Coupling from the Past with Randomized Quasi-Monte Carlo

Pierre L'Ecuyer and Charles Sanvido

Département d'Informatique et de Recherche Opérationnelle
Université de Montréal, C.P. 6128, Succ. Centre-Ville
Montréal (Québec), H3C 3J7, CANADA

May 17, 2007

Abstract

The coupling-from-the-past (CFTP) algorithm of Propp and Wilson, also called perfect sampling, permits one to sample exactly from the stationary distribution of an ergodic Markov chain. By using it n times independently, we obtain an independent sample from that distribution. A more representative sample can be obtained by creating negative dependence between these n replicates; other authors have already proposed to do this via antithetic variates, Latin hypercube sampling, and randomized quasi-Monte Carlo (RQMC). We study a new, often more effective, way of combining CFTP with RQMC, based on the array-RQMC algorithm. We provide numerical illustrations for Markov chains with both finite and continuous state spaces, and compare with the RQMC combinations proposed earlier.

KEY WORDS: Variance reduction, randomized quasi-Monte Carlo, Markov chain, exact sampling, perfect sampling, coupling from the past.

1 Introduction

The Monte Carlo (MC) method is a key tool for estimating mathematical expectations of the form

$$\mu = \int_{\mathcal{S}} c(x) d\pi(x), \tag{1}$$

where π is a probability measure defined over some multidimensional measurable space $(\mathcal{S}, \mathcal{F})$, and $c : \mathcal{S} \to \mathbb{R}$ is a measurable cost function. This setting often occurs naturally because we want to estimate the steady-state average cost for a system whose evolution is modeled by an ergodic Markov chain $\{X_j,\, j \geq 0\}$ with huge state space $\mathcal{S}$ and with complicated dynamics (Law and Kelton, 2000). Typically, in this case, we have little a priori clue of how π might look like but we can easily simulate the Markov chain. In other cases, we have the reverse situation: the form of π is given, sometimes up to a normalizing constant, but we have no direct way of generating samples from π. However, we can construct an artificial Markov chain with steady-state distribution π, e.g., via the Metropolis-Hastings algorithm. This is known as Markov chain Monte Carlo (MCMC).

Regardless of where the Markov chain originates from, a classical way of estimating μ in (1) by MC is to select two large integers $t > t_0 > 0$, start the chain from some fixed initial state $X_0 = x_0$, simulate it for t steps, and take the average cost over the steps $t_0 + 1$ to t as an estimator of μ. The first t_0 steps are discarded in order to reduce the bias due to the arbitrary choice of the initial state. But some bias usually remains regardless of the choice of t_0, and it is often difficult to select t_0 in a way that the bias is guaranteed to be negligible while the estimator remains efficient (Heidelberger and Welch, 1983; Law and Kelton, 2000; Glynn, 2006; Awad and Glynn, 2007). This initial bias problem would disappear if we knew how to generate the initial state X_0 exactly from π.

Propp and Wilson (1996) proposed an algorithm that does precisely this. Their method, sometimes named *perfect sampling*, or *exact sampling*, uses a concept called *coupling from the past* (CFTP) to generate a state from π. Conceptually, one simulates the chain from all possible states, in parallel, from some time $-T_1 < 0$ to time 0. If all copies of the chain are in the same state at time 0, then this state has distribution π. Otherwise, one can try again from time $-T_2 < -T_1$, making sure that the same random numbers are used to simulate the chain at all steps from $-T_1$ to 0. This process is repeated by going further in the past until all the chains have coalesced (are in the same state) at time 0.

Constructing an unbiased estimator of μ is one thing, but controlling the variance of this estimator is also an important issue. The variance can of course be reduced by simulating a larger number of independent copies of the estimator and taking the average; with n independent copies, the variance is divided by n. A further improvement is to induce

negative dependence between the copies, in some sense, to reduce the variance of the average. Some ways of doing this are based on the idea of antithetic variates (AV) and their generalizations, which include Latin hypercube sampling (LHS), iterative LHS (ILHS), and randomized quasi-Monte Carlo (RQMC) (Owen, 1998; L'Ecuyer and Lemieux, 2002; Craiu and Meng, 2005). The combination of CFTP with dependence-induction methods such as AV, ILHS, and RQMC has been studied and experimented by Craiu and Meng (2000, 2005); Lemieux and Sidorsky (2006). In empirical experiments with small examples, these methods did reduce the variance significantly and RQMC was the best performer (Lemieux and Sidorsky, 2006).

In general, RQMC can be quite effective to estimate integrals of smooth functions in small or moderate dimension. But simulating a Markov chain over a large number of steps can be viewed as applying MC to estimate a large-dimensional integral, and RQMC usually loses its punch when faced with such integrals. However, a new RQMC method specially designed for Markov chains, called array-RQMC (L'Ecuyer et al., 2007), is often very effective in this situation. The idea of this method is to simulate n copies of the chain in parallel, advancing all copies by one step at each iteration, and to induce negative dependence between these copies, in a way that the empirical distribution of the n states at any given step provides a better estimate of the theoretical distribution of the state at that step, and a lower-variance for the average cost, than if the n copies were simulated independently. Given that both CFTP and array-RQMC work by simulating several copies of the Markov chain in parallel, the idea of using them together appears natural at first sight.

The aim of this article is to examine how CFTP and array-RQMC can be effectively combined to produce a lower-variance and more efficient estimator than CFTP alone. In Section 2, we define the Markov chain model considered here. We also recall the CFTP technique and some of its variants. The RQMC and array-RQMC methods are discussed in Section 3. We then examine how CFTP can be combined with RQMC sampling in Section 4, and with array-RQMC in Section 5. This second combination turns out to be less obvious than expected but we propose a practical way of making it work. In Section 6, we report some empirical results. In Section 7, we conclude by discussing directions for further research.

2 Markov Chain Model and Coupling from the Past

2.1 Markov Chain Setting

Our basic model is a Markov chain $\{X_j,\, j \geq 0\}$ with state space $\mathcal{S}$, defined via the stochastic recurrence:

$$X_j = \varphi(X_{j-1}, \mathbf{U}_j), \quad j \geq 1, \tag{2}$$

where $\mathbf{U}_1, \mathbf{U}_2, \ldots$ are independent random vectors uniformly distributed over the d-dimensional unit cube $[0,1)^d$ (i.i.d. $U(0,1)^d$, for short). In our numerical examples, we will have $d = 1$. For each $j \geq 1$, there is a state-dependent cost $c(X_j)$, where $c : \mathcal{S} \to \mathbb{R}$ is the cost function. We implicitly make all the required measurability assumptions on $\mathcal{S}$, φ, c, and so on. We also suppose that the chain has an equilibrium and limiting distribution π, so that

$$\mu = \int_{\mathcal{S}} c(x) d\pi(x) = \lim_{t\to\infty} \frac{1}{t} \sum_{j=1}^{t} \mathbb{E}[c(X_j)]$$

is the steady-state average cost per step, which we want to estimate. This implies that if we can generate $X_0 \sim \pi$, then $X_j \sim \pi$ for all $j \geq 0$ and $(1/t)\sum_{j=1}^{t} c(X_j)$ is an unbiased estimator of μ for any t.

2.2 The CFTP Algorithm

Backward algorithm. The conceptual idea of the CFTP algorithm is to generate $\mathbf{U}_0, \mathbf{U}_{-1}$, $\mathbf{U}_{-2}, \ldots$ (backward) to find a random time $-T$ in the past such that

$$X_0 = \varphi(\varphi(\cdots\varphi(X_{-T}, \mathbf{U}_{-T+1}), \cdots, \mathbf{U}_{-1}), \mathbf{U}_0) \tag{3}$$

takes the same value for all states $X_{-T} \in \mathcal{S}$. When this happens, we say that *total coalescence has occurred at time 0* for this particular sample path and this particular T. This means that if we start one copy of the chain in each state $x \in \mathcal{S}$ at step $-T$, then all the copies would have collapsed into a single state X_0 at step 0. Any T that satisfies this condition is a *backward coupling time* and the corresponding X_0 is the *coalescence state*. The smallest such T, say T_*, is the minimal backward coupling time. Propp and Wilson (1996) have proved that whenever we have total coalescence at time 0, the coalescence state X_0 has distribution π, exactly. This is easy to explain: when we have coalescence, X_0 is independent of the state at time T_* and its realization would remain the same if that state X_{T_*} was generated from the stationary distribution.

Note that for any fixed $T \geq 0$, (3) defines a random mapping $\Psi_T : \mathcal{S} \to \mathcal{S}$ via $\Psi_T(X_{-T}) = X_0$. In other words, Ψ_0 is the identity and $\Psi_T(\cdot) = \Psi_{T-1}(\varphi(\cdot, \mathbf{U}_{-T+1}))$. Each mapping Ψ_T is random because it is a function of $\mathbf{U}_{-T+1}, \ldots, \mathbf{U}_0$. In terms of those mappings, T_* can be defined as the smallest T such that $\Psi_T(\mathcal{S})$ contains a single state. A conceptually simple way of finding T_* is to compute the mappings Ψ_T for $T = 1, 2, \ldots$, until Ψ_T maps the state space to a singleton. The evolution of this backward CFTP algorithm can be described by the Markov chain $\mathcal{Y} = \{Y_j = \Psi_j,\ j \geq 0\}$, whose state at step j is the mapping Ψ_j. If $\mathcal{S}$ is finite, say $\mathcal{S} = \{0, \ldots, M-1\}$, the state of $\mathcal{Y}$ at step j can be represented by an M-dimensional vector $Y_j = \Psi_j = (\Psi_j(0), \ldots, \Psi_j(M-1))$.

For large state spaces, this backward simulation approach, where the mappings are entirely computed in succession, is generally inefficient. Forward simulation is usually more convenient.

Forward algorithm. In the case where $\mathcal{S}$ is finite, a direct way of finding a backward coupling time T and coalescence state X_0 is by simulating in the forward direction as many copies of the chain as the number of states, as follows. Select some integer $T_1 > 0$, start one chain in each state at time $-T_1$, simulate all these chains from time $-T_1$ to time 0 using the same sequence of random numbers $\mathbf{U}_{-T_1+1}, \mathbf{U}_{-T_1+2}, \ldots, \mathbf{U}_{-1}, \mathbf{U}_0$, and check if total coalescence has occurred at time 0. If not, then select a larger integer $T_2 > T_1$, and try again, making sure that exactly the same sequence of random numbers $\mathbf{U}_{-T_1+1}, \mathbf{U}_{-T_1+2}, \ldots, \mathbf{U}_{-1}, \mathbf{U}_0$ is used for all the chains over the last T_1 steps. If coalescence has not yet been achieved, repeat with $T_3 > T_2$, making sure again that the same random numbers are reused over the last T_2 steps, and so on. We may choose $T_j = 2T_{j-1}$, for example. The requirement of reusing the same random numbers at the same steps is crucial; without it, (3) does not hold and the theorem of Propp and Wilson does not apply. This is the reason why simulating the chains forward from time 0 until the first time T when they are all in the same state, and returning this common state X_T, is not equivalent to CFTP; it does not provide a state X_T generated from π. With the latter method, when T is increased by 1, the additional random number is used for the last step of the chains, whereas with CFTP, it is used for the first step.

Large, possibly infinite, state spaces can often be handled as follows. Suppose that $\mathcal{S}$ has a partial order $\leq$ and contains two finite subsets $\mathcal{S}_0$ and $\mathcal{S}_1$ such that for each $x \in \mathcal{S}$, there are two states $x_0 \in \mathcal{S}_0$ and $x_1 \in \mathcal{S}_1$ such that $x_0 \leq x \leq x_1$. Suppose also that $\varphi(\cdot, \mathbf{u})$ is nondecreasing with respect to that partial order, for each $\mathbf{u} \in (0,1)^d$. Then, it suffice to simulate copies of the chains from all states of $\mathcal{S}_0 \cup \mathcal{S}_1$. Whenever these chains have coalesced, we know that if we had started chains from all states $x \in \mathcal{S}$ rather than only from the states $x \in \mathcal{S}_0 \cup \mathcal{S}_1$, all the chain would also have coalesced at that point, even if $\mathcal{S}$ is infinite and nondenumerable. A special case of this is when the state space has a largest state and a smallest state.

In the following, we assume that $\mathcal{S}_0 \cup \mathcal{S}_1 = \{0, \ldots, M-1\}$, for notational simplicity. If $\mathcal{S}$ is finite and unordered, we just take $\mathcal{S}_0 = \mathcal{S}_1 = \mathcal{S} = \{0, \ldots, M-1\}$. Under this assumption, the forward version of the CFTP algorithm simulates the chain $\mathcal{X} = \{X_j,\, j \geq 0\}$ from the M initial states $0, \ldots, M-1$, and the CFTP process can be represented by a single Markov chain $\mathcal{Y} = \{Y_j,\, j \geq 0\}$, where the vector $Y_j = (X_{0,j}, \ldots, X_{M-1,j})$ represents the states of the M instances of the original chain $\mathcal{X}$ at step j. The chain $\mathcal{Y}$ is simulated from time $-T_1$ to time 0, and we then verify if all coordinates of Y_0 are the same.

3 MC, RQMC, and Array-RQMC

We now consider an arbitrary DTMC $\mathcal{Y} = \{Y_j,\, j \geq 0\}$ and recall how the *Monte Carlo* (MC), *randomized quasi Monte carlo* (RQMC), and *array-RQMC* methods would simulate n copies of this chain from time 0 up to some stopping time τ. This chain may represent the backward or forward CFTP process. We assume that it can be realized via the recurrence

$$Y_j = \psi_j(Y_{j-1}, \mathbf{U}_j), \tag{4}$$

where the vectors $\mathbf{U}_j = (U_{(j-1)d+1}, \ldots, U_{jd})$ are i.i.d. $U(0,1)^d$, and ψ_j is an appropriate function that defines the recurrence at step j. Suppose we want to estimate the expectation $\mu = \mathbb{E}[\gamma(Y_\tau)]$ for some cost function $\gamma : \mathcal{S} \to \mathbb{R}$.

We can estimate μ by simulating n copies of the chain via (4) and taking the average of the n copies of $\gamma(Y_\tau)$. If $Y_{i,j}$ is the state of the ith chain at step j, we have

$$Y_{i,j} = \psi_j(Y_{i,j-1}, \mathbf{U}_{i,j}), \tag{5}$$

where each $\mathbf{U}_{i,j} = (U_{i,(j-1)d+1}, \ldots, U_{i,jd})$ is a vector (or point) in the unit hypercube $(0,1)^d$. Let τ_i be the realization of τ for the ith copy. Our estimate of μ is then

$$\bar{Y}_n = \frac{1}{n} \sum_{i=0}^{n-1} \gamma(Y_{i,\tau_i}). \tag{6}$$

In the *MC method*, the $U_{i,j}$ are taken as "good" imitations of independent random variables uniformly distributed over $(0,1)$, obtained from a (pseudo)random number generator (Law and Kelton, 2000; L'Ecuyer, 2006). The n copies of the chain are considered as independent.

Let s be the smallest integer so that $\mathbb{P}[\tau d \leq s] = 1$; if no such integer exists, we take $s = \infty$. Note that the random variable $\gamma(Y_\tau)$ can be written as a $\gamma(Y_\tau) = f(U_1, U_2, \ldots)$ for some function $f : (0,1)^s \to \mathbb{R}$, and μ is simply the integral of this function f over the s-dimensional unit hypercube $(0,1)^s$. The *(classical) RQMC method* is designed to approximate such integrals. For this method, we use a set of s-dimensional points $\mathbf{V}_i = (U_{i,1}, \ldots, U_{i,s})$, for $i = 0, \ldots, n-1$, with the following two properties:

(a) for each i, the coordinates $U_{i,1}, U_{i,2}, \ldots$ of $\mathbf{V}_i$ are i.i.d. $U(0,1)$, i.e., $\mathbf{V}_i$ has the *uniform distribution* over $[0,1)^s$;

(b) the point set $P_n = \{\mathbf{V}_0, \ldots, \mathbf{V}_{n-1}\}$ is *more evenly distributed* over the unit cube $(0,1)^s$ than a typical set of independent random points.

Precise definitions of "more evenly distributed," in terms of measures of discrepancy with respect to the uniform distribution, are detailed in Niederreiter (1992); Owen (1998); L'Ecuyer and Lemieux (2002), and other references given there. A point set P_n that satisfies these two conditions is called an *RQMC point set*. Examples of RQMC point sets include randomly-shifted lattice rules, digitally shifted digital nets, and scrambled nets (L'Ecuyer and Lemieux, 2002). Intuitively, the aim is that the empirical distribution of $\{\gamma(Y_{i,\tau_i}),\, i = 0, \ldots, n-1\}$ provides a better approximation of the theoretical distribution of the random variable $\gamma(Y_\tau)$ than with MC, in order to reduce the variance of the average (6). This is equivalent to inducing negative correlation between $\gamma(Y_{i,\tau_i})$ and $\gamma(Y_{j,\tau_j})$, on average over all pairs $i \neq j$. This approach is typically more efficient than MC if s is small or if f has *low effective dimension* in some sense (Owen, 1998; L'Ecuyer and Lemieux, 2002). The smoothness of f also plays an important role; under sufficient smoothness conditions, one can prove that the variance converges to zero (as a function of n) at a faster rate for RQMC than for MC.

In the *Array-RQMC method*, detailed in L'Ecuyer et al. (2007), we simulate n copies of the chain $\mathcal{Y}$ in parallel. At step j, for $j \geq 1$, we advance the n copies by one transition, using a $(d+1)$-dimensional *modified RQMC point set*, defined as a set

$$P'_{n,j} = \{\mathbf{U}'_{i,j} = ((i + 0.5)/n, \mathbf{U}_{i,j}),\, 0 \leq i < n\}$$

with the following properties:

(c) $\mathbf{U}_{i,j}$ is a random vector uniformly distributed over $[0,1)^d$ for each i;

(d) $P'_{n,j}$ is "highly uniform" in $[0,1)^{d+1}$, in a sense that we leave open (as in our definition of RQMC point set).

(d) $P_{n,j} = \{\mathbf{U}_{0,j}, \ldots, \mathbf{U}_{n-1,j}\}$ is an RQMC point set in $[0,1)^d$;

Such point sets $P'_{n,j}$ are easy to construct: just take a $(d+1)$-dimensional RQMC point set, sort the points by order of their first coordinate, and replace the first coordinate of point i by $(i + 0.5)/n$. For the most common types of highly-uniform point sets, including digital nets and lattice rules of rank 1, the points can be enumerated in a way that the first coordinate of point i before the randomization is i/n; then it suffices to randomize only the other coordinates. This is what we have used in all our experiments.

Array-RQMC also uses a *sorting function* $v : \mathcal{Y} \to \mathbb{R}$ to sort the n states by increasing order of $v(y)$ at each step. Ideally, v should be chosen so that any two states x and y with $v(x) = v(y)$ should be approximately *equivalent*. The intuition is that if this is true, and if the empirical distribution of $v(Y_{0,j}), \ldots, v(Y_{n-1,j})$ at step j provides a better approximation of the theoretical distribution F_j of $v(Y_j)$ than for MC, for each j, then $\bar{Y}_n$ would have

smaller variance than for MC. To maintain the good approximation from step $j-1$ to step j, the idea is to sort the chains according to their values of $v(Y_{i,j-1})$ at step $j-1$, and use the modified RQMC point set $P'_{n,j}$ to determine the next states in a way that the empirical conditional distribution of $v(Y_{i,j})$, given that $v(Y_{i,j-1})$ belongs to a specific interval, matches very well the theoretical conditional distribution. More detailed explanations can be found in L'Ecuyer et al. (2007). With a good choice of v, this method often outperforms classical RQMC when s is large. A description of this algorithm will be given later, in the context of CFTP.

For both the RQMC and array-RQMC method, the variance can be estimated by replicating the whole scheme, say, r times, independently, and using the empirical variance of the r independent copies of (6) as an unbiased variance estimator.

4 Combining CFTP with RQMC

Lemieux and Sidorsky (2006) propose a direct application of RQMC to the Markov chain $\mathcal{Y}$ that describes the *backward CFTP algorithm*. This goes as follows. Select an infinite-dimensional RQMC point set of cardinality n and use it to run n replicates of (3), one for each point $\mathbf{V}_i = (U_{i,1}, U_{i,2}, \dots)$, by putting $\mathbf{U}_0 = (U_{i,1}, \dots, U_{i,d})$, $\mathbf{U}_{-1} = (U_{i,d+1}, \dots, U_{i,2d})$, $\mathbf{U}_{-2} = (U_{i,2d+1}, \dots, U_{i,3d})$, and so on. In other words, the algorithm uses each new block of d coordinates of the point $\mathbf{V}_i$ to go one more step backward with the chain. Their implementation does the backward simulation explicitly. This is practical only if the state space $\mathcal{S}$ is finite and not too large. The dimension s must be infinite because there is no deterministic upper bound on the number of steps required for coalescence.

An equivalent forward implementation would proceed as follows. In the first stage, we start the n copies of the CFTP process at time $-T_1$. For the ith copy, we simulate the T_1 steps using the first $T_1 d$ coordinates of $\mathbf{V}_i$, where blocks of d coordinates are taken in reverse order. That is, we put $X_{-j+1} = \varphi(X_{-j}, U_{i,(j-1)d+1}, \dots, U_{i,jd})$ for $j = T_1, \dots, 1$. If coalescence does not occur at step 0 for this ith copy, we start again from $-T_2 < -T_1$, using the coordinates $T_1 d + 1, \dots, T_2 d$ of $\mathbf{V}_i$ by blocks of size d taken in reverse order to simulate the first $T_2 - T_1$ steps of the chain. For the copies that did not coalesce after starting from $-T_2$, we start from $-T_3 < -T_2$, and so on.

This could be awkward to implement, especially if $d > 1$, because software implementations of infinite-dimensional RQMC point sets are normally not designed to enumerate the coordinates in a different order than the natural order. A simple solution is to just use the coordinates of $\mathbf{V}_i$ in the most convenient order: the first d coordinates to go from time $-T_1$ to $-T_1 + 1$, ..., the coordinates $(T_1 - 1)d + 1$ to $T_1 d$ to go from time

-1 to time 0, then the coordinates $T_1 d + 1$ to $(T_1 + 1)d$ to go from time $-T_2$ to $-T_2 + 1$, and so on. This gives $X_{-T_1+j} = \varphi(X_{-T_1+j-1}, U_{i,(j-1)d+1}, \ldots, U_{i,jd})$ for $j = 1, \ldots, T_1$, then $X_{-T_2+j} = \varphi(X_{-T_2+j-1}, U_{i,(T_1+j-1)d+1}, \ldots, U_{i,(T_1+j)d})$ for $j = 1, \ldots, T_2 - T_1$, and so on. This is how we define our implementation of *RQMC with forward CFTP*. Thus, it differs from that of Lemieux and Sidorsky (2006).

5 Combination with Array-RQMC

We now examine the combination with array-RQMC, first for the backward CFTP algorithm, then for the forward approach.

For the backward algorithm, we suppose again that $\mathcal{S} = \{0, \ldots, M-1\}$ and apply the array-RQMC method to simulate the Markov chain $\mathcal{Y}$ defined earlier, with $Y_j = (\Psi_j(0), \ldots, \Psi_j(M-1))$. For this, we need to define a sorting function $v : \mathcal{S}^M \to \mathbb{R}$. Finding a good v is the least obvious part of the approach. The algorithm simulates n dependent copies of the chain $\mathcal{Y}$, in parallel. The state of the ith copy at step j is $Y_{i,j} = \Psi_j^{(i)} = (\Psi_j^{(i)}(0), \ldots, \Psi_j^{(i)}(M-1)$, where $\Psi_j^{(i)}(m)$ is the realization of $\Psi_j(m)$ for the ith copy of the chain. These realizations evolve according to the recurrence $\Psi_j(m) = \Psi_{j-1}(\varphi(m, \mathbf{U}_{-j+1}))$, as explained earlier, at any step and for any copy. At each step j, we use a modified RQMC point set $P'_{n,j}$ and take the randomized point $\mathbf{U}_{i,j}$ in place of $\mathbf{U}_{-j+1}$ in the recurrence, for the ith copy of the chain. Note that the chains are reordered at each step, using the sorting function v, so the ith chain at step j is generally not the same chain as the ith chain at step $j-1$. For this reason, we cannot write $\Psi_j^{(i)}(m) = \Psi_{j-1}^{(i)}(\varphi(m, \mathbf{U}_{i,j}))$. Sorting the chains is done only to achieve a specific assignment of the points to the chains; it is in fact equivalent to using the points of $P'_{n,j}$ in a different order at the successive steps.

Algorithm 1 summarizes the *backward algorithm with array-RQMC*. The variable N denotes the number of chains $\mathcal{Y}$ that have not yet coalesced. We start with $N = n$ and the algorithm stops when $N = 0$. The variable S accumulates the sum of values of $c(\Psi_j^{(i)}(0))$ at coalescence, whose average is $\bar{Y}_n$.

A naive way of combining the *forward algorithm with array-RQMC* could proceed as follows. Suppose that $\mathcal{S}_0 \cup \mathcal{S}_1 = \{0, \ldots, M-1\}$. We simulate n CFTP processes in parallel, from time $-T_1 < 0$ to time 0. At step j, for $j = -T_1 + 1$ to 0, let $Y_j^{(i)} = (X_{0,j}^{(i)}, \ldots, X_{M-1,j}^{(i)})$ be the state of the CFTP process i; we sort the n processes by order of $v(Y_j^{(i)})$, we randomize afresh the RQMC point set P_n and use these points to advance all the CFTP processes by one step. For those processes that did not reach coalescence at step 0, we start again from time $-T_2 < -T_1$, using new randomizations of the RQMC point set from step $-T_2 + 1$ to $-T_1$, but the same randomizations as in the first pass from step $-T_1 + 1$ onward. A major

Algorithm 1 Array-RQMC with the Backward CFTP Algorithm

$N \leftarrow n$; $S := 0.0$;
for $(i = 0; i < n; i{+}{+})$ **do**
  for $(m = 0; m < M; m{+}{+})$ **do**
    $\Psi_0^{(i)}(m) \leftarrow m$;
  end for
end for
for $(j = 1; N > 0; j{+}{+})$ **do**
  Randomize P_n afresh into $P_{n,j} = \{\mathbf{U}_{0,j}, \ldots, \mathbf{U}_{n-1,j}\}$;
  for $(i = 0; i < N; i{+}{+})$ **do**
    for $(m = 0; m < M; m{+}{+})$ **do**
      $\Psi_j^{(i)}(m) \leftarrow \varphi(\Psi_{j-1}^{(i)}(m), \mathbf{U}_{i,j})$;
    end for
    if process i has just collapsed **then**
      $N \leftarrow N - 1$; $S \leftarrow S + c(\Psi_j^{(i)}(0))$;
    end if
  end for
  Sort (and renumber) the N processes that did not yet collapse by order of their values of $v(\Psi_j^{(i)}(0), \ldots, \Psi_j^{(i)}(M-1))$;
end for
Return $\bar{Y}_n \leftarrow S/n$.

problem with this approach, however, is that even if we use the same randomizations of the point sets from step $-T_1 + 1$, the randomized points will generally not be assigned to the same CFTP processes in those steps, because these processes do not visit the same sequence of states and are therefore sorted in a different order at each step. Thus, a given chain is not necessarily using the same sequence of vectors $\mathbf{U}_{i,j}$, so the CFTP implementation is no longer valid.

We can handle this difficulty in the following way. The first pass, from $-T_1$ to 0, is done as just described, with a value of T_1 large enough so that only a small fraction of the processes (e.g., something between 10^{-3} and 10^{-5}) are expected not to have coalesced by time 0. On the second and further passes, for the CFTP processes that did not coalesce, we restart further back in the past, but using Monte Carlo instead of array-RQMC before time T_1, and reusing the same random numbers for the same processes thereafter, without sorting the processes. There will be practically no variance reduction for those processes, but they are only a small fraction of all the processes, so a good overall variance reduction can still be achieved. This procedure is described in Algorithm 2

A second option would be to apply array-RQMC also on the second and further passes, but when we start in the past from time $-T_k$, we apply array-RQMC only until we reach time $-T_{k-1}$, and then we stop sorting the processes and we make sure that we reuse the

same random numbers for the same processes until time 0. When T_1 is large enough, this alternative involves only a very small fraction of the chains, so it does not make much of a difference in practice. On the other hand, it may permit one to take a smaller T_1. For $T_k = k$, it becomes equivalent to the backward algorithm (but the computational cost is different).

In the implementation, it is important that the points $\mathbf{U}_{i,j}$ are memorized together with the chains $\mathcal{Y}$ for which they are used, to make sure that each point is associated with the same chain, at each step. In an object-oriented implementation, each process $\mathcal{Y}$ would be represented as an object, and the sequence of points used by that process would be memorized in that object, e.g., in a list. A new point is added to the list each time that process moves ahead by one step. These lists are in fact used only for the processes that did not coalesce by time 0, but since we do not know in advance which are those processes, the lists must be filled for all the processes. This may require a significant amount of memory in some cases.

It is important to underline that with this approach, all the processes must be be simulated for at least T_1 time steps. If we choose T_1 so that most processes have collapsed by time 0, then T_1 must be much larger than the average value of T_* in the backward algorithm, which means that this forward approach requires much more simulation work (counted in terms of the total number of steps of the Markov chain that are simulated) than the backward method. This must be taken into account when comparing efficiencies. This also means that the choice of T_1 is a question of compromise: When T_1 is very large, increasing it further increases the cost linearly and has almost no impact on the variance, whereas when T_1 is small, increasing it can improve the efficiency by reducing the variance, due to a better exploitation of the array-RQMC effect by reducing the number of processes that have to be started back from $-T_2$. The efficiency usually increases as a function of T_1 up to some optimal value, then it decreases roughly as $\mathcal{O}(1/T_1)$. Our numerical examples will illustrate this.

6 Numerical Experiments

We compare the CFTP algorithms with MC, classical RQMC, and array-RQMC on small examples. We first specify the selected RQMC point sets and the experimental setting, which are the same as in L'Ecuyer et al. (2007) (the interested readers can consult this reference for further details), then we describe the examples and give the results. In all the examples, we have $d = 1$, so the modified RQMC point sets $P'_{n,j}$ are always two-dimensional: the first coordinate is used to sort the chains $\mathcal{Y}$ and the second one to determine the next transition. We will denote the one-dimensional vectors $\mathbf{U}_j$ and $\mathbf{U}_{i,j}$ by U_j and $U_{i,j}$.

Algorithm 2 Array-RQMC with the Forward CFTP Algorithm

$T \leftarrow T_1$;
for $(i = 0;\ i < n;\ i{+}{+})$ **do**
  for $(m = 0;\ m < M;\ m{+}{+})$ **do**
    $X_{-T}^{(i)}(m) \leftarrow m$;
  end for
end for
for $(j = -T + 1;\ j <= 0;\ j{+}{+})$ **do**
  Randomize P_n afresh into $P_{n,j} = \{\mathbf{U}_{0,j}, \ldots, \mathbf{U}_{n-1,j}\}$;
  for $(i = 0;\ i < n;\ i{+}{+})$ **do**
    for $(m = 0;\ m < M;\ m{+}{+})$ **do**
      $X_j^{(i)}(m) \leftarrow \varphi(X_{j-1}^{(i)}(m), \mathbf{U}_{i,j})$;
    end for
    Memorize $\mathbf{U}_{i,j}$ in the "history" of process i;
  end for
  Sort (and renumber) the n processes by order of values of $v(X_j^{(i)}(0), \ldots, X_j^{(i)}(M_i - 1))$;
end for
Let N be the number of processes that have not yet collapsed;
while $N > 0$ **do**
  $T = 2 * T$;
  Simulate the N processes that remain, from time $-T$ to time 0, using MC until time $-T/2$ and reusing thereafter the same random numbers as before, for each process;
  Let N be the number of processes that have not yet collapsed at time 0;
end while
Return $\bar{Y}_n \leftarrow [c(X^{(0)}(0)) + \cdots + c(X^{(n-1)}(0))]/n$.

6.1 Point sets and experimental setting

For the RQMC methods, we use Korobov lattice rules and Sobol' nets. A Korobov rule uses the infinite-dimensional point set

$$P_n = \{\mathbf{v}_i = (i/n,\ (ia \bmod n)/n, (ia^2 \bmod n)/n, \dots),\ i = 0, \dots, n-1\}$$

defined by two parameters $0 < a < n$ (Niederreiter, 1992; Sloan and Joe, 1994; L'Ecuyer and Lemieux, 2000). We take n equal to the largest prime number smaller than 2^k for $k = 10, 12, \dots, 20$. For array-RQMC, where $P'_{n,j}$ is two-dimensional, we take a equal to the odd integer nearest to $n/1.61803399$, so a/n is close to the golden ratio (this always gives a good two-dimensional lattice). For classical RQMC, the point set must be infinite-dimensional and we use several of its coordinates; for that case, a is taken from Table 1 of L'Ecuyer and Lemieux (2000). We randomize this point set by a (single) random shift modulo 1 applied simultaneously to all the points, followed by a baker's transformation, which transforms each coordinate u to $2u$ if $u < 1/2$ and to $2(1-u)$ if $u \geq 1/2$. The random shift consists in generating a single random point $\tilde{\mathbf{V}}$ uniformly in $[0,1)^s$ and then replacing each point $\mathbf{v}_i$ of P_n by $\mathbf{V}_i = (\mathbf{v}_i + \tilde{\mathbf{V}}) \bmod 1$.

For array-RQMC, we also use Sobol' nets with $n = 2^k$ points, for $k = 10, 12, \dots, 20$, randomized by a left (upper triangular) matrix scrambling followed by a random digital shift (L'Ecuyer and Lemieux, 2002; Owen, 2003). The matrix scrambling left-multiplies the generator matrix for each coordinate of the net by a random nonsingular lower-triangular matrix, and the digital shift is similar to the random shift, except that addition is applied bitwise modulo 2. Since we do not have an infinite-dimensional implementation of these point sets, we use them only for array-RQMC.

All these point sets and randomizations are available in the SSJ software (L'Ecuyer, 2004), which we have used for all our experiments. For array-RQMC, the randomization is only applied to the second coordinate and the points are enumerated by order of the first coordinate for the lattices and by order of their Gray code for the Sobol' nets, as explained in L'Ecuyer et al. (2007).

For each RQMC method and each value of n considered, we estimate the *variance reduction factor* (VRF) compared with standard MC, defined as $\mathrm{Var}[Y]/(n\ \mathrm{Var}[\bar{Y}_n])$, where $\bar{Y}_n$ is the estimator with the RQMC method considered. These variances were estimated from 2^{20} independent replicates of the CFTP process for the MC method and 100 independent copies of $\bar{Y}_n$ for the RQMC methods. This gives a more accurate variance estimate for MC than for RQMC; for the latter we have roughly about 10 to 20% relative accuracy (as a crude estimate).

We also compare the work-normalized variance of estimators, defined as the variance

multiplied by the computational work, where the work is measured as the total number of Markov chains steps that are simulated. The *efficiency improvement* of any given estimator compared with the MC estimator is defined as the work-normalized variance of the MC estimator divided by that of the given estimator.

6.2 Some Markov chains with small state spaces

Our first set of examples is taken directly from Lemieux and Sidorsky (2006). It is comprised of three finite-state Markov chains, with state spaces of size 3, 4, and 16, respectively, combined with the following three choices of the cost function c:

$$c_1(x) = x, \qquad c_2(x) = (x-2)(x-5), \qquad \text{and} \qquad c_3(x) = \sin(3x).$$

For each chain, we want to estimate $\mu_k = \mathbb{E}_\pi[c_k(X)]$ for $k = 1, 2, 3$. The probability transition matrices of the first two chains are

$$\mathbf{P}_1 = \begin{pmatrix} 0.5 & 0.4 & 0.1 \\ 0.3 & 0.4 & 0.3 \\ 0.2 & 0.3 & 0.5 \end{pmatrix},$$

$$\mathbf{P}_2 = \begin{pmatrix} 0.7 & 0.0 & 0.3 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.4 & 0.0 & 0.6 \\ 0.0 & 0.2 & 0.0 & 0.8 \end{pmatrix},$$

while the third chain is a random walk whose transition matrix $\mathbf{P}_3$ has elements

$$P_{i,j} = \begin{cases} 0.2 & \text{If } i+1 = j \le 15 \text{ or } i = j = 15 \\ 0.8 & \text{If } i-1 = j \le 15 \text{ or } i = j = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Each transition is generated by inversion from the relevant uniform; i.e., $\varphi(i, U) = \min\{j | P_{i,0} + \cdots + P_{i,j} \ge U)$. For the sorting function v, we take the average of the states of the M chains in the CFTP process, and sort the processes by increasing order of that average. That is, for the process i at step j, we have $v(Y_j^{(i)}) = \frac{1}{M}\sum_{m=0}^{M-1} X_{m,j}^{(i)}$. We tried other sorting functions, such as the average value of the cost function, and the result were not better.

Table 1 reports the estimated VRFs compared with MC, per copy of the CFTP process. Since the computing times in the backward algorithm are very similar for all the MC and RQMC methods, the efficiency improvement is essentially the same as the VRF. In all our tables, Classical-Korobov means classical RQMC with a Korobov point set, a random shift, and a baker's transformation; Array-Korobov means array-RQMC with a Korobov point set,

Table 1: VRFs for the first set of examples, with the backward algorithm

	n	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
	n for Korobov	1021	4093	16381	65521	262139	1048573
	a for Classical-Korobov	306	1397	5693	944	118068	802275
	a for Array-Korobov	633	2531	10125	40503	162013	648055
Matrix $\mathbf{P}_1$							
c_1	Classical-Korobov	23	50	34	68	106	339
	Array-Korobov	240	826	1951	4886	2638	21560
	Array-Sobol	155	378	1281	4882	16150	51470
c_2	Classical-Korobov	33	51	25	43	102	342
	Array-Korobov	199	720	1820	5467	2962	25260
	Array-Sobol	123	540	1483	3982	16910	41410
c_3	Classical-Korobov	17	30	36	26	25	40
	Array-Korobov	97	317	1067	2380	1464	5763
	Array-Sobol	63	156	503	1015	6496	19150
Matrix $\mathbf{P}_2$							
c_1	Classical-Korobov	5	12	3	17	44	24
	Array-Korobov	32	59	186	607	774	3555
	Array-Sobol	22	46	104	285	812	2031
c_2	Classical-Korobov	5	10	2	12	31	20
	Array-Korobov	33	62	211	491	585	3165
	Array-Sobol	23	48	100	282	742	2228
c_3	Classical-Korobov	5	8	3	14	33	18
	Array-Korobov	31	78	230	596	673	3690
	Array-Sobol	15	68	101	273	673	1912
Matrix $\mathbf{P}_3$							
c_1	Classical-Korobov	7	13	26	47	54	72
	Array-Korobov	39	84	155	345	598	1484
	Array-Sobol	26	49	102	220	559	1108
c_2	Classical-Korobov	12	16	16	38	37	68
	Array-Korobov	51	114	265	587	858	2140
	Array-Sobol	34	64	172	369	917	1718
c_3	Classical-Korobov	1	2	3	6	6	7
	Array-Korobov	4	8	15	22	53	101
	Array-Sobol	3	4	9	14	34	69

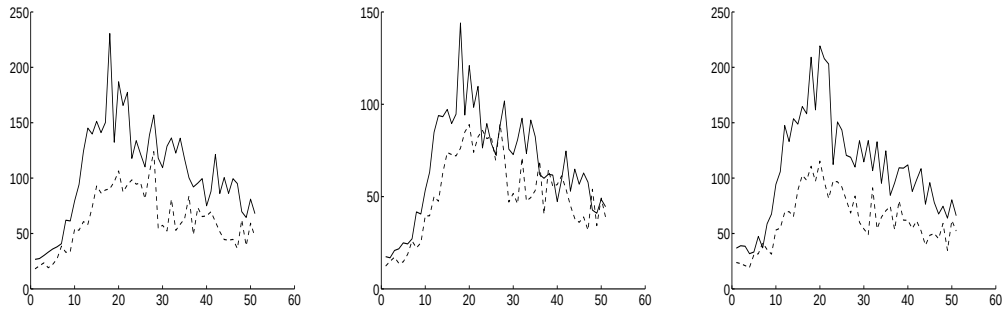

Figure 1: Efficiency improvement of array-RQMC over MC, as a function of T_1, for $\mathbf{P}_3$. The graphs are for c_1, c_2, and c_3 (right to left), for the Korobov lattices (solid lines) and the Sobol' nets (dotted lines).

a random shift, and a baker's transformation; and Array-Sobol means array-RQMC with a Sobol' point set, a left matrix scramble and a random digital shift.

For Classical-Korobov, we observe larger VRFs than Lemieux and Sidorsky (2006) for the same examples. This can be explained by the fact that we have tried larger values of n (the VRF tends to increase with n) and that applying the baker's transform brings some additional improvement. However, the array-RQMC method provides much larger improvements, for both point sets and for all examples. The improvement factor exceeds several thousands in many cases. The worst case is the random walk (matrix $\mathbf{P}_3$) with function c_3, for which we still get an improvement factor of about 100 with $n = 2^{20}$.

Table 2 gives the efficiency improvement factors obtained with the forward algorithm, for the same examples. For the small matrix $\mathbf{P}_1$, we are not doing better than with the backward algorithm, but for the other matrices, we do. With array-RQMC, it is for $\mathbf{P}_3$, for which the state space is the largest, that the forward method provides the best improvement over the backward method. This last observation tends to be true in general; with array-RQMC, the forward method is usually more advantageous than the backward method when the state space is large, which is the case of most practical interest.

The values of T_1 used in Table 2 were selected based on $n = 2^{20}$ preliminary runs to estimate the optimal values (that maximize the efficiency) for the array-RQMC method. With the selected values, the proportion of processes that do not reach coalescence by time 0 is approximately 10^{-4} in the three cases.

As an illustration, in Figure 1 we plot the efficiency improvement factor of array-RQMC compared with MC, as a function of T_1, where T_1 varies from 0 to 50, for the matrix $\mathbf{P}_2$ with $n = 2^{10}$. The maximal efficiency is reached for T_1 near 20.

Table 2: Efficiency improvement factors for the forward algorithm with fixed T_1

	n	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
	n for Korobov	1021	4093	16381	65521	262139	1048573
	a for Classical-Korobov	306	1397	5693	944	118068	802275
	a for Array-Korobov	633	2531	10125	40503	162013	648055
Matrix $\mathbf{P}_1$, with $T_1 = 9$							
c_1	Classical-Korobov	23	50	34	68	106	339
	Array-Korobov	71	149	737	124	6506	8347
	Array-Sobol	42	139	522	1660	4176	13689
c_2	Classical-Korobov	33	51	25	43	102	342
	Array-Korobov	84	104	416	114	5844	10044
	Array-Sobol	50	156	598	1496	6236	15771
c_3	Classical-Korobov	17	30	36	26	25	40
	Array-Korobov	40	23	240	303	1728	2090
	Array-Sobol	22	72	178	879	3307	9358
Matrix $\mathbf{P}_2$, with $T_1 = 20$							
c_1	Classical-Korobov	5	12	3	17	44	24
	Array-Korobov	31	62	248	71	1321	1431
	Array-Sobol	16	59	185	618	1925	5005
c_2	Classical-Korobov	5	10	2	12	31	20
	Array-Korobov	43	55	205	73	102	1642
	Array-Sobol	26	65	249	558	1702	3986
c_3	Classical-Korobov	5	8	3	14	33	18
	Array-Korobov	37	44	177	130	1070	3103
	Array-Sobol	14	52	162	512	1215	4368
Matrix $\mathbf{P}_3$, with $T_1 = 62$							
c_1	Classical-Korobov	7	13	26	47	54	72
	Array-Korobov	33	58	239	1418	921	2107
	Array-Sobol	65	179	772	2548	5226	14330
c_2	Classical-Korobov	12	16	16	38	37	68
	Array-Korobov	46	81	219	991	941	1240
	Array-Sobol	41	1287	428	1504	3285	9027
c_3	Classical-Korobov	1	2	3	6	6	7
	Array-Korobov	7	13	42	111	123	297
	Array-Sobol	5	15	59	144	417	1217

Table 3: Efficiency improvement factors for the forward CFTP method with fixed T_1, for the random walk over $[0, 5]$

Parameters		$n = 2^{10}$		$n = 2^{12}$		$n = 2^{14}$	
		T_1		T_1		T_1	
σ=5	Classical-Korobov		17		42		14
	Array-Korobov	10	227	10	1428	12	2485
	Array-Sobol	9	2554	11	12911	12	36449
σ=1	Classical-Korobov		11		13		18
	Array-Korobov	81	11	80	727	98	1411
	Array-Sobol	86	588	86	2406	92	6846
σ=0.5	Classical-Korobov		5		2		9
	Array-Korobov	303	11	319	637	298	1753
	Array-Sobol	302	2969	326	1255	326	3195

6.3 A random walk over a finite segment of the real line

We now consider a random walk on a continuous state space, namely $\mathcal{S} = [0, \overline{w}]$ where $\overline{w} > 0$. The Markov chain evolves according to the stochastic recurrence

$$X_0 = x_0, \qquad X_j = \max(0, \min(X_{j-1} + Z_j, \overline{w})), \qquad j \geq 1$$

where the Z_j are independent and normally distributed with mean 0 and variance σ^2. We can write $Z_j = \sigma\Phi^{-1}(U_j)$, where Φ is the standard normal distribution function; then the recurrence (2) becomes

$$X_j = \varphi(X_{j-1}, U_j) = \max(0, \min(X_{j-1} + \sigma\Phi^{-1}(U_j), \overline{w})).$$

We want to estimate $\mu = \mathbb{E}_\pi[c(X)]$ for $c(x) = x$. Here, φ is monotone in U_j, and we have $\mathcal{S}_0 = \{0\}$ and $\mathcal{S}_1 = \{\overline{w}\}$, so the forward CFTP algorithm requires the simulation of only two copies of the chain $\mathcal{X}$, one from state 0 and the other from state $\overline{w}$. The sorting function v is taken as the average between the states of these two copies of $\mathcal{X}$.

Table 3 gives the estimated efficiency improvement factors of the RQMC methods compared with MC, for the forward algorithm with fixed T_1. We take $\overline{w} = 5$ and three different values of σ. For array-RQMC, the optimal value of T_1 was estimated for each case by pilot runs. Figure 2 illustrates one case; it shows the (empirical) efficiency improvement factor as a function of T_1 for the random walk over $[0, 5]$ with $\sigma = 1$, for array-RQMC with $n = 2^{12}$. The maximum is reached for T_1 around 90. The other cases are similar.

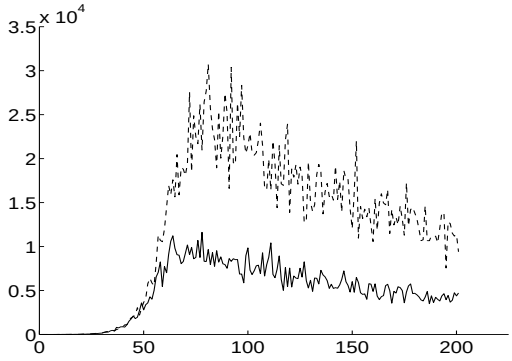

Figure 2: Efficiency improvement factor as a function of T_1 for the random walk with $\overline{w} = 5$ and $\sigma = 1$. Again, the solid line is for Array-Korobov and the dotted line is for Array-Sobol.

7 Conclusion

We have shown how the array-RQMC method can be combined with both the forward and backward CFTP algorithms. In our numerical examples, these combinations have outperformed the combination of CFTP with a classical RQMC approach. The latter was already shown in Lemieux and Sidorsky (2006) to be superior to the more traditional antithetic and Latin hypercube sampling approaches. Admittedly, our numerical illustrations are with small academic examples. The aim was to show the potential of the method on simple cases for which we know the exact answer. The next step will be to experiment it on larger, more complex, Markov chains. For that, the forward method with fixed T_1 appears to be the most promising.

Acknowledgments

This research has been supported by Grant OGP-0110050 and a Canada Research Chair to the first author. This article was written while the first author was a visiting researcher at IRISA, in Rennes, France.

References

Awad, H. P., and P. W. Glynn. 2007. On the theoretical comparison of low-bias steady-state simulation estimators. *ACM Transactions on Modeling and Computer Simulation* 17 (1): 4.

Craiu, R. V., and X.-L. Meng. 2000. Antithetic coupling for perfect sampling. In *Bayesian*

Methods with Applications to Science, Policy, and Official Statistics (Selected Papers from ISBA 2000), ed. E. I. George, 99–108.

Craiu, R. V., and X.-L. Meng. 2005. Multiprocess parallel antithetic coupling for backward and forward Markov chain Monte Carlo. *Annals of Statistics* 33 (2): 661–697.

Glynn, P. W. 2006. Simulation algorithms for regenerative processes. In *Simulation*, ed. S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science, 477–500. Amsterdam, The Netherlands: Elsevier. Chapter 16.

Heidelberger, P., and P. D. Welch. 1983. Simulation run length control in the presence of an initial transient. *Operations Research* 31:1109–1144.

Law, A. M., and W. D. Kelton. 2000. *Simulation modeling and analysis*. Third ed. New York: McGraw-Hill.

L'Ecuyer, P. 2004. *SSJ: A Java library for stochastic simulation.* Software user's guide, Available at `http://www.iro.umontreal.ca/~lecuyer`.

L'Ecuyer, P. 2006. Uniform random number generation. In *Simulation*, ed. S. G. Henderson and B. L. Nelson, Handbooks in Operations Research and Management Science, 55–81. Amsterdam, The Netherlands: Elsevier. Chapter 3.

L'Ecuyer, P., C. Lécot, and B. Tuffin. 2007. A randomized quasi-Monte Carlo simulation method for Markov chains. *Operations Research*. To appear.

L'Ecuyer, P., and C. Lemieux. 2000. Variance reduction via lattice rules. *Management Science* 46 (9): 1214–1235.

L'Ecuyer, P., and C. Lemieux. 2002. Recent advances in randomized quasi-Monte Carlo methods. In *Modeling Uncertainty: An Examination of Stochastic Theory, Methods, and Applications*, ed. M. Dror, P. L'Ecuyer, and F. Szidarovszky, 419–474. Boston: Kluwer Academic.

Lemieux, C., and P. Sidorsky. 2006. Exact sampling with highly-uniform point sets. *Mathematical and Computer Modelling* 43:339–349.

Niederreiter, H. 1992. *Random number generation and quasi-Monte Carlo methods*, Volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics*. Philadelphia: SIAM.

Owen, A. B. 1998. Latin supercube sampling for very high-dimensional simulations. *ACM Transactions on Modeling and Computer Simulation* 8 (1): 71–102.

Owen, A. B. 2003. Variance with alternative scramblings of digital nets. *ACM Transactions on Modeling and Computer Simulation* 13 (4): 363–378.

Propp, J. G., and D. B. Wilson. 1996. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms* 9 (1&2): 223–252.

Sloan, I. H., and S. Joe. 1994. *Lattice methods for multiple integration.* Oxford: Clarendon Press.