the number of observations, assuming some independence among the observations. In many examples we consider, LS can run for hours without retaining a single sample.

Likelihood weighting

The most commonly implemented simulation technique, called *likelihood weighting* (LW), is a variation on logical sampling which can converge much faster in some cases (Shachter and Peot 1989; Fung and Chang 1989; Dagum and Pradhan 1996).

LW also samples the network repeatedly but weights each instantiation by the probability it could have produced the observations. Consider a simple case in which there is a single observation node O_1 with observed value o_1. Logical sampling would repeatedly simulate the network and discard any samples in which $O_1 \neq o_1$. LW, however, weights each sample by the probability that O_1 would be assigned o_1 under the network's CPT function given the values of O_1's parents in the sample.[1] For example, if O_1 had a .08 probability of being assigned o_1 in a sample then LW would weight the sample .08, regardless of what value O_1 was actually assigned.

In the more general case, LW weights each sample s as the Likelihood$(\mathcal{O}|s)$, where **Likelihood**$(\mathcal{O}|s)$ denotes the product of the individual conditional probabilities for the evidence in $\mathcal{O}$ given the sampled values for their parents in s.

LW is also unbiased and has been shown to be effective on many problems. However, as demonstrated in (Kanazawa *et al.* 1995), LW does not work well on DBNs. When the network models a temporal process with an exponential number of possible execution paths, the vast majority of random samples can have a likelihood of, or near, zero.

SOF

Of the algorithms we investigated, the *Survival Of The Fittest* (SOF) algorithm is the best suited for plan monitoring. SOF uses the evidence to guide the simulation, rather than simply to weight it afterwards (Kanazawa *et al.* 1995). We now provide an informal description

[1] We assume observation variables have no children. Thus, their values do not effect the value of other variables.

of the SOF algorithm. Figure 5 contains pseudo-code for a slight generalization.

SOF was designed specifically for DBNs. The idea is to seed the next round of simulations, at each time point, with the samples that best matched the evidence in the previous round. SOF maintains a fixed number of possible world states, but re-generates a sample population of world states for each time t by a weighted random selection from the world states at time $t-1$ where the weight for a world state is given by the likelihood of the observations at time t in that world state.

Initially, SOF generates a fixed number, say 100, of samples of the state variables in the first time slice. The weight, w_i, of the ith sample, s_i, is the likelihood of the observed evidence for time 1 in sample s_i. SOF now generates a new set of 100 samples by randomly selecting from samples $s_1, ..., s_{100}$ using weights $w_1, ..., w_{100}$. Some samples may be chosen (and copied) multiple times and others may not be chosen at all. SOF next samples the state variables in the second slice in each of its 100 samples. SOF then weights each sample by the likelihood of the evidence at time 2 and re-populates its samples by random selection using these weights. SOF repeats this process for all time slices.

Investigation of SOF

We now describe two DBN's designed to expose potential problems with using SOF for plan monitoring.

Discarding plausible hypotheses

Consider the following thought experiment. Suppose you put 1,000 distinct names in a hat and repeatedly draw a name from the hat, write it down, and return the name to the hat. If you repeat the process 1,000 times you will have, on average, about 632 distinct names.[2] SOF can discard hypotheses by random chance in a similar manner. Even though the evidence favors selecting the most likely hypotheses, SOF can lose plausible hypothesis if there are hidden state variables whose value becomes apparent only by integrating information over time from a series of reports.

We designed the following network to demonstrate this problem. The network contains one state and one sensor node for each time slice. At time 1, the state node is assigned an integer value between 1 and 50 from a uniform distribution. For $t > 1$, the state simply inherits its value from the state node at time $t-1$. At each time step $t \geq 1$, if the state node's value is evenly divisible by t then the sensor returns *Yes* with .9 probability or *No* with .1 probability and otherwise returns *Yes* with .1 probability or *No* with .9 probability. If, for example, we observe Yes, Yes, Yes, No, Yes, then there is a 22.5% chance that the state's value is 30.

The goal is to guess the state node's value. This seems like an easy problem for SOF to solve with, say, 1000 samples. There are only 50 distinct hypotheses, and thus SOF initially has several copies of each of them. As time progresses, the more likely hypotheses should be steadily re-enforced by the evidence. We tested SOF by summing the weights associated with each hypothesis, and seeing if the one with the highest weight is, in fact, one of the most likely hypotheses.[3] Based on 1,000 trials, however, SOF with $N = 1000$ achieves only an 81% accuracy on this problem. Even with $N = 2000$, SOF achieves only 92% accuracy.

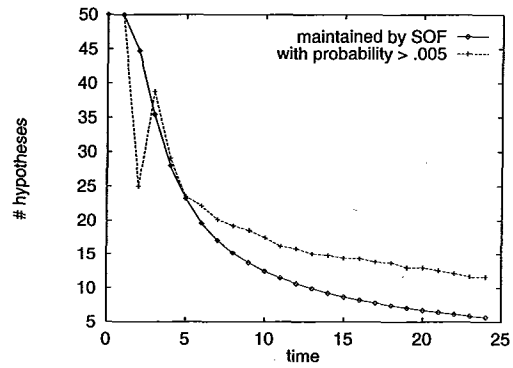

Figure 2: Number of hypotheses maintained by SOF compared to the number of hypotheses with probability greater than .005 on the NumberNet DBN.

The graph in figure 2 shows the average number of distinct hypotheses that SOF maintains, with $N = 1000$, after t time steps for $0 \leq t \leq 25$ as well as the average number of hypotheses that have a .005 or greater probability of being true given the evidence at time t.[4] We counted the hypotheses maintained by SOF by counting the number of distinct values for the state node contained in the 1000 samples maintained by SOF. As the graph shows, the number of hypotheses that SOF maintains is noticeably lower than the number of plausible hypotheses.

Premature variable assignment

We designed the following problem to reduce the usefulness of SOF's re-population strategy. We consider a case in which the state variables are assigned values several time steps before they are observed. We believe this represents an important aspect of plan monitoring. Probabilistic planners (e.g. (Kushmerick *et al.* 1995)) take as input a probability distribution over initial states. The value of conditions in the initial state are decided before they are observed, but may be crucial to the success of a plan. For example, in evacuation plans, there might be uncertainty about whether a certain bridge is usable, and the first report on its condition may come when a vehicle encounters the bridge during plan execution.

[2] Each name has a 1 - $.999^{1000}$ chance of being selected.

[3] We can easily compute the exact probability of each hypothesis given the evidence, but there may be ties.

[4] The set of hypotheses with probability $\geq .005$ can grow as evidence favors some hypotheses and eliminates others.

Consider a simple DBN in which there are K fair coins, all flipped at time 1. The ith coin is reported with 95% accuracy at times $i+1$ and $K+i+1$. The network for this DBN contains a state node c_i for each coin and a single sensor node. Initially all the coins are given a random value from $\{Heads, Tails\}$. For all times $j > 1$ the coin c_i simply inherits its value at time $j-1$. At time $j+1$ the sensor outputs the value of coin c_j at time $j+1$, with .95 accuracy. Similarly, at time $K+j+1$ the sensor outputs the value of coin c_j at time $K+j+1$, with .95 accuracy.[5]

The goal is to compute the probability that coin $c_i = Heads$ for each coin, given all $2K$ observations. To evaluate SOF, we compare the actual probability of $c_i = Heads$ with the estimate computed from the simulations returned by SOF. We compute the estimate from SOF by dividing the sum of the weights of the simulations in which $c_i = Heads$ by the sum of the weights of all simulations.

The graph in figure 3 shows the average error rate on a problem with 15 coins for SOF with $N = 1000$ samples. The graph reports the absolute error between the actual probability of $c_i = Heads$ given the evidence and the weighted estimate from SOF. The graph shows both that the errors are high (the average error is .24) and that the error is worse for the higher number coins. Our explanation is as follows. At time 1, SOF picks 1000 of the possible 2^{15} combinations of *Heads* or *Tails* for the coins. At time 2, SOF re-populates its samples based on the first report on coin 1. At time 3, SOF does the same for coin 2. As SOF gets to the higher number coins, its sample population becomes more and more skewed as it generates more copies of the combinations of coin flips that match the lower number coins.

Figure 3 also shows the performance of a modified version of SOF, called SOF-Bayes, which we describe below. As shown in the graph, SOF-Bayes performs much better on this problem.

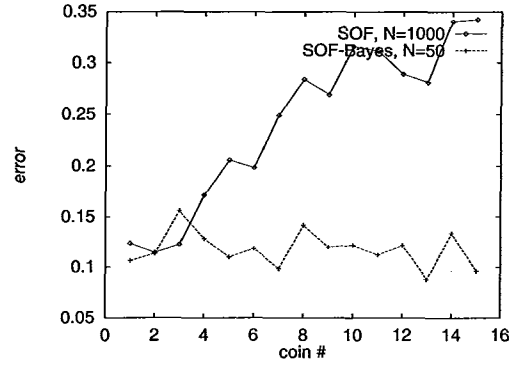

Figure 3: Error for SOF with $N = 1000$ and SOF-Bayes with $N = 50$ on the CoinNet problem. Results averaged from 200 trials.

New algorithms

We now present new methods.

Rewind/replay algorithm

We now introduce the *Rewind/Replay* (RR) algorithm, which is closely related to the sampling method called sequential imputation (discussed below).

We first describe RR informally in terms of solving a DBN. RR samples each time slice of the network a fixed number of times, and then randomly chooses one sample from those that contain the observations for that time slice. It discards all the other samples of the current time slice and simulates forward from the chosen sample into the next time slice. If the observed evidence does not arise in any of the samples of a time slice, then RR abandons the current instantiation of the DBN it is constructing, and begins again from the first time slice.

The number of times RR samples each time slice is an input integer, R_{max}, to the RR algorithm. Each time RR makes it through all the time slices it generates a single sample of the entire network that matches all the evidence. RR assigns the sample a weight of $F_1 \times F_2 \times ... \times F_n$ where F_i is the fraction of samples of time slice i that matched the observations for time i. The justification for this weighting method is that the probability of the sample given the evidence is $\mathbf{P}(E_1 \mid S_1) \times \mathbf{P}(E_2 \mid S_2) \times ... \times \mathbf{P}(E_n \mid S_n)$, where $\mathbf{P}(E_i \mid S_i)$ is the probability of the evidence at time i given the state at time i and is approximated by F_i.

An optimization in our implementation of RR is to proceed to the next time slice as soon as the evidence arises in any sample. This is as random as choosing from the samples that matched the evidence after R_{max} tries, since the trials are independent. If RR manages to fully instantiate the network, it computes the weight for the network by returning to each time slice j and sampling it $R_{max} - k_j$ times, where k_j is the number of samples already performed on the jth time slice. This optimization reduces the computation expended in RR's failed attempts to sample the network.

The intuition behind RR can be explained in terms of the task of flipping twenty coins until all of them are heads. The logical sampling approach is to flip all twenty coins, and if they do not all come up heads, flip them all again. This is expected to succeed in 1 out of 2^{20} trials. The Rewind/Replay approach is to flip each coin until it is heads and then move on to the next coin. For $R_{max} = 5$, RR should succeed with $1 - (\frac{2^5-1}{2^5})^{20}$ probability, or about 1 in 2 times. With $R_{max} = 10$, RR will succeed with over .99 probability.

We now describe RR in terms of an arbitrary Bayesian network. To do so, we need to distribute the input observations $\mathcal{O} = \{O_1 = o_1, ..., O_n = o_n\}$ into a sequence of sets of observations $\mathcal{E}_1, \mathcal{E}_2, ...\mathcal{E}_m$ such that

[5]Note that not all time slices are identical in this DBN, because different state nodes are connected to the observation node in different time slices.

procedure: RR($B, \mathcal{E} = \{\mathcal{E}_1 ... \mathcal{E}_m\}, N, R_{max}$)
$\mathcal{SIMS} \leftarrow \emptyset$
for $i = 1$ to N
  $s_i \leftarrow \emptyset$; $w_i \leftarrow 1$; $step \leftarrow 0$
  while ($step \leq m$ **and** $w_i > 0$)
    $nodes \leftarrow$ ANC($\mathcal{E}_{step}, \mathcal{E}, B$)
    $\mathcal{M} \leftarrow \emptyset$
    for $l = 1$ **to** R_{max}
      add a sample of $nodes$ to s_i
      if HOLDS($\mathcal{E}_{step}, s_i$)
        then add copy of s_i to $\mathcal{M}$
      remove sampled values of $nodes$ from s_i
    $w_i \leftarrow w_i \times \frac{|\mathcal{M}|}{R_{max}}$
    if $|\mathcal{M}| > 0$
      then $s_i \leftarrow$ random selection from $\mathcal{M}$
    $step \leftarrow step + 1$
  if $w_i > 0$,
    then add $\langle s_i, w_i \rangle$ to $\mathcal{SIMS}$
return $\mathcal{SIMS}$

function: ANC($\mathcal{E}_i, \{\mathcal{E}_1 ... \mathcal{E}_m\}, B$)
  Return set of all variables V_i in B from which there is a directed path in B from V_i to a variable in $\mathcal{E}_j$ but not a directed path to any variable $e_j \in \mathcal{E}_1 \cup \mathcal{E}_2 \cup ... \cup \mathcal{E}_{j-1}$.

Figure 4: The Rewind/Replay algorithm.

each observation goes into exactly one evidence set. Although it will not effect correctness, the distribution of the evidence can have a significant impact on the performance of RR. For example, if all the evidence is put into a single set, i.e, $\mathcal{E}_1 = \{\mathcal{O}\}$, then RR is equivalent to LS. Unless otherwise stated, we will assume the opposite extreme of putting each observation in its own set, i.e, for $1 \leq i \leq n$, $\mathcal{E}_i = \{O_i = o_i\}$.

RR needs to determine which nodes influence the observations in each set $\mathcal{E}_j$. When sampling a Bayesian network, the only nodes which can influence the probability distribution over the value of some node O_i when O_i is sampled are the ancestors of O_i (i.e., the parents of O_i, the parents of the parents O_i, and so on). As we process each evidence set $\mathcal{E}_j$, we want to sample only the variables which are ancestors of $\mathcal{E}_j$ and that we have not already sampled, i.e., that are not ancestors of any observation in $\mathcal{E}_1, ..., \mathcal{E}_{j-1}$. This set is determined by the function ANC, defined in figure 4.

The pseudo code for RR is shown in figure 4. For each simulation, RR iterates through the evidence sets $\mathcal{E}_1$,..., $\mathcal{E}_m$. For each $\mathcal{E}_j$, RR samples the unsampled ancestors of $\mathcal{E}_j$ a total of R_{max} times and selects one of the samples that satisfies every variable assignment in $\mathcal{E}_j$. The weight for each returned sample is $\frac{M_1}{R_{max}} \times \frac{M_2}{R_{max}} \times ... \times \frac{M_m}{R_{max}}$, where M_i is the number of samples that matched evidence set $\mathcal{E}_i$.

RR uses the evidence to guide the simulations, but can avoid the problems caused by SOF's re-population phase. With $R_{max} = 10$ and $N = 500$, RR achieved .96 accuracy on the number guessing problem described above, on 1000 random trials. In the worst case, RR might effectively sample the network $10 \times 500 = 5000$ times, but in practice RR samples it much less frequently and, on average, only generates 69.4 complete instantiations of the network per 500 attempts. The other 430.6 attempts to instantiate the network require many fewer than 5000 samples of the network. In our Lisp implementations, RR required 3.3 CPU seconds to complete its inference, which is comparable to the 4.8 CPU seconds for SOF with $N = 1000$ to achieve an accuracy of .82. Additionally, RR achieved .09 error on the CoinNet problem, with $R_{max} = 10$ and $N = 1,000$.

procedure: SOF-Bayes ($B, \mathcal{E} = \{\mathcal{E}_1 ... \mathcal{E}_m\}, N$)
for $i = 1$ to N
  $s_{1,i} \leftarrow \emptyset$; $w_i \leftarrow 1$
for $j = 1$ to m
  $X_j \leftarrow$ ANC($\mathcal{E}_j, \mathcal{E}, B$)
  for $i = 1$ **to** N
    $s_{j,i} \leftarrow$ randomized selection from $s_{j-1,1}, ..., s_{j-1,N}$
      weighted by $w_1, ..., w_N$.
    add a sample of X_j to $s_{j,i}$
    $w_i \leftarrow$ LIKELIHOOD($E_j | s_i$)
for $i = 1$ to N
  sample of any unsampled nodes in $s_{m,i}$
return $\langle s_{m,1}, w_1 \rangle ... \langle s_{m,N}, w_N \rangle$

Figure 5: The Survival-of-the-fittest algorithm for an arbitrary Bayesian network

The RR algorithm is closely related to the statistical sampling technique called sequential imputation (Kong *et al.* 1994; Liu 1996). Sequential imputation is a Monte Carlo technique for solving nonparametric Bayes models given some data $d_1, ..., d_i$ by incrementally sampling from the probability distribution $\mathbf{P}(d_i \mid d_1, ..., d_{i-1})$ for increasing values of i. RR is perhaps best viewed as one of many possible instantiations of sequential imputation for the task of inference on Bayesian networks.

SOF-Bayes

Figure 5 shows pseudo code for a variation of SOF that works on arbitrary Bayesian networks (SOF was designed for DBNs). Like RR, SOF-Bayes requires that the evidence be distributed into evidence sets $\mathcal{E}_1, ..., \mathcal{E}_m$. Like SOF, SOF-Bayes incrementally instantiates a set of N samples of the given network in parallel. However, while SOF samples the nodes time slice by time slice, SOF-Bayes first samples the ancestors of $\mathcal{E}_1$, then the (unsampled) ancestors of $\mathcal{E}_2$, and so on until all the evidence is accounted for. As a final step, SOF-Bayes samples any nodes which are not ancestors of any of the evidence variables.

There are two advantages of SOF-Bayes over SOF even for solving DBNs, both of which are also enjoyed by RR. First, SOF-Bayes lazily samples the network, only assigning values to variables when necessary for determining the likelihood of the evidence currently being used to re-populate the samples. Note that in any

DBN, a state variable in time slice i is only observed in time slice i if it is directly connected to an observation variable. In the CoinNet problem, for example, SOF-Bayes generates N independent samples of a coin just before the evidence re-enforces the most likely value of that coin. In contrast, SOF samples all the coins at time 0. By the time a coin's value is observed, SOF may have copied and re-populated the samples so many times that only a few independent samples of the coin remain, copied hundreds of times each.

A second advantage, somewhat conflated with the first advantage in the CoinNet problem, is that SOF-Bayes does not have to sample an entire time slice of nodes at once. Suppose K coins are flipped at time 1 and then all are reported with 100% accuracy at time 2. For SOF, only one of out 2^K samples will likely match the evidence. SOF-Bayes can work on each coin flip independently and thus will very likely match all the evidence with as few as $N = 10$ samples.

Experiments

We now describe our experiments. Our goal was to generate a large, non-trivial DBN in order to demonstrate the potential value of simulation-based reasoning for plan monitoring in uncertain domains.

We constructed a simple forest-fire domain based loosely on the Phoenix fire simulator (Hart and Cohen 1992).We use a grid representation of the terrain. During the course of the simulation, each cell is either burning or not. At each time step, the probability that a cell will be burning is computed (by a function not described here) based on the status of that cell and that cell's neighbors at the previous time slice.

There are ten noisy fire sensors, each of which reports on a 3×3 block of grid cells. The sensor can output *Low*, *Med*, or *High*, indicating that 0, 1–3, or 4–9 of the cells are on fire, respectively (with 10% noise added in). Additionally, each sensor has a solar battery with a 75% chance of being charged in each time slice. If the battery is not charged at time t then the sensor outputs *Recharging* at time $t + 1$.

A fire fighter is initially situated in the middle of the grid. The fire fighter monitors the fire sensors in order to decide whether to flee to one of four helipads situated on the edges of the grid. Each helipad, however, has only a .65 chance of being functional. The fire fighter receives reports of varying reliability about the condition of four helipads.

In these experiments, we used a 10×10 grid, and created a DBN with 30 identical time slices, resulting in a network with 3660 nodes. Most nodes in the network have nine parents. There are a total of 14 observations per time slice: 10 fire sensors and 4 helipad reports.[6] Figure 6 contains an ASCII representation of several time slices from one execution trace.

[6]A complete description of the domain is available at http://www.merl.com/projects/sim-inference/.

	LW	SOF $N = 1000$	SOF-Bayes $N = 1000$	RR $R_{max} = 50$
CPU seconds	181	235	168	128
Acc. on fire	–	.64	.83	.82
Acc. on helipad	–	.53	.66	.94

Table 1: Accuracy at predicting if a cell will contain fire and the condition of the helipads. Numbers averaged over 100 trial runs.

	LW	SOF $N = 1000$	SOF-Bayes $N = 1000$	RR $R_{max} = 75$
Survival rate	–	.61	.75	.95

Table 2: Survival rate of simulated fire fighter. Numbers averaged over 25 trial runs.

Results

The first set of experiments measured how accurately the algorithms could predict aspects of the world state from past observations. The results in table 1 were produced as follows. First we generated a sample of the network, s_r. We then set O_r to be the variable assignments for all the sensor nodes in s_r in the first 6 time steps. We then called LW, SOF, SOF-Bayes, and RR with observations O_r. The algorithms ran until they produced at least 50 sample simulations or a 3 minute time limit elapsed. If the algorithm was still running after 3 minutes, we allowed it to finish its current run and included the simulations it returned (note that some of the reported times exceed 3 minutes). The SOF and SOF-Bayes algorithms require more time per run because they compute N simulations in parallel.

We then evaluated the sample simulations returned by the inference algorithms against the sample s_r. For each grid cell, we compared the status (*Burning* or *Safe*) of the cell at time 10 with the weighted estimate of the returned cell. We credit the algorithm with a correct prediction if the weighted estimate of the cell being *Safe* was greater than .5 and the cell's status in s_r is *Safe* or if the weighted estimate is less than .5 and the cell's status is *Burning*. Similarly, for each helipad, we compared the value of the helipad in s_r (either *Usable* or *Unusable*) with the weighted estimate of the returned distributions. If no distributions were returned, we counted this as guessing that all cells are *Safe* and all helipads *Usable*. As shown in table 1, RR performs best, and SOF-Bayes performed better than SOF. LW never generated any matches, primarily because it never matched all the *Recharging* signals, and thus we did not list any accuracies for LW.

In the second set of experiments, we used the weighted simulations to control the movements of the fire fighter and then compared the survival rate with the different inference algorithms. In each iteration the fire fighter can stay in its current cell or move to one of the eight adjacent cells. The procedure for decid-

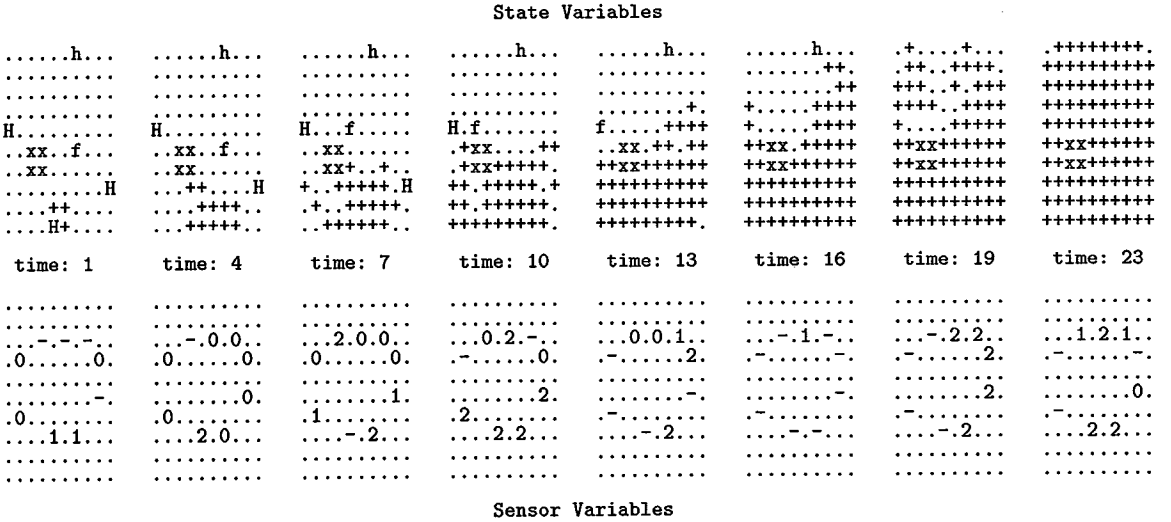

Figure 6: ASCII representation of several time slices of an example simulation of the fire world domain. A '+' indicates fire, an 'H' indicates a working helipad, an 'h' indicates a non-working 'helipad', an 'x' indicates an impassable terrain. The fire fighter is denoted by an 'f'. In the sensor readings, a 0 indicates a low reading, a 1 indicates a medium reading, a 2 indicates a high reading, and a '-' indicates that the sensor is recharging. The messages indicating the condition of the helipads are not shown here.

ing where to go computes, for each weighted simulation and each cell, whether there is a path to safety from the cell. Based on the weights of the simulations, the procedure then moves to one of the cells with the highest probability of being safe, with a preference for remaining in the same cell. To speed the experiments, we only re-computed the weighted simulations given the observations at time 6, 8, and 10. As shown in table 2, the fire fighter survives much more often when using the Rewind/Replay algorithm than the other algorithms.

Discussion- plan monitoring

Although the above domain contains very little planning, recall that our initial motivation was to monitor and improve plans during their execution in dynamic domains. We now discuss how simulation-based inference can be applied to the task of plan monitoring.

Due to space limitations we will not describe in detail how a probabilistic plan and domain can be encoded as a DBN. Roughly speaking, however, the given information about possible initial states must be encoded into the first time slice of the network. The model of how actions effect the world must be encoded into all subsequent time slices. Additionally, the plan being executed must be incorporated into the network in order to indicate when actions will be executed.

We now briefly discuss how the simulation-based inference algorithms can be used to evaluate a possible modification to the plan. We assume that some of the actions in the plan will generate observable results. Thus, at any point during plan execution there will be observed values for some of the variables in the DBN. These values can be input to RR or SOF-Bayes which will return weighted instantiations, or samples, of the DBN. For each returned instantiation, we can extract the time slice corresponding to the current state of the world. We now have a set of world states, each weighted by the probability that it is the current world state. In order to evaluate a potential modification to the plan, we can simulate the execution of the remaining steps of the original plan from each of the returned states, and do the same for the modified plan. These simulations will give us an estimate of the probability that the plan will succeed with and without the proposed modification. The more simulations we perform, the better our estimates will be.

Related and future work

One previous technique we have not yet mentioned is arc reversal (Shachter 1986; Fung and Chang 1989), which has been applied to DBNs (Cheuk and Boutilier 1997). Arc reversal involves reversing the arcs that point to the evidence nodes. This technique essentially uses the information in the CPT of the Bayesian network to perform reverse simulation from the observed evidence. Arc reversal can increase the size of the network exponentially and would do so on our networks. When reversing an arc from node n_1 to node n_2, all parents of n_1 and n_2 become parents to both nodes. Reversing a single arc in the fire network can increase the number of parents of a node from 9 to 17. This

increases the size of the CPT for that node from 2^9 to 2^{17}. Furthermore, in our algorithm the original CPT is encoded compactly as a function, but would have to be expanded out into a table in order to reverse an arc.

There have been many other approaches to approximate inference on Bayesian networks (e.g., (Dechter and Rish 1997)) though most have not been specifically evaluated on complex temporal processes. (Boyen and Koller 1998) propose a method for approximating the belief state for DBNs and prove that, under certain assumptions, the error in the belief state can be bounded over time. Additionally, this approach allows for continuous monitoring of a system while RR, as presented here, requires an initial starting time. Prior work has suggested the use of sequential imputation for solving Bayes nets (Hanks *et al.* 1995), but did not propose our specific implementation of this idea or compare RR to SOF.

In this paper we have investigated simulation techniques for plan monitoring and have shown that some techniques have significant advantages over others. We have extended existing simulation algorithms by generalizing SOF to arbitrary Bayesian networks and introducing the Rewind/Replay algorithm.

Our plans for future work include determining whether RR is unbiased, substituting in LW rather than LS on the sampling step for RR, and developing strategies for distributing the evidence into evidence sets and for choosing the value for R_{max}. Additionally, we would like to compare RR and SOF-Bayes both with each other and previous methods on the task of solving arbitrary Bayesian networks.

Acknowledgments

Many thanks to Andy Golding and the AAAI reviewers for their comments on earlier drafts of this paper. This material is based upon work supported by ARPA under Grant number F30602-95-1-0025

References

X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *UAI'98*, pages 33–42, 1998.

A. Y.W Cheuk and C. Boutilier. Structured arc reversal and simulation of dynamic probabilistic networks. In *UAI'97*, pages 72–79, 1997.

P. Dagum and M. Pradhan. Optimal Monte Carlo Estimation Of Belief Network Inference. In *UAI'96*, pages 446–453, 1996.

Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

R. Dechter and I. Rish. A scheme for approximating probabilistic inference. In *UAI'97*, 1997.

R. Fung and K. Chang. Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *UAI'89*, pages 209–219, 1989.

S. Hanks, D. Madigan, and J. Gavrin. Probabilistic temporal reasoning with endogenous change. In *UAI'95*, pages 245–254, 1995.

D.M Hart and P.R Cohen. Predicting and explaining success and task duration in the phoenix planner. In *Proceedings of the First International Conference on AI Planning Systems*, pages 106–115, 1992.

M. Henrion. Propagation of uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence 2*, pages 149–163, 1988.

K. Kanazawa, Koller D., and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI'95*, pages 346–351, 1995.

U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *UAI'92*, pages 121–129, July 1992.

A. Kong, J.S. Liu, and W.H. Wong. Sequential imputation and Bayesian missing data problems. *J. Americian Statistical Association*, pages 278–288, 1994.

N. Kushmerick, S. Hanks, and D. Weld. An Algorithm for Probabilistic Planning. *J. Artificial Intelligence*, 76:239–286, 1995.

J.S. Liu. Nonparametric hierarchical bayes via sequential imputation. *Ann. Statist.*, pages 910–930, 1996.

J. Pearl. *Probablistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, CA, 1988.

R.D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *UAI'89*, 1989.

R.D. Shachter. Evaluating influence diagrams. *Operations Research*, 33(6):871–882, 1986.