Testing MCMC algorithms with randomly generated Bayesian networks

Jaime S. Ide and Fabio G. Cozman

Escola Politécnica,
University of São Paulo
Av. Prof. Mello Moraes, 2231 - São Paulo, SP - Brazil
jaime.ide@poli.usp.br, fgcozman@usp.br

Abstract. In this work we show how to generate random Bayesian networks and how to test inference algorithms using these samples. First, we present a new method to generate random networks through Markov chains. We then use random networks to investigate the performance of quasi-random numbers in Gibbs sampling algorithms for inference. We present experimental results and describe code that implements our methods.

1 Introduction

Bayesian networks occupy a prominent position in Artificial Intelligence, mainly as a representation for uncertainty and for causal dependencies [2, 1, 4]. Bayesian networks can be used for *inference*; to build a Bayesian network, we may have to resort to *learning*. There are several algorithms for inference and learning in Bayesian networks; given an algorithm, one may ask how to test the algorithm.

The purpose of this work is to present random generation of Bayesian networks, and to show how to assess the performance of inference algorithms using these random networks. This work complements another paper from the authors that focuses on the random generation of Bayesian networks [6]; here we present both the generation and the testing of algorithms. The basic idea is to test inference algorithms with a set of networks that is a representative sample of all possible networks with given properties.

In this paper we test inference algorithms that use quasi-Monte Carlo methods in MCMC inference. Cheng and Druzdzel [23] used with success quasi-Monte Carlo methods in importance sampling algorithms, but the performance of quasi-Monte Carlo methods in MCMC inference is still open [24]. We present new and interesting results concerning Gibbs sampling based on quasi-Monte Carlo sampling.

In Section 2 we review the theory of Bayesian networks. Quasi-Monte Carlo methods are discussed in Section 3. In Section 4 we present the problem of Bayesian network generation and existing approaches; we briefly describe our methodology for generate connected directed acyclic graphs given the number of nodes and the maximum number of arcs per node. In Section 5 we show test

results obtained combining quasi-Monte Carlo methods and MCMC algorithms in randomly generated Bayesian networks.

2 Bayesian networks

This section summarizes the theory of Bayesian networks and introduces terminology used throughout the paper. All random variables are assumed to have a finite number of possible values. Denote by $p(X)$ the *probability density* of X, and by $p(X|Y)$ the probability density of X conditional on values of Y.

A Bayesian network represents a joint probability density over a set of variables $\mathbf{X}$ [1]. The joint density is specified through a directed acyclic graph.

In a Bayesian network, each node of its graph represents a random variable X_i in $\mathbf{X}$. The *parents* of X_i are denoted by $\text{pa}(X_i)$. The semantics of the Bayesian network model is determined by the *Markov condition*: Every variable is independent of its nondescendants nonparents given its parents. This condition leads to a unique joint probability density [2]:

$$p(\mathbf{X}) = \prod_i p(X_i|\text{pa}(X_i))\,. \tag{1}$$

Every random variable X_i is associated with a conditional probability density $p(X_i|\text{pa}(X_i))$. Figure 4 depicts examples of DAGs as Bayesian networks.

An inference with a Bayesian network consists of computing the conditional distribution of a variable or a set of variables given a set of observed variables. A general expression for inference (for discrete variables) is:

$$p(X_Q|\mathbf{E}) = \frac{\sum_{\mathbf{X}\setminus X_Q,\mathbf{E}} p(\mathbf{X})}{\sum_{\mathbf{X}\setminus\mathbf{E}} p(\mathbf{X})}, \tag{2}$$

where X_Q is the queried single variable or a set of variables and $\mathbf{E}$ is a set of observed variables.

There are several algorithms for calculating $p(X_Q|\mathbf{E})$ [1], even though the problem is NP-hard [8]. Approximate methods, often based on sampling through simulation [9], or searching [11], or conditioning [12]), have been used with success in complex systems that have many variables [13].

Approximate methods based on Monte Carlo sampling [14] are employed in many areas, from Physics to Economics. For high dimensional problems, a popular solution for approximate inference is offered by Monte Carlo Markov Chain methods [15, 16] (MCMC) that include Gibbs sampling and Metropolis-Hasting algorithms[17].

Gibbs sampling is a special case of Metropolis-Hasting algorithms [17, 18] that was proposed by Geman and Geman [19] for image-processing model studies. The method generates random variables from a marginal distribution indirectly, without having to calculate the joint density [20]. The basic idea is to generate samples from the *full conditional distribution* (conditional distribution

Algorithm ***Gibbs sampler***

Input:queried variable(X_Q), observed variables($\mathbf{E}$), number of iterations(n) **Output**: Return the conditional distribution, $p(X_Q|\mathbf{E})$
01. Inicialize state of all non-observed variables;
02. Repeat the next loop n times;
03. Run-out all variables and for each non-observed variable:
04. Compute the *full conditional distribution*(*fcd*) and normalize at the end;
05. Generate a random variable $u \sim U(0,1)$;
06. Set the next state of this variable given by the *fcd* and u;
07. Increment position of value assumed by queried variable(X_Q) at vector n_Q;
08. Compute the conditional distribution given by estimator $\frac{n_Q}{n}$;
09. Return conditional distribution estimator;

Fig. 1. Simple Gibbs sampler in Bayesian network with discrete variables.

of a variable given all other variables) and use these samples as a estimator of equation 2.

Gibbs sampling technique has been emerged as a very popular tool for complex statistical model analysis [21]. In Bayesian computation we have to calculate expressions with high dimension to get the conditional distribution (see Equation 2) and Gibbs sampling is a very simple way to approximate this expression.

Algorithm *Gibbs sampler*(figure 1) implements a basic Gibbs sampler for estimating the conditional distribution of a queried variable given a set of observed variables. Choosing the *full conditional distribution*(*fcd*) as a transition function (line 6), we get a ergodic Markov chain and after a sufficient number n of iterations, we get a stationary distribution. Each generated chain is a sample of Monte Carlo estimator(line 7).

Gibbs sampling techniques are well suited for embedded systems that requires low-memory consumption [5], because of simplicity of the method, and for *anytime* inference, that is, inference processes that can produce results in a given time T as required [7].

3 Quasi-Monte Carlo methods

In short, quasi-Monte Carlo use quasi-random numbers instead of pseudo random numbers — quasi-random numbers form low-discrepancy sequences. Discrepancy is a measure of non uniformity of a sequence of points placed in a hypercube $[0,1]^d$. The most popular distance measure is the star-discrepancy[22], D_N^*. A sequence $x_1, \ldots, x_N$ of points in $[0,1]^d$ is a low-discrepancy sequence if for any $N > 1$:

$$D_N^*(x_1, \ldots, x_N) < c(d).\frac{(logN)^d}{N} \tag{3}$$

Where the constant $c(d)$ depends on the the problem dimension d. The idea behind low-discrepancy sequences is to get a set of points in $[0,1]^d$ as close as possible to its volume.

Basic low-discrepancy sequences proposed in the literature are those of Halton[33], Sobol[35] and Faure[36]. These sequences can be viewed as a special case of generalized (t, d)-sequences[22].

Methods based in standard Monte Carlo have a probabilistic error bound with order $O(N^{-1/2})$ derived with the central limit theorem. In quasi-Monte Carlo methods, we obtain a deterministic error bound of $O((logN)^d/N)$ [22], where N is the number of generated samples and d is the dimension of quasi-random number generated. We can see that quasi-Monte Carlo methods may offer better convergence rates than Monte Carlo methods as we increase N. In fact, there have been quite successful application of quasi-Monte Carlo methods to computer graphics, computational physics, financial engineering, and approximate integrals in low-dimensionality problems. In high-dimensional problems, there has been conflicting evidence regarding the performance of quasi-Monte Carlo methods [23]. As a positive example, Cheng and Druzdzel [23] applied quasi-Monte Carlo methods in importance sampling algorithms [10] and obtained good results in networks ranging from 5 to 179 variables. There has been no work applying quasi-Monte Carlo methods with Gibbs sampling algorithms for Bayesian inference; in fact, the problem of finding an efficient algorithms based on a combination of quasi-random numbers and Gibbs sampling is still open [24]. Liao [27] applied quasi-random numbers for variance reduction in Gibbs sampler for classical statistical models, obtaining good results through randomly permuted quasi-random sequences.

4 Markov chains for generating Bayesian networks

The basic premise of this paper is that, to test new algorithms such as Gibbs sampling based on quasi-random numbers, we need randomly generated models with known average properties. Our goal then is to generate random Bayesian networks that are *uniformly* distributed. To generate random Bayesian networks, the obvious method is to generate a random DAG (directed acyclic graph), and then to generate the conditional probability distributions for that graph. Given a DAG, it is relatively easy to generate random conditional distributions using Dirichlet distributions [6]. The real difficulty is to generate random DAGs that are uniformly distributed. Many authors have used random graphs to test Bayesian network algorithms, generating these graphs in some ad hoc manner. A typical example of such methods is given by the work of Xiang and Miller [3]. By creating some heuristic graph generator, it is usually impossible to guarantee any distribution on the generated neworks.

It can be argued that any generator that produces a uniform distribution on the space of *all* DAGs is not very useful. The problem is that Bayesian networks usually have a reasonably small degree; if a generator produces graphs that are too dense, these graphs are not representative examples of Bayesian networks. So, we must be able to generate graphs uniformly over the space of graphs that are *connected*, *acyclic*, and *not very dense*. We assume that the number of arcs in a graph is a good indicator of how dense the graph is, so we assume that our

Algorithm 1: Generating Multi-connected DAG's

Input: number of nodes (n), maximum number of arcs per node ($maxD$) , number of iterations (N).
Output: Return a connected DAG with n nodes and controlled number of arcs.
01. Inicialize a simple ordered tree with n nodes, where all nodes have just one parent, except the first one that does not have any parent;
02. Repeat the next loop N times:
03. Generate uniformly a pair of distinct nodes i and j;
04. If the arc (i, j) exists in the actual graph, delete the arc, provided that the underlying graph remains connected;
05. else
06. Add the arc, provided that the underlying graph remains acyclic and satisfy $maxD$ condition;
07. Otherwise keep the same state;
08. Return the current graph after N iterations.

Fig. 2. Algorithm for Generating multi-connected DAGs.

problem is to uniformly generate connected DAGs with a given number of arcs or with restrictions on degrees.

Our approach to generate random graphs is to use Markov chains [30]. We are directly inspired by the work of Melançon et al and Sinclair on random graph generation [31, 32]. The main difference between Melançon et al's work and ours is that they let their graphs be disconnected, a detail that makes considerable difference in the correctness proofs. In this section, we just describe briefly the algorithms that we have developed for generating multi-connected graphs and polytrees, given the number of nodes and the maximum number of arcs per node. Theoretical details and proofs about this generating process can be find at Ide[6].

We can generate random graphs by simulating Markov chains. To have a Markov chain, it is enough that we can "move" from a graph to another graph in some probabilistic way that depends only on the current graph. This Markov chain will be irreducible if it can reach any graph from any graph. Also, the chain will be aperiodic if there exists a self-loop probability, i.e. there is a chance that the next generated graph is the same as the current one. Irreducibility and aperiodicity are necessary and sufficient conditions for ergoticity (that is, the chain has a stationary distribution). Then, if we prove that the transition matrix of this generating process is doubly stochastic, the unique stationary distribution is uniform.

Consider a set of n nodes (from 0 to $n-1$) and the Markov chain described by Algorithm 1. We start with a connected graph. The loop between lines 3 and 7 construct the next state from the current state. This procedure defines a transition matrix for a Markov chain. Our transitions are limited to 2 operations: add and remove arcs, providing the graph is still acyclic and connected.

Algorithm 2: Generating Polytree

Input: number of nodes (n), maximum number of arcs per node ($maxD$) , number of iterations (N).
Output: Return a polytree with n nodes and controlled number of arcs.
01. Inicialize a simple ordered tree with n nodes as in Algorithm 1.
02. Repeat the next loop N times:
03. Generate uniformly a pair of distinct nodes i and j;
04. If the arc (i, j) exists in the actual graph or if the resultant graph do not satisfy $maxD$ condition, keep the same state;
05. else
06. Invert the arc with probability 1/2 to (j, i), and then
07. Find the predecessor node k in the path between i and j, remove the arc between k and j, and add an arc (i, j) or arc (j, i) depending on the result of line 06.
08. Return the current graph after N iterations.

Fig. 3. Algorithm for generating polytrees.

A type of Bayesian network that is of great practical interest is represented by polytree structures [2]. So, we can establish another problem: to uniformly generate polytrees with n nodes. To the best of our knowledge, there exists no algorithm for random generation of polytrees so far. The process of generating polytrees is similar to Algorithm 1; consider again n nodes, and the transition matrix defined by Algorithm 2. Notice that we have constructed a ergodic Markov chain (that is, it has a stationary distribution), ensuring the irreducibility and the aperiodicity of the chain. Line 6 is important to obtain symmetry for the transition matrix, what ensure the doubly-stochastic characteristic, and make the stationary distribution to be uniform.

The maximum number of arcs per node($maxD$) condition (line 6 of Algorithm 1 and line 4 of Algorithm 2) provide us *density* control over generated Bayesian network structure. That is interesting to obtain more representative networks.

5 Experimental Results

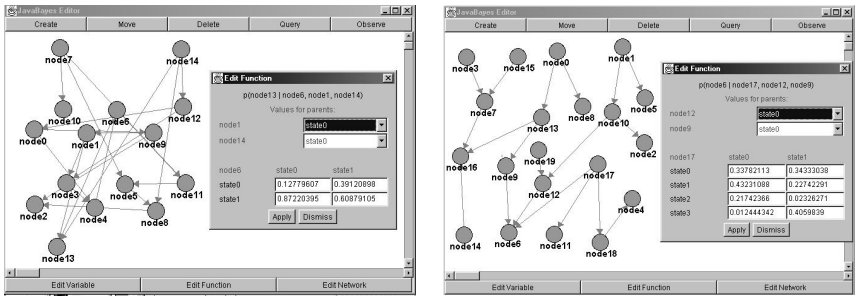
We have implemented Gibbs sampling for Bayesian networks and coupled this algorithm with pseudo and quasi random numbers. For generating (pseudo)random numbers, we have used an efficient generator from the literature, the MersenneTwister [37]. For generating quasi-random sequences, we have used an efficient variant of the Sobol sequence, *Gray code*, that was proposed by Antonov and Saleev[34], and implemented by Bratley and Fox[25]. A Faure sequence generator implemented by Fox[26] was used, and we noticed that results given by Faure sequences are always worse than results by Sobol sequences. Quasi-random sequences are used in place of (pseudo) random numbers (line 5 of Algorithm *Gibbs sampler*) , where for each different variable we use one dimension of quasi-random

sequence. For example, for a network with 10 nodes, we need a quasi-random sequence with dimension 10.

We have implemented the Gibbs sampler according to Gelfand and Smiths purposes[29], that was used by Casella[20]. Their suggestion is to use m samples picked at the end of k-iterations. The idea is to pick samples with period of k-iterations, after burn-in phase. Determining the value of k is a difficult issue. We tested the Gibbs sampling for several values of k, but no substantial difference was noted from the direct implementation (figure 1).

Following the purpose of Cheng and Druzdzel[23], we used the Mean Square Error (MSE) as a measure of efficiency for Gibbs sampling algorithms. MSE is the square root of the sum of square differences between the exact result and the approximate result obtained, for all set of variables of the network.

Algorithms 1 and 2 were implemented to construct random structures of networks, and random conditional distributions were generated with Dirichlet distributions [6]. Network structures are generated given the number of nodes and the maximum number of arcs per node, and some examples can be found at figure 4.



(a) (b)

Fig. 4. Bayesian networks are generated in format compatible with freely available JavaBayes system (http://www.cs.cmu.edu/~javabayes):(a) Multi-connected graph with 15 nodes and maximum of 3 arcs per node, (b) Polytree with 20 nodes and maximum of 3 arcs per node.

At figures 5 and 6, we have results showing relationships between number of samples and accuracy of both approximation (Gibbs sampling with pseudo-sequence and Sobol sequence), measured by MSE. For each of the tests, we have generated 1000 uniformly distributed Bayesian networks and tested quasi-Monte Carlo methods in Gibbs sampling. We varied the number of samples from 250 to 250.000 and generated networks with 5, 10, 15 and 35 nodes. The graphs shown the mean value of MSE trough all 1000 networks, and their deviation. Notice that, for left example at figure 5, with 250 samples, we have a deviation of 53 percent. It means that MSE result depends on the particular structure of the network and their probabilities.

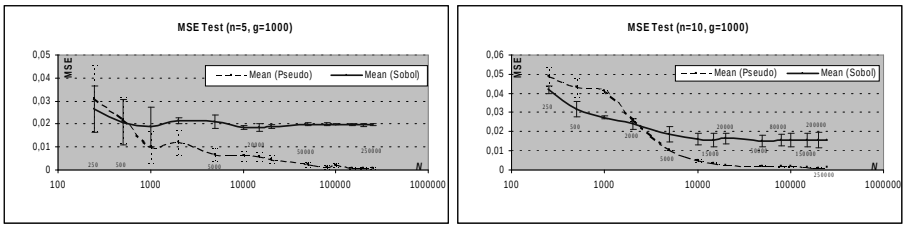


Fig. 5. Mean of MSE's and their deviation of Gibbs sampler tested with 1000 uniformly generated Bayesian network with 5 and 10 nodes, varying between N=250 to 250.000 number of simulation

We can see (figure 5 and 6) that Gibbs sampling with Sobol sequence has better initial convergence rates than pseudo random numbers, but after some number of sample (at example, 500 to 5000), it converges to a constant error and MSE given by pseudo numbers converges to zero. This constant error is possibly due to statistical dependencies between quasi-random sequences and Gibbs sampling successive draws. Such dependencies could be broken trough a renewal process[27]. We can state then, based on empirical results, that the direct use of quasi-random sequences does not work well at inference in Bayesian networks, using Gibbs sampling, refuting the hypothesis stated at Section 3.

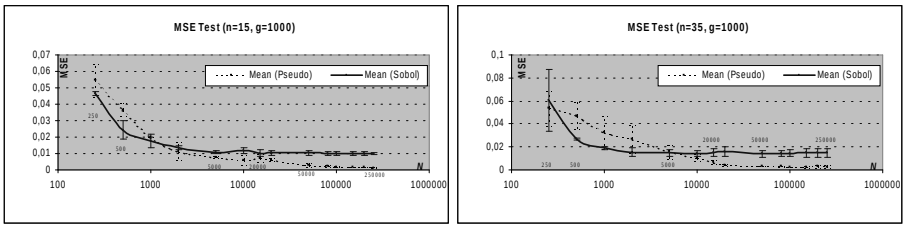


Fig. 6. Mean of MSE's and their deviation of Gibbs sampler tested with 1000 uniformly generated Bayesian network with 15 and 35 nodes, varying between N=250 to 250.000 number of simulation

6 Conclusion

We can summarize this work as follows: we have introduced a consistent method for testing inference algorithms and obtained new empirical results on quasi-Monte Carlo methods in Gibbs sampling. To produce such tests, we have introduced algorithms for generation of uniformly distributed random Bayesian networks, both as multi-connected networks and polytrees. Our algorithms are flexible enough to allow specification of maximum numbers of arcs and maximum degrees, and to incorporate any of the usual characteristics of Bayesian

networks. We suggest that the methods presented here provide the best available scheme at the moment for producing valid tests and experiments with Bayesian networks.

Unfortunately, we have not solved the problem of using quasi-random sequences in Gibbs sampling algorithms. The scheme that we provide in this work allows us to reach this kind of empirical conclusion and gives supports for producing valid tests in future works. Renewal process could be applied to broke dependencies as mentioned at Section 5.

Acknowledgements

We thank Nir Friedman for suggesting the Dirichlet distribution method, and Robert Castelo for pointing us to Melançon et al's work. We thank Guy Melançon for confirming that the idea of Algorithm 1 was sound and for making his DagAlea software available. We also thank Jaap Suermondt and Alessandra Potrich for providing important ideas, and Y. Xiang, P. Smets, D. Dash, M. Horsh, E. Santos, and B. D'Ambrosio for suggesting valuable procedures. This work was supported by FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo).

References

1. Jensen, F.V.: An Introduction to Bayesian Networks. Springer-Verlag, NewYork (1996)
2. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kauffman (1988)
3. Xiang, Y., Miller, T.: A Well-Behaved Algorithm for Simulating Dependence Structures of bayesian Networks. International Journal of Applied Mathematics, Vol. 1, No. 8 (1999), 923–932
4. Castillo, E., Gutierrez, J.M., Hadi, A.S.: Expert Systems and Probabilistic Network Models. Springer-Verlag, New Yor (1997)
5. Ide, J.S.: Raciocnio Probabilstico em Sistemas Embarcados. Anais do XXI Congresso da Sociedade Brasileira de Computao, Ed. Ana Teresa Martins, Fortaleza (2001)
6. Ide, J.S., Cozman, F.G.: Random generation of Bayesian networks. Proc. of XVI Brazilian Symposium on Artificial Intelligence, Springer-Verlag (2002)
7. Ramos, F.T., Cozman, F.G., Ide, J.S.: Embedded bayesian Networks: Anyspace, Anytime Probabilistic Inference. AAAI/KDD/UAI-2002 Joint Workshop on Real-Time Decision Support and Diagnosis Systems, AAAI Press, California (2002), 13–19
8. Cooper, G.F.: The Computational Complexity of Probabilistic Inference using Bayesian Belief Networks. AI Journal, Vol.42, Oct. (1997), 393–405
9. Henrion, M.: Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Uncertainty in Artificial Intellgience 2, New York, N.Y., Elsevier Science Publishing Company, Inc.(1988), 149–163
10. Shachter, R.D., Peot, M.A.: Simulation approaches to general probabilistic inference on belief networks. In Uncertainty in Artificial Intelligence 5, New York, N.Y., Elsevier Science Publishing Company, Inc.(1989), 221–231

11. Cooper, G.: NESTOR: A computer-based medical diagnostic aid that integrates causal and probabilistic knowledge. Ph.D. Dissertation, Medical Informatics Sciences, Stanford University, Stanford, CA. (1985)
12. Horvitz, E., Suermondt, H.J., Cooper, G.F.: Bounded conditioning: Flexible inference for decisions under scarce resources. In: Proceedings of Conference on Uncertainty in Artificial Intelligence, Windsor, Aug. (1989), 182–193
13. Jaakkola, T., Jordan, M.: Variational probabilistic inference and the qmr-dt database. Journal of Artificial Intelligence Research, Vol.10 (1999), 291–322
14. Sobol, I.M.: A Primer for Monte Carlo methods. CRC Press, Inc., Florida (1994)
15. Gilks, W.R., Richardson, S., Spiegelhalter, D.J.: Markov Chain Monte Carlo in Practice. Chapman and Hall (1996)
16. Gamerman, D.: Markov Chain Monte Carlo. Stochastic simulation for Bayesian inference. Texts in Statistical Science Series. Chapman and Hall, London (1997)
17. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equations of state calculations by fast computing machine. Journal Chem. Phys., Vol. 21 (1953), 1087–1091
18. Hastings, W.K.: Monte Carlo Sampling Methods Using Markov Chains and Thier Applications. Biometrika, Vol.57 (1970), 97–109
19. Geman, S., Geman, D.: Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. IEEE Trans. Pattn. Anal. March. Intel., 6 (1984), 721–741
20. Casella, G., George, E.I.: Explaining the Gibbs Sampler. The American Statistician, Vol.46 (1992), 167–174
21. Gilks, W.R., Thomas, A., Spiegelhalter, D.J.: A language and program for complex Bayesian modelling. The Statistician, Vol.43 (1994), 169–178
22. Niederreiter, H.: Random Number Generation and Quasi-Monte Carlo Methods. SIAM, CBMS-NSF Regional Conf. Series in Applied Mathematics, No.63 (1992)
23. Cheng, J., Druzdzel, M.J.:Computational Investigation of Low-Discrepancy Sequences in Simulation Algorithms for bayesian Networks. In Proc. 16th Conf. On Uncertainty in Artificial Intelligence, Morgan Kaufmann (1999), 72–81
24. Fang, K.T., Wang, Y.: Number-theoretic Methods in Statistics. Chapman and Hall, London (1994), 98–100
25. Bratley, P., Fox, B.L.: Algorithm 659: Implementing Sobols quasi-random sequence generator. ACM Transactions on Mathematical Software, 14(1) (1988), 88–100
26. Fox, B.L.: Algorithm 647: Implementation and relative efficiency of quasirandom sequence generators. ACM Transactions on Mathematical Software, 12(4) (1986), 362–376
27. Liao, J.G.: Variance Reduction in Gibbs Sampler Using Quasi Random Numbers. Journal of Computational and Graphical Statistics, No.3 (1998), 253–266
28. Shaw, J.E.H.: A Quasirandom Approach to Integration in Bayesian Statistics. The Annals of Statistics, Vol. 16, No. 2 (1988), 895-914
29. Gelfand, A.E., Smith, A.F.M.: Sampling-Based Approaches to Calculating Marginal Densities. Journal of American Statistical Association, 85 (1990), 398–409
30. Resnick, S.I.: Adventures in Stochastic Processes. Birkhäuser, Boston (1992)
31. Melançon, G., Bousque-Melou, M.: Random Generation of Dags for Graph Drawing. Dutch Research Center for Mathematical and Computer Science (CWI). Technical Report INS-R0005 February (2000)
32. Sinclair, A.: Algoritms for Random Generation and Counting: A Markov Chain Approach. Progress in Theoretical Computer Science. Birkhaüser, Boston (1993).
33. Halton, J.H.: On the efficiency of certain quasirandom sequences of points in evaluating multidimensional integrals. Numerische Mathmatik, Vol.2 (1960), 84–90

34. Antonov, I.A., Saleev, V.M.: An economic method of computing lp_t-sequences. U.S.S.R. Computational Mathematics and Mathematical Physics, 19 (1979), 252–256
35. Sobol, I.M.: On the distribution of points in a cube and the approximate evaluation of integrals. U.S.S.R. Computational Mathematics and Mathematical Phisics, Vol.7(4) , Moscow (1967), 86–112
36. Faure, H.: Discrepance de suites associees a un systeme de numeration en dimension s. Acta Arithmetica, Vol.41 (1982), 337–351
37. Matsumoto, M., Nishimura, T.: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM Transactions on Modeling and Computer Simulation, Vol.8, No.1 (1998), 3–30