Sorting methods and convergence rates for Array-RQMC: some empirical comparisons

Pierre L'Ecuyer[a], David Munger[b], Christian Lécot[c], Bruno Tuffin[d]

[a] *DIRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada; and Inria Rennes – Bretagne Atlantique,* lecuyer@iro.umontreal.ca
[b] *DIRO, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, Canada,* mungerd@iro.umontreal.ca
[c] *Université Savoie Mont Blanc, LAMA, UMR 5127 CNRS, 73376 Le Bourget-du-Lac Cedex, France,* Christian.Lecot@univ-savoie.fr
[d] *Inria Rennes – Bretagne Atlantique, Campus Universitaire de Beaulieu, 35042 Rennes Cedex, France,* bruno.tuffin@inria.fr

Abstract

We review the Array-RQMC method, its variants, sorting strategies, and convergence results. We are interested in the convergence rate of measures of discrepancy of the states at a given step of the chain, as a function of the sample size n, and also the convergence rate of the variance of the sample average of a (cost) function of the state at a given step, viewed as an estimator of the expected cost. We summarize known convergence rate results and show empirical results that suggest much better convergence rates than those that are proved. We also compare different types of multivariate sorts to match the chains with the RQMC points, including a sort based on a Hilbert curve.

Keywords: Low discrepancy, quasi-Monte Carlo, Markov chain, variance reduction, simulation, Array-RQMC

1. Introduction

Array-RQMC is a method to simulate an array of n dependent realizations of a Markov chain in a way that each chain is generated from its exact probability law, and with the aim that the empirical distribution of the states at a given step of the chain provides a "low-discrepancy" approximation of the theoretical distribution of the state at that step. At each step, the n copies of the chain

are sorted in a particular order and then moved forward by one step using a randomized quasi-Monte Carlo (RQMC) point set of cardinality n. If the state space has more than one dimension, the sort can be multidimensional and the performance may depend strongly on the sorting method. More details on the method, intuitive justifications, convergence results, applications, and empirical evaluations, can be found in [4, 5, 7, 8, 13, 15, 16, 17, 18, 20, 25].

The aim of this paper is to review briefly what is known and what has been done so far with this method, report new experimental results on convergence rates, and compare different types of multivariate sorting methods, including sorts based on space-filling curves. Those types of curves are widely used to map points from the multivariate unit cube $[0,1]^\ell$ to the unit interval $[0,1]$ in various areas of applications such as sorting multivariate objects, mapping them to memory addresses in database systems and multiprocessing computers, storing and reconstructing images in computer graphics, etc., see [1]. They also provide one of the most effective heuristics to quickly obtain a good solution for a traveling salesman problem with a large number of points in two or more dimensions [2, 3, 23]. Thus, their use to reduce the dimension and order the states in Array-RQMC seems natural, and was suggested in [25]. A Hilbert sort to map two-dimensional points to $[0,1)$ is also proposed in [24] for QMC sampling in computer graphics. Recently, Gerber and Chopin [9] proposed to use the Hilbert space-filling curve to sort multidimensional states in a variant of Array-RQMC combined with particle filtering, which they named *sequential quasi-Monte Carlo*, and proved that under some conditions, the variance of the filtering estimator converges as $o(n^{-1/2})$, i.e., faster than for Monte Carlo. He and Owen [11] study the Hilbert curve as a way of reducing the dimension from $d > 1$ to 1 in the context of estimating a d-dimensional integral by QMC or RQMC (not for Markov chains). They prove convergence rate bounds on the MSE under different sets of conditions on the integrand and the points. We discuss this type of sort and compare it empirically to other multivariate sorts proposed previously. We also survey currently known convergence rate results for Array-RQMC, and show examples in which the observed convergence rates

are much better than those that are proved.

The remainder is organized as follows. We review the Array-RQMC algorithm in Section 2, multivariate sorts in Section 3, and theoretical convergence results in Section 4. In Section 5, we compare the convergence rates observed empirically in some examples with those that are proved, for the mean square L_2-discrepancy and the variance of cost estimators.

Note: It is customary to define space-filling curves over the closed hypercube $[0,1]^s$, QMC points over the semi-open hypercube $[0,1)^s$ in s dimensions, and uniform random variables over the open interval $(0,1)$, because the inverse cdf may be infinite at 0 or 1. We follow these conventions as much as we can in the paper, but there are inevitable occasional clashes (when two different conventions would apply to the same box). This may appear inconsistent, but in our practical implementations, no coordinate will ever equal 1, so it does not matter if the interval is open or closed.

2. Array-RQMC

We consider a discrete-time Markov chain whose state evolves in a measurable space $\mathcal{X}$ according to a stochastic recurrence defined by

$$X_0 = x_0 \in \mathcal{X} \quad \text{and} \quad X_j = \varphi_j(X_{j-1}, \mathbf{U}_j) \text{ for } j \geq 1,$$

in which $\mathbf{U}_1, \mathbf{U}_2, \ldots$ are independent and identically distributed (i.i.d.) uniform random variables over the unit hypercube $(0,1)^d$, and each $\varphi_j : \mathcal{X} \times (0,1)^d \to \mathcal{X}$ is a measurable mapping. Suppose we want to estimate

$$\mu = \mathbb{E}[Y], \quad \text{where} \quad Y = \sum_{j=1}^{\tau} g_j(X_j)$$

for some measurable *cost (or reward) functions* $g_j : \mathcal{X} \to \mathbb{R}$, where τ is a fixed time horizon (a positive integer). The methods we describe also work if τ is a random stopping time, as explained in [18].

The *standard Monte Carlo* (MC) method estimates μ by $\bar{Y}_n = \frac{1}{n}\sum_{i=0}^{n-1} Y_i$ where $Y_0, \ldots, Y_{n-1}$ are n independent realizations of Y. One has $\mathbb{E}[\bar{Y}_n] = \mu$ and $\mathrm{Var}[\bar{Y}_n] = \mathrm{Var}[Y]/n = \mathcal{O}(n^{-1})$ if $\mathbb{E}[Y^2] < \infty$.

A naive way of using RQMC in this setting would be to replace the n independent $d\tau$-dimensional uniformly distributed vectors $\mathbf{V}_i = (\mathbf{U}_{i,1}, \ldots, \mathbf{U}_{i,\tau}) \in (0,1)^{d\tau}$ by a $d\tau$-dimensional RQMC point set, in $(0,1)^{d\tau}$. However, when $d\tau$ increases, this RQMC scheme typically becomes ineffective, because $\mathbb{E}[Y]$ is a high-dimensional integral.

The Array-RQMC method was designed to address this issue. To provide an intuitive explanation of how it works, let us assume that we have a one-to-one mapping $h : \mathcal{X} \to [0,1)^c$, for some small integer $c \geq 1$ (to implement the method, it is not essential to define such a mapping h explicitly, and h does not have to be one-to-one). If h is one-to-one, $\tilde{X}_j = h(X_j)$ contains all the information in X_j that is relevant for the probability law of the future evolution of the chain. At each step j, the n realizations of the chains are "sorted" in some order, based on the realizations of the transformed state $\tilde{X}_j$, and then matched to n RQMC points based on that order. In [18], it was assumed that $c = 1$, so h maps the states to the interval $[0,1)$, and the states were sorted by increasing order of $\tilde{X}_j$. The function h was then called a *sorting function*. Implementations with $c > 1$ were also examined in [7, 8] in a QMC setting and in [16], in which multidimensional sorts match the chains to RQMC points.

Under our assumptions, we can write $X_j = \varphi_j(X_{j-1}, \mathbf{U}_j) = \tilde{\varphi}_j(\tilde{X}_{j-1}, \mathbf{U}_j)$ for some function $\tilde{\varphi}_j$. If $\tilde{X}_{j-1}$ has a density f_j over $[0,1)^c$, then

$$\mu_j = \mathbb{E}[g_j(X_j)] = \mathbb{E}[g_j(\tilde{\varphi}_j(\tilde{X}_{j-1}, \mathbf{U}_j))] = \int_{[0,1)^{c+d}} f_j(\mathbf{w}) g_j(\tilde{\varphi}_j(\mathbf{w}, \mathbf{u})) d\mathbf{u} d\mathbf{w}.$$

To simplify the argument, we assume that f_j is uniform over $[0,1)^c$. This can generally be achieved by incorporating an appropriate change of variable in h. Since the choice of h will only affect the pairing between the states and the RQMC points at each step j, this change of variable does not have to be computed explicitly if it does not change this pairing. In particular, if $c = 1$ and if $\tilde{X}_{j-1}$ has a continuous cdf F_j, then $F_j(\tilde{X}_{j-1}) \sim U(0,1)$, so we can replace h by $F_j \circ h$ to obtain the required uniformity. But in a situation where the method sorts the states by increasing order of value of $\tilde{X}_{j-1}$, this replacement would never change the ordering, so there is no need to implement it explicitly.

Let $X_{0,j}, \ldots, X_{n-1,j}$ be the n realizations of X_j and $\tilde{X}_{i,j} = h(X_{i,j})$. Given that $\tilde{X}_{j-1}$ is assumed uniform over $[0,1)^c$, the idea is to construct a (semi)-randomized point set $Q_n = \{(\tilde{X}_{i,j-1}, \mathbf{U}_{i,j}),\ 0 \le i < n\}$ such that each $\mathbf{U}_{i,j}$ has the uniform distribution over $[0,1)^d$ and Q_n as a whole has low discrepancy with respect to the uniform distribution over $[0,1)^{c+d}$. Note that the $\tilde{X}_{i,j-1}$ cannot be chosen, since they come from the previous evolution of the chain. We will assume that they form a low-discrepancy point set with respect to the uniform distribution over $[0,1)^c$. This property will have to be preserved at the next step, for the points $\tilde{X}_{i,j}$, so we can use induction to argue that it holds at all steps. We construct a point set

$$\tilde{Q}_n = \{(\mathbf{w}_0, \mathbf{U}_{0,j}), \ldots, (\mathbf{w}_{n-1}, \mathbf{U}_{n-1,j})\}$$

in which the $\mathbf{w}_i \in [0,1)^c$ are fixed, $\mathbf{U}_{i,j} \sim U(0,1)^d$ for each i, and $\tilde{Q}_n$ has low discrepancy with probability 1. Then we find a permutation π_j of the n states $X_{i,j-1}$ for which $\tilde{X}_{\pi_j(i),j-1}$ is "close" to $\mathbf{w}_i$ for each i, so $\tilde{Q}_n$ is close to Q_n after the points are permuted, and we compute $X_{i,j} = \varphi_j(X_{\pi_j(i),j-1}, \mathbf{U}_{i,j})$ for each i. The integral $\mu_j = \mathbb{E}[g_j(X_j)]$ is then estimated by

$$\begin{aligned}\hat{\mu}_{\mathrm{arqmc},j,n} = \bar{Y}_{n,j} &= \frac{1}{n}\sum_{i=0}^{n-1} g_j(X_{i,j}) \\ &= \frac{1}{n}\sum_{i=0}^{n-1} g_j(\tilde{\varphi}_j(\tilde{X}_{\pi_j(i),j-1}, \mathbf{U}_{i,j})) \approx \frac{1}{n}\sum_{i=0}^{n-1}(g_j \circ \tilde{\varphi}_j)(\mathbf{w}_i, \mathbf{U}_{i,j})),\end{aligned}$$

which can be seen as a semi-RQMC estimator. The first c coordinates of the points are (in general) not randomized; they are only used to match the points to the chains. As a special case, if $c = 1$, one can take $\mathbf{w}_i = (i + 0.5)/n$ or i/n and the best match is obtained by just sorting the states by increasing order of $\tilde{X}_{i,j-1}$. In this case, this $\mathbf{w}_i$ can be only implicit (not stored explicitly) because the points do not need to be sorted at each step.

The estimator $\hat{\mu}_{\mathrm{arqmc},j,n}$ is essentially a low-discrepancy estimator of μ_j. An intuitive explanation for why the discrepancy of the points $h(X_{i,j})$ is expected to be small is that this discrepancy can be written (at least for some discrepancy measures) as an average of the form $\frac{1}{n}\sum_{i=0}^{n-1} g(X_{i,j})$, similar to $\hat{\mu}_{\mathrm{arqmc},j,n}$.

Algorithm 1: : Array-RQMC procedure (general outline)

$X_{i,0} \leftarrow x_0$ for $i = 0, \ldots, n-1$;

for $j = 1, 2, \ldots, \tau$ **do**

Compute an appropriate permutation π_j of the n states $X_{0,j}, \ldots, X_{n-1,j}$, based on the $h(X_{i,j-1})$, to match them with the RQMC points;

Randomize afresh $\{\mathbf{U}_{0,j}, \ldots, \mathbf{U}_{n-1,j}\}$ in $\tilde{Q}_n$;

$X_{i,j} = \varphi_j(X_{\pi_j(i),j-1}, \mathbf{U}_{i,j})$, for $i = 0, \ldots, n-1$;

$\hat{\mu}_{\text{arqmc},j,n} = \bar{Y}_{n,j} = \frac{1}{n}\sum_{i=0}^{n-1} g(X_{i,j})$;

end for

Estimate μ by the average $\bar{Y}_n = \hat{\mu}_{\text{arqmc},n} = \sum_{j=1}^{\tau} \hat{\mu}_{\text{arqmc},j,n}$.

With these ingredients, Algorithm 1 gives a general (high-level) description of the Array-RQMC method. There are of course many possibilities for the choices of QMC point sets, randomization, and permutation strategy. Regardless of the choice of h and of the distribution of the $\tilde{X}_j$, we have that (see [18], Propositions 1 and 2): (i) the average $\bar{Y}_n = \hat{\mu}_{\text{arqmc},n}$ is an *unbiased* estimator of μ; and (ii) the *empirical variance* of m independent realizations of $\hat{\mu}_{\text{arqmc},n}$ gives an unbiased estimator of $\text{Var}[\bar{Y}_n]$.

3. How to sort the states

When the state space $\mathcal{X}$ is one-dimensional, e.g, $\mathcal{X} \subseteq \mathbb{R}$, the states are simply sorted in natural order. Otherwise, one can either define a mapping $h : \mathcal{X} \to \mathbb{R}$ (of which $h : \mathcal{X} \to [0, 1)$ is a special case) and then use the natural order, or define $h : \mathcal{X} \to \mathbb{R}^c$ for $c > 1$ and use a multivariate sort as follows.

3.1. Multivariate sorts

Suppose the (potentially transformed) states $\tilde{X}_{i,j}$ are in $[0, 1)^c$ for $c > 1$. A *multivariate batch sort* considered in [7, 12, 16] operates as follows. We select positive integers $n_1, \ldots, n_c$ and assume that $n = n_1 \cdots n_c$. We first sort the states by their first coordinate in n_1 packets of size n/n_1, in a way that the

states in any given packet have first coordinates that are no larger than those in the next packet. Then we sort each packet in n_2 packets of size $n/(n_1 n_2)$ in the same way but by the second coordinate, and so on. At the last level, we sort each packet of size n_c by the last coordinate. To do the matching, we also sort the RQMC points $\tilde{Q}_n$ in exactly the same way, using their first c coordinates, and then match the corresponding states and points. Figure 1 illustrates this procedure for $c = 2$ and $n_1 = n_2 = 4$. The best choice of $n_1, \ldots, n_c$ depends on the application.

If the same sort is used at all steps and the method is applied several times with the same n, and the first c coordinates of $\tilde{Q}_n$ are not randomized, it suffices to sort the points $\tilde{Q}_n$ in the right order once and for all at the beginning, and randomize their last d coordinates at each step.

The *multivariate split sort* is a variant in which we assume that $n = 2^e$ (a power of 2), and we take $n_1 = n_2 = \cdots = n_e = 2$. We first sort (split) the points in 2 packets by the first coordinate, then split each packet in two by the second coordinate, and so on. If $e > c$, after having reached the last coordinate we start again with the first one and continue in a round-robin fashion.

3.2. Mapping the states to the one-dimensional unit interval

In [18], several examples are given in which sorting functions $h : \mathcal{X} \to \mathbb{R}$ are defined in a heuristic manner, in a similar way as the *importance functions* in the splitting methodology for rare-event simulation (this is discussed in [15]). The idea is to try to select h so that $h(x)$ approximates in some way the expected future costs when we are in state x at a given stage. We could also make the function h depend on the step number j, because an "optimal" h (in terms of variance) may generally depend on j.

The choice of sorting function is further discussed in [25], in which the authors suggest using a space-filling curve, a widely-used tool for ordering multivariate objects [1, 3]. Suppose that the objects already correspond to points in $[0,1]^\ell$. A *space-filling curve* for $[0,1]^\ell$ is defined as a surjective continuous mapping $\psi : [0,1] \to [0,1]^\ell$. It is known that ψ cannot be invertible, but one

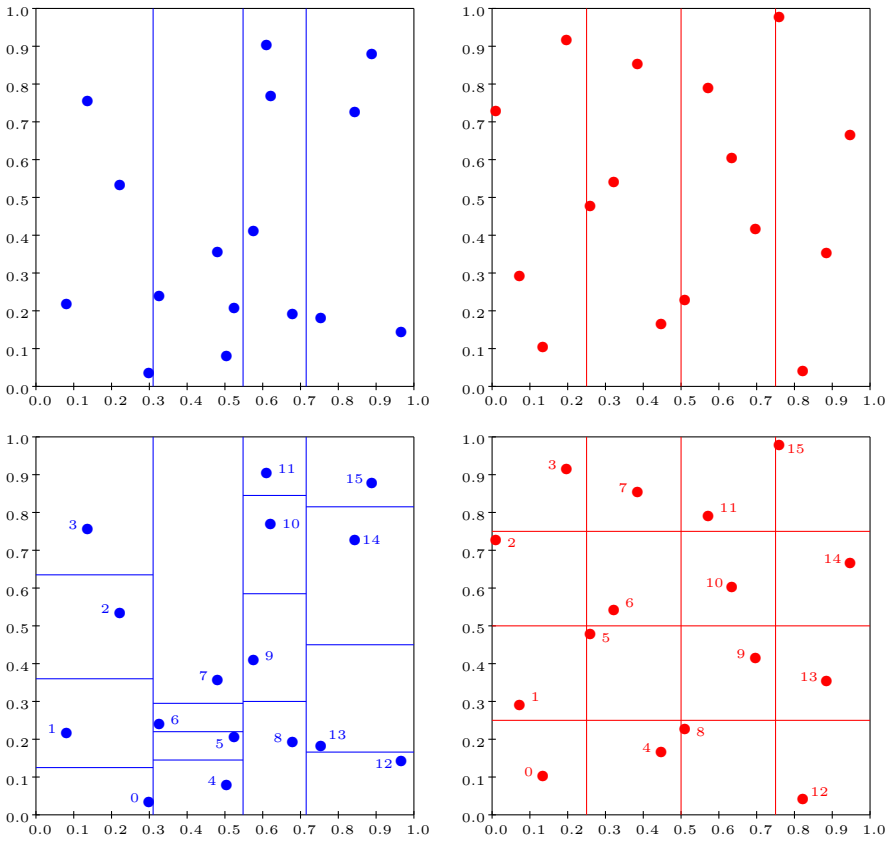

Figure 1: Illustration of the batch sort with 16 points in 2 dimensions, with $n_1 = n_2 = 4$. The states are in the left panels and the points of a Sobol' net after random digital shift are in the right panels. Above, the states and points are split in four batches of size 4 using the first coordinate. Below, each batch is then split according to the second coordinate. After that, the boxes (and the points inside them) in the first column in the left picture are matched to the corresponding ones on the first column in the right picture, then similarly for the second column, etc. Each point on the left is matched with the point having the same number on the right.

can define a pseudoinverse $h = \psi^{-1}$, use it to map any set of n points in $[0,1]^\ell$ to $[0,1]$, and sort these points by natural order of their mapping.

In practice, one would use a *discrete* version of the curve, as follows. In base $b \geq 2$ at iteration level $m \geq 1$, the inverse is a map $h_m : \{0, 1, \ldots, b^m - 1\}^\ell \to \{0, 1, \ldots, b^{m\ell} - 1\}$. Suppose we partition $[0,1)^\ell$ into $b^{m\ell}$ subcubes of equal size, and define the integer coordinates of a point $\mathbf{x} = (x_1, \ldots, x_\ell) \in [0,1)^\ell$ by $(i_1, \ldots, i_\ell)$ where $i_k = \lfloor b^m x_k \rfloor$. These integer coordinates identify the subcube that contains the point $\mathbf{x}$ and h_m enumerates these subcubes from 0 to $b^{m\ell}-1$. If no subcube contains more than one point, this sorts the points uniquely. If some subcube contains more than one point (a collision), we may divide it again in b^ℓ smaller subcubes by splitting each of its edges in b equal parts. But it is simpler and usually more effective to just select a fixed m large enough so that collisions are not too frequent (if they happen occasionally, the two points that collide can be ordered arbitrarily and this will not make much difference in numerical computations). For fixed ℓ and m, the mapping h_m can be precomputed once and for all and stored as an index that gives direct access to the subcube number given the integer coordinates of the point.

A Hilbert space-filling curve in $[0,1]^\ell$ uses base $b = 2$. It visits the subcubes in a way that successive subcubes are always adjacent to each other. A Peano curve also has this property, but is defined in base $b = 3$. For a partition in $b^{m\ell}$ subcubes, if the discrete curve is assumed to start and end on the cube boundary and visit each subcube center, then it has at least $b^{m\ell} - 1$ linear segments of length b^{-m} each, plus two of length b^{-m-1}, so its total length is at least $b^{m(\ell-1)}$. A Hilbert curve and a Peano curve reach this minimal length. For $\ell = 1$ (one dimension), this length is 1, otherwise it increases exponentially in both m and ℓ. The true space-filling curve is actually defined as the limiting (continuous) curve obtained when $m \to \infty$, and its length is infinite, but in practice we use a finite m.

For a crude intuition for wanting a shorter curve that visits all the n states $X_{i,j}$, suppose $g : [0,1)^\ell \to \mathbb{R}$ is a continuous cost function whose integral is estimated by the average value at the n evaluation points. For a given ordering of

the points, consider a curve that connect them in this order, by linear segments. The average can also be seen (roughly) as an RQMC estimator of the integral of g along this curve, after rescaling the curve to length 1. Generally speaking, the shorter the curve, the smaller the total variation of g along it is likely to be. For example, if the absolute slope of the gradient of g in any direction never exceeds K, the total variation along the curve does not exceed K times the length of the curve. This suggests that a good strategy to sort the points in $[0,1)^\ell$ would be to find the shortest curve that connects all the points. This is nothing else than a traveling salesman problem (open loop version). Solving this problem exactly is known to be NP-hard (as a function of n), but there are effective heuristics that provide good solutions rapidly. Interestingly, one of the fastest good heuristics when n is very large is to sort the points using a Hilbert or Peano curve [2, 3, 23]. It is proved for $\ell = 2$ (and conjectured for larger ℓ) that the ratio of the length of the path found by this heuristic divided by the length of the shortest path is $\mathcal{O}(\log n)$ in the worst case. For n independent random points in $[0,1)^\ell$, the average length of the shortest path is $\mathcal{O}(\sqrt{n})$.

Using the Hilbert curve for sorting the points was mentioned in [25], but the authors ended up using a Lebesgue Z-curve, which visits successive subcubes that are not necessarily adjacent. Gerber and Chopin [9] proposed sorting the states with a Hilbert curve after mapping them to the unit hypercube via a component-wise rescaled logistic transformation. They proved that the Hilbert map preserves low discrepancy and that the resulting unbiased Array-RQMC estimator has $o(1/n)$ variance, under certain smoothness conditions.

Note that the batch sort and split sort discussed earlier do not require to map the states to the unit hypercube. As an alternative mixed strategy, one may first apply one of these two sorts to the states, so the n states are mapped to n subboxes that partition $\mathbb{R}^\ell$, and then enumerate these subboxes (and states) in the same order as they would be by a Hilbert curve. We call these sorts *Hilbert curve batch sort* and *Hilbert curve split sort*. One advantage of this strategy is it does not require an explicit mapping to the unit hypercube; it suffices that the states can be ordered w.r.t. any coordinate.

4. Some theoretical convergence results

Here we summarize known convergence results for Array-RQMC. Suppose $\mathcal{X} = [0,1]$ and X_j has a continuous distribution with cdf F_j, for each $j > 0$. Let $\hat{F}_j$ be the empirical cdf of the n states at step j. The *star discrepancy* of the states at step j is

$$\Delta_j = \sup_{x \in \mathcal{X}} |\hat{F}_j(x) - F_j(x)|.$$

The corresponding *variation* of a function $g_j : [0,1] \to \mathbb{R}$ is

$$V_\infty(g_j) \quad = \quad \int_0^1 \left| \frac{dg_j(x)}{dx} \right| dx$$

and the standard Koksma-Hlawka inequality gives

$$\left| \bar{Y}_{n,j} - \mathbb{E}[g_j(X_j)] \right| \le \Delta_j V_\infty(g_j). \tag{1}$$

The square L_2 *discrepancy* (or Cramer von Mises statistic) at step j is

$$D_j^2 = \int_0^1 (\hat{F}_j(x) - F_j(x))^2 dx = \frac{1}{12n^2} + \frac{1}{n} \sum_{i=0}^{n-1} ((i + 0.5/n) - F_j(X_{(i),j}))^2,$$

where the $X_{(i),j}$ are the order statistics of $X_{0,j}, \ldots, X_{n-1,j}$ (see, e.g., [6], Chapter 4). Obviously, $D_j^2 \le \Delta_j^2$. With the corresponding square variation

$$V_2^2(g_j) \quad = \quad \int_0^1 (dg_j(x)/dx)^2 dx,$$

one obtains the variance bound (it suffices to square the two sides and take the expectation in the second displayed inequality on page 965 of [22], for example):

$$\mathrm{Var}[\bar{Y}_{n,j}] = \mathbb{E}[(\bar{Y}_{n,j} - \mathbb{E}[g_j(X_j)])^2] \le \mathbb{E}[D_j^2] V_2^2(g_j). \tag{2}$$

We recall some bounds on Δ_j which hold under the following assumptions, in the Array-RQMC setting.

Assumption 1. *Suppose that $\ell = d = 1$ and $\varphi_j(x,u)$ is non-decreasing in u. That is, we use inversion to generate the next state from the cdf $F_j(z \mid \cdot) = P[X_j \le z \mid X_{j-1} = \cdot\,]$. Suppose also that $n = k^2$ for some integer k and that each square of the $k \times k$ grid contains exactly one RQMC point.*

Define

$$\Lambda_j \quad = \quad \sup_{0\le z\le 1} V(F_j(z \mid \cdot\,)),$$

which bounds the variation of the cdf of X_j as a function of X_{j-1}. The following Proposition was proved in [18]:

Proposition 1. *(Worst-case error.) Under Assumption 1,*

$$\Delta_j \le n^{-1/2} \sum_{\ell=1}^{j} (\Lambda_\ell + 1) \prod_{i=\ell+1}^{j} \Lambda_i. \tag{3}$$

Corollary 2. *If $\Lambda_j \le \rho < 1$ for all j, then*

$$\Delta_j \le \frac{1+\rho}{1-\rho} n^{-1/2}. \tag{4}$$

Using this to bound D_j^2 in (2) can only give an $\mathcal{O}(1/n)$ bound on the variance, which does not beat the MC rate. However, [18] also proved Proposition 3 below, which gives a variance rate of $\mathcal{O}(n^{-3/2})$ when combined with (2).

Assumption 2. *Suppose that Assumption 1 holds, that φ_j is also non-decreasing in u, and that the randomized parts of the points are uniformly distributed in the cubes and pairwise independent, conditional on the cubes in which they lie.*

Under Assumption 2, the RQMC points are a *stratified sample*, except for their first coordinates which can be deterministic. That is, the first coordinate of point i can be fixed to w_i, for each i. Alternatively, one can generate the two coordinates of each point in its subcube, and then sort the points by their first coordinate. From the viewpoint of the algorithm, this is equivalent. The proof of the next proposition could actually be extended to a situation where the points are pairwise negatively dependent in the following sense. If we have an indicator random variable for each subcube that indicates if the point is below a certain function of x, and we take two subcubes at random, the expected correlation between the two selected indicators is not positive. Certain randomized digital nets, for example, would satisfy this assumption.

Proposition 3. *Under Assumption 2, we have*

$$\mathbb{E}[D_j^2] \le \left(\frac{1}{4} \sum_{\ell=1}^{j} (\Lambda_\ell + 1) \prod_{i=\ell+1}^{j} \Lambda_i^2 \right) n^{-3/2}.$$

Corollary 4. *If $\Lambda_j \le \rho < 1$ for all j, then*

$$\mathbb{E}[D_j^2] \le \frac{1+\rho}{4(1-\rho^2)} n^{-3/2} = \frac{1}{4(1-\rho)} n^{-3/2}$$

and therefore

$$\mathrm{Var}[\bar{Y}_{n,j}] \le \frac{1}{4(1-\rho)} V_2^2(g_j) n^{-3/2}.$$

Note that these bounds are uniform in j.

For arbitrary $\ell \ge 1$, a convergence rate of $\mathcal{O}(n^{-1/(\ell+1)})$ for the worst-case error is proved in a deterministic setting in [7] for a discrete (potentially infinite) state space in $\mathcal{X} \subseteq \mathbb{Z}^\ell$, and in [8] for a continuous state space $\mathcal{X} \subseteq \mathbb{R}^\ell$, under conditions on the φ_j and using a batch sort. This rate is not given explicitly in [8], but it can be obtained by applying Proposition 1 of that paper with $n = b^m$, $d_1 = ... = d_{\ell-1} = m/(\ell+1)$ and $d_\ell = 2m/(\ell+1)$. The best proven rate is $o(n^{-1})$ for the variance, proved by [9] in a setting where a Hilbert sort is used.

5. Examples

Example 1. Our first example has $\ell = d = 1$ and $X_j \sim U(0,1)$ at each step. Let $\theta \in (0, 1/2)$ be a fixed parameter and let G_θ be the cdf of $Y = \theta U + (1-\theta)V$, where U and V are two independent $U(0,1)$ random variables. We have

$$G_\theta(y) = \begin{cases} \dfrac{y^2}{2\theta(1-\theta)} & \text{, if} \quad 0 \le y \le \theta, \\ \dfrac{2y-\theta}{2(1-\theta)} & \text{, if} \quad \theta < y \le 1-\theta, \\ 1 - \dfrac{(1-y)^2}{2\theta(1-\theta)} & \text{, if} \quad 1-\theta < y \le 1. \end{cases}$$

We define the Markov chain $\{X_j, j \ge 1\}$ in $\mathcal{X} = [0,1]$ via the recurrence

$$X_1 = U_1, \qquad X_j = \varphi(X_{j-1}, U_j) = G_\theta(\theta X_{j-1} + (1-\theta)U_j), \quad j \ge 2,$$

where the U_j's are i.i.d. $U(0,1)$. Then $X_j \sim U(0,1)$ for each $j \geq 1$, and one can show that $\Lambda_j = \theta/(1-\theta) \stackrel{\text{def}}{=} \rho < 1$. Corollaries 2 and 4 give

$$\Delta_j \leq \frac{1+\rho}{1-\rho} n^{-1/2} = \frac{n^{-1/2}}{1-2\theta} \quad \text{and} \quad \mathbb{E}[D_j^2] \leq \frac{n^{-3/2}}{4(1-\rho)} = \frac{1-\theta}{4(1-2\theta)} n^{-3/2}.$$

We tried the following cost functions, all defined to have mean zero over $[0,1]$: $g_j(x) = x - 1/2$, $g_j(x) = x^2 - 1/3$, $g_j(x) = \sin(2\pi x)$, $g_j(x) = e^x - e + 1$, $g_j(x) = \max(0, x-1/2) - 1/8 = (x-1/2)^+ - 1/8$, and $g_j(x) = \mathbb{I}[x \leq 1/3] - 1/3$. The first four are continuously differentiable, the fifth is continuous but not differentiable at $1/2$, and the sixth is discontinuous at $1/3$. We made some experiments to estimate the (empirical) convergence rate of $\mathbb{E}[D_j^2]$ and of $\mathrm{Var}[\bar{Y}_{n,j}]$ for these choices of g_j, as a function of n, first for standard Monte Carlo (MC) with n independent realizations of the chain, then for selected RQMC point sets defined over the unit square $[0,1)^2$. For n, we took all powers of 2 from 2^9 to 2^{21}. In each case, we performed $m = 200$ independent RQMC replicates, and took the average of square values to estimate the mean square discrepancy $\mathbb{E}[D_j^2]$ and the RQMC variance $\mathrm{Var}[\bar{Y}_{n,j}]$. We then fitted linear regression models for $\log_2 \mathbb{E}[D_j^2]$ and for $\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$ as functions of $\log_2 n$, to estimate the slope.

For Array-RQMC, the n two-dimensional RQMC points are always sorted by their first coordinate, and the second coordinate is used to advance the state by one step. We tried the following types of point sets, which are further described below (we describe the two-dimensional versions but all these point sets are also defined in more than two dimensions): Latin hypercube sampling (LHS); stratified sampling (SS); stratified sampling with antithetic variates in each stratum (SSA); a Sobol' point set with a random linear matrix scrambling and a digital random shift (Sobol); Sobol with a baker's transformation of the points after the shift (Sobol+baker); a Korobov lattice rule in two dimensions with a random shift modulo 1 (Korobov); and Korobov with a baker's transformation of the points after the shift (Korobov+baker). LHS divides each axis of the square in n equal intervals to produce n^2 square boxes, draw one point randomly in each box that lies on the diagonal, and permute the boxes by randomly permuting the rows. SS divides the unit square into $n = k^2$ square boxes and draws one

point randomly in each box. In this case, n must be a square and we take the closest square to the power of 2 of interest. SSA uses a grid of $n/2$ square boxes instead ($n/2$ must be a square), and in each box it takes two antithetic random points. Sobol uses the two-dimensional Sobol (or Hammersley) point set: the first coordinate of point i is i/n and the second coordinate is defined by the binary expansion of i in reverse order. Then a random left matrix scramble [21] is applied, followed by a digital random shift in base 2, to the second coordinate. Sobol+baker adds a baker's transformation after the digital shift (see [10, 14] and other references therein for the details). It has been shown in [10] that the latter combination gives a convergence rate of $\mathcal{O}(n^{-1}\sqrt{\log n})$ for the L_p discrepancy, for any $p \geq 1$, in two dimensions. Korobov uses a two-dimensional rank-1 lattice with generating vector $(1/n, a/n)$ for some integer $a > 1$ and n equal to a power of 2, with the second coordinate randomized by a random shift modulo 1. Korobov+baker adds a baker's transformation after the shift. For the lattice rules, we found that $\mathbb{E}[D_1^2]$ depends significantly on the choice of a. For this reason, for each n, we tried 2 to 4 values of a that performed well with respect to the $\mathcal{P}_2$ criterion, found using Lattice Builder [19], and we selected the one that gave the smallest mean square discrepancy $\mathbb{E}[D_1^2]$. Without doing this, the plot was more erratic. The selected values of a for $n = 2^9, \ldots, 2^{21}$ are (149, 275, 857, 1731, 2431, 6915, 12545, 19463, 50673, 96407, 204843, 443165, 768165).

The upper part of Table 1 gives the estimated regression slopes for $\log_2 \mathbb{E}[D_j^2]$ as well as $\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$ for four choices of g_j, as functions of $\log_2 n$, all for $\theta = 0.3$ and $j = 5$. For the other smooth choices of g_j (sin and exp), the slopes are essentially the same as for the linear and quadratic (smooth) functions. The lower part of the table gives $-\log_{10} \mathrm{Var}[\bar{Y}_{n,j}]$ for $n = 2^{21}$, for three choices of g_j, and the total CPU time needed for the 200 replications for $n = 2^{21}$. The performances for all six functions g_j were computed simultaneously in each replication. We tried other values of θ, such as 0.1 and 0.9, and other values of j from 1 to 100, and the results were almost identical. The variances are essentially constant in j.

Table 1: Upper panel: Regression slopes for $\log_2 \mathbb{E}[D_j^2]$ and $\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$ as functions of $\log_2 n$. Lower panel: Values of $-\log_{10} \mathrm{Var}[\bar{Y}_{n,j}]$ and total CPU time in seconds for $n = 2^{21}$.

point sets	$\log_2 \mathbb{E}[D_j^2]$	$\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$			
		$X_j - \frac{1}{2}$	$X_j^2 - \frac{1}{3}$	$(X_j - \frac{1}{2})^+ - \frac{1}{8}$	$\mathbb{I}[X_j \le \frac{1}{3}] - \frac{1}{3}$
MC	-1.01	-1.02	-1.01	-1.00	-1.02
LHS	-1.02	-0.99	-1.00	-1.00	-1.00
SS	-1.50	-1.98	-2.00	-2.00	-1.49
SSA	-1.50	-2.65	-2.56	-2.50	-1.50
Sobol	-1.51	-3.22	-3.14	-2.52	-1.49
Sobol+baker	-1.50	**-3.41**	**-3.36**	**-2.54**	-1.50
Korobov	-1.87	-2.00	-1.98	-1.98	-1.85
Korobov+baker	**-1.92**	-2.01	-2.02	-2.01	**-1.90**

point sets	$-\log_{10} \mathrm{Var}[\bar{Y}_{n,j}]$ for $n = 2^{21}$			CPU time (sec)
	$X_j^2 - \frac{1}{3}$	$(X_j - \frac{1}{2})^+ - \frac{1}{8}$	$\mathbb{I}[X_j \le \frac{1}{3}] - \frac{1}{3}$	
MC	7.35	7.86	6.98	270
LHS	8.82	8.93	7.61	992
SS	13.73	14.10	10.20	2334
SSA	18.12	17.41	10.38	1576
Sobol	19.86	**17.51**	10.36	443
Sobol+baker	**20.07**	16.98	10.04	462
Korobov	13.55	14.03	11.98	359
Korobov+baker	14.59	14.56	**12.22**	**352**

We see that all methods except LHS improve the rate for both the mean square L_2 discrepancy and the variance. In all cases, this improvement is from $\mathcal{O}(n^{-1})$ to at least $\mathcal{O}(n^{-3/2})$ for the mean square discrepancy and at least $\mathcal{O}(n^{-2})$ for the variance of the five continuous functions. For the (discontinuous) indicator function, the variance decreases at roughly the same rate as $\mathbb{E}[D_j^2]$ for all point sets. In most cases, the variance converges significantly faster than the mean square discrepancy. This suggests that there should be another measure of discrepancy that would be more appropriate for this situation and that would converge at the same rate as the variance. For the four choices of smooth functions g_j (linear, quadratic, sin, and exp), the variance decreases at (essentially) the same rate. The differences can be attributed to random noise. The rates of the max function differ from those of the smooth functions only for Sobol, for which the rate deteriorates and is similar to that obtained with SSA. For $\mathbb{E}[D_j^2]$, the lattice rules give the fastest convergence rate, significantly faster than all others, whereas for the variance for all smooth functions, Sobol gives the fastest rate, and SSA also performs quite well. Adding a baker's (or tent) transformation improves the rate slightly for Sobol, and makes practically no difference for Korobov. Proving all these convergence rates seems an interesting challenge that would provide valuable insight in the method.

We find in the table that for $g_j(x) = x^2 - 1/3$ and $n = 2^{21}$, Sobol reduces the variance by a factor of about 3×10^{12} (3 trillions) with respect to MC, SSA reduces it by a factor of about 6×10^{10}, and LHS improves it by a small factor (near 30), without changing the rate.

Example 2. Our second example has a two-dimensional state. It is an Asian option, with the same parameters as in Section 4.5 of [18] and Example 2 of [16]. Given observation times $t_1, t_2, \ldots, t_\tau$, we have

$$S(t_j) = S(t_{j-1}) \exp[(r - \sigma^2/2)(t_j - t_{j-1}) + \sigma(t_j - t_{j-1})^{1/2} \Phi^{-1}(U_j)],$$

where Φ is the standard normal cdf, $U_j \sim U(0,1)$, and $S(t_0) = s_0$ is fixed. At step j, we have the running average $\bar{S}_j = \frac{1}{j} \sum_{i=1}^{j} S(t_i)$ and the state is

$X_j = (S(t_j), \bar{S}_j)$. This state evolves as

$$X_j = (S(t_j), \bar{S}_j) = \varphi_j(S(t_{j-1}), \bar{S}_{j-1}, U_j) = \left(S(t_j), \frac{(j-1)\bar{S}_{j-1} + S(t_j)}{j} \right).$$

The total payoff is $Y = g_\tau(X_\tau) = \max\left[0, \bar{S}_\tau - K\right]$. The selected parameters are $S(0) = 100$, $K = 100$, $r = \ln(1.09)$, $\sigma = 0.2$, and $t_j = (230 + j)/365$, for $j = 1, \ldots, \tau = 10$.

We tried the split sort, batch sort, Hilbert batch sort, and Hilbert sort with a logistic transformation as in [9]. For the last two sorts, the two-dimensional states are mapped to $[0, 1]$, so we only need two-dimensional RQMC point sets, as in Example 1. But with the split and batch sorts, we need three-dimensional RQMC point sets. The first two coordinates are used to match the points with the chains. We used SS, Sobol, and Korobov+baker, defined as in Example 1, but in three dimensions when needed. For SS, all coordinates are randomized (we generate a fresh stratified sample) and the points are re-sorted at each step, whereas for Sobol and Korobov, only the last coordinate is randomized. Table 2 reports the regression slopes of $\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$ as a function of $\log_2 n$, estimated from $m = 100$ replications with $n = 2^e$ for $e = 9, \ldots, 20$, as well as the variance reduction factors (VRF) w.r.t. MC and the CPU times to perform the 100 replications for $n = 2^{20}$.

With Sobol and Korobov+baker, the variance decreases roughly as n^{-2}, and for $n = 2^{20}$ it is about the same for these two point sets and all sorting algorithm, except for the Hilbert batch sort for which the rate is slightly slower. The best VRFs are obtained with the batch sort. For SS, computations take more time and the variance decreases slower than with other point sets. Unlike the other point sets, SS is sensitive to the sort method: the variance decreases faster with the Hilbert and Hilbert batch sorts than with the split or batch sorts. The split sort is implemented recursively, and for this reason is significantly slower than the other sort methods. The logistic map used with the Hilbert sort is centered at zero with a half-width of two standard deviations, as suggested in [9]. We found that the results significantly deteriorate when using other continuous, strictly increasing functions for which the transition from 0 to 1 is either very

Table 2: Regression slopes for $\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$ vs $\log_2 n$, VRF for RQMC vs MC, and CPU time in seconds for $m = 100$, for $n = 2^{20}$, for the Asian option.

Sort	RQMC points	$\log_2 \mathrm{Var}[\bar{Y}_{n,j}]$	VRF	CPU time (sec)
Split sort	SS	-1.38	2.0×10^2	3093
	Sobol	-2.04	4.0×10^6	1116
	Korobov+baker	-2.00	2.2×10^6	903
Batch sort	SS	-1.38	2.0×10^2	744
($n_1 = n_2$)	Sobol	-2.03	4.2×10^6	532
	Korobov+baker	-2.04	4.4×10^6	482
Hilbert batch sort	SS	-1.54	2.3×10^3	835
	Sobol	-1.79	1.4×10^5	555
	Korobov+baker	-1.92	3.4×10^6	528
Hilbert sort	SS	-1.55	2.4×10^3	840
(with logistic map)	Sobol	-2.03	2.6×10^6	534
	Korobov+baker	-2.01	3.3×10^6	567

steep or occurs far from the origin.

6. Conclusion

In a stylized one-dimensional numerical example, we found that the convergence rate for $\mathbb{E}[D_j^2]$ matches the proved rate of $\mathcal{O}(n^{-3/2})$ (except for lattice rules, for which it does better), and the convergence rate of the variance matches that of $\mathbb{E}[D_j^2]$ for a discontinuous cost function g_j. But much faster rates are obtained for the variance (up to about $\mathcal{O}(n^{-3.4})$) when g_j is continuously-differentiable, and still about $\mathcal{O}(n^{-2.5})$ when g_j is continuous but not differentiable. In a two-dimensional example with a continuous but not differentiable cost function, we observed a $\mathcal{O}(n^{-2})$ convergence rate for the variance, while $o(n^{-1})$ is the best that has been proved so far. The observed rate and also the variance values at large n were about the same for different sorting methods: batch sort, split sort, and Hilbert-curve sort. Proving those empirically-observed rates is an interesting challenge and is the object of our ongoing research.

Acknowledgements

This work has been supported by a Canada Research Chair, an Inria International Chair, and a NSERC Discovery Grant to P. L'Ecuyer.

References

[1] M. Bader, Space-Filling Curves: An Introduction with Applications in Scientific Computing, Springer, Berlin, 2013.

[2] J.J. Bartholdi III, L.K. Platzman, Heuristics based on space-filling curves for combinatorial problems in the Euclidean space, Management Science 34 (1988) 291–305.

[3] K. Buchin, Organizing Point Sets: Space-Filling Curves, Delaunay Tessellations of Random Point Sets, and Flow Complexes, Ph.D. thesis, Freien

Universität Berlin, Department of Mathematics and Computer Science, 2008.

[4] V. Demers, P. L'Ecuyer, B. Tuffin, A combination of randomized quasi-Monte Carlo with splitting for rare-event simulation, in: Proceedings of the 2005 European Simulation and Modeling Conference, EUROSIS, Ghent, Belgium, pp. 25–32.

[5] M. Dion, P. L'Ecuyer, American option pricing with randomized quasi-Monte Carlo simulations, in: B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, E. Yücesan (Eds.), Proceedings of the 2010 Winter Simulation Conference, pp. 2705–2720.

[6] J. Durbin, Distribution Theory for Tests Based on the Sample Distribution Function, SIAM CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, PA, 1973.

[7] R. El Haddad, C. Lécot, P. L'Ecuyer, Quasi-Monte Carlo simulation of discrete-time Markov chains on multidimensional state spaces, in: A. Keller, S. Heinrich, H. Niederreiter (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer-Verlag, Berlin, 2008, pp. 413–429.

[8] R. El Haddad, C. Lécot, P. L'Ecuyer, N. Nassif, Quasi-Monte Carlo methods for Markov chains with continuous multidimensional state space, Mathematics and Computers in Simulation 81 (2010) 560–567.

[9] M. Gerber, N. Chopin, Sequential quasi-Monte Carlo, Journal of the Royal Statistical Society, Series B 77 (2015) 509–579.

[10] T. Goda, On the L_p discrepancy of two-dimensional folded Hammersley point sets, Archiv der Mathematik 103 (2014) 389–398.

[11] Z. He, A.B. Owen, Extensible grids: uniform sampling on a space filling curve, Journal of the Royal Statistical Society: Series B (Statistical Methodology) (2015).

[12] C. Lécot, I. Coulibaly, A quasi-Monte Carlo scheme using nets for a linear Boltzmann equation, SIAM Journal on Numerical Analysis 35 (1998) 51–70.

[13] C. Lécot, B. Tuffin, Quasi-Monte Carlo methods for estimating transient measures of discrete time Markov chains, in: H. Niederreiter (Ed.), Monte Carlo and Quasi-Monte Carlo Methods 2002, Springer-Verlag, Berlin, 2004, pp. 329–343.

[14] P. L'Ecuyer, Quasi-Monte Carlo methods with applications in finance, Finance and Stochastics 13 (2009) 307–349.

[15] P. L'Ecuyer, V. Demers, B. Tuffin, Rare-events, splitting, and quasi-Monte Carlo, ACM Transactions on Modeling and Computer Simulation 17 (2007) Article 9.

[16] P. L'Ecuyer, C. Lécot, A. L'Archevêque-Gaudet, On array-RQMC for Markov chains: Mapping alternatives and convergence rates, in: P. L'Ecuyer, A.B. Owen (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer-Verlag, Berlin, 2009, pp. 485–500.

[17] P. L'Ecuyer, C. Lécot, B. Tuffin, Randomized quasi-Monte Carlo simulation of Markov chains with an ordered state space, in: H. Niederreiter, D. Talay (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2004, Springer-Verlag, Berlin, 2006, pp. 331–342.

[18] P. L'Ecuyer, C. Lécot, B. Tuffin, A randomized quasi-Monte Carlo simulation method for Markov chains, Operations Research 56 (2008) 958–975.

[19] P. L'Ecuyer, D. Munger, Lattice builder: A general software tool for constructing rank-1 lattice rules, ACM Transactions on Mathematical Software (2015). To appear, see `http://www.iro.umontreal.ca/~lecuyer/papers.html`.

[20] P. L'Ecuyer, C. Sanvido, Coupling from the past with randomized quasi-Monte Carlo, Mathematics and Computers in Simulation 81 (2010) 476–489.

[21] J. Matoušek, Geometric Discrepancy: An Illustrated Guide, Springer-Verlag, Berlin, 1999.

[22] H. Niederreiter, Quasi-Monte Carlo methods and pseudorandom numbers, Bulletin of the American Mathematical Society 84 (1978) 957–1041.

[23] L.K. Platzman, J.J. Bartholdi III, Space-filling curves and the planar traveling salesman problem, Journal of the ACM 36 (1989) 719–737.

[24] C. Schretter, H. Niederreiter, A direct inversion method for non-uniform quasi-random point sequences, Monte Carlo Methods and Applications 19 (2013) 1–9.

[25] C. Wächter, A. Keller, Efficient simultaneous simulation of Markov chains, in: A. Keller, S. Heinrich, H. Niederreiter (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer-Verlag, Berlin, 2008, pp. 669–684.