

Latin Hypercube Sampling in Bayesian Networks

Jian Cheng & Marek J. Druzdzel

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

Abstract

We propose a scheme for producing Latin hypercube samples that can enhance any of the existing sampling algorithms in Bayesian networks. We test this scheme in combination with the likelihood weighting algorithm and show that it can lead to a significant improvement in the convergence rate. While performance of sampling algorithms in general depends on the numerical properties of a network, in our experiments Latin hypercube sampling performed always better than random sampling. In some cases we observed as much as an order of magnitude improvement in convergence rates. We discuss practical issues related to storage requirements of Latin hypercube sample generation and propose a low-storage, anytime cascaded version of Latin hypercube sampling that introduces a minimal performance loss compared to the original scheme.¹

Keywords: Bayesian networks, algorithms, stochastic sampling, Latin hypercube sampling

Introduction

Bayesian networks (Pearl 1988) are increasingly popular tools for modeling problems involving uncertainty. Practical models based on Bayesian networks often reach the size of hundreds of variables (e.g., (Pradhan *et al.* 1994; Conati *et al.* 1997)). Although a number of ingenious inference algorithms have been developed, the problem of exact belief updating in Bayesian networks is NP-hard (Cooper 1990). Approximate inference schemes may often be the only feasible alternative for very large and complex models. The problem of achieving a required precision in approximate inference is also NP-hard (Dagum & Luby 1993). However, precision guarantees may not be critical for some types of problems and can be traded off against the speed of computation. In all situations where precision is not of essence, stochastic simulation methods can offer several advantages compared to exact algorithms. Execution time required of exact algorithms depends on the topology of the network and is generally exponential in its size and connectivity. In case of stochastic sampling, computation is measured in terms of the number of samples — in general, the more samples are generated, the higher precision is obtained. Execution time

of stochastic sampling algorithms is less dependent on the topology of the network and is linear in the number of samples. Computation can be interrupted at any time, yielding an anytime property of the algorithms, important in time-critical applications.

In this paper, we investigate the advantages of applying Latin hypercube sampling to existing sampling algorithms in Bayesian networks. While Henrion (1988) suggested that applying Latin hypercube sampling can lead to modest improvements in performance of stochastic sampling algorithms, neither he nor anybody else has to our knowledge studied this systematically or proposed a scheme for generating Latin hypercube samples in Bayesian networks. We first propose a scheme for producing Latin hypercube samples. We then test this scheme in the likelihood weighting algorithm (Fung & Chang 1990; Shachter & Peot 1990) and show that it can lead to a significant improvement in the convergence rate over random sampling. While performance of sampling algorithms in general depends on the numerical properties of a network, in our experiments Latin hypercube sampling performed always better than random sampling. In some cases we observed as much as an order of magnitude improvement in convergence rates. We discuss practical issues related to storage requirements of the Latin hypercube sample generation and propose a low-storage, anytime cascaded version of Latin hypercube sampling that introduces a minimal performance loss compared to the original scheme.

All random variables used in this paper are multiple-valued, discrete variables. Capital letters, such as A , B , or C will denote random variables. Bold capital letters, such as \mathbf{E} , will denote sets of variables. Lower case letters a , b , c will denote particular instantiations of variables A , B , and C respectively.

Latin hypercube sampling

Latin hypercube sampling (McKay, Beckman, & Conover 1979) is inspired by the Latin square experimental design, which tries to eliminate confounding effect of various experimental factors without increasing the number of subjects in the experiment. The purpose of Latin hypercube sampling is to ensure that each value (or a range of values) of a variable is represented in the

¹Copyright 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

samples, no matter which value might turn out to be more important.

Consider the problem of generating five samples from two independent, uniformly distributed variables X and Y , such that $X, Y \in [0, 1]$. According to the Latin hypercube sampling method, we first divide the range of both X and Y into five equal intervals, resulting in $5 \times 5 = 25$ cells (Figure 1). The requirement of Latin

| | | | | | |
|-----|---|---|---|---|---|
| 5 | | | | 4 | |
| 4 | | | | | 5 |
| 3 | | 1 | | | |
| 2 | 2 | | | | |
| 1 | | | 3 | | |
| Y/X | 1 | 2 | 3 | 4 | 5 |

Figure 1: A Latin hypercube sample on the unit square, with $n = 5$.

hypercube sampling is that each row and each column of the constructed table contain only one sample. This ensures that even though there are only five samples, each of the five intervals of both X and Y will be sampled. For each sample $[i, j]$, the sample values of X, Y are determined by:

$$\begin{aligned} X &= F_X^{-1}((i - 1 + \xi_X)/n) \\ Y &= F_Y^{-1}((j - 1 + \xi_Y)/n), \end{aligned} \quad (1)$$

where n is the sample size, ξ_X and ξ_Y are random numbers ($\xi_X, \xi_Y \in [0, 1]$), and F_X and F_Y are the cumulative probability distributions functions of X and Y respectively.

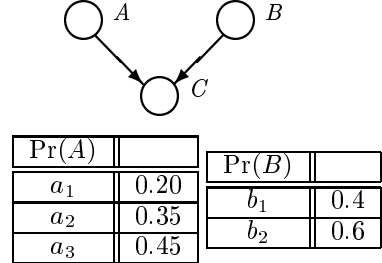
Valid Latin hypercube samples can be generated by starting with a sequence of n numbers $1, 2, \dots, n$ and taking various permutations of this sequence. Figure 2 illustrates the samples represented by Figure 1. Each

| | | |
|--------|---|---|
| Sample | X | Y |
| 1 | 2 | 3 |
| 2 | 1 | 2 |
| 3 | 3 | 1 |
| 4 | 4 | 5 |
| 5 | 5 | 4 |

Figure 2: A Latin hypercube matrix ($LHS_{2,5}$) for $m = 2$ variables and sample size $n = 5$.

column of the table corresponds to a variable, each row to a sample. Each column (please note that the columns list the row and column number of the corresponding sample in Figure 1) is a random permutation of the sequence of integers $1, 2, 3, 4, 5$. In general, we can obtain Latin hypercube samples of size n for m variables through randomly selecting m permutations of the sequence of integers $1, 2, \dots, n$ and assigning them to each of the m columns of the table. We will call this table, a *Latin hypercube sampling matrix* (LHS matrix) $LHS_{m,n}$.

We will now illustrate the process of generation of $n = 100$ Latin hypercube samples in the context of a forward sampling algorithm on a simple network consisting of $m = 3$ random variables A, B , and C , presented in Figure 3. We first construct the $LHS_{3,100}$



| | | | | | | |
|------------|-------|-------|-------|-------|-------|-------|
| Pr(C A, B) | a_1 | | a_2 | | a_3 | |
| | b_1 | b_2 | b_1 | b_2 | b_1 | b_2 |
| c_1 | 0.01 | 0.04 | 0.28 | 0.06 | 0.18 | 0.82 |
| c_2 | 0.68 | 0.93 | 0.52 | 0.12 | 0.50 | 0.10 |
| c_3 | 0.31 | 0.03 | 0.20 | 0.82 | 0.32 | 0.08 |

Figure 3: A simple Bayesian network with three nodes.

matrix for this network, a fragment of which is illustrated in Figure 4. Since the algorithm applied will

| | | | |
|--------|-----|-----|-----|
| Sample | A | B | C |
| 1 | 25 | 13 | 74 |
| 2 | 14 | 91 | 7 |
| ... | ... | ... | ... |
| 39 | 47 | 32 | 56 |
| ... | ... | ... | ... |
| 100 | 69 | 4 | 84 |

Figure 4: A $LHS_{3,100}$ matrix for $m = 3$ variables and sample size $n = 100$.

be forward sampling, we need to make sure that the columns of the LHS matrix follow the topological order, i.e., that parents always precede their children. The process of sample generation proceeds as follows. We divide each of the variables A, B , and C into 100 equally sized intervals. If $LHS_{A,i} \leq \text{Pr}(a_1) \times 100$, we set the state of variable A in the i th sample to a_1 . If $\text{Pr}(a_1) \times 100 < LHS_{A,i} \leq (\text{Pr}(a_1) + \text{Pr}(a_2)) \times 100$, we set the state of variable A to a_2 , otherwise we set it to a_3 . If $LHS_{B,i} \leq \text{Pr}(b_1) \times 100$, we set the state of variable B to b_1 , otherwise we set it to b_2 . The state of variable C is generated similarly from the conditional probability table based on the states of A and B . In our example, row 39 of the LHS matrix contains $\{47, 32, 56\}$. According to the above rule, in sample number 39, variable A is instantiated to the state a_2 and variable B is instantiated to the state b_1 . Variable C is instantiated to c_2 since $\text{Pr}(c_1|a_2, b_1) \times 100 < LHS_{C,i} \leq (\text{Pr}(c_1|a_2, b_1) + \text{Pr}(c_2|a_2, b_1)) \times 100$. Please note that Equation 1 requires that we add a random

number between 0 and 1 into $LHS_{m,i}$. Since the sample size n is usually very large and, consequently $1/n$ is very small, much smaller than the required precision, we can ignore this random number in case of discrete variables.

The above Latin hypercube sampling scheme gives us a way of instantiating variables to their states that is applicable to any stochastic sampling algorithm. A generalized procedure based on Latin hypercube sampling, is shown in Figure 5.

1. Order the nodes according to their topological order, as required by the algorithm.
2. Generate the *LHS* matrix.
3. For each sample (row of the *LHS* matrix) generate corresponding variable states.
4. Use these samples to calculate the desired probability distribution.

Figure 5: A generalized stochastic sampling procedure based on Latin hypercube sampling.

The complexity of Latin hypercube sampling is the same as that of random sampling. A random sample in the latter corresponds to generation of one element of the *LHS* matrix, which involves a call to the random number generator. An additional step in generation of one element of the *LHS* matrix is swapping of matrix elements, equal to three assignments.

Using the crossed analysis of variance (ANOVA) decomposition (Efron & Stein 1981), Stein (1987) has proven that in the problem of estimating $I = \int_{[0,1]^d} f(\mathbf{X})d\mathbf{X}$, the variance of stochastic simulation based on Latin hypercube sampling is asymptotically smaller than that based on simple random sampling. In finite samples, Owen (1997) shows that the variance of Latin hypercube sampling is never much worse than that of simple random sampling. Roughly speaking, because Latin hypercube sampling stratifies each dimension as much as possible and otherwise picks the relation between different dimensions randomly, it can significantly decrease the variance coming from individual dimensions. Since the stochastic sampling in Bayesian networks is very similar to stochastic integration, Latin hypercube sampling can be expected to lead to smaller variance here as well.

Some improvements to the Latin hypercube sampling

The main practical problem related to Latin hypercube sampling is that the *LHS* matrix, which is usually very large, has to be stored in memory for the duration of the sampling. When both the network size and the number of samples are very large, this may become prohibitive. For example, when a network consists of 500 nodes that we want to sample 100,000 times,

we will need at least $\log_2 100,000 \approx 17$ bits for each of the elements of the *LHS* matrix. This means that to store the entire *LHS* matrix we will need at least $17 \times 500 \times 100,000 = 850,000$ bits ≈ 106 M bytes.

The first solution to this problem that we applied in our implementation is to store variable instantiations instead of permutations in the *LHS* matrix. If we generate the permutations in the order of sampling, and the columns of the *LHS* matrix follow the parent ordering, we can compute at each step the outcome of the variable in question. As most variables have a relatively small number of outcomes, a few bits usually suffice. If the maximum number of outcomes of a variable in the above example is 8, we can use $\log_2 8 = 3$ bits per element of the *LHS* matrix instead of 17. Then the entire matrix will take $3 \times 500 \times 100,000 = 150$ M bits ≈ 19 M bytes. In case of networks consisting of only binary variables, this number can be reduced even further to approximately 6 M bytes. Another way of looking at this solution is that it combines steps 2 and 3 in the algorithm of Figure 5 into a single step.

The second improvement is treatment of the root nodes in the network. Latin hypercube sampling requires that the states of these nodes are sampled according to their prior probability. To achieve this sampling, instead of generating permutations for the root nodes, we can assign randomly a proportion of elements in their columns to their states s_i : $\text{Pr}(s_1) \times n$ positions for state s_1 , $\text{Pr}(s_2) \times n$ positions for state s_2 , etc., assigning the remainder of the positions to the last state. In case of the network in Figure 3, this can save us $P_{a_3} \times n + P_{b_2} \times n = 105$ calls to the random number generator (in this simple example, this amounts to 35% of all calls!).

Finally, our third proposed improvement is dividing the *LHS* matrix into smaller matrices and instead of working with a large $m \times n$ matrix, working with k $m \times n/k$ *LHS* matrices. While this solution reduces the precision of sampling (please note that since we use the number of samples to divide the interval $[0, 1]$, the number of samples determines the precision), it may be unimportant when the number of samples is sufficiently large and the probabilities in the model are not extreme. We will confirm this observation empirically in the next section. This method, that we call *cascaded Latin hypercube sampling* has the advantage of being anytime, as processing each of the k $LHS_{m,n/k}$ matrices increases precision of the algorithm but the result is available as soon as the first matrix has been processed. In the sequel of the paper, we will use the symbolic notation $k \times LHS$ to denote a cascaded Latin hypercube sampling scheme that consists of k steps.

Experimental results

We performed empirical tests comparing Latin hypercube sampling to random sampling in the context of the likelihood sampling algorithm (Fung & Chang 1990; Shachter & Peot 1990). We used two networks in our tests. The first network is *Coma*, a sim-

ple multiply-connected network originally proposed by Cooper (1984).

The second network used in our tests is a subset of the *CPCS* (Computer-based Patient Case Study) model (Pradhan *et al.* 1994), a large multiply-connected multi-layer network consisting of 422 multi-valued nodes and covering a subset of the domain of internal medicine. Among the 422 nodes, 14 nodes describe diseases, 33 nodes describe history and risk factors, and the remaining 375 nodes describe various findings related to the diseases. The *CPCS* network is among the largest real networks available to the research community at present time. Our analysis is based on a subset of 179 nodes of the *CPCS* network, created by Max Henrion and Malcolm Pradhan. We used this smaller version in order to be able to compute the exact solution for the purpose of measuring approximation error. We used the clustering algorithm (Lauritzen & Spiegelhalter 1988) to provide the exact results for the experiment.

We focused on the relationship between the number of samples in random sampling (in all experiments, we run between 1,000 and 10,000 samples with 1,000 increments) and in Latin hypercube sampling (we tried the straight and cascaded Latin hypercube sampling) on the accuracy of approximation achieved by the simulation. We measured the latter in terms of the Mean-Squared-Error (*MSE*), i.e., square root of the sum of square differences between $\text{Pr}'(X_{ij})$, the computed marginal probability of state j ($j = 1, 2, \dots, n_i$) of node i and $\text{Pr}(X_{ij})$, the exact marginal probability, for all non-evidence nodes. In all diagrams, the reported *MSE* is average over 20 trials. More precisely,

$$MSE(t) = \sqrt{\frac{1}{\sum_{i \in \mathbf{N} \setminus \mathbf{E}} n_i} \sum_{i \in \mathbf{N} \setminus \mathbf{E}} \sum_{j=1}^{n_i} (\text{Pr}'(x_{ij}) - \text{Pr}(x_{ij}))^2}$$

where \mathbf{N} is the set of all nodes, \mathbf{E} is the set of evidence nodes, and n_i is the number of outcomes of node i .

Figure 6 shows the *MSE* as a function of the number of samples for the Coma network. It is clear that the Latin hypercube sampling in all three variants outperforms random sampling. For a given sample size, the improvement of *MSE* can be as high as 75%. The improvement looks even more dramatic when we look for the number of samples that are needed to achieve a given precision. Latin hypercube sampling with only 2,000 samples can perform better than random sampling with 10,000 samples. In other words, Latin hypercube sampling can achieve a better precision in one fifth of the time. For the cascaded versions of the algorithm, as expected, performance deterioration is minimal when the number of samples is large enough. For small samples sizes (e.g., for 1,000 samples), a cascaded version with 20 steps of 50 samples each performed somewhat worse because 50 samples divide the interval $[0,1]$ too coarsely to achieve a desired precision. Analogous results for the *CPCS* network are presented in Figure 7. Without evidence, typical improvement in *MSE* due to

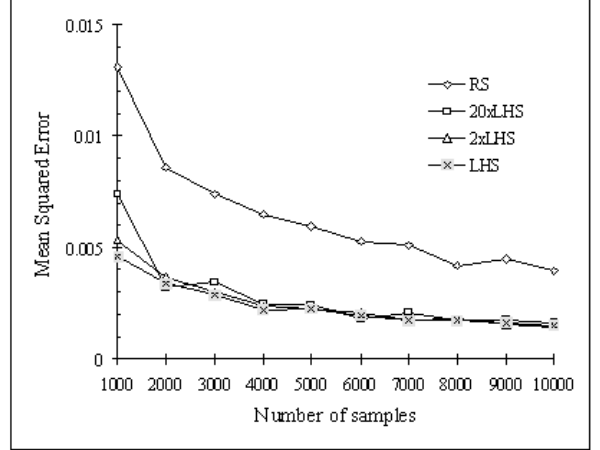


Figure 6: Mean Squared Error as a function of the number of samples for the Coma network without evidence for the random sampling (RS) and three variants of Latin Hypercube sampling (LHS).

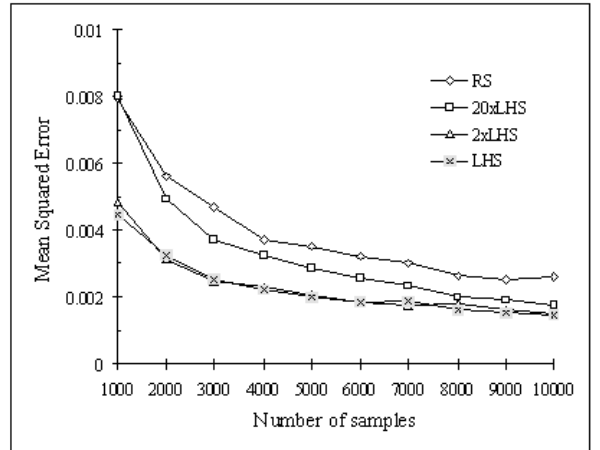


Figure 7: Mean Squared Error as a function of the number of samples for the *CPCS* network without evidence for the random sampling (RS) and three variants of Latin Hypercube sampling (LHS).

Latin hypercube sampling is on the order of 50%. For a given error level, typically fewer than one quarter of the samples was used by the Latin hypercube sampling compared to random sampling. Cascaded inference in the *CPCS* network requires a larger number of samples to achieve the same precision as pure Latin hypercube sampling. This is due to the fact that the probabilities in the *CPCS* network are more extreme.

Figures 8 and 9 show the results for the Coma and *CPCS* networks with evidence. In case of the Coma network, the evidence was observation of severe headaches without coma. In case of the *CPCS* network, we were able to generate evidence randomly

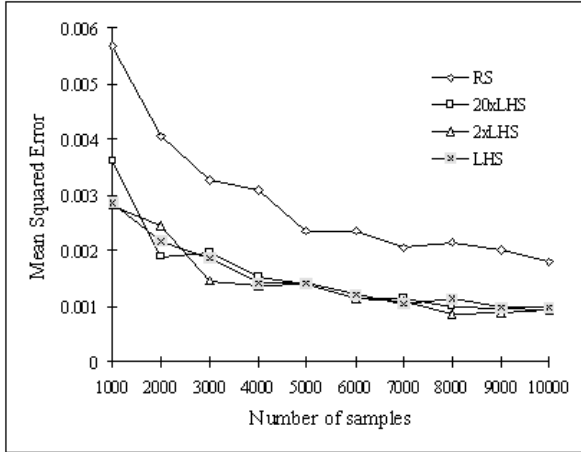


Figure 8: Mean Squared Error as a function of the number of samples for the Coma network with evidence (severe headaches without coma) for the random sampling (RS) and three variants of Latin Hypercube sampling (LHS).

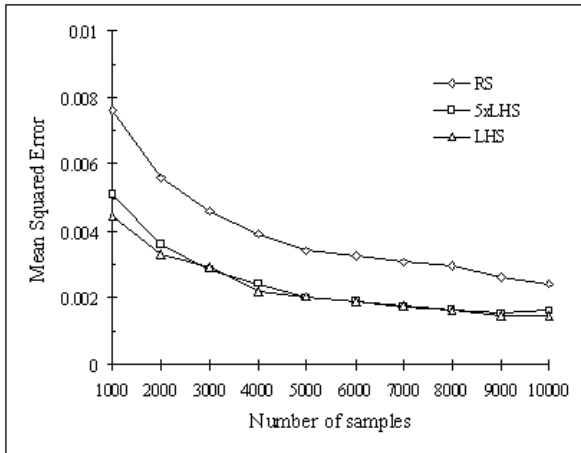


Figure 9: Mean Squared Error as a function of the number of samples for the CPCS network with evidence (five evidence nodes chosen randomly from among plausible medical observations) for the random sampling (RS) and three variants of Latin Hypercube sampling (LHS).

from among those nodes that described various plausible medical findings. These nodes were almost all leaf nodes in the network. We report results for five evidence nodes. When the number of evidence nodes is much larger, the convergence rates deteriorate for all versions of the likelihood sampling algorithm. The improvement due to Latin hypercube sampling in terms of *MSE* is not as dramatic as in the test runs without or with a small number of evidence nodes.

We have observed that in case of large networks, performance of sampling algorithms depends strongly on the number of evidence nodes and their topological lo-

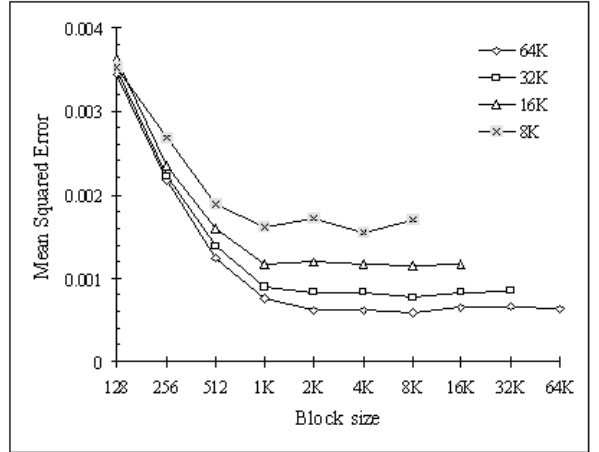


Figure 10: Mean Squared Error as a function of block size and sample size for the CPCS network without evidence for the Latin Hypercube sampling (LHS).

cation in the network — the closer they are to the leaves of the tree, the worse. The reason for this is a mismatch between the sample space (which is in case of likelihood sampling dominated by the prior probability distribution) and the posterior probability distribution. This matches the observations made by Cousins (1993), who concluded that performance of different algorithms depends strongly on the properties of the network. While in the networks that we have tested Latin hypercube sampling was equal to or better than random sampling, sometimes the improvement was not large. We believe that in case of large mismatch between the sampling distribution and the posterior distribution, sampling simply performs poorly and its performance cannot be improved much by the Latin hypercube sampling technique. This does not diminish the value of Latin hypercube sampling, which can be used in combination with any sampling algorithm, including one that will match the posterior distribution better.

We also tested the relationship between the block size in the cascaded version of the Latin hypercube sampling and the *MSE* for different sample sizes. Figure 10 shows that the cascaded version very quickly achieves the performance of the original Latin hypercube sampling scheme. As soon as the number of samples in individual blocks allows for achieving a desired precision, there is little difference between the performance of the cascaded and the original scheme. In the networks that we tested, block size of 2,000 turned out to be sufficiently large to achieve the desired performance.

In terms of absolute computation time, we have observed that generation of one Latin hypercube sample takes about 25% more time than generation of a random sample. We believe that this may be related to large data structures of the Latin hypercube sampling scheme (the *LHS* matrix) and resulting decrease in efficiency of hardware caching. We would like to point

out that despite this difference, Latin hypercube sampling outperforms random sampling in terms of mean squared error within the same absolute time.

Conclusion

Computational complexity remains a major problem in application of probability theory and decision theory in knowledge-based systems. It is important to develop schemes that will reduce it — even though the worst case will remain NP-hard, many practical cases may become tractable. In this paper, we studied application of Latin hypercube sampling to stochastic sampling algorithms in Bayesian networks. We have outlined a method for generating Latin hypercube samples and demonstrated empirically that it leads to significant performance increases in terms of convergence rate. We proposed several ways of reducing storage requirements, the main problem related to practical implementations of Latin hypercube sampling, and proposed a cascaded version of sampling that is anytime and that introduces minimal performance losses. Even though we tested it only with likelihood sampling, Latin hypercube sampling is applicable to any sampling algorithm and we believe that it should in each case lead to performance improvements. The precise advantages will depend on the numerical parameters of the networks and the algorithm applied. We expect, however, that Latin hypercube sampling should never perform worse than random sampling.

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, grants F49620-97-1-0225 and F49620-00-1-0112. Malcolm Pradhan and Max Henrion of the Institute for Decision Systems Research shared with us the CPCS network with a kind permission from the developers of the Internist system at the University of Pittsburgh. All experimental data have been obtained using SMILE, a Bayesian inference engine developed at the Decision Systems Laboratory and available at <http://www2.sis.pitt.edu/~genie>.

References

- Conati, C.; Gertner, A. S.; VanLehn, K.; and Druzdzel, M. J. 1997. On-line student modeling for coached problem solving using Bayesian networks. In *Proceedings of the Sixth International Conference on User Modeling (UM-96)*, 231–242. Vienna, New York: Springer Verlag.
- Cooper, G. F. 1984. *NESTOR: A Computer-based Medical Diagnostic Aid that Integrates Causal and Probabilistic Knowledge*. Ph.D. Dissertation, Stanford University, Computer Science Department.
- Cooper, G. F. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42(2–3):393–405.
- Cousins, S. B.; Chen, W.; and Frisse, M. E. 1993. A tutorial introduction to stochastic simulation algorithm for belief networks. In *Artificial Intelligence in Medicine*. Elsevier Science Publishers B.V. chapter 5, 315–340.
- Dagum, P., and Luby, M. 1993. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence* 60(1):141–153.
- Efron, B., and Stein, C. 1981. The jackknife estimate of variance. *Annals of Statistics* 9(3):586–596.
- Fung, R., and Chang, K.-C. 1990. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In Henrion, M.; Shachter, R.; Kanal, L.; and Lemmer, J., eds., *Uncertainty in Artificial Intelligence 5*. North Holland: Elsevier Science Publishers B.V. 209–219.
- Henrion, M. 1988. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J., and Kanal, L., eds., *Uncertainty in Artificial Intelligence 2*. Elsevier Science Publishers B.V. (North Holland). 149–163.
- Lauritzen, S. L., and Spiegelhalter, D. J. 1988. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B (Methodological)* 50(2):157–224.
- McKay, M. D.; Beckman, R. J.; and Conover, W. J. 1979. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 21(2):239–245.
- Owen, A. B. 1997. Monte Carlo variance of scrambled equidistribution quadrature. *SIAM Journal of Numerical Analysis* 34(5):1884–1910.
- Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann Publishers, Inc.
- Pradhan, M.; Provan, G.; Middleton, B.; and Henrion, M. 1994. Knowledge engineering for large belief networks. In *Proceedings of the Tenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*, 484–490. San Mateo, CA: Morgan Kaufmann Publishers, Inc.
- Shachter, R. D., and Peot, M. A. 1990. Simulation approaches to general probabilistic inference on belief networks. In Henrion, M.; Shachter, R.; Kanal, L.; and Lemmer, J., eds., *Uncertainty in Artificial Intelligence 5*. North Holland: Elsevier Science Publishers B.V. 221–231.
- Stein, M. 1987. Large sample properties of simulations using Latin hypercube sampling. *Technometrics* 29(2):143–151.