Probability in the Engineering and Informational Sciences, **24**, 2010, 405–422.
doi:10.1017/S0269964810000057

HOW TO GENERATE UNIFORM SAMPLES ON DISCRETE SETS USING THE SPLITTING METHOD

PETER W. GLYNN, ANDREY DOLGIN, REUVEN Y. RUBINSTEIN, AND RADISLAV VAISMAN
Faculty of Industrial Engineering and Management Technion
Israel Institute of Technology
Haifa, Israel
E-mail: ierrr01@ie.technion.ac.il; iew3.technion.ac.il:8080/ierrr01.phtml

The goal of this work is twofold. We show the following:

1. In spite of the common consensus on the classic Markov chain Monte Carlo (MCMC) as a universal tool for generating samples on complex sets, it fails to generate points uniformly distributed on discrete ones, such as that defined by the constraints of integer programming. In fact, we will demonstrate empirically that not only does it fail to generate uniform points on the desired set, but typically it misses some of the points of the set.
2. The *splitting*, also called the *cloning* method – originally designed for combinatorial optimization and for counting on discrete sets and presenting a combination of MCMC, like the Gibbs sampler, with a specially designed splitting mechanism—can also be efficiently used for generating uniform samples on these sets. Without introducing the appropriate splitting mechanism, MCMC fails. Although we do not have a formal proof, we guess (conjecture) that the main reason that the classic MCMC is not working is that its resulting chain is not irreducible. We provide valid statistical tests supporting the uniformity of generated samples by the splitting method and present supportive numerical results.

1. INTRODUCTION: THE SPLITTING METHOD

The goal of this work is to show the following:

1. The classic MCMC (Markov chain Monte Carlo) *fails* to generate points uniformly distributed on discrete sets, such as that defined by the constraints

 0269-9648/10 $25.00

of integer programming with both equality and inequality constraints; that is,

$$\begin{aligned}
&\sum_{k=1}^{n} a_{ik}x_k = b_i, \qquad i = 1, \ldots, m_1, \\
&\sum_{k=1}^{n} a_{jk}x_k \geq b_j, \qquad j = m_1 + 1, \ldots, m_1 + m_2, \\
&\boldsymbol{x} = (x_1, \ldots, x_n) \geq \mathbf{0}, \quad x_k \text{ is integer } \forall k = 1, \ldots, n.
\end{aligned} \tag{1}$$

We demonstrate empirically that starting MCMC from any initial point in the desired set $\mathcal{X}^*$ given in (1) and running it for a very long time not only *fails* to generate uniform points on $\mathcal{X}^*$ but samples only in some subset of $\mathcal{X}^*$, rather than in the entire set $\mathcal{X}^*$. We observed that this is the case even if $\mathcal{X}^*$ is very small containing only view points. Thus, in spite of the common consensus (MCMC) as a universal tool for generating samples on complex sets, our empirical studies on discrete sets, like (1), have proved quite negative. Although we do not have a formal proof, we guess (conjecture) that the main reason the classic MCMC is not working is that its resulting chain is not irreducible.

2. In contrast to MCMC, the *splitting* method, also called the *cloning* method, recently introduced in [11,12] can be efficiently used for generating uniform samples on sets like (1). We provide valid statistical tests supporting the uniformity of generated samples on $\mathcal{X}^*$ and present supportive numerical results.

At first glance one might think that the classic MCMC [1,9,14,15] should be a good alternative sets like (1). Indeed, MCMC has been successfully used for generating points on different complex regions. In all such cases, given an arbitrary initial point in $\mathcal{X}^*$, one runs MCMC for some time until it reaches steady state and then collects the necessary data. One of the most popular MCMC is the hit-and-run method [15] for generation of uniform points on continuous regions. Applications of hit-and-run for convex optimization are given in [7].

As mentioned, we will show that this is not always the case: MCMC fails when one deals with discrete sets like (1). To emphasize this point, observe that most decision making, optimization, and counting problems associated with the set (1) are NP-hard; thus, one should not expect an easy way of generating points uniformly on (1) since it is shown in [11,12] that counting on $\mathcal{X}^*$ (which is NP-hard) is directly associated with uniform sampling on $\mathcal{X}^*$. It is also shown that the splitting method [11,12] presents a combination of MCMC with a specially designed splitting mechanism. Again, without the appropriate splitting mechanism, MCMC fails.

Although this article is mainly of empirical nature, we believe that it provides a good insight of the state of the art of generating uniform points on discrete sets $\mathcal{X}^*$ like (1) and that it will motivate further research.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

We start by presenting some background on the splitting method following [11,12]. For related references, see [2–6,8].

Like the classic cross-entropy (CE) method [10,13], the splitting one in [11,12] was originally designed for counting and combinatorial optimization. As mentioned, the counting algorithm in [11,12] assumes, in fact, uniform generation on that set $\mathcal{X}^*$. So, from that retrospective, one can view this generation as a nice "free" byproduct of this algorithm.

The rest of this section deals with the splitting method from [12] for counting. The main idea is to design a sequential sampling plan, with a view to decomposing a "difficult" counting problem defined on some set $\mathcal{X}^*$ into a number of "easy" ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ and such that $\mathcal{X}_m = \mathcal{X}^*$. Typically, splitting algorithms explore the connection between counting and sampling problems—in particular, reduction from approximate counting on a discrete set to approximate sampling of its elements by the classic MCMC method [14].

A typical splitting algorithm comprises the following steps:

1. Formulate the counting problem as that of estimating the cardinality $|\mathcal{X}^*|$ of some set $\mathcal{X}^*$.
2. Find a sequence of sets $\mathcal{X} = \mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ such that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ and $|\mathcal{X}| = |\mathcal{X}_0|$ is known.
3. Write $|\mathcal{X}^*| = |\mathcal{X}_m|$ as

$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = \ell |\mathcal{X}_0|, \tag{2}$$

   where $\ell = \prod_{t=1}^{m} (|\mathcal{X}_t|/|\mathcal{X}_{t-1}|)$. Note that ℓ is typically very small (e.g., $\ell = 10^{-100}$) and each ratio

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \tag{3}$$

   should not be small (e.g., $c_t = 10^{-2}$ or larger). Clearly, estimating ℓ directly while sampling in $|\mathcal{X}_0|$ is meaningless, but estimating each c_t separately seems to be a good alternative.
4. Develop an efficient estimator $\widehat{c}_t = |\widehat{\mathcal{X}}_t|/|\widehat{\mathcal{X}}_{t-1}|$ for each $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}|$.
5. Estimate $|\mathcal{X}^*|$ by

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}| \prod_{t=1}^{m} \widehat{c}_t, \tag{4}$$

   where $|\widehat{\mathcal{X}}_t|$, $t = 1, \ldots, m$, is an estimator of $|\mathcal{X}_t|$, and similarly for the rare-event probability ℓ.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

It is readily seen that in order to obtain a meaningful estimator of $|\mathcal{X}^*|$, we have to solve the following two major problems:

(i) Put the well-known NP-hard counting problems into the framework (2) by making sure that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ and each c_t is not a rare-event probability.

(ii) Obtain a low variance estimator $\widehat{c}_t$ for each $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}|$.

Whereas task (i) is typically not difficult [12], task (ii) is quite complicated and associated with the generation of uniform samples at each subregion $\mathcal{X}_t$ separately. As we will see below, this can be done by combining the Gibbs sampler with a specially designed splitting mechanism. The resulting algorithm is called the *splitting* or *cloning* algorithm [12].

The main goal of this work is to show empirically that the *splitting* algorithm [12] is able to generate points uniformly distributed on different discrete sets.

To proceed, note that ℓ can be also written as

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) \geq m\}} \right], \tag{5}$$

where $\boldsymbol{X} \sim f(\boldsymbol{x})$, where $f(\boldsymbol{x})$ is a uniform distribution on the entire set $\mathcal{X}$, as before; m is a fixed parameter (e.g., the total number of constraints in an integer program); and $S(\boldsymbol{X})$ is the sample performance (e.g., the number of feasible solutions generated by the above constraints). Alternatively (see(2)), ℓ can be written as

$$\ell = \prod_{t=1}^{T} c_t, \tag{6}$$

where

$$c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}| = \mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X}) \geq m_{t-1}\}}]. \tag{7}$$

Here,

$$g^*_{t-1} = g^*(\boldsymbol{x}, m_{t-1}) = \ell(m_{t-1})^{-1} f(\boldsymbol{x}) I_{\{S(\boldsymbol{x}) \geq m_{t-1}\}}, \tag{8}$$

where $\ell(m_{t-1})^{-1}$ is the normalization constant and similarly to (2) the sequence m_t, and $t = 0, 1, \ldots, T$ represents a fixed grid satisfying $-\infty < m_0 < m_1 < \cdots < m_T = m$. Note that in contrast to (2), we use in (6) a product of T terms instead of m terms, where T might be a random variable. The latter case is associated with adaptive choice of the level sets $\{\widehat{m}_t\}_{t=0}^{T}$ resulting in $T \leq m$. Since for counting problems the probability density function (p.d.f.) $f(\boldsymbol{x})$ should be *uniformly* distributed on $\mathcal{X}$, which we denote by $\mathcal{U}(\mathcal{X})$, it follows from (8) that the p.d.f. $g^*(\boldsymbol{x}, m_{t-1})$ should be *uniformly* distributed on each set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_{t-1}\}, t = 1, \ldots, T$; that is, $g^*(\boldsymbol{x}, m_{t-1})$ should be equal to $\mathcal{U}(\mathcal{X}_t)$. Recall that the goal of the article is to show that this is indeed the case for $\mathcal{X}_T = \mathcal{X}^* = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_T\}$, where $m_T = m$.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

Once sampling from $g_t^* = \mathcal{U}(\mathcal{X}_t)$ becomes feasible, the final estimator of ℓ (based on the estimators of $c_t = \mathbb{E}_{g_{t-1}^*}[I_{\{S(\boldsymbol{X}) \geq m_{t-1}\}}],\ t = 0, \ldots, T$), can be written as

$$\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t = \frac{1}{N^T} \prod_{t=1}^{T} N_t, \tag{9}$$

where

$$\widehat{c}_t = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq m_{t-1}\}} = \frac{N_t}{N}, \tag{10}$$

$N_t = \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq m_{t-1}\}}$, $\boldsymbol{X}_i \sim g_{t-1}^*$, and $g_{-1}^* = f$.

We next show how to put the counting problem of finding the number of feasible solutions of the set of integer programming constraints into the framework (5)–(8).

Example 1.1: The Set of Integer Programming Constraints: Consider again the set $\mathcal{X}^*$ of integer programming constraints given in (1). Our goal is to generate points uniformly distributed of this set. We assume that each component $x_k, k = 1, \ldots, n$, has d different values, labeled $1, \ldots, d$. Note that the SAT problem represents a particular case of (1) with inequality constraints and where $x_1, \ldots, x_n$ are binary components. Unless stated otherwise, we will bear in mind the counting problem on the set (1)—in particular, counting the true (valid) assignments in a SAT problem.

It is shown in [12] that in order to count the points of the set (1), one can associate it with the following rare-event probability problem:

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) = m\}} \right] = \mathbb{E}_f \left[I_{\{\sum_{i=1}^{m} C_i(\boldsymbol{X}) = m\}} \right], \tag{11}$$

where the first m_1 terms $C_i(\boldsymbol{X})$' in (11) are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^{n} a_{ik} X_k = b_i\}}, \qquad i = 1, \ldots, m_1 \tag{12}$$

and the remaining m_2 ones are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^{n} a_{ik} X_k \geq b_i\}}, \qquad i = m_1 + 1, \ldots, m_1 + m_2 \tag{13}$$

and $S(\boldsymbol{X}) = \sum_{i=1}^{m} C_i(\boldsymbol{X})$. Thus, in order to count the number of feasible solutions on the set (1), one can consider an associated rare-event probability estimation problem (11) involving a *sum of dependent Bernoulli random variables* $C_i, i = m_1 + 1, \ldots, m$, and then apply $|\widehat{\mathcal{X}^*}| = \widehat{\ell}|\mathcal{X}|$. In other words, in order to count on $\mathcal{X}^*$, one needs to estimate efficiently the rare-event probability ℓ in (11). A framework similar to (11) can be readily established for many NP-hard counting problems [12].

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

It follows from the above that the splitting algorithm will generate an adaptive sequence of tuples:

$$\{(m_0, g^*(\boldsymbol{x}, m_{-1})), (m_1, g^*(\boldsymbol{x}, m_0)), (m_2, g^*(\boldsymbol{x}, m_1)), \ldots, (m_T, g^*(\boldsymbol{x}, m_{T-1}))\}. \quad \textbf{(14)}$$

Here, as earlier, $g^*(\boldsymbol{x}, m_{-1}) = f(\boldsymbol{x})$ and m_t is obtained from the solution of the following nonlinear equation:

$$\mathbb{E}_{g^*_{t-1}} I_{\{S(\boldsymbol{X}) \geq m_t\}} = \rho, \quad \textbf{(15)}$$

where ρ is called the *rarity* parameter [12]. Typically, one sets $0.1 \leq \rho \leq 0.01$. Note that in contrast to the CE method [10,13], where one generates a sequence of tuples

$$\{(m_0, \boldsymbol{v}_0), (m_1, \boldsymbol{v}_1), \ldots, (m_T, \boldsymbol{v}_T)\} \quad \textbf{(16)}$$

and where $\{\boldsymbol{v}_t,\ t = 1, \ldots, T\}$ is a sequence of parameters in the parametric family of distributions $f(\boldsymbol{x}, \boldsymbol{v}_t)$, in (14), $\{g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1},\ t = 0, 1, \ldots, T\}$ is a sequence of *nonparametric* IS distributions. Otherwise, the CE and the splitting algorithms are similar.

In the Appendix, following [12], we present two versions of the splitting algorithm for counting: the so-called *basic* Algorithm A.1 and the *enhanced* one, Algorithm A.2, bearing in mind Example 1.1. Recall that the crucial point is to ensure that the points generated from the p.d.f. $g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1}$ are uniformly distributed on the corresponding set $\mathcal{X}_t = \{S(\boldsymbol{X}) \geq m_t\}$, $t = 1, \ldots, T$.

To understand that this is so, consider the enhanced Algorithm A.2, bearing in mind the following:

1. The samples generated on the set $\mathcal{X}_1 = \{S(\boldsymbol{X}) \geq \widehat{m}_0\}$ from the p.d.f. $g^*(\boldsymbol{x}, \widehat{m}_0) = g^*_0$ are *exactly* distributed uniformly since the original distribution f is a uniform one on the entire space $\mathcal{X} = \mathcal{X}_0$ and since use of acceptance–rejection (see Step 1) yields uniform points on $\mathcal{X}_1$.
2. The samples generated on the sets $\mathcal{X}_t = \{S(\boldsymbol{X}) \geq \widehat{m}_{t-1}\}$ from the corresponding p.d.f.s $g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1}$, $t = 2, \ldots, T$, are distributed only *approximately* uniformly. This is so since starting from iteration $t = 2$ we first split the elite samples and then apply to each of them the Gibbs sampler, which runs for some burn-in periods (see Step 2). This in turn means that we run N Markov chains in parallel. The goal of the Gibbs sampler is, therefore, to keep the N Markov chains in steady state while sampling at $\mathcal{X}_t = \{S(\boldsymbol{X}) \geq \widehat{m}_{t-1}\}$, $t = 2, \ldots, T$. This is an easy task achievable by running the Gibbs sampler for a number of burn-in periods.

Note that the splitting algorithm in [12] is also suitable for optimization. Here, we use the same sequence of tuples (14) but *without involving the product of the estimators* $\widehat{c}_t$, $t = 1, \ldots, T$.

The rest of our article is organized as follows. Section 2 deals with the Gibbs sampler, which is an important element of the splitting algorithm. In particular, we

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

show how to generate points uniformly on the set (1) avoiding acceptance–rejection. Section 3 presents supporting numerical results. In Section 4 conclusions and some directions for further research are given. Finally, in the Appendix the basic and the enhanced versions of the splitting algorithm are presented.

2. THE GIBBS SAMPLER

In this section we show how to use efficiently the Gibbs sampler to generate points uniformly on the set (1). We start with some background [14] on the generation of points from a given joint p.d.f. $g(x_1, \ldots, x_n)$. In the latter, instead of sampling directly from $g(x_1, \ldots, x_n)$, which might be very difficult, one samples from the one-dimensional conditional p.d.f.s $g(x_i|X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$, $i = 1, \ldots, n$, which is typically much simpler. Two basic versions of the Gibbs sampler are available: *systematic* and *random*. In the former, the components of the vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ are updated in a fixed, say increasing order, whereas in the latter they are chosen randomly according to a discrete uniform n-point p.d.f. Below, we present the systematic Gibbs sampler algorithm. In the systematic version, for a given vector $\boldsymbol{X} = (X_1, \ldots, X_n) \sim g(\boldsymbol{x})$, one generates a *new* vector $\widetilde{\boldsymbol{X}} = (\widetilde{X}_1, \ldots, \widetilde{X}_n)$ with the same distribution $\sim g(\boldsymbol{x})$ using Algorithm 2.1.

Algorithm 2.1 (Systematic Gibbs Sampler)

1. Draw $\widetilde{X}_1$ from the conditional p.d.f. $g(x_1|X_2, \ldots, X_n)$.
2. Draw $\widetilde{X}_i$ from the conditional p.d.f. $g(x_i|\widetilde{X}_1, \ldots, \widetilde{X}_{i-1}, X_{i+1}, \ldots, X_n)$, $i = 2, \ldots, n-1$.
3. Draw $\widetilde{X}_n$ from the conditional p.d.f. $g(x_n|\widetilde{X}_1, \ldots, \widetilde{X}_{n-1})$.

Iterating with Algorithm 2.1, the Gibbs sampler generates (under some mild conditions [14]), a sample distributed $g(x_1, \ldots, x_n)$.

We denote for convenience each conditional p.d.f. $g(x_i|\widetilde{X}_1, \ldots, \widetilde{X}_{i-1}, X_{i+1}, \ldots, X_n)$ by $g(x_i|\boldsymbol{x}_{-i})$, where $|\boldsymbol{x}_{-i}$ denotes conditioning on all random variables except the i-th component.

Next we present a random Gibbs sampler taken from [9] for estimating each $c_t = \mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X}) \geq m_{t-1}\}}]$, $t = 0, 1, \ldots, T$, separately according to (10); that is,

$$\widehat{c}_t = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq m_{t-1}\}} = \frac{N_t^{(e)}}{N}.$$

Algorithm 2.2 (Ross's Acceptance–Rejection Algorithm for Estimating c_t)

1. Set $N_t^{(e)} = N = 0$.
2. Choose a vector $\boldsymbol{x}$ such that $S(\boldsymbol{x}) \geq m_{t-1}$.
3. Generate a random number $U \sim U(0, 1)$ and set $I = \text{Int}(nU) + 1$.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

4. If $I = k$, generate Y_k from the conditional one-dimensional distribution $g(x_k|\boldsymbol{x}_{-k})$ (see Algorithm 2.1).
5. If $S(\widetilde{X}_1, \ldots, \widetilde{X}_{k-1}, Y_k, X_{k+1}, \ldots, X_n) < m_{t-1}$, return to part 4.
6. Set $N = N + 1$ and $Y_k = \widetilde{X}_k$.
7. If $S(\boldsymbol{x}) \geq m_t$, then $N_t^{(e)} = N_t^{(e)} + 1$.
8. Go to part 3.
9. Estimate c_t as $\widehat{c}_t = N_t^{(e)}/N$.

Note that Algorithm 2.2 (see Step 5) is based on the acceptance–rejection method. For many rare-event and counting problems, generation from the conditional pdf $g(x_i|\boldsymbol{x}_{-i})$ can be done directly; that is, skipping part 5 in it. This should clearly result in a speed-up.

Example 2.1: *Sum of Independent Random Variables*: Consider the estimation of ℓ with $S(\boldsymbol{x}) = \sum_{i=1}^n X_i$; that is,

$$\ell = \mathbb{E}_f \left[I_{\{\sum_{i=1}^n X_i \geq m\}} \right]. \tag{17}$$

In this case, random variables X_i, $i = 1, \ldots, n$, for a fixed value m can be easily generated by the Gibbs sampler based on the following conditional p.d.f.:

$$g^*(x_i, m|\boldsymbol{x}_{-i}) = \propto f_i(x_i) I_{\{x_i \geq m - \sum_{j \neq i} x_j\}}, \tag{18}$$

where $\propto$ means proportional to.

Note also that each of the n conditional p.d.f.s $g^*(x_i, m|\boldsymbol{x}_{-i})$ represents a truncated version of the proposed marginal p.d.f. $f_i(x_i)$ with the truncation point at $m - \sum_{j \neq i} x_j$. In short, the random variable $\widetilde{X}$ from $g^*(x_i, m|\boldsymbol{x}_{-i})$ represents a shifted original random variable $X \sim f_i(x_i)$. Generation from a such a truncated one-dimensional p.d.f. $g^*(x_i, m|\boldsymbol{x}_{-i})$ is easy and can be typically done by the inverse-transform method, thus dispensing with Step 5.

Generating a Bernoulli random variable $\widetilde{X}_i$ from (18) with the Gibbs sampler can be done as follows. Generate $Y \sim \text{Ber}(p)$. If $I_{\{Y \geq m - \sum_{j \neq i} X_j\}}$ is unity, then set $\widetilde{X}_i = Y$; otherwise, set $\widetilde{X}_i = 1 - Y$.

Example 2.2: Assume that all $X_i \sim f_i(x_i)$ are independent and identically distributed (i.i.d.) and each $f_i(x_i)$ is a uniform d-point discrete p.d.f. with mass equal to $1/d$ at points $1, 2, \ldots, d$; that is

$$f_i(x_i) = \mathcal{U}(1, 2, \ldots, d) = \mathcal{U}\left(\frac{1}{d}\right).$$

We first apply for this example the original Algorithm 2.2, (i.e., using acceptance–rejection) and then show how one can dispense with it.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

Procedure 1: Acceptance–Rejection. In this case, given a fixed point $\boldsymbol{X} = (X_1, \ldots, X_n)$, generating $\widetilde{X}_i$ from $g(x_i, m|\boldsymbol{x}_{-i})$ (see part 5 of Algorithm 2.2) can be done as follows:

Generate $Y \sim f_i(x_i)$. *If* $Y \geq m - \sum_{j \neq i} X_j$, *then accept* Y; *that is*,

set $\widetilde{X}_i = Y$; *otherwise, reject* Y *and try again.*

For instance, consider generation with a systematic Gibbs sampler of a two-dimensional random vector $(\widetilde{X}_1, \widetilde{X}_2)$ on the set $\{\mathcal{X} : x_1 + x_2 \geq m\}$, given a fixed two-dimensional vector $\boldsymbol{x} = (x_1, x_2)$. Assume that both random variables X_1 and X_2 are i.i.d. symmetric dice, $m = 8$, and the initial point $(x_1, x_2) = (3, 5)$. Consider the following dynamic while simulating $(\widetilde{X}_1, \widetilde{X}_2)$ according to Procedure 1.

1. *Generating* $\widetilde{X}_1$. Generate $Y \sim f_1(x_1)$. Let $Y = 2$. Check if $Y \geq m - \sum_{j \neq 1} X_j$ holds. We have $2 \geq 8 - 5 = 3$. This is false; so we reject Y and try again. Next, let $Y = 4$. In this case, $Y \geq m - \sum_{j \neq 1} X_j$ holds; so we set $\widetilde{X}_1 = Y = 4$.
2. *Generating* $\widetilde{X}_2$. Generate $Y \sim f_2(x_2)$. Let $Y = 3$. Check if $Y \geq m - \sum_{j \neq 2} X_j$ holds. We have $3 \geq 8 - 4 = 4$. This is false; so we reject Y and try again. Next let $Y = 6$. In this case, $Y \geq m - \sum_{j \neq 2} X_j$ holds; so we set $\widetilde{X}_2 = Y = 6$.

The resulting point is therefore $(\widetilde{X}_1, \widetilde{X}_2) = (4, 6)$, with $S(\widetilde{X}_1, \widetilde{X}_2) = 10$.

Let us proceed from $(\widetilde{X}_1, \widetilde{X}_2) = (4, 6)$ to generate one more point using the Gibbs sampler and the same level $m = 8$. Denote $(X_1, X_2) = (\widetilde{X}_1, \widetilde{X}_2)$.

1. *Generating* $\widetilde{X}_1$. Generate $Y \sim f_1(x_1)$. Let $Y = 2$. Check if $Y \geq m - \sum_{j \neq 1} X_j$ holds. We have $2 \geq 8 - 6 = 2$. This is true; so we set $\widetilde{X}_1 = Y = 2$.
2. *Generating* $\widetilde{X}_2$. Generate $Y \sim f_2(x_2)$. Let $Y = 3$. Check if $Y \geq m - \sum_{j \neq 2} X_j$ holds. We have $3 \geq 8 - 3 = 5$. This is false; so we reject Y and try again. Next, let $Y = 6$. In this case, $Y \geq m - \sum_{j \neq 2} X_j$ holds; so we set $\widetilde{X}_2 = Y = 6$.

The resulting point is therefore $(\widetilde{X}_1, \widetilde{X}_2) = (2, 6)$ and $S(\widetilde{X}_1, \widetilde{X}_2) = 8$.

We could alternatively view the above experiment as one with two simultaneously given independent initial points—namely $\boldsymbol{X}_1 = (3, 5)$ and $\boldsymbol{X}_2 = (4, 6)$—each of them run independently using the Gibbs sampler. Assume that the results of such a run (from $\boldsymbol{X}_1 = (3, 5)$ and $\boldsymbol{X}_2 = (4, 6)$) are again $\widehat{\boldsymbol{X}_1} = (4, 6)$ and $\widehat{\boldsymbol{X}_2} = (2, 6)$, respectively. If, in addition, we denote $m = m_{t-1}$ and we set a new level $m_t = 10$, then we have $N_t^{(e)} = 1$, $N = 2$ and we obtain $\widehat{c}_t = N_t^{(e)}/N = 1/2$ (by accepting the point $\widehat{\boldsymbol{X}_1} = (4, 6)$ and rejecting the point $\widehat{\boldsymbol{X}_2} = (2, 6)$).

Example 2.3: *Sum of Independent Random Variables*: *Example* 2.1 *Continued:* We now modify the above **Procedure 1** such that all Gibbs samples $\widetilde{\boldsymbol{X}} = (\widetilde{X}_1, \ldots, \widetilde{X}_n)$ are accepted. The modified procedure takes into account availability of the quantity

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

$m - \sum_{j \neq i} X_j$ and the fact that Y is a truncated version of X_i with the truncation point $m - \sum_{j \neq i} X_j$.

Define

$$r_i = \begin{cases} m - \sum_{j \neq i} X_j, & \text{if } m - \sum_{j \neq i} X_j \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{19}$$

Once r_i, $(r_i \geq 0)$ is available, we sample a point $Z \sim \mathcal{U}(1/(d - r_i + 1))$, instead of $Y \sim \mathcal{U}(\frac{1}{d})$. Recall that d is the number of different values taken by each random variable X_i.

Procedure 2: Without Acceptance–Rejection

Generate Y from the truncated uniform distribution $\mathcal{U}(1/(d - r_i + 1))$*, where* r_i *is an online parameter defined in (*19*).*

We demonstrate now how this works in practice. Again let $m = 8$ and let the initial point be $(X_1, X_2) = (3, 5)$.

1. *Generating* $\widetilde{X}_1$ *without rejection*. Find $r_1 = m - \sum_{j \neq i} X_j$. We have $r_1 = 8 - 5 = 3$. So, the truncated distribution is uniform over the points (3, 4, 5, 6) rather than over all six points (1, 2, 3, 4, 5, 6) as in the case of acceptance–rejection. Generate Y uniformly over the points (3, 4, 5, 6). Let the outcome be $Y = 4$. Set $\widetilde{X}_1 = Y = 4$.
2. *Generating* $\widetilde{X}_2$ *without rejection.* Find $r_2 = m - \sum_{j \neq i} X_j$. We have $r_2 = 8 - 4 = 4$. So, the truncated distribution is uniform on the points (4, 5, 6). Generate Y uniformly on these points. Let the result be $Y = 6$. Set $\widetilde{X}_2 = Y = 6$.

Thus, the generated point is $(\widetilde{X}_1, \widetilde{X}_2) = (4, 6)$ and $S(\widetilde{X}_1, \widetilde{X}_2) = 10$.

Let us generate one more point proceeding from $(\widetilde{X}_1, \widetilde{X}_2) = (4, 6)$ using the same $m = 8$. Denote $(X_1, X_2) = (\widetilde{X}_1, \widetilde{X}_2)$.

1. *Generating* $\widetilde{X}_1$ *without rejection*. Find $r_1 = m - \sum_{j \neq i} X_j$. We have $r_1 = 8 - 6 = 2$. So, the truncated distribution is uniform over the points (2, 3, 4, 5, 6). Generate Y uniformly over the points (2, 3, 4, 5, 6). Let the outcome be $Y = 2$. Set $\widetilde{X}_1 = Y = 2$.
2. *Generating* $\widetilde{X}_2$ *without rejection*. Find $r_2 = m - \sum_{j \neq i} X_j$. We have $r_2 = 8 - 2 = 6$. So, the truncated distribution is a degenerated one with the whole mass at point 6 and we automatically set $Y = 6$. We deliver $\widetilde{X}_2 = Y = 6$. The generated point is therefore $(\widetilde{X}_1, \widetilde{X}_2) = (2, 6)$ with $S(\widetilde{X}_1, \widetilde{X}_2) = 8$.

Note that we deliberately made the results of Examples 2.1 and 2.3 identical.

Clearly, the above enhancement can be used for more complex models as in Example 1.1 for SAT and also for continuous p.d.f.s. Our numerical results show that it can be typically achieve a speed-up by a factor of 2–3 compared with the acceptance–rejection approach.

Example 2.4: *Counting on the Set of an Integer Program*: *Example* 1.1 *continued:* Consider the set (1). It is readily seen (see also [11]) that in order to count on this

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

set with given matrix $\boldsymbol{A} = \{a_{ij}\}$, one only needs to sample from the one-dimensional conditional p.d.f.s:

$$g^*(x_i, \widehat{m}_{t-1} | \boldsymbol{x}_{-i}) = \propto \mathcal{U}(1/d) I_{\{\sum_{r \in R_i} C_r(\boldsymbol{X}) \geq (\widehat{m}_{t-1} - c_{-i}) - \sum_{r \notin R_i} C_r(\boldsymbol{X})\}}\ , \tag{20}$$

where $R_i = \{j :\ a_{ij} \neq 0\}$ and $c_{-i} = m - |R_i|$. Note that R_i represents the set of indexes of all the constraints affected by x_i, and c_{-i} counts the number of all those unaffected ones.

Remark 2.1: The goal of the set R_i is to avoid calculation of every C_r. It is used mainly for speed-up, which can be significant for sparse matrixes $\boldsymbol{A}$, where matrix calculations are performed in loops and when using low-level programming languages (e.g., MatLab). The latter operates very fast with matrixes and has its own inner optimizer.

Sampling a random variable $\widetilde{X}_i$ from (20) using the Gibbs sampler is simple. In particular, for the Bernoulli case with $\boldsymbol{x} \in \{0, 1\}^n$, this can be done as follows. Generate $Y \sim \text{Ber}(1/2)$. If

$$\sum_{r \in R_i} C_r(x_1, \ldots, x_{i-1}, Y, x_{i+1}, \ldots, x_n) \geq \widehat{m}_{t-1}, \tag{21}$$

then set $\widetilde{X}_i = Y$; otherwise, set $\widetilde{X}_i = 1 - Y$.

3. UNIFORMITY RESULTS

Here, we demonstrate empirically the following:

1. The original MCMC (without splitting) fails to generate points uniformly distributed on discrete sets of type (1). As mentioned, we will demonstrate that not only does MCMC fail to generate uniform points on the set $\mathcal{X}^*$, but typically it samples only on some subset of $\mathcal{X}^*$ instead of the entire one.
2. The splitting Algorithm A.2 handles successfully the uniformity problem in the sense that it generates uniform points on the set $\mathcal{X}^*$.

We consider both issues separately.

3.1. MCMC Without Splitting

Our first model is the random 3-SAT model with the instance matrix ($A = 20 \times 80$) adapted from [11]. Table 1 presents the performance of the splitting Algorithm A.1 based on 10 independent runs for the 3-SAT problem with with $N = 1000$, rarity parameter $\rho = 0.1$, and burn-in parameter $b = 1$.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

TABLE 1. Performance of Splitting Algorithm A.1 for SAT Problem with Instance Matrix $\boldsymbol{A} = (20 \times 80)$

Run N_0	Iterations	$\widehat{\lvert\mathcal{X}^*\rvert}$	RE of $\widehat{\lvert\mathcal{X}^*\rvert}$	$\widehat{\lvert\mathcal{X}^*_{\text{dir}}\rvert}$	RE of $\widehat{\lvert\mathcal{X}^*_{\text{dir}}\rvert}$	CPU
1	10	14.612	0.026	15	0.000	5.143
2	10	14.376	0.042	15	0.000	5.168
3	10	16.304	0.087	15	0.000	5.154
4	10	19.589	0.306	15	0.000	5.178
5	10	13.253	0.116	15	0.000	5.140
6	10	17.104	0.140	15	0.000	5.137
7	10	14.908	0.006	15	0.000	5.173
8	10	13.853	0.076	15	0.000	5.149
9	10	18.376	0.225	15	0.000	5.135
10	10	12.668	0.155	15	0.000	5.156
Average	10	15.504	0.118	15.000	0.000	5.153

Here, we use the following notation:

1. $N_t^{(e)}$ and $N_t^{(s)}$ denote the actual number of elites and the one after screening, respectively.
2. m_t^* and m_{*t} denote the upper and the lower elite levels reached, respectively.
3. $\rho_t = N_t^{(e)}/N$ denotes the adaptive proposal rarity parameter.
4. $\widehat{\lvert\mathcal{X}^*\rvert}$ and $\widehat{\lvert\mathcal{X}^*_{\text{dir}}\rvert}$ denote the product and what is called the *direct* estimator. The latter counts all distinct values of $\boldsymbol{X}_i$, $i = 1, \ldots, N$, satisfying $S(\boldsymbol{X}_i) \geq m$; that is, it can be written as

$$\widehat{\lvert\mathcal{X}^*_{\text{dir}}\rvert} = \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i^{(d)}) \geq m\}}, \tag{22}$$

where $\boldsymbol{X}_i^{(d)} = \boldsymbol{X}_i$ if $\boldsymbol{X}_i \neq \boldsymbol{X}_j, \forall j = 1, \ldots, i-1$ and $\boldsymbol{X}_i^{(d)} = 0$, otherwise. For more details on $\widehat{\lvert\mathcal{X}^*_{\text{dir}}\rvert}$, see [12].

Table 2 presents the dynamics for one run of the splitting Algorithm A.1 for the same model.

To demonstrate that the original Gibbs sampler (without splitting) fails to allocate all 15 valid SAT assignments as per Table 2, we took 1 of these 15 points and ran it for a very long time (allowing 1,000,000 Gibbs steps). We found that depending on the initial point $\boldsymbol{X} \in \mathcal{X}^*$, the Gibbs sampler converges either to 6 or 9, that is, it is able to find only 6 or 9 points out of the 15.

It is interesting to note that a similar phenomenon occurs with the splitting Algorithm A.2 if instead of keeping all 15 elites $N_T^{(s)}$ for $m_T = m = 80$, we leave only one of them and then proceed with the Gibbs sampler for a long time. Clearly, setting $N_t^{(s)} = 1$ is exactly the same as dispensing with the splitting, (i.e., staying with the original Gibbs sampler).

A similar phenomenon was observed with some other SAT models: Starting Gibbs from a *single* point $\boldsymbol{X} \in \mathcal{X}^*$, it was able to generate points inside *only* some subsets

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

TABLE 2. Dynamics of Algorithm A.1 for SAT 20 × 80 Model.

t	$\widehat{\lvert\mathcal{X}^*\rvert}$	$\lvert\widehat{\mathcal{X}}^*_{\text{dir}}\rvert$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	1.59E+05	0	152	152	78	56	0.152
2	3.16E+04	0	198	198	78	74	0.198
3	8.84E+03	0	280	276	79	76	0.280
4	1.78E+03	3	201	190	80	77	0.201
5	229.11	6	129	93	80	78	0.129
6	15.580	15	68	15	80	79	0.068
7	15.580	15	1000	15	80	80	1.000
8	15.580	15	1000	15	80	80	1.000
9	15.580	15	1000	15	80	80	1.000
10	15.580	15	1000	15	80	80	1.000

of $\mathcal{X}^*$, rather than in the entire $\mathcal{X}^*$. In other words, the Gibbs sampler was stacked at some local minima.

We found that the picture changes dramatically if $\mathcal{X}^*$ is a nice continuous convex set, like that of linear constraints. In this case, starting from any $\boldsymbol{X} \in \mathcal{X}^*$ and running the Gibbs sampler alone for a long time, we are able to obtain uniform samples, (i.e., to pass the χ^2 test).

3.2. Uniformity of the Splitting Method

To get uniform samples on $\mathcal{X}^*$, we modify Algorithm A.2 as follows. Once it reaches the desired level $m_T = m$, we perform several more steps (burn-in periods) with the corresponding elite samples; that is, we use the screening and splitting (cloning) steps exactly as we did for $m_t < m$. Clearly, this will only improve the uniformity of the samples of the desired set $\mathcal{X}^*$.

Observe also that the number of additional steps k needed for the resulting sample to pass the χ^2 test depends on the quality of the original elite sample at level m, which in turn depends on the values of ρ and b set in Algorithm A.2. We found numerically that the closer ρ is to 1 the more uniform is the sample. However, clearly, running the splitting Algorithm A.2 at ρ close to 1 is time-consuming. Thus, there exists a trade-off between the quality (uniformity) of the sample and the number of additional iterations k required.

Consider again the 3-SAT problem with the instance matrix $A = (20 \times 80)$ and $|\mathcal{X}^*| = 15$ (see Table 1). Figure 1 presents the histogram for $k = 0$, $N = 100$, $\rho = 0.05$, and $b = 1$. We found that the corresponding sample passes the χ^2 test, with $\chi^2 = 12.8333$. With $\rho > 0.05$, instead of $\rho = 0.05$ and with $k > 0$ instead of $k = 0$, we found that χ^2 was even smaller as expected. In particular, $\rho = 0.5$ and $k = 1$, we found that $\chi^2 = 9.7647$ and $\chi^2 = 10.3524$, respectively.

Note again that using a single elite $\boldsymbol{X} \in \mathcal{X}^*$, [i.e., using one of the 15 points (valid SAT assignments)], the Gibbs sampler was unable to find all of them. More precisely, depending on the initial value of the point $\boldsymbol{X} \in \mathcal{X}^*$, Algorithm A.2 was stacked at a

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

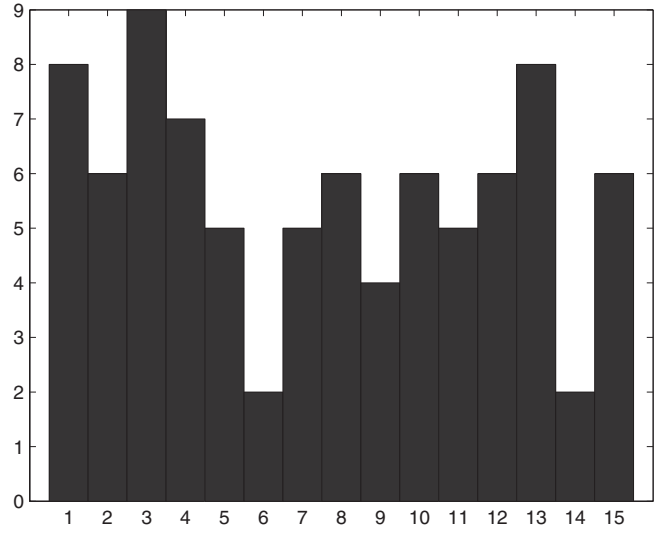

FIGURE 1. Histogram for the 3-SAT problem with the instance matrix $A = (20 \times 80)$ for $b = 1, k = 0, N = 100$, and $\rho = 0.05$.

local extremum, delivering as an estimator of $|\mathcal{X}^*|$ either 6 or 9 instead of the true value $|\mathcal{X}^*| = 15$.

We checked the uniformity for many SAT problems and found that typically (about 95%) the resulting sample passes the χ^2 test, provided we set $k = 2, 3$; that is, we perform two to three more steps (burn-in) with the corresponding elite sample after reaching the desired level m.

4. CONCLUSION AND FURTHER RESEARCH

In this article we showed the following empirically:

1. In spite of the common consensus on MCMC as a universal tool for generating samples on complex sets, we show that this is not the case when one needs to generate points uniformly distributed on discrete sets, such as on that defined in (1) (i.e., one containing the constraints of integer programming). We have demonstrated empirically that not only does the original MCMC fail to generate uniform points on the set $\mathcal{X}^*$, but typically it generates points only at some subset of $\mathcal{X}^*$ instead on the entire set $\mathcal{X}^*$.
2. In contrast to the classic MCMC, the splitting Algorithm A.2, which represents a combination of MCMC with a specially designed splitting mechanism, handles efficiently the uniformity problem in the sense that, under some mild requirements, the generated samples pass the χ^2 test.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

We intend to present rigorous statistical treatment of the splitting Algorithm A.2, and, in particular, find the associated parameters N, ρ, b, and η ensuring that generated samples are uniformly distributed on different discrete sets $\mathcal{X}^*$.

Acknowledgment

This research was supported by the BSF (Binational Science Foundation), grant No. 2008482.

References

1. Asmussen, S. & Glynn, P.W. (2007). *Stochastic simulation: Algorithms and analyses*. New York: Springer.
2. Botev, Z.I. & Kroese, D.P. (2008). An efficient algorithm for rare-event probability estimation, combinatorial optimization, and counting. *Methodology and Computing in Applied Probability* 10: 471–505.
3. Garvels, M.J.J. (2000). The splitting method in rare-event simulation. Ph.D. thesis, University of Twente.
4. Garvels, M.J.J. & Rubinstein, R.Y. (2000). A combined splitting–cross entropy method for rare event probability estimation of single queues and ATM networks. Unpublished manuscript
5. Lagnoux-Renaudie, A. (2009). A two-steps branching splitting model under cost constraint to. *Journal of Applied Probability* 46: 429–452.
6. Melas, V.B. (1997). On the efficiency of the splitting and roulette approach for sensitivity analysis. Winter Simulation Conference, Atlanta, GA, pp. 269 – 274.
7. Gryazina, E. & Polyak, B. (2010). Randomized methods based on new Monte Carlo schemes for control and optimizations. *Annals of Operations Research*.
8. L'Ecuyer, P., Demers, V. & Tuffin, B. (2007). Rare-events, cloning, and quasi-Monte Carlo. *ACM Transactions on Modeling and Computer Simulation*, 17(2).
9. Ross, S.M. (2006). *Simulation*. New York: Wiley.
10. Rubinstein, R.Y. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability* 1: 127–190.
11. Rubinstein, R.Y. (2008). The Gibbs cloner for combinatorial optimization, counting and sampling. *Methodology and Computing in Applied Probability* 11: 491–549.
12. Rubinstein, R.Y. (2010). Randomized algorithms with splitting: Why the classic randomized algorithms do not work and how to make them work. *Methodology and Computing in Applied Probability* 12: 1–41.
13. Rubinstein, R.Y. & Kroese, D.P. (2004). *The cross-entropy method: A unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. New York: Springer.
14. Rubinstein, R.Y. & Kroese, D.P. (2007). *Simulation and the Monte Carlo method*; 2nd ed. New York: Wiley.
15. Smith, R.L. (1984). Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32: 1296–1308.

APPENDIX

Splitting Algorithms

Following [12], we present the two versions of the splitting algorithm: the so-called *basic* version and the *enhanced* version, bearing in mind Example 1.1.

A.1. BASIC SPLITTING ALGORITHM

Let N, ρ_t, and N_t be the fixed sample size, the adaptive rarity parameter, and the number of elite samples at iteration t, respectively (see [12] for details). Recall that the elite sample $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ corresponds to the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_t}\}$, for which $S(\boldsymbol{X}_i) \geq \widehat{m}_t$; that is, $\widehat{m}_t$ is the $(1-\rho_t)$ sample quantile of the ordered statistics values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$. It follows that the number of elites $N_t = \lceil N\rho_t \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the largest integer.

In the basic version at iteration t, we *split* each elite sample $\eta_t = \left\lceil \rho_t^{-1} \right\rceil$ times. By doing so, we generate $\left\lceil \rho_t^{-1} N_t \right\rceil \approx N$ new samples for the next iteration $t+1$. The rationale is based on the fact that if all ρ_t are not small, say $\rho_t \geq 0.01$, we have enough stationary elite samples and all the Gibbs sampler has to do is to continue with these N_t elites and generate N *new* stationary samples for the next level.

Algorithm A.1 (Basic Splitting Algorithm for Counting)
Given the initial parameter ρ_0, say $\rho_0 \in (0.01, 0.25)$, and the sample size N, say $N = nm$, execute the following steps:

1. **Acceptance–Rejection.** Set a counter $t = 1$. Generate a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ uniformly on $\mathcal{X}_0$. Let $\widehat{\mathcal{X}}_0 = \{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_0}\}$ be the elite samples. Take

$$\widehat{c}_0 = \widehat{\ell}(\widehat{m}_0) = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_0\}} = \frac{N_0}{N} \tag{A.1}$$

   as an *unbiased* estimator of c_0. Note that $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_0} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$, where $g^*(\boldsymbol{x}, \widehat{m}_0)$ is a *uniform distribution* on the set $\mathcal{X}_1 = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_0\}$.
2. **Splitting.** Let $\widehat{\mathcal{X}}_{t-1} = \{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ be the elite sample at iteration $(t-1)$, (i.e., the subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$ for which $S(\boldsymbol{X}_i) \geq \widehat{m}_{t-1}$). Reproduce $\eta_{t-1} = \left\lceil \rho_{t-1}^{-1} \right\rceil$ times each vector $\widehat{\boldsymbol{X}}_k = (\widehat{X}_{1k}, \ldots, \widehat{X}_{nk})$ of the elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$; that is, take η_{t-1} identical copies of each vector $\widehat{\boldsymbol{X}}_k$. Denote the entire new population ($\eta_{t-1} N_{t-1}$ cloned vectors plus the original elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{\mathrm{cl}} = \{(\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_1), \ldots, (\widehat{\boldsymbol{X}}_{N_{t-1}}, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each cloned vector of the population $\mathcal{X}_{\mathrm{cl}}$ apply MCMC (and, in particular, the random Gibbs sampler) for a single period (single burn-in). Denote the *new entire* population by $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$. Note that each vector in the sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is distributed $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$, where $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$ has approximately a uniform distribution on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_{t-1}\}$.
3. **Estimating** c_t**.** Take $\widehat{c}_t = (N_t/N)$ (see (10)) as an estimator of c_t in (8). Note again that each vector of $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ of the elite sample is distributed $g^*(\boldsymbol{x}, \widehat{m}_t)$, where $g^*(\boldsymbol{x}, \widehat{m}_t)$ has approximately a uniform distribution on the set $\mathcal{X}_{t+1} = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_t\}$.
4. **Stopping rule.** If $m_t = m$, go to part 5; otherwise, set $t = t+1$ and repeat from part 2.
5. **Final Estimator.** Deliver $\widehat{\ell}$ given in (9) as an estimator of ℓ and $|\widehat{\mathcal{X}}^*| = \widehat{\ell}|\mathcal{X}|$ as an estimator of $|\mathcal{X}^*|$.

Figure A.1 presents a typical dynamics of the splitting algorithm, which terminates after two iterations. The set of points denoted $\star$ and $\bullet$ is associated with these two iterations. In particular the points marked $\star$ are uniformly distributed on the sets $\mathcal{X}_0$ and $\mathcal{X}_1$ (those in $\mathcal{X}_1$

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

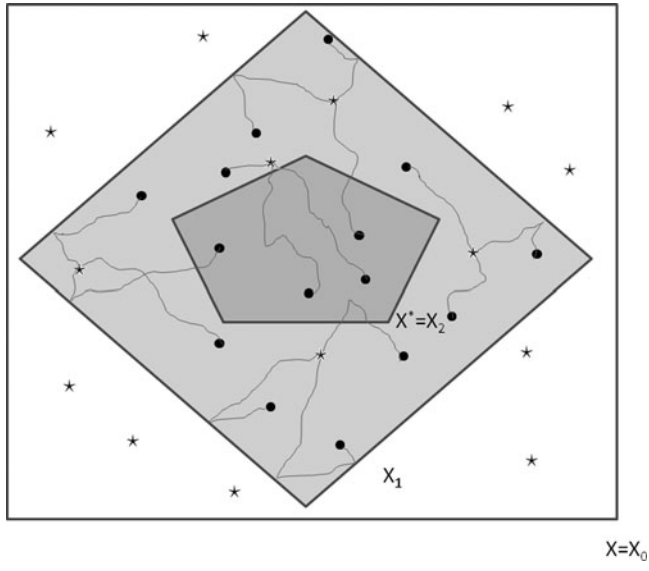

FIGURE A.1. Dynamics of Algorithm A.1

correspond to the elite samples). The points marked • are approximately uniformly distributed on the sets $\mathcal{X}_1$ and $\mathcal{X}_2$ (those in $\mathcal{X}_2 = \mathcal{X}^*$ correspond to the elite samples).

A.2. ENHANCED SPLITTING ALGORITHM FOR COUNTING

The improved version of the basic splitting Algorithm A.1, which contains (i) an enhanced splitting step instead of the original one as in Algorithms A.1 and (ii) a new screening step.

(i) Enhanced Splitting Step. Denote by η_t the number of times each of the N_t elite samples is reproduced at iteration t and call it the *splitting parameter*. Denote by b_t the *burn-in parameter*, (i.e., the number of times each elite sample has to follow through the MCMC (Gibbs) sampler). The purpose of the enhanced splitting step is to find a good balance, in terms of bias variance of the estimator of $|\mathcal{X}^*|$, between η_t and b_t, provided the number of samples N is given.

Let us assume for a moment that $b_t = b$ is fixed. Then for fixed N, we can define the adaptive *cloning* parameter η_{t-1} at iteration $t-1$ as follows

$$\eta_{t-1} = \left\lceil \frac{N}{bN_{t-1}} \right\rceil - 1 = \left\lceil \frac{N_{\mathrm{cl}}}{N_{t-1}} \right\rceil - 1. \qquad \textbf{(A.2)}$$

Here, $N_{\mathrm{cl}} = N/b$ is called the *cloned sample size*, and as earlier, $N_{t-1} = \rho_{t-1}N$ denotes the number of elites and ρ_{t-1} is the adaptive rarety parameter at iteration $t-1$ (see [14] for details).

As an example, let $N = 1000$, and $b = 10$. Consider two cases: $N_{t-1} = 21$ and $N_{t-1} = 121$. We obtain $\eta_{t-1} = 4$ and $\eta_{t-1} = 0$ (no splitting), respectively.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press

As an alternative to (A.2) one can use the following heuristic strategy in defining b and η: Find b_{t-1} and η_{t-1} from $b_{t-1}\eta_{t-1} \approx N/N_{t-1}$ and take $b_{t-1} \approx \eta_{t-1}$. In short, one can take

$$b_{t-1} \approx \eta_{t-1} \approx \left(\frac{N}{N_{t-1}}\right)^{1/2}. \tag{A.3}$$

Consider again the same two cases for N_{t-1} and N We have $b_{t-1} \approx \eta_{t-1} = 7$ and $b_{t-1} \approx \eta_{t-1} = 3$, respectively. We found numerically that both versions work well, but unless stated otherwise, we will use (A.3).

(ii) Screening Step. Since the IS p.d.f. $g^*(\boldsymbol{x}, m_t)$ must be *uniformly distributed* for each fixed m_t, the splitting algorithm checks at each iteration whether *all elite vectors* $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ *are different*. If this is not the case, we screen out (eliminate) all redundant elite samples. We denote the resulting elite sample as $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ and call it the *screened elite sample*. Note that this procedure prevents (at least partially) deviation of the empirical p.d.f. associated with $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ from the uniform.

Algorithm A.2 (Enhanced Splitting Algorithm for Counting)
Given the parameter ρ, say $\rho \in (0.01, 0.25)$ and the sample size N, say $N = nm$, execute the following steps:

1. **Acceptance–Rejection.** Same as in Algorithm A.1.
2. **Screening.** Denote the elite sample obtained at iteration $(t-1)$ by $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$. Screen out the redundant elements from the subset $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ and denote the resulting (reduced) one as $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$.
3. **Splitting (Cloning).** Given the size N_{t-1} of the screened elites $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ at iteration $(t-1)$, find the splitting and the burn-in parameters η_{t-1} and b_{t-1}, respectively, according to (A.3). Reproduce η_{t-1} times each vector $\widehat{\boldsymbol{X}}_k = (\widehat{X}_{1k}, \ldots, \widehat{X}_{nk})$ of the screened elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$; that is, take η_{t-1} identical copies of each vector $\widehat{\boldsymbol{X}}_k$ obtained at the $(t-1)$st iteration. Denote the entire new population ($\eta_{t-1}N_{t-1}$ cloned vectors plus the original screened elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{\mathrm{cl}} = \{(\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_1), \ldots, (\widehat{\boldsymbol{X}}_{N_{t-1}}, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each of the cloned vectors of the population $\mathcal{X}_{\mathrm{cl}}$, apply the Gibbs sampler for b_{t-1} burn-in periods. Denote the *new entire* population by $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$. Note that each vector in the sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is distributed approximately $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$, where $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$ is a uniform distribution on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_{t-1}\}$.
4. **Estimating c_t.** Same as in Algorithm A.1.
5. **Stopping Rule.** Same as in Algorithm A.1.
6. **Final Estimator.** Same as in Algorithm A.1.

https://doi.org/10.1017/S0269964810000057 Published online by Cambridge University Press