Randomized Algorithms with Splitting: Why the Classic Randomized Algorithms do not Work and how to Make them Work

Reuven Rubinstein

Faculty of Industrial Engineering and
Technion, Israel Institute of Technology,
Haifa , Israel

ierrr01@ie.technion.ac.il

iew3.technion.ac.il:8080/ierrr01.phtml

March 27, 2009

Abstract

We show that the *original classic randomized algorithms* for approximate counting in NP-hard problems, like for counting the number of satisfiability assignments in a SAT problem, counting the number of feasible colorings in a graph and calculating the permanent, typically fail. They either do not converge at all or are heavily biased (converge to a local extremum). Exceptions are convex counting problems, like estimating the volume of a convex polytope. We also show how their performance could be dramatically improved by combining them with the classic *splitting* method, which is based on simulating simultaneously multiple Markov chains. We present several algorithms of the combined version, which we simple call the *splitting algorithms*. We show that the most advance splitting version coincides with the *cloning* algorithm suggested earlier by the author. As compared to the randomized algorithms, the proposed splitting algorithms require very little warm-up time while running the MCMC from iteration to iteration, since the underlying Markov chains are already in steady-state from the beginning. What required is only fine tuning, i.e. keeping the Markov chains in steady-state while moving from iteration to iteration. We present extensive simulation studies with both the splitting and randomized algorithms for different NP-hard counting problems.

Keywords. Combinatorial Optimization, Counting, Cross-Entropy, Gibbs Sampler, Importance Sampling, Rare-Event, Randomized Algorithms, Splitting.

[0†] This research was supported by the BSF (Binational Science Foundation, grant No 2008482)

Contents

1 Introduction: Randomized Algorithms for Counting

In this work we show that the *original classic randomized algorithms* for approximate counting in NP-hard problems (Mitzenmacher and Upfal, 2005; Motwani and Raghavan, 1997) typically fail. We also show how their performance could be dramatically improved by combining them with the classic *splitting* method (Garvels, 2000), which is based on simulating simultaneously multiple Markov chains. We present several algorithms of the combined version and we show that the most advance algorithm coincides with the *cloning* algorithm suggested earlier in Rubinstein (2008).

Below we present some background on randomized algorithms. The main idea of *randomized algorithms* for counting (Mitzenmacher and Upfal, 2005; Motwani and Raghavan, 1997) is to design a sequential sampling plan, with a view to decomposing a "difficult" counting problem defined on the 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, randomized algorithms explore the connection between counting and sampling problems and in particular the reduction from approximate counting of a discrete set to approximate sampling of elements of this set, where the sampling is performed by the classic MCMC method (Rubinstein and Kroese, 2007). A typical randomized 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 $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ such that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$, $|\mathcal{X}_m| = |\mathcal{X}^*|$ and $|\mathcal{X}_0|$ is known.

3. Write $|\mathcal{X}^*| = |\mathcal{X}_m|$ as
$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}. \tag{1}$$
Note that the quantity
$$\ell = \frac{|\mathcal{X}^*|}{|\mathcal{X}_0|}$$
is very small, like $\ell = 10^{-100}$, while each ratio
$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \tag{2}$$
should not be small, like $c_t = 10^{-2}$ or greater. As we shall see below in typical applications such c_t will be available. 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 for each $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}|$.

5. Estimate $|\mathcal{X}^*|$ by
$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0| \prod_{t=1}^{m} \widehat{c}_t = \prod_{t=1}^{m} \frac{|\widehat{\mathcal{X}}_t|}{|\widehat{\mathcal{X}}_{t-1}|}, \tag{3}$$
where $|\widehat{\mathcal{X}}_t|,\ t = 1, \ldots, m$ is an estimator of $|\mathcal{X}_t|$, $\widehat{c}_t = \frac{|\widehat{\mathcal{X}}_t|}{|\widehat{\mathcal{X}}_{t-1}|}$, and similarly for the rare-event probability ℓ.

Algorithms based on the sequential Monte Carlo sampling estimator (3) are called in computer literature (Mitzenmacher and Upfal, 2005; Motwani and Raghavan, 1997), *randomized algorithms*. We shall call them simply, the *RAN* algorithms.

It is readily seen that in order to deliver 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 (1) 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) Present a low variance unbiased estimator of each $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}|$, such that the resulting estimator of ℓ is of low variance and unbiased.

We shall see below that task (i) is not difficult and shall proceed with it in this section. Task (ii) is quite complicated and is associated with uniform sampling separately at each sub-region $\mathcal{X}_t$. This will be done by combining the Gibbs sampler with the classic splitting method (Garvels, 2000) and will be considered in the subsequent sections. Note that some alternative MCMC samplers and in particular the *hit-and-run* sampler can be used as well.

It readily follows that as soon as both tasks (i) and (ii) are resolved one can obtain an efficient estimators for each c_t, and, thus a low variance estimator $\widehat{|\mathcal{X}^*|}$ in (3). We therefore proceed with task (i) by considering several well- known NP-hard counting problems.

Example 1.1 (Independent Sets) Consider a graph $G = (V, E)$ with m edges and n vertices. Our goal is to count the number of independent node (vertex) sets of this graph. A node set is called *independent* if no two nodes are connected by an edge, that is, no two nodes are adjacent; see Figure 1 for an illustration of this concept.

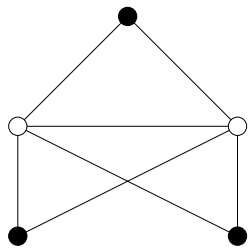

Figure 1: The black nodes form an independent set since they are not adjacent to each other.

Consider an arbitrary ordering of the edges. Let E_j be the set of the first j edges and let $G_j = (V, E_j) = (V, \{e_1, \ldots, e_j\})$ be the associated sub-graph. Note that $G_m = G$, and that G_{j+1} is obtained from G_j by adding the edge e_{j+1}, which is not in G_j. Denoting by $\mathcal{X}_j$ the set of independent sets of G_i we can write $|\mathcal{X}^*| = |\mathcal{X}_m|$ in the form (1). Here $|\mathcal{X}_0| = 2^n$, since G_0 has no edges and thus every subset of V is an independent set, including the empty set. Note that here $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$.

Example 1.2 (Vertex Coloring) Given a graph $G = (V, E)$ with m edges and n vertices, color the vertices of V with given q colors, such that for each edge $(i, j) \in E$, vertices i and j have different colors. Note that in an optimization problem one has to find the minimum number of colors to color the vertices of V such that for each edge $(i, j) \in E$, vertices i and j have different colors. The procedure for vertex coloring while applying the randomized algorithm is the same as for independent

sets. Indeed, we again consider an arbitrary ordering of the edges. Let E_j be the set of the first j edges and let $G_j = (V, E_j)$ be the associated sub-graph. Note that $G_m = G$, and that G_{j+1} is obtained from G_j by adding the edge e_{j+1}. Denoting by $|\mathcal{X}_i|$ the cardinality of the set $\mathcal{X}_i$ corresponding to G_i we can write again $|\mathcal{X}^*| = |\mathcal{X}_m|$ in the form (1), where $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$. Here $|\mathcal{X}_0| = q^n$, since G_0 has no edges.

Again, application of the splitting algorithm for the vertex coloring problem is similar to independent set and thus, to multiple events. The parameters n and m are the numbers of vertices and nodes in the graph G, respectively.

Example 1.3 (Hamiltonian Cycles) Given a graph $G = (V, E)$ with m edges, each of length 1 and n vertices, find all Hamiltonian cycles, that is, those corresponding to the tours of length n.

Figure 2 presents a graph with 8 nodes and several Hamiltonian cycles, one of which is marked in bold lines.

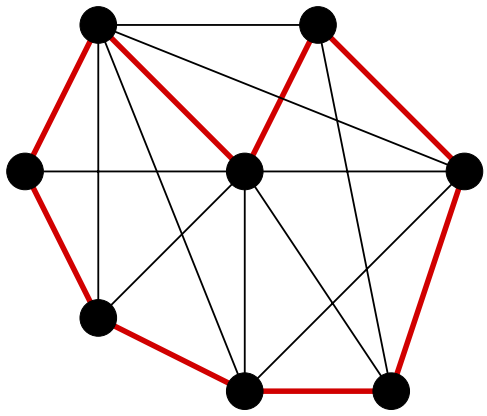

Figure 2: A Hamiltonian graph. The bold edges form a Hamiltonian cycle.

The procedure for Hamiltonian cycles is similar to independent sets. As before, consider an arbitrary ordering of the edges of length 0, that is, those without connection between the vertices. Define a sub-graph $G_j = (V, E_j)$, where the first j edges of length 0 remain the same while the other $n(n-1)/(2-m-j)$ edges of length 0 are replaced by 1. Note that $G_m = G$, and G_{j+1} is obtained from G_j by replacing 1 by 0 for the edge e_{j+1}. Denoting by $|\mathcal{X}_i|$ the cardinality of the set $\mathcal{X}_i$ corresponding to G_i we can write again $|\mathcal{X}^*| = |\mathcal{X}_m|$ in the form (1), where $\mathcal{X} = (n-1)!$

Example 1.4 (Knapsack Problem) Given items of sizes $a_1, \ldots, a_m > 0$ and a positive integer $b \geq \min_i a_i$, find the numbers of vectors $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$, such that

$$\sum_{i=1}^{n} a_i\, x_i \leq b.$$

The integer b re-presents the size of the knapsack, and x_i indicates whether or not item i is put into the knapsack. Let $\mathcal{X}^*$ denote the set of all feasible solutions, that is, all different combinations of items that can be placed into the knapsack without exceeding its capacity. The goal is to determine $|\mathcal{X}^*|$.

To put the knapsack problem into the framework (1), assume without loss of generality that $a_1 \leq a_2 \leq \cdots \leq a_n$ and define $b_j = \sum_{i=1}^{j} a_i$, with $b_0 = 0$. Denote by $\mathcal{X}_j$ the set of vectors $\boldsymbol{x}$ that satisfy $\sum_{i=1}^{n} a_i\, x_i \leq b_j$, and let m be the largest integer such that $b_m \leq b$. Clearly, $\mathcal{X}_m = \mathcal{X}^*$. Thus, (1) is established again.

Example 1.5 (Counting the Permanent) The permanent of a general $n \times n$ binary matrix $A = (a_{ij})$ is defined as

$$\text{per}(A) = |\mathcal{X}^*| = \sum_{\boldsymbol{x} \in \mathcal{X}} \prod_{i=1}^{n} a_{ix_i}, \tag{4}$$

where $\mathcal{X}$ is the set of all permutations $\boldsymbol{x} = (x_1, \ldots, x_n)$ of $(1, \ldots, n)$. It is well-known that calculation of the permanent of a *binary* matrix is equivalent to the calculation of the number of perfect matchings in a certain bipartite graph. A *bipartite graph* $G(V, E)$ is a graph in which the node set V is the union of two disjoint sets V_1 and V_2, and in which each edge joins a node in V_1 to a node in V_2. A *matching* of size m is a collection of m edges in which each node occurs at most once. A *perfect matching* is a matching of size n.

To see the relation between the permanent of a binary matrix $A = (a_{ij})$ and the number of perfect matchings in a graph, consider the bipartite graph $G = (V, E)$ where V_1 and V_2 are disjoint copies of $\{1, \ldots, n\}$, and $(i, j) \in E$ if and only if $a_{ij} = 1$, for all i and j. As an example, let A be the 3×3 matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}. \tag{5}$$

The corresponding bipartite graph is given in Figure 3. The graph has 3 perfect matchings, one of which is displayed in the figure. These correspond to all permutations $\boldsymbol{x}$ for which the product $\prod_{i=1}^{n} a_{ix_i} = 1$.

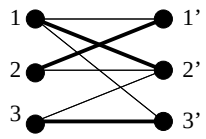

Figure 3: A bipartite graph. The bold edges form a perfect matching.

For a general binary matrix A let $\mathcal{X}_i$ denote the set of matchings of size i in the corresponding bipartite graph G. Assume that $\mathcal{X}_n$ is non-empty, so that G has a perfect matching of nodes V_1 and V_2. We are interested in calculating $|\mathcal{X}_n| = \text{per}(A)$. Taking into account that $|\mathcal{X}_0| = |E|$ we obtain (1).

Let us write $|\mathcal{X}^*|$ as

$$|\mathcal{X}^*| = \ell(m)|\mathcal{X}|,$$

where as before $\ell(m) = \frac{|\mathcal{X}^*|}{|\mathcal{X}|}$ is the rare-event probability. It can be also written as

$$\ell(m) = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) \geq m\}} \right]. \tag{6}$$

Here $\boldsymbol{X} \sim f(\boldsymbol{x})$, $f(\boldsymbol{x})$ is a uniform distribution on the set of points of $\mathcal{X}$, as before, m is a fixed parameter, like the total number of edges in a coloring graph, and $S(\boldsymbol{X})$ is the sample performance.

To proceed, write $\ell(m)$ similar to (1) as

$$\ell(m) = c_0 \prod_{t=1}^{T} c_t, \tag{7}$$

where $c_0 = \mathbb{E}_f[I_{\{S(\boldsymbol{X}) \geq m_0\}}]$ and, as before

$$c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}| = \mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X}) \geq m_{t-1}\}}]. \tag{8}$$

Here

$$g^*_{t-1} = g^*(\boldsymbol{x}, m_{t-1}) = \ell(m_{t-1})^{-1} f(\boldsymbol{x}) I_{\{S(\boldsymbol{x}) \geq m_{t-1}\}}, \tag{9}$$

$\ell(m_{t-1})^{-1}$ is the normalization constant and similar to (1) the sequence $m_t,\ t = 0, 1, \ldots, T$ represents a fixed grid satisfying $-\infty < m_0 < m_1 < \cdots < m_T = m$. Note that in contrast to (1) we use in (7) a product of T terms instead of m terms. As we shall see below, one can typically make $T \leq m$. In addition, T might be a random variable. The later case is associated with adaptive choice of the level sets $\{\widehat{m}_t\}_{t=0}^T$. This in turn involves an additional parameter denoted by ρ, which is called the *rarity parameter*. For more details see (Botev and Kroese, 2008 and Rubinstein, 2008). Unless stated otherwise, we assume without loss of generality that $T = m$.

Since for counting problems the pdf $f(\boldsymbol{x})$ should be *uniformly* distributed on $\mathcal{X}$, which we denote by $\mathcal{U}(\mathcal{X})$, it follows from (9) that the pdf $g^*(\boldsymbol{x}, m_{t-1})$ should be *uniformly* distributed on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_{t-1}\}$, that is, $g^*(\boldsymbol{x}, m_{t-1})$ must be equal to $\mathcal{U}(\mathcal{X}_t)$. As mentioned, generating points uniformly distributed on each sub-region $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_{t-1}\}$ is one the main issues of this paper and will be addressed in the further sections.

Once sampling from $g^*_t = \mathcal{U}(\mathcal{X}_t)$ becomes feasible, the final estimator of $\ell(m)$ (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

$$\widehat{\ell}(m) = \prod_{t=1}^{T} \widehat{c}_t = \frac{1}{N^{T+1}} \prod_{t=0}^{T} N_t, \tag{10}$$

where

$$\widehat{c}_t = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq m_{t-1}\}} = \frac{N_t}{N}, \tag{11}$$

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

The estimator (10) is often called the *product estimator* (Mitzenmacher and Upfal, 2005).

We next show how to cast the problem of counting the number of feasible solutions of the set defined by integer programming constraints into the framework (6)-(9).

Example 1.6 Counting on the set of an integer programming constraints Consider a set $\mathcal{X}^*$ containing both equality and inequality constraints of an integer program, that is,

$$\begin{aligned} &\textstyle\sum_{k=1}^n a_{ik} x_k = b_i,\ i = 1, \ldots, m_1, \\ &\textstyle\sum_{k=1}^n a_{jk} x_k \geq b_j,\ j = m_1 + 1, \ldots, m_1 + m_2, \\ &\boldsymbol{x} = (x_1, \ldots, x_n) \geq \boldsymbol{0}, \quad x_k \text{ is integer } \forall k = 1, \ldots, n. \end{aligned} \tag{12}$$

Our goal is to count the number of feasible points of the set (12). We assume that each component $x_k,\ k = 1, \ldots, n$ may have d different values, labeled $1, \ldots, d$. Note that the SAT problem represents a particular case of (12) with inequality constraints and where $x_1, \ldots, x_n$ are binary components.

It is easy to show (see Rubinstein, 2008) that in order to count the number of points of the set (12) one can associate it with the following rare-event probability problem

$$\ell = \mathbb{E}_f \left[I_{\{\sum_{i=1}^m C_i(\boldsymbol{X}) = m\}} \right], \tag{13}$$

where the first m_1 terms $C_i(\boldsymbol{X})$'s in (13) are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^n a_{ik} X_k = b_i\}},\ i = 1, \ldots, m_1, \tag{14}$$

while the remaining m_2 ones are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^{n} a_{ik} X_k \geq b_i\}}, \; i = m_1 + 1, \ldots, m_1 + m_2. \tag{15}$$

Thus, in order to count the number of feasible solution on the set (12) one can consider an associated rare-event probability estimation problem (13) involving a *sum of dependent Bernoulli random variables* C_i $i = m_1 + 1, \ldots, m$. A rare-event probability estimation framework similar to (13) can be readily established for all the above NP-hard counting problems. It follow from the above while using the product estimator (3) for counting on $\mathcal{X}^*$ our main goal will be still estimating efficiently the rare event probability ℓ in (13).

In this paper we show how using the above product estimators for ℓ and $|\mathcal{X}^*|$ to improve the performance of the classic *randomized algorithms* for counting in NP-hard problems, like counting the number of satisfiability assignments in a SAT problem, counting the number of valid colorings in a graph and calculating the permanent, by combining them with the classic *splitting* method. We present several algorithms of the combined versions, which we simply call the *splitting* algorithms. We show that the most advanced splitting version coincides with the *cloning* algorithm suggested earlier by the author (Rubinstein, 2008).

The basic idea of splitting is due to Kahn and Harris (1951). The idea is to partition the state-space of the system into a series of nested subsets and to consider the rare-event as the intersection of a nested sequence of events. When a given subset is entered by a sample trajectory during the simulation, numerous random retrials are generated with the initial state for each retrial being the state of the system at the entry point. By doing so, the system trajectory is split into a number of new sub-trajectories, hence the name splitting. For references on the splitting method see (Frederic Cerou, et al (2005), Del Moral (2004), Garvels and D.P. Kroese (1998), Garvels, Kroese and van Ommeren (2000), Garvels (2000), Glasserman, et al (1997), (1999) and Del Moral, (2004). For a nice recent paper on splitting see L'Ecuyer, Demers, and Tuffin (2007).

We show numerically that

1. For counting in NP-hard problems the original randomized algorithms typically fail. They either do not converge at all or are heavily biased (converge to a local extremum). Exceptions are convex counting problems, like estimating the volume of a convex plytope. By contrast, the splitting algorithm, which simulates simultaneously multiple Markov chains, performs nicely, that is, typically converges to the true counting quantity.

2. As compared to the randomized algorithms, the proposed splitting algorithms require very little warm-up time while running the MCMC from iteration to iteration, since the underlying Markov chains are already in steady-state just from the beginning. What required is only fine tuning, i.e. keeping the Markov chains in steady-state while moving from iteration to iteration. As a result we obtain dramatic variance reduction and thus, dramatic speedup compared to the randomized algorithms.

The remaining sections are organized as follows. In Section 2 we discuss application of the Gibbs sampler for generating samples uniformly distributed on the subspaces $\mathcal{X}_j, \; j = 1, \ldots m$. In Section 3 we show how to combine the MCMC method and in particular the Gibbs sampler with the classical splitting method (Garvels, 2000) and in particular how to incorporate the splitting method into the randomized algorithms and substantially improve their performance. More precisely, we present several combined versions of randomized algorithms with splitting, one of

which coincides with the *cloning* algorithm introduced. Section 4 represents numerical results with the original randomized algorithms and several of its combined versions with splitting, including the cloning algorithm. As mentioned, we show numerically that the original randomized algorithms typically fail; they are either trapped at some local extremum or, more often, stacked at some intermediate level m_t, that is, unable to deliver any meaningful estimator of $|\mathcal{X}^*|$. By contrast, the splitting algorithm delivers quite reliable estimators of the true global extremum. In Section 5 conclusions and some directions directions for further research are given. Finally, in Appendix some mathematical grounding is given of the complexity of the randomized algorithms (recapitulated from (Mitzenmacher and Upfal, 2005)) and of the product estimator (10)-(11).

2 Gibbs Sampler and Randomized Algorithms

In this section we show how to sample from a given joint pdf $g(x_1, \ldots, x_n)$ using Gibbs sampler. In the latter, instead of sampling from $g(x_1, \ldots, x_n)$ directly, which might be very difficult, one samples from the one-dimensional conditional pdfs $g(x_i|X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n)$, $i = 1, \ldots, n$, which is typically much simpler. Under some mild conditions (see Rubinstein and Kroese, 2007), the Gibbs sampler generates a sample distributed $g(x_1, \ldots, x_n)$. Two basic version of the Gibbs sampler (Rubinstein and Kroese, 2007) are available: *systematic* and *random*. In the former one the components of the vector $\boldsymbol{X} = (X_1, \ldots, X_n)$ are updated in a fixed, say increasing order, while in the latter, they are chosen randomly, that is, according to a discrete uniform n-point pdf. 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 $\sim g(\boldsymbol{x})$ as follows:

Algorithm 2.1 (Systematic Gibbs Sampler)

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

We denote for convenience each conditional pdf $g(x_i|\widetilde{X}_1, \ldots, \widetilde{X}_{i-1}, X_{i+1}, \ldots, X_n)$ as $g(x_i|\boldsymbol{x}_{-i})$, where $|\boldsymbol{x}_{-i}$ denotes conditioning on all random variables except the i-th component. Below we present a random Gibbs sampler taken from Ross, 2006) 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 (11), that is,

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

Algorithm 2.2 (Ross' Acceptance-Rejection Algorithm for Estimating c_t)

1. Set $N_t = N$=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=Int$(nU) + 1$.
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 4.

6. Set $N = N + 1$ and $Y_k = \widetilde{X}_k$.

7. If $S(\boldsymbol{x}) \geq m_t$, then $N_t = N_t + 1$.

8. Go to 3.

9. Estimate c_t as $\widehat{c}_t = \frac{N_t}{N}$.

Note that Algorithm 2.2 (see step 5) is based on the acceptance-rejection method. For many rare-events and counting problems, generation from the conditional pdf $g(x_i|\boldsymbol{x}_{-i})$ can be performed directly, that is, skipping step 5 in it. We present several relevant examples.

Example 2.1 Sum of Independent Random Variables

Consider estimation of ℓ with $S(\boldsymbol{x}) = \sum_{i=1}^n X_i$, that is,

$$\ell = \mathbb{E}_f \left[I_{\{\sum_{i=1}^n X_i \geq m\}} \right] . \tag{16}$$

In this case, generating random variables $X_i,\ i = 1, \ldots, n$ for a fixed value m can be easily performed by using the Gibbs sampler based on the following conditional pdf

$$g^*(x_i, m|\boldsymbol{x}_{-i}) = \propto f_i(x_i) I_{\{x_i \geq m - \sum_{j \neq i} x_j\}} , \tag{17}$$

where $\propto$ means proportional to.

Note also that each of the n conditional pdfs $g^*(x_i, m|\boldsymbol{x}_{-i})$ represents a truncated version of the proposed marginal pdf $f_i(x_i)$ with the truncating 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 truncated single dimensional pdf $g^*(x_i, m|\boldsymbol{x}_{-i})$ is easy and can be typically performed by the inverse-transform method.

Sampling a Bernoulli random variable $\widetilde{X}_i$ from (17) using the Gibbs sampler can be performed as follows. Generate $Y \sim$ Ber (p). If

$$Y \geq m - \sum_{j \neq i} X_j ,$$

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

Example 2.2 Counting on the set of an integer program: Example 1.6 continued

Consider the set (12). It readily follows (see also Rubinstein, 2008), that in order to count on the set (12) with given matrix $\boldsymbol{A} = \{a_{ij}\}$, one only needs to sample from the one-dimensional conditional pdfs

$$g^*(x_i, \widehat{m}_{t-1}|\boldsymbol{x}_{-i}) = \propto \mathcal{U}(1/d) I_{\{\sum_{r \in R_i} C_r(\boldsymbol{X}) \geq (\widehat{m}_{t-1} - c_{-i}) - \sum_{r \notin R_i} C_r(\boldsymbol{X})\}} , \tag{18}$$

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 that are affected by x_i, and c_{-i} counts the number of all those unaffected by x_i.

Remark 2.1 The goal of the set R_i is to avoid calculation of every C_r. It is used mainly for speedup. The speedup could be significant for sparse matrices $\boldsymbol{A}$, where matrix calculations are performed in loops and when one uses low level programming languages, unlike MatLab. The latter operates with matrices very fast and it has its own inner optimizer.

Sampling a random variable $\widetilde{X}_i$ from (18) using the Gibbs sampler is simple. In particular for the Bernoulli case with $\boldsymbol{x} \in \{0,1\}^n$ this can be performed as follows. Generate $Y \sim$ Ber (1/2). If

$$\sum_{r \in R_i} C_r(x_1, \ldots, x_{i-1}, Y, x_{i+1}, \ldots, x_n) \geq \widehat{m}_{t-1}, \tag{19}$$

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

Example 2.3 Vertex coloring: Example 1.2 continued It follows from Example 1.2 that the number of levels m required by for vertex coloring equals to the number of edges in the graph. Also note that in order to implement (3), a randomized algorithm assumes running a separate MCMC (Gibbs) sampler for each sub-graph G_j, $j = 1, \ldots, m$. The samples thus obtained must eventually (after some warm-up period) be distributed uniformly over the set $\mathcal{X}_j$ of all feasible states, that is, we must sample uniformly on all feasible sets of q colorings associated with the sub-graph G_j. In other words, in order to sample uniformly on $\mathcal{X}_j$ we need to construct an irreducible and aperiodic Markov chain (MC) with the state space defined on $\{\zeta \in [1, \ldots q]^n : \zeta \text{ is feasible}\}$, where n is the number of vertices.

The underlying Markov chain for generating proper coloring is very simple: at each step, choose a vertex $v \in V$ uniformly at random and also a color r uniformly at random. Recolor the v with r if the new coloring is proper, that is, if the two end points of every edge acquire two different colors.

More formally, the Gibbs sampler algorithm for proper q-coloring (see also Algorithm 2.2) can be written as follows.

Algorithm 2.3 Gibbs Sampler for Vertex Coloring

Let X_0 be an arbitrary q-coloring in G_j. To generate X_{t+1}

1. Choose a vertex $v \in V$ uniformly at random from the set of n vertices.
2. Choose X_{t+1} according to the uniform distribution over the set of colors that are not assigned at any neighbor of v.
3. Leave the colors unchanged at all other vertices, i.e., set $X_{t+1} = X_t$ for all vertices $w \in V$ except v.

Note that in contrast to Algorithm 2.2, the acceptance - rejection step is skipped in Algorithm 2.3.

Remark 2.2 For clarity of the presentation, we still provide details on how to estimate $c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{j-1}|}$ in (1), (8) for the vertex coloring problem. Denote by a_t and b_t the end vertices of the edge e_t, which is in G_t but not in G_{j-1}. By definition, $|\mathcal{X}_t|$ is the number of q- colorings of the sub-graph G_t. Note that the q- colorings of G_t are exactly those configurations $\zeta \in \{1, \ldots, q\}^n$ that are in q- colorings of G_{j-1} and that in addition satisfy $\zeta(a_t) \neq \zeta(b_t)$. Hence the ratio $\frac{|\mathcal{X}_t|}{|\mathcal{X}_{j-1}|}$ in (1) is exactly the proportion of q-colorings of G_t that satisfy $\zeta(a_t) \neq \zeta(b_t)$. In other words, $\frac{|\mathcal{X}_t|}{|\mathcal{X}_{j-1}|}$ equals the probability of a random coloring ζ of G_{j-1}, chosen according to the uniform distribution $\mathcal{U}_{G_{j-1}}$, satisfying $\zeta(a_t) \neq \zeta(b_t)$. The crucial point here is that each probability $c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{j-1}|}$ could be estimated separately and independently while running Algorithm 2.3 (for $\tau(\varepsilon)$ steps, see Theorem 6.2 of the Appendix) until it reaches steady-state, provided that we condition on the events $\zeta(a_t) \neq \zeta(b_t)$.

Similar Gibbs sampling algorithms can be written for some other combinatorial problems. In particular, for the independent sets problem we have (Mitzenmacher and Upfal, 2005)

Algorithm 2.4 Gibbs Sampler for Independent Sets

Let X_0 be an arbitrary independent set in G_j. To generate X_{t+1}

1. Choose a vertex $v \in V$ uniformly at random from the set of n vertices.
2. If $v \in X_t$, then $X_{t+1} = X_t \setminus \{v\}$.
3. If v is not in X_t and if adding v to X_t still preserves its independency, then $X_{t+1} = X_t \cup \{v\}$.
4. Otherwise, $X_{t+1} = X_t$.

Note that in Algorithm 2.4 the neighbors of a state X_t are all independent sets that differ from X_t by one vertex.

We call the algorithms, like the coloring Algorithm 2.3 and the independent sets Algorithm 2.4, where each c_t is estimated separately - the *RAN* algorithms. As will follow from our numerical results below the original RAN algorithms have two problems: (i) they are quite slow, since one needs to run the Gibbs sampler m times from scratch while estimating each $c_t,\ t = 0, 1, \ldots, m$; (ii) they are typically either stacked at some intermediate level m_t or generate a heavily biased estimator of $\mathcal{X}^*$ and thus converge to a local extremum.

We next proceed by introducing the original cloning algorithm from (Rubinstein, 2008), which overcomes the difficulties of RAN algorithms.

3 Randomized Algorithms Combined with Splitting

To speed up convergence and improve the statistical properties of the RAN algorithms, we shall combine them with the classic splitting algorithms (Asmussen and Glynn, 2007), Garvels, 2001). We call such combined algorithms *randomized splitting* or simply *splitting* algorithms.

In splitting algorithms one runs the MCMC sampler, like the Gibbs sampler or the hit-and-run one, by simulating simultaneously many Markov chains instead of a single one as in Algorithm 2.1. For details see Algorithm 3.1 below, where we use the Gibbs sampler. We found that for counting and optimization problems based on the rare event probabilities of type (13) the Gibbs sampler is very useful. Here we outline the main steps of the splitting algorithm.

1. We start by generating a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ uniformly on $\mathcal{X}_0$. We denote by $\widetilde{\mathcal{X}}_0 = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0}\}$ the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$ and call it the *elite* sample. The elite sample corresponds to the ordered statistics values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$ for which $S(\boldsymbol{X}_i) \geq \widehat{m}_0$. Here $\widehat{m}_0$ corresponds to the $(1-\rho)$ sample, ρ being the *rarity parameter*. Typically we set $0.01 \leq \rho \leq 0.25$. In short, $\widetilde{\mathcal{X}}_0 = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0}\}$ corresponds to the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$, for which $S(\boldsymbol{X}_i) \geq \widehat{m}_0$. It is crucial to note that the elite sample $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$, where $g^*(\boldsymbol{x}, \widehat{m}_0)$ is a *uniform* distribution on the set $\mathcal{X}_1 = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_0\}$. In other words, the elite sample $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$ can be viewed as the one in steady-state.
2. We split each elite sample $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0}$ η_0 times, that is, we take η_0 copies of each of them, with η chosen such that $\eta_0 N_0 \approx N$.
3. At the next iteration we apply the Gibbs Algorithms 2.1 for burn-in periods to each of the above $N \approx \eta_0 N_0$ samples. It is crucial to note that after applying the Gibbs sampler each vector in the *new* sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is distributed

approximately uniformly on the same set $\mathcal{X}_1 = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_0\}$. This is so since after the first iteration all elite samples $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0}$ are exactly in steady-state (uniformly distributed on $\mathcal{X}_1$), and by applying the Gibbs sampler (for one-two loops) will ensure that the new sample $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_N$ remains distributed nearly uniformly on $\mathcal{X}_1$.

4. We keep iterating as above (applying the Gibbs sampler simultaneously to N splitted elites) until convergence. We call this procedure, the *fine tuning* procedure. As mentioned, its goal is to keep the resulting sample $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_N$ at each iteration t in the steady-state.

Unless stated otherwise the reader should bear in mind either the coloring or the SAT problem.

We consider the following 4 splitting versions:

(i) Splitting of the sub-graphs $G_j,\ j = 1, \ldots, m$ with strictly fixed topology configurations and with levels $m_j,\ j = 1, \ldots, m$ strictly fixed in advance.

(ii) Splitting of the sub-graphs $G_j,\ j = 1, \ldots, m$ with strictly fixed topology configurations, but with adaptive levels $m_j,\ j = 1, \ldots, m$.

(iii) Splitting of the sub-graphs $G_j,\ j = 1, \ldots, m$ with adaptive choice of the topology configurations and with levels $m_j,\ j = 1, \ldots, m$ strictly fixed in advance.

(iv) Splitting of the sub-graphs $G_j,\ j = 1, \ldots, m$ with both adaptive choice of the topology configurations and of levels $m_j,\ j = 1, \ldots, m$.

We shall call the above four versions *Split1, Split2, Split3* and *cloning*, respectively. Investigation of their performance and their comparison with RAN is one of the main tasks in this work.

Note also that

- When we say that the sub-graphs $G_j,\ j = 1, \ldots, m$ are strictly fixed in advance, like in RAN, Split1 and Split2 - we mean that their corresponding edges $j = 1, \ldots, m$ (say, in the graph coloring problem) are *labeled in advance*. When we say that they are chosen adaptively, like in Split3 and in the cloning method, we mean that there is *no labeling* of the edges $j = 1, \ldots, m$ in G_j in advance, that is, we accept *all samples* from the sub-space $\mathcal{X}_j$ rather than those restricted by labeling.

  In short, in RAN, Split1 and Split2 the configurations G_j are fixed in advance, that is, G_j contains exactly j edges labeled in advance, while in Split3 and in cloning, the edges are not fixed (controlled) in advance. As we shall see below, labeled and unlabeled sub-graphs involve one of the crucial convergence issues.

- The number of levels (iterations) in Split1 and Split3 is fixed and equal to m, while in Split2 and cloning it is chosen adaptive and denoted by T. Clearly, in the latter cases typically $T < m$.

- Although the cloning version, where both the levels m_j and the sub-graphs G_j are chosen adaptively, is the most natural one - our numerical results suggest that Split3 also nicely converges to $|\mathcal{X}^*|$. We observed that the crucial issue for global convergence to $|\mathcal{X}^*|$ is *not the number of iterations* (fixed m versus adaptive m), but rather labeled G_j versus adaptive G_j. More on this below.

We shall also show below that

1. The RAN and Split1, Split2, Split3, can be viewed as particular cases of the cloning Algorithm 3.1 below.

2. The RAN algorithm can be viewed as a particular case of Split1, when the number of elite samples equal unity (no splitting).

3. By contrast to RAN, there is no need in any of the 4 splitting versions for a warm up (see, for example, Theorem 6.2 of Appendix), since it will follow directly from the cloning Algorithm 3.1 below that the underlying model is in *steady state* from the very first iteration. What is required from the 4 algorithms is fine tuning only, that is keeping the corresponding Markov chains (MC's) in steady state, while moving from iteration to iteration. As a result, we obtain dramatic variance reduction (speed up) while estimating $|\mathcal{X}^*|$ in (1).

4. With the algorithms arranged in the sequence

   RAN $\rightarrow$ Split1 $\rightarrow$ Split2 $\rightarrow$ Split3 $\rightarrow$ cloning

   the statistical properties of the counting estimators of $|\mathcal{X}^*|$ substantially improve. The main performance breakthrough occurs at the link Split2 $\rightarrow$ Split3.

We next present two versions of the cloning algorithm: the so-called *basic* version and the *enhanced* version.

3.1 Basic Cloning Algorithm

Let N, ρ_t and N_t be the fixed sample size, the adaptive rarity parameter (see Rubinstein, 2008 for details) and the number of elites at iteration t, respectively. Note that the number of elites equals $N_t = \lceil N\rho_t \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the largest integer.

In the basic version at iteration t we split each elite sample $\eta_t = \lceil \rho_t^{-1} \rceil$ times. By doing so we generate $\lceil \rho_t^{-1} N_t \rceil \approx N$ new samples for the next iteration $t+1$. Our 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 stationary N_t elites and generate N *new* stationary samples for the next level.

Algorithm 3.1 (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 $\widetilde{\mathcal{X}}_0 = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0}\}$ be the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$, the elite samples for which $S(\boldsymbol{X}_i) \geq \widehat{m}_0$, where $\widehat{m}_0$ is the $(1-\rho_0)$ sample quantile of the ordered statistics values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$. Take
$$\widehat{c}_0 = \widehat{\ell}(\widehat{m}_0) = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_0\}} = \frac{N_0}{N} \tag{20}$$
as an *unbiased* estimator of c_0. Note that $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$, 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 $\widetilde{\mathcal{X}}_{t-1} = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}}\}$ be the elite sample at iteration $(t-1)$, that is the subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$ for which $S(\boldsymbol{X}_i) \geq \widehat{m}_{t-1}$. Reproduce $\eta_{t-1} = \lceil \rho_{t-1}^{-1} \rceil$ times each vector $\widetilde{\boldsymbol{X}}_k = (\widetilde{X}_{1k}, \ldots, \widetilde{X}_{nk})$ of the elite sample $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}}\}$, that is take η_{t-1} identical copies of

each vector $\widetilde{\boldsymbol{X}}_k$. Denote the entire new population ($\eta_{t-1}N_{t-1}$ cloned vectors plus the original elite sample $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{cl} = \{(\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_1), \ldots, (\widetilde{\boldsymbol{X}}_{N_{t-1}}, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}})\}$. To each of the cloned vectors of the population $\mathcal{X}_{cl}$ apply the 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 $\widehat{c}_t = \frac{N_t}{N}$ (see (11)) as an estimator of c_t in (9). Note again that each vector of $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_t}$ 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 step 5, otherwise set $t = t + 1$ and repeat from step 2.

5. **Final Estimator** Deliver $\widehat{\ell}(m)$ given in (10) as an estimator of $\ell(m)$ and $|\widehat{\mathcal{X}}^*| = \widehat{\ell}(m)|\mathcal{X}|$ as an estimator of $|\mathcal{X}^*|$.

In the Appendix we derive (under certain simplifying assumptions), some analytical results on the complexity of the product estimator (10). More on the complexity of the cloning Algorithm 3.1 can be found in (Rubinstein, 2008).

To provide more insight on the splitting Algorithm 3.1, consider a toy example associated with a 6-sided polytope. Let $N = 6$ and $\rho = 1/3$. This corresponds to a number of elite samples equal two. Figures 4- 10 present typical dynamics (3 iterations) with the splitting Algorithm 3.1 for the polytope (marked as bold lines) with $m = 6$ inequality constraints (see (12)). It follows from Figures 4- 10 that

- At iterations 1, 2 and 3, we have $m_1 = 4,\ m_2 = 5$ and $m_3 = 6$, respectively. The corresponding two elite values of $S(\boldsymbol{X})$ after iterations 1, 2 and 3 are $(S = 4, S = 5)$, $(S = 5, S = 5)$ and $(S = 6, S = 6)$, respectively.

- The last two elite samples in Figure 10 (both corresponding to $S(\boldsymbol{X}) = m = 6$) hit the desired polytope, which means that Algorithm 3.1 converged after 3 iterations.

The purpose of Figures 4- 10 is also to demonstrate that Algorithm 3.1 is a global search one in the sense that at each iteration the generated random vectors $\boldsymbol{X}_i,\ i = 1, \ldots, 6$ are randomly distributed inside their corresponding sub-spaces. Consider, for example, the 4 *non-elite* samples in Figure 4 (after the first iteration), corresponding to $S(\boldsymbol{X}) = 3$. One can clearly see that they are spread quite randomly in the two-dimensional space, and similarly for the other elite and non-elite samples at iterations 2 and 3.

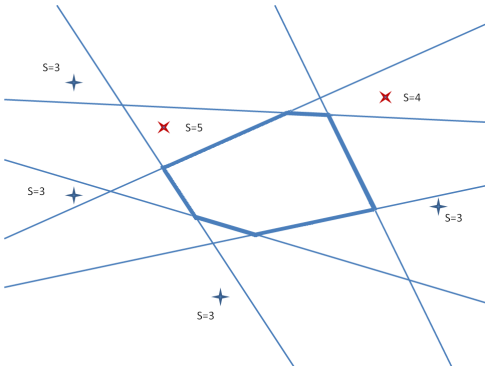

Figure 4: Iteration 1 of splitting Algorithm for 6-sided polytope.

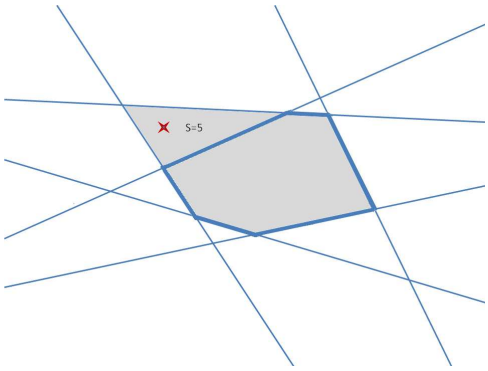

Figure 5: Iteration 1: the sub-region corresponding to the elite point with $S = 5$.

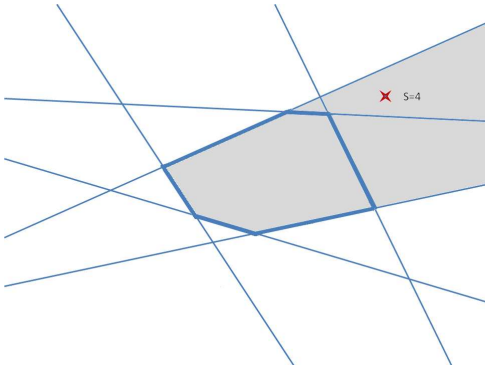

Figure 6: Iteration 1: the sub-region corresponding to the elite point with $S = 4$.

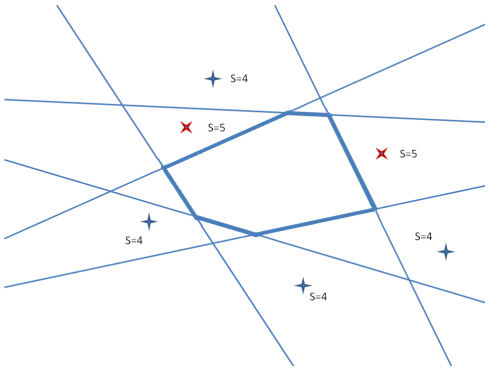

Figure 7: Iteration 2 of splitting Algorithm for 6-sided polytope.

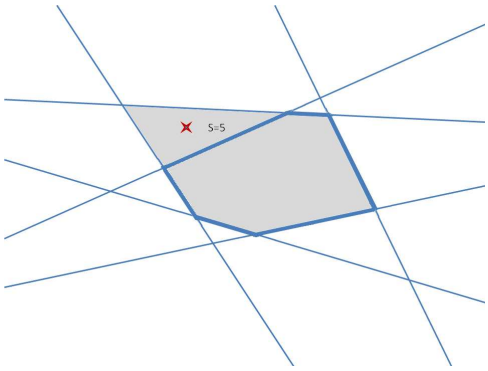

Figure 8: Iteration 2: the sub-region corresponding to the elite point with $S = 5$.

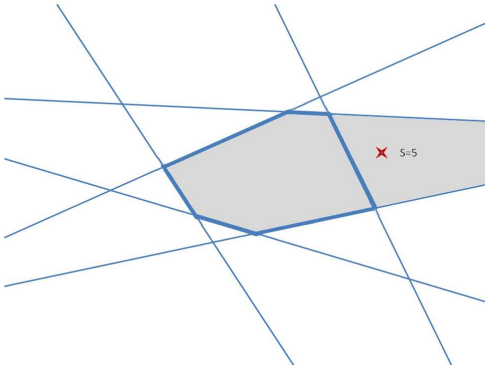

Figure 9: Iteration 2: the sub-region corresponding to the elite point with $S = 5$.

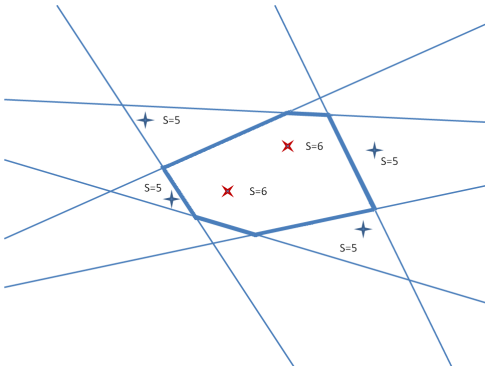

Figure 10: Iteration 3 of splitting Algorithm for 6-sided polytope.

Figure 11 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 by $\star$ are uniformly distributed on the sets $\mathcal{X}_0$ and $\mathcal{X}_1$. (Those, which are in $\mathcal{X}_1$ correspond to the elite samples). The points marked by $\bullet$ are approximately uniformly distributed on the sets $\mathcal{X}_1$ and $\mathcal{X}_2$. (Those, which are in $\mathcal{X}_2 = \mathcal{X}^*$ likewise correspond to the elite samples).

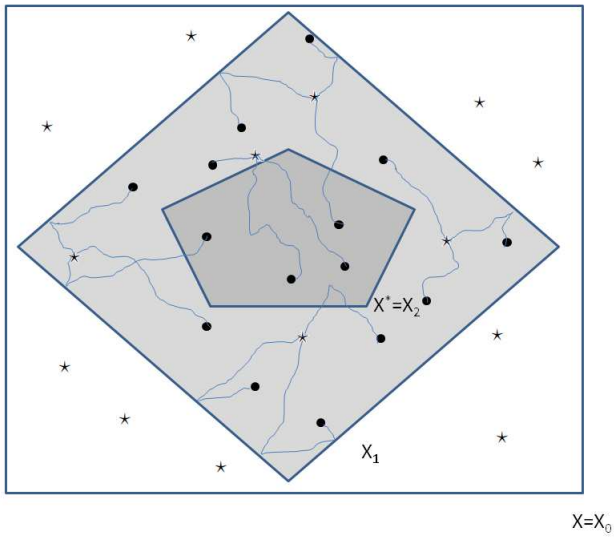

Figure 11: Dynamics of Algorithm 3.1

3.2 Enhanced Cloning Algorithm for Counting

Here we introduce an improved version of the basic cloning Algorithm 3.1, which contains (i) an enhanced cloning (splitting) step instead of the original one as in Algorithms 3.1 and a (ii) new screening step.

(i) **Enhanced cloning step** Its purpose is to find a good balance, in terms of bias-variance of the estimator of $|\mathcal{X}^*|$, between the cloning (splitting) parameter η_t, and the length of the burn-in parameter b_t, provided the number of samples N is given. Recall that the purpose of η_t is to reproduce the N_t elites η_t times.

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

$$\eta_{t-1} = \left\lceil \frac{N}{bN_{t-1}} \right\rceil - 1 = \left\lceil \frac{N_{cl}}{N_{t-1}} \right\rceil - 1. \tag{21}$$

Here $N_{cl} = N/b$ is called the *cloned sample size*, $N_{t-1} = \rho_{t-1}N$ denotes the number of elites and ρ_{t-1} is the adaptive rarety parameter at iteration $t-1$ [see (Rubinstein, 2008)] for details).

As an example, let $N = 1,000$, $b = 10$. 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.

As an alternative to (21) one can use the following heuristic strategy in defining b and η: find b_{t-1} and η_{t-1} from $b_{t-1}\eta_{t-1} \approx \frac{N}{N_{t-1}}$ and take $b_{t-1} \approx \eta_{t-1}$. In short, one can take

$$b_{t-1} \approx \eta_{t-1} \approx \left(\frac{N}{N_{t-1}} \right)^{1/2}. \tag{22}$$

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. Unless stated otherwise we shall use (22).

(ii) **Screening step**. Since the IS pdf $g^*(\boldsymbol{x}, m_t)$ must be *uniformly distributed* for each fixed m_t, the cloning algorithm checks at each iteration whether or not *all elite vectors* $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_t}$ *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) the empirical pdf associated with $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ from deviating from the uniform.

With this to hand, we can recapitulate from (Rubinstein, 2008) the cloning algorithm.

Algorithm 3.2 (Cloning 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** - the same as in Algorithm 3.1.

2. **Screening** Denote the elite sample obtained at iteration $(t-1)$ by $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}}\}$. Screen out the redundant elements from the subset $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}}\}$, and denote the resulting (reduced) one as $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$.

3. **Cloning (Splitting)** 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 cloning (splitting) and the burn-in parameters η_{t-1} and b_{t-1} according to (22). 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 $(t-1)$-th 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 $\mathcal{X}_{cl} = \{(\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_1), \ldots, (\widehat{\boldsymbol{X}}_{N_{t-1}}, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each of the cloned vectors of the population $\mathcal{X}_{cl}$ apply the MCMC (and in particular 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 - the same as in Algorithm 3.1.

5. **Stopping rule** -the same as in Algorithm 3.1.

6. **Final estimator** - the same as in Algorithm 3.1.

Note that the basic Algorithm 3.1 (with $b = 1$ and without screening) presents a particular case of the enhanced Algorithm 3.2. Note also that RAN algorithm presents a particular case of the enhanced Algorithm 3.2 without splitting ($\eta = 1$) and thus, with burn-in $b = N$. Note, finally, that although with a good balance between η and b, like in (22) and by adding the screening step, we can get a more accurate estimator of $|\mathcal{X}^*|$, while using the enhanced Algorithm 3.2, nevertheless to keep things simple we shall use below the basic cloning Algorithm 3.1.

3.3 Direct Estimator

The direct estimator below can be viewed as an alternative to the estimator $|\widehat{\mathcal{X}}^*|$ obtained by Algorithm 3.1. This estimator is based on *direct counting* of the number of screened samples obtained immediately after crossing the level m. Such a counting estimator, denoted by $|\widehat{\mathcal{X}}^*_{dir}|$, is associated with the *empirical* distribution of the uniform distribution $g^*(\boldsymbol{x}, m)$. We found numerically that $|\widehat{\mathcal{X}}^*_{dir}|$ is extremely useful and very accurate. Note that it is applicable only for counting problems with $|\mathcal{X}^*|$ not too large. Specifically $|\mathcal{X}^*|$ should be less than the sample size N, that is $|\mathcal{X}^*| < N$. Note also that counting problems with values small relative to $|\mathcal{X}|$ are the most difficult ones and in many counting problems one is interested in the cases where $|\mathcal{X}^*|$ does not exceed some fixed quantity, say $\mathcal{N}$. Clearly, this is possible only if $N \geq \mathcal{N}$. It is important to note that $|\widehat{\mathcal{X}}^*_{dir}|$ is typically much more accurate than its counterpart, the standard estimator $|\widehat{\mathcal{X}}^*| = \widehat{\ell}|\mathcal{X}|$. The reason is that $|\widehat{\mathcal{X}}^*_{dir}|$ is obtained *directly* by counting 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

$$|\widehat{\mathcal{X}}^*_{dir}| = \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i^{(d)}) \geq m\}}, \tag{23}$$

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. Note that we set in advance $\boldsymbol{X}_1^{(d)} = \boldsymbol{X}_1$. To increase further the accuracy of $|\widehat{\mathcal{X}}^*_{dir}|$ we can take a larger sample at the last step of Algorithm 3.1, that is after reaching the level m.

Since we shall extensively use the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$ in our numerical results, we present next the relevant algorithm.

Algorithm 3.3 (Direct Splitting Algorithm for Counting) Given the rarity parameter ρ, say $\rho = 0.1$, the parameters a_1 and a_2, say $a_1 = 0.01$ and $a_2 = 0.25$, such that $\rho \in (a_1, a_2)$, and the sample size N, execute the following steps:

1. **Acceptance-Rejection** - the same as in Algorithm 3.1.
2. **Adaptive choice of** ρ - the same as in Algorithm 3.1.
3. **Screening** - the same as in Algorithm 3.2.
4. **Cloning** - the same as in Algorithm 3.1.
5. **Estimating** m_t - the same as in Algorithm 3.1.
6. **Stopping rule** - the same as in Algorithm 3.1.
7. **Final Estimator** For $m_T = m$, take a sample of size N, and deliver $|\widetilde{\mathcal{X}}^*_{dir}|$ in (23) as an estimator of $|\mathcal{X}^*|$.

Note that there is no need to calculate $\widehat{c}_t$ in Algorithm 3.3 at any step. Note also that the counting Algorithm 3.3 can be readily modified for combinatorial optimization, since an optimization problem can be can be viewed as a particular case of counting, where the counting quantity $|\mathcal{X}^*| = 1$. For details see (Rubinstein 2008).

3.4 Split1 Estimator and its Convergence

Although Algorithms Split1-Split3 directly follow from the splitting Algorithm 3.1, for clarity we will elaborate a little more on the Split1 version, which can be also viewed as an extended version of the RAN algorithm, for which the number of elite samples =1, (no splitting, $\eta = 1$). Unless stated otherwise we shall have in mind the coloring problem.

Consider a configuration G_{j-1} which, as before, has a fixed number (equal to $j-1$) vertices. Assume for a moment that we have a feasible q-coloring in G_{j-1} and we run Algorithm 3.1 until it reaches steady state as per Definitions 6.2 and 6.3 and Theorem 6.2 of the Appendix. This means that after $\tau(\varepsilon)$ transitions we can start collecting the samples, since according to Theorem 6.2 they will be approximately distributed uniformly on the set $\mathcal{X}_{j-1}$. Clearly, part of these samples will satisfy $\zeta(a_j) \neq \zeta(b_j)$, that is, will also be uniformly distributed on the next sub-graph G_j. Denoting by N the total sample size generated in steady state for the graph G_{j-1} and by N_j the size of the sample, called the *elite sample*, we can estimate

$$c_t = \frac{|\mathcal{X}_j|}{|\mathcal{X}_{j-1}|}$$

by

$$\widetilde{c}_t = \frac{N_j}{N}.$$

Since $b = 1$, it follows that we shall split $\eta_j = \lceil \widetilde{c}_t \rceil$ times each of the elites of size N_j.

It is important to note that if we continue running in parallel the N_j Markov chains for several additional burn-in periods b, we would obtain a sample "closer" to the uniform distribution on $\mathcal{X}_j$ than that with $b = 1$. However, in fact the value of b does not matter much. The crucial point here is that all elites $N_j, j = 1, \ldots, j-1$ at all previous configurations G_j are already in the *steady state (warm up)*. What follows is that using some manageable burn-in period b, we only need, iteration-wise to keep the resulting Gibbs samples (of the size N) in the warm position, rather than run each MC from scratch with a large b, like in RAN. In other words, in the splitting versions, for each sub-graph G_j, we automatically derive an elite sample of size N_{j+1} from the generated sample of size N. As mentioned earlier, for coloring all these elite samples are obtained by conditioning on $\zeta(a_{j+1}) \neq \zeta(b_{j+1})$, and will be used for the next sub-graph G_{j+1}.

Remark 3.1 It is important to note that in contrast to Algorithm 3.1 the rarity parameter ρ in Split1 is not explicitly defined. The reason is that the number of elites N_j at iteration j is defined on-line. More precisely, as soon as the sample of size N has been generated (using the Gibbs sampler), the number N_j of elites is found by labeling the j edges, with their order $1, \ldots, j$ strictly followed in advance, while for the remaining $m - j$ ones the order is arbitrary. It is readily seen that as j approaches m the number of elites N_j may become small, since the labeling reduces the sample space of the elites. If at some iteration $N_j < k$, say $k = 10$, we have to increase the sample size till $N_j \geq k$. A similar policy should be employed for Split2 where the levels m_t are chosen adaptively.

Convergence of Split1

Assume for a moment that similar to RAN we use a long warm-up periods for each of the N parallel independent Markov chains in Split1. In this case the convergence theorems established for RAN (Hayes, Vera and Vigoda, 2007; Mitzenmacher and Upfal, 2005; Motwani and Raghavan, 1997) automatically hold for Split1. The reason is that Split1 extends RAN in the sense that in the former case we independently run in N parallel Markov chains instead of a single one. Clearly, by taking the best performance from N parallel simulations one can not do worse than when using a single one.

3.5 Why Splitting Method Works

Consider c_t in (8), which is used in $\ell(m)$ in (7). Note that

- c_t can be written as $c_t = \mathbb{P}(\mathcal{X}_t|\mathcal{X}_{t-1})$. That is, c_t represents conditional probability that the Markov process reaches the sub-space $\mathcal{X}_t$ from $\mathcal{X}_{t-1}$, or in other words, it presents the conditional probability that the process reaches the level m_t from level m_{t-1}. Consider for concreteness the set of integer program constraints as in Example 1.6 and assume without loss of generality that $T = m$. Note that this implies that $m_t = t$ and $m_T = m$, where m is the number of constraints.

- $c_t = \mathbb{P}(\mathcal{X}_t|\mathcal{X}_{t-1})$ can also be viewed as the conditional probability of exactly t *arbitrary* constraints being satisfied at iteration t, $(t = 1, \ldots, m)$, provided that at iteration $t-1$ any of the $t-1$ constraints have already been satisfied. We emphasize the word *arbitrary*, since without labeling, as in Algorithm 3.1, any point generated in the sub-space $\mathcal{X}_{t-1}$ is considered as a candidate for acceptance to $\mathcal{X}_t$. This is in contrast to RAN, where because of the labeling, either no points or a very limited number of points can be accepted to $\mathcal{X}_t$ from $\mathcal{X}_{t-1}$.

- Since $c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}$ is the ratio of the number of feasible points in $\mathcal{X}_t$ to that in the larger sub-space $\mathcal{X}_{t-1}$, where both sub-spaces are *arbitrary*, satisfying t and $t-1$ *arbitrary* constraints respectively - it readily follows that typically c_t is not a rare-event, say $c_t \geq 10^{-3}$, hence the splitting method should work.

Note that in the continuous case $|\mathcal{X}_t|$ represents a volume of a body, and if we consider, say, the constraints of linear programming, then c_t is the ratio of the volume of a convex sub-set $\mathcal{X}_t$ to the volume of a convex sub-set $\mathcal{X}_{t-1}$, where the former volume is obtained by adding an *arbitrary* constraints from the latter one.

Veracity of the splitting method

For certain pathological models, some of the probabilities $c_t,\ t = 1, \ldots, m$ in the product $c_1, \ldots, c_m$ can still be small, say $c_t < 10^{-4}$. In such a case Algorithm 3.1 stops at that intermediate level $m_t < m$, with an announcement - "pathological model".

As a simple "pathological model" consider estimation of ℓ in the sum of independent random variables, that is estimation of

$$\ell = \mathbb{E}_f \left[I_{\{\sum_{i=1}^n X_i = m\}} \right], \tag{24}$$

where $X_i \sim \text{Ber}(p_i)$. Assume for concreteness that $p_1 = 10^{-5}$, while the remaining ones, $p_i,\ i = 2, \ldots, n$ equal $1/2$. Assume also that $m = n$. It is readily seen that if one takes a sample, say $N = 1,000$, then there is a high probability that Algorithm 3.1 will be stacked with an announcement - "pathological model". This will happen because of the element $p_1 = 10^{-5}$, which by itself presents a rare event probability.

4 Numerical Results

Here we present numerical results on all 5 versions of counting the number of SAT assignments, the number of valid colorings in a network and some other problems, such as the calculating the permanent and the volume of a convex body.

As mentioned we shall show below that

- The RAN algorithm with a single Markov chain (MC), that is without splitting, typically performs poorly; it is either stacked at some intermediate level m_t and thus unable to deliver any value $|\widetilde{\mathcal{X}}^*|$ or trapped at some local extremum, that is its estimator $|\widetilde{\mathcal{X}}^*|$ of $|\mathcal{X}^*|$ is heavily biased. Exceptions are convex counting problems, like estimating the volume of a convex polytope and when the counting quantity is a huge number. In the latter case one can typically use the crude Monte Carlo method. The main reason for the poor performance is that RAN is a local rather than a global search algorithm. This is due to the absence of splitting in RAN, that is by running only a single Markov chain instead of multiple chains in parallel. Another reason is labeling of the sub-graphs G_j in RAN, whereby they are chosen *deterministically* in advance instead of randomly. Note that, when simulating multiple Markov chains in parallel at each level m_t, the splitting algorithms can be viewed as *level-based multi-start* algorithms. In this sense they resemble the multi-start algorithm in multi-extremuml optimization.

- The estimator $|\widetilde{\mathcal{X}}^*|$ of $|\mathcal{X}^*|$ in RAN will heavily depend on a particular (deterministic) combination of the sequences of sub-graphs $G_0, G_1, \ldots, G_m$. In short, the statistical properties of the counting estimators $|\widetilde{\mathcal{X}}_0|, |\widetilde{\mathcal{X}}_1|, \ldots, |\widetilde{\mathcal{X}}_m|$ will differ for different sequences (permutations) of $G_0, G_1, \ldots, G_m$. Once we stop controlling the sequences, that is, we choose them *randomly* , as in Split3 and in the splitting Algorithm 3.2, the associated counting estimators $|\widetilde{\mathcal{X}}_0|$, $|\widetilde{\mathcal{X}}_1|, \ldots,\ |\widetilde{\mathcal{X}}_m|$ will be able to avoid local extrema. In summary, *there is not enough randomness in the RAN algorithm* as compared to Split3 and the cloning Algorithm 3.2.

Note also that

1. The Split1 and Split2 versions (performing similar to RAN in the labeled space, but using splitting), will serve as bridges between the RAN and the splitting Algorithm 3.1.

2. We shall also use enhanced versions of RAN, the so-called RAN1 and RAN2. Both of them are *without splitting*, but otherwise they coincide with Split3 and the splitting Algorithm 2.3, respectively, that is, in contrast to RAN we sample in them without labeling in advance. We shall see that although in the sequence

   RAN $\rightarrow$ RAN1 $\rightarrow$ RAN2

   the statistical properties improve, both RAN1 and RAN2 are typically trapped in local extremum, or sometimes even stacked at an intermediate level m_t.

Remark 4.1 To find the initial sub-graph in Split3, and thus speed up its performance, we can use the following Monte Carlo acceptance-rejection procedure:

- Generate a sample of size N of iid random vectors $\boldsymbol{X}_i = (X_{i1}, \ldots, X_{id})$ each of size n (number of vertices). Associate each component of the n-dimensional random vector with a vertex in the graph G.

- For each new sample, assign randomly a color r, $r = 1, \ldots, d$ to the corresponding vertex.
- Estimate c_0 by using acceptance-rejection, that is as

$$\widetilde{c}_0 = \frac{N_0}{N},$$

  where N_0 is the number of elites.

In practice one can determine the initial sub-graph as follows. Pick *any* configuration G_j corresponding to the elite sample N_0. Here subscript j corresponds to the sub-graph G_j with exactly j labeled edges. With G_j to hand, we proceed next to G_{j+1}, etc, for the remaining $m - j$ iterations until we reach $G = G_m$. We call this Split3, the *enhanced* Split3 algorithm.

Recall, finally that we shall use the basic splitting Algorithm 3.1, that is, set the burn-in period $b = 1$ and employ the random Gibbs sampler.

4.1 SAT Problem

To study the performance (variability in the solutions) of all 5 algorithms, we ran each problem 10 times with each algorithm and report the statistics.

We took some benchmarks from
http://www.satcompetition.org and
http://fmv.jku.at/sat- race-2006/downloads.html

We present data for two different SAT models, one of a small size and the other of a moderate size. As we shall see below, for the former case RAN delivers a local extremum, while for the latter case it is always stacked at an intermediate level m_t. For the splitting method the results are as follows: for the former case all 4 splitting methods are unbiased, while for the latter case Split1 and Split2 are biased (because of the labeling issue), while Split3 and the splitting Algorithm 3.1 are unbiased.

First Model: 3-SAT with instance matrix $A = (20 \times 80)$.

Tables 1-5 summarize the performance of the 5 algorithms for the for a random 3-SAT problem with an instance matrix $A = (20 \times 80)$ taken from (Rubinstein, 2008) with $N = 1,000$. For the splitting algorithms we set in addition $\rho = 0.1$ and $b = 1$ (no warm up).

One can readily see that the accuracy of the algorithm increases in the sequence RAN $\rightarrow$ Split1 $\rightarrow$ Split2 $\rightarrow$ Split3 $\rightarrow$ cloning.

In particular one can see from Table 5 that the RAN algorithm is either stacked at some intermediate level (6 out of 10 cases) or is trapped in a local extremum. In the latter case, depending on on particular on particular labeling, it delivers either 6 or 9 SAT assignments instead of 15 for all 4 splitting algorithms. The increase of the sample size from $N = 1,000$, say to $N = 10,000$ did not help much. In this case, instead of 6 times, it was stacked 3 times. In the remaining 7 cases it delivered again either 6 or 9, but never 15. We also ran both RAN1 and RAN2. They were never stacked at some intermediate level m_t, but were consistently trapped in a local extremum, delivering again either 6 or 9 valid SAT assignments instead of 15.

Table 1: Performance of splitting Algorithm 3.1 for SAT 20×80 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	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

Table 2: Performance of Split3 Algorithm for SAT 20×80 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	11	17.515	0.168	15	0.000	5.635
2	11	15.622	0.041	15	0.000	5.652
3	11	9.024	0.398	15	0.000	5.635
4	11	15.512	0.034	15	0.000	5.650
5	11	14.372	0.042	15	0.000	5.623
6	11	15.051	0.003	15	0.000	5.652
7	11	13.607	0.093	15	0.000	6.654
8	11	11.650	0.223	15	0.000	5.668
9	11	15.968	0.065	15	0.000	5.643
10	11	11.925	0.205	15	0.000	5.648
Average	11	14.025	0.127	15.000	0.000	5.746

Table 3: Performance of Split2 Algorithm for SAT 20×80 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	10	14.557	0.030	15	0.000	6.660
2	10	12.434	0.171	15	0.000	6.734
3	10	16.748	0.117	15	0.000	6.038
4	10	17.235	0.149	15	0.000	6.055
5	10	15.783	0.052	15	0.000	6.005
6	10	17.198	0.147	15	0.000	6.719
7	10	14.029	0.065	15	0.000	6.687
8	10	17.198	0.147	15	0.000	6.704
9	10	20.373	0.358	15	0.000	6.676
10	10	15.992	0.066	15	0.000	6.064
Average	10	16.155	0.130	15.000	0.000	6.434

Table 4: Performance of Split1 Algorithm for SAT 20×80 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	80	14.624	0.025	15	0.000	56.606
2	80	9.854	0.343	15	0.000	54.400
3	80	12.689	0.154	15	0.000	57.265
4	80	13.236	0.118	15	0.000	56.855
5	80	15.134	0.009	15	0.000	57.333
6	80	13.195	0.120	15	0.000	54.324
7	80	8.154	0.456	15	0.000	56.312
8	80	17.039	0.136	15	0.000	54.835
9	80	14.354	0.043	15	0.000	55.169
10	80	17.796	0.186	15	0.000	54.355
Average	80	13.608	0.159	15.000	0.000	55.745

Table 5: Performance of RAN Algorithm for SAT 20×80 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	80	25.607	0.707	9	0.400	61.130
2	78	NaN	NaN	NaN	NaN	58.638
3	76	NaN	NaN	NaN	NaN	57.337
4	80	144.542	8.636	9	0.400	57.051
5	80	261.825	16.455	9	0.400	57.075
6	60	NaN	NaN	NaN	NaN	43.619
7	80	11.373	0.242	9	0.400	57.229
8	80	34.684	1.312	6	0.600	57.034
9	71	NaN	NaN	NaN	NaN	53.681
10	61	NaN	NaN	NaN	NaN	45.714
Average	74.6	95.606	5.470	8.400	0.440	54.851

Table 6 presents the dynamics for one of the runs of the splitting Algorithm 3.1 for the same model.

Table 6: Dynamics of Algorithm 3.1 for SAT 20×80 model.

t	$\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	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

Here we used the following notations

1. N_t 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. $\rho_t = N_t/N$ denote the adaptive rarity parameter.

Tables 7 and 8 present summary statistics of RAN, Split1- Split3 and the splitting (cloning) Algorithm 3.1 for the SAT 20×80 model. In particular, Table 7 presents the average statistics of $|\widetilde{\mathcal{X}}^*|$ and the relative errors, denoted as $|\bar{\mathcal{X}}^*|$ and $\bar{RE}$, respectively, while table 8 presents similar data but with $|\widetilde{\mathcal{X}}^*|$ replaced by $|\widehat{\mathcal{X}}^*|_{dir}$.

Table 7: Comparative studies of Algorithms RAN, Split1, Split 2, Split 3 and cloning Algorithm 3.1 for the product estimator $|\bar{\mathcal{X}}^*|$.

RAN		Split1		Split2		Split3		Cloning	
$\bar{\mathcal{X}}^*\|$	$\bar{RE}$	$\bar{\mathcal{X}}^*\|$	$\bar{RE}$	$\bar{\mathcal{X}}^*\|$	$\bar{RE}$	$\bar{\mathcal{X}}^*\|$	$\bar{RE}$	$\bar{\mathcal{X}}^*\|$	$\bar{RE}$
95.606	5.470	13.608	0.159	16.155	0.130	14.025	0.127	15.504	0.118

Table 8: Comparative studies of Algorithms RAN, Split1, Split 2, Split 3 and cloning Algorithm 3.1 for the direct estimator $|\bar{\mathcal{X}}^*|_{dir}$.

RAN		Split1		Split2		Split3		Cloning	
$\bar{\mathcal{X}}^*\|_{dir}$	$\bar{RE}$	$\|\bar{\mathcal{X}}^*_{dir}\|$	$\bar{RE}$	$\|\bar{\mathcal{X}}^*_{dir}\|$	$\bar{RE}$	$\|\bar{\mathcal{X}}^*_{dir}\|$	$\bar{RE}$	$\bar{RE}$	$\|\bar{\mathcal{X}}^*_{dir}\|$
8.400	0.440	15	0.000	15	0.000	15.0	0.000	15	0.000

We also considered an extended version of the above random 20×80 3-SAT model, namely one with 140 clauses instead of 80 and such that the first 80 clauses are the same as in model 20×80. Table 9 presents the dynamic of Algorithm 3.1 for the extended SAT model. One can clearly see that in this case $|\widetilde{\mathcal{X}}^*| = 0$, that is, there is no feasible solution starting from $m = 140$. Note that for $m = 139$ we still obtain $|\widetilde{\mathcal{X}}^*| = 1$.

Table 9: Dynamics of Algorithm 3.1 for SAT 20×140 model.

t	$\lvert\mathcal{X}^*\rvert$	Empirical	$N_{t,e}$	$N_{t,e}^{(s)}$	m_t^*	m_{*t}	ρ_t
1	1.39e+005	1.33e+002	133	1	133	127	0.13
2	6.97e+004	1.00e+000	1	1	129	129	0.50
3	6.97e+004	2.00e+000	2	1	130	130	1.00
4	3.49e+004	1.00e+000	1	1	131	131	0.50
5	3.49e+004	2.00e+000	2	1	131	131	1.00
6	1.74e+004	1.00e+000	1	1	133	133	0.50
7	8.72e+003	1.00e+000	1	1	134	134	0.50
8	4.36e+003	1.00e+000	1	1	137	137	0.50
9	4.36e+003	2.00e+000	2	1	137	137	1.00
10	2.18e+003	1.00e+000	1	1	139	139	0.50
11	2.18e+003	1.00e+000	2	1	139	139	1.00
12	2.18e+003	1.00e+000	2	1	139	139	1.00
13	2.18e+003	1.00e+000	2	1	139	139	1.00
14	2.18e+003	1.00e+000	2	1	139	139	1.00
15	2.18e+003	1.00e+000	2	1	139	139	1.00
16	2.18e+003	1.00e+000	2	1	139	139	1.00
17	2.18e+003	1.00e+000	2	1	139	139	1.00
18	2.18e+003	1.00e+000	2	1	139	139	1.00
19	0.00e+000	0.00e+000	0	0	140	140	0.00

Second model: Random 3-SAT with instance matrix $A = (75 \times 325)$.

Our next model is the random 3-SAT with the instance matrix $A = (75 \times 325)$ taken from www.satlib.org. Table 10 presents the dynamic of the splitting Algorithm 3.1. We set $N = 10,000$ and $\rho = 0.1$ for all iterations until Algorithm 3.1 has reached the desired level 325. After that we switched to $N = 100,000$ for the last iteration. The results are self- explanatory.

Table 10: Dynamics of Algorithm 3.1 for the random 3-SAT with matrix $A = (75 \times 325)$.

t	$\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	5.4e+020	0.0	1020	1020	305	292	0.11
2	5.6e+019	0.0	1714	1714	307	297	0.14
3	6.5e+018	0.0	1070	1070	309	301	0.10
4	1.2e+018	0.0	1462	1462	310	304	0.12
5	1.7e+017	0.0	2436	2436	312	306	0.18
6	2.0e+016	0.0	2166	2166	314	308	0.14
7	6.1e+015	0.0	1501	1501	316	310	0.12
8	4.6e+014	0.0	1115	1115	316	312	0.09
9	2.5e+013	0.0	636	636	319	314	0.06
10	5.0e+012	0.0	2213	2213	320	315	0.23
11	9.5e+011	0.0	2674	2674	321	316	0.20
12	1.6e+011	0.0	1969	1969	320	317	0.19
13	2.5e+010	0.0	1962	1962	321	318	0.17
14	3.3e+009	0.0	1775	1775	322	319	0.16
15	3.9e+008	0.0	1350	1350	323	320	0.13
16	3.5e+008	0.0	1437	1437	324	321	0.12
17	3.8e+007	0.0	1270	1270	324	322	0.10
18	2.8e+006	8.0	924	924	325	323	0.08
19	1.4e+005	179.0	537	534	325	324	0.05
20	2341.0	2203.0	196	187	325	325	0.01
21	2341.0	2225.0	10472	2199	325	325	1.00

We found that the average relative error for the product estimator $|\widetilde{\mathcal{X}}^*|$ it is RE = 0.08 while for the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$ is RE = 0.015, and the average CPU time is about 6 minutes for each ran. We ran also this instance with the remaining Split1-Split3 and with the RAN algorithms. We found that

1. RAN is always stacked at some intermediate level.
2. Split1 and Split2 are never stacked, but both are much slower than splitting Algorithm 3.1. In addition, both are trapped at some local extremum. A similar phenomenon was observed for RAN1 and RAN2. For example, Split2 delivered the following results for the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$:

   $$(2092, 1504, 2184, 2092, 2063, 612, 2068, 1618, 2116, 2152).$$

   One can see that $|\widehat{\mathcal{X}}^*_{dir}|$ in Split2 is heavily biased as compared with $|\widehat{\mathcal{X}}^*_{dir}| \approx 2225$ in Algorithm 3.1.
3. Split3 performs similar to splitting Algorithm 3.1, although its CPU time is longer by a factor of 5-6.

4.2 Coloring

Here we ran the vertex coloring models in the following two settings: (i) Condition $q \geq 2\Delta + 1$ of Theorem 6.2 of the Appendix holds. (ii) The condition is violated, that is, q is any integer number satisfying $q \geq 2$. We found that

1. The RAN algorithm and its associates Split1 and Split2 still perform satisfactorily, as predicted by Theorem 6.2 of the Appendix for case (i), but fail for case (ii), in particular when q is small.

2. The splitting algorithms Split3 and the Algorithm 3.1 perform nicely irrespective of the value of q. Note that case (ii), in particular when q is small, is the most interesting and most difficult, since $|\mathcal{X}^*|$ is very small relative to $|\mathcal{X}|$, thus $\ell = \frac{|\mathcal{X}^*|}{|\mathcal{X}|}$ is a very low probability.

We consider here two coloring models: a small one and a moderate size.

First Model: $n = 20$ **vertices,** $m = 62$ **edges,** $\Delta = 10$**,** $q = 21$**.**

Tables 11- 12 present the performance of the splitting Algorithm 3.1 and RAN respectively for the benchmark problem with $n = 20$ vertices, $m = 62$ edges, $\Delta = 10$, $q = 21$ and the following $A = 20 \times 20$ matrix.

$$\boldsymbol{A} = \begin{matrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0
\end{matrix} \tag{25}$$

We set the following parameters: $N = 1,000, \rho = 0.05$ and $b = 1$.

One can readily see that the splitting Algorithm is faster than RAN about 10 times, since the number of iterations in the former is 6 and in the latter 62, which is equal to the number of edges.

Table 11: Performance of splitting Algorithm 3.1 for 21-coloring problem with $n = 20$ nodes.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	6	1.18E+25	0.05	0.788
2	6	1.09E+25	0.12	0.827
3	6	1.12E+25	0.10	0.834
4	6	1.38E+25	0.11	0.793
5	6	1.33E+25	0.07	0.821
6	6	1.45E+25	0.16	0.827
7	6	1.33E+25	0.07	0.778
8	6	1.19E+25	0.04	0.824
9	6	1.09E+25	0.12	0.805
10	6	1.29E+25	0.04	0.808
Average	6	1.25E+25	0.09	0.811

Table 12: Performance of RAN algorithm for 21-coloring problem with $n = 20$ nodes.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	62	1.22E+25	0.01	8.441
2	62	1.22E+25	0.01	8.437
3	62	1.27E+25	0.03	8.387
4	62	1.15E+25	0.07	8.445
5	62	1.24E+25	0.00	8.443
6	62	1.41E+25	0.14	8.478
7	62	1.33E+25	0.08	8.482
8	62	1.10E+25	0.11	8.421
9	62	1.13E+25	0.09	8.533
10	62	1.29E+25	0.04	8.494
Average	62	1.24E+25	0.06	8.455

As expected, for $q = 21$ the probability $\ell = \frac{|\mathcal{X}^*|}{|\mathcal{X}|}$ is not a rare-event one, so even the crude Monte Carlo (CMC) method can also be used here. Table 13 presents the performance of the CMC estimator based on 10 independent runs, which matches both Tables 11 and 12.

Table 13: Performance of CMC method for 21-coloring problem with $n = 20$ nodes.

Run N_0	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	1.24E+25	4.20E-02	1.045
2	1.24E+25	4.20E-02	1.016
3	1.16E+25	2.52E-02	1.051
4	1.17E+25	1.68E-02	1.026
5	1.16E+25	2.52E-02	1.027
6	1.15E+25	3.36E-02	1.054
7	1.15E+25	3.36E-02	1.010
8	1.22E+25	2.52E-02	1.014
9	1.17E+25	1.68E-02	1.042
10	1.24E+25	4.20E-02	1.045
Average	1.19E+25	3.03E-02	1.033

Second Model: $n = 40$ vertices, $m = 162$ edges, $\Delta = 12$, $q = 4$.

We next consider a larger and much sparser coloring model, namely one with $n = 40$ vertices, $m = 162$ edges, $\Delta = 12$, $q = 4$ and the following 40×40 matrix

$$
\begin{array}{cccccccccccccccccccccccccccccccccccccccc}
0&1&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0\\
1&0&1&0&1&0&0&0&0&1&1&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&1&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&1&0&0&0&1&0&0\\
0&0&1&0&1&1&0&1&0&1&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&1\\
1&1&1&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&1&1&0&1&0&1&1&0&0&0&0&0&1&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&1&0&1&0&0&0&0&0&0\\
0&0&0&1&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&0&1&0&1&1&0&1&0&0&0&0&1&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&1&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&0\\
0&1&0&0&0&0&0&0&1&1&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&1&0&1&0&0&0&0&1&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&1&0&1&0&0&0&1&0&0&1&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&1&0&1&1&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&1&1&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0\\
0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0
\end{array}
$$

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1
0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1
0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0
0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 0 1 0 0 1 0 1
0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 1 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0
0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0
0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 1
0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
```

We set the $N = 100,000, \rho = 0.05$ and $b = 1$. Note that we used here $q = 4$ colorings (instead of $q = 2 \times 12 + 1 = 25$-colorings as Theorem 6.2 suggests). By doing so (see tables below) $|\mathcal{X}^*|$ becomes very small relative to $|\mathcal{X}|$ and $\ell = \frac{|\mathcal{X}^*|}{|\mathcal{X}|}$ is a rare-event probability.

We found that for this model RAN does not work at all. It always stops before reaching the final level $m = 162$. The RAN1 and RAN2 versions always reach the level $m = 162$, but, because of lack of splitting, converge to a local extremum and thus produce a heavily biased estimator of $|\mathcal{X}^*|$.

Tables 14- 16 present the performance of the splitting Algorithm 3.1, Split3 and Split2, respectively for the above benchmark problem with $n = 40$ vertices. We used the enhanced version of Split3 (see Remark 4.1) to determine the initial subgraph G_j. We do not present simulation results for Split1 since we always found that it is less accurate than Split2. In addition, it requires $m = 162$ iterations (equal to the number of edges) and is thus time- consuming. It follows from the results of Tables 14- 16 that

- Split2 performs poorly. Because of labeling it is trapped in different local extremum, which are typically far away from the true global one.

- Both, splitting Algorithm 3.1 and Split3 perform well. The splitting Algorithm 3.1 is faster about 3 times than Split3, since the number of iterations in Algorithm 3.1 is about 3 times less than in Split3. It is of interest to note that the product estimator $|\widetilde{\mathcal{X}^*}|$ is slightly better in Algorithm 3.1 while vice versa for the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$. This can be explained as follows: on the one hand the product estimator $|\widetilde{\mathcal{X}^*}|$ in Split3 has approximately 3 times more terms $\widehat{c}_t,\ t = 1, \ldots, T$ than in Algorithm 3.1, while on the other hand, it consumes more time (iterations) and thus more samples. This makes the direct Split3 estimator $|\widehat{\mathcal{X}}^*_{dir}|$ more accurate.

Note that if we stop both Algorithm 3.1 and the Split3 not immediately after they reach the final level $m = 162$, but continue them for 3-5 further iterations, then the difference of their accuracy (for the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$) becomes negligible.

Table 14: Performance of splitting Algorithm 3.1 for 4-coloring problem with $n = 40$ nodes.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	24	1415.62	0.055	1318	0.018	257.168
2	24	1194.38	0.110	1322	0.015	257.098
3	24	1356.09	0.010	1316	0.019	256.942
4	24	1596.61	0.190	1264	0.058	258.255
5	24	1348.93	0.005	1316	0.019	256.568
6	24	1627.90	0.213	1328	0.010	258.743
7	24	1353.55	0.009	1304	0.028	257.663
8	24	1293.32	0.036	1330	0.009	260.002
9	24	1229.90	0.084	1312	0.022	259.695
10	24	1590.67	0.185	1334	0.006	259.309
Average	24	1400.7	0.090	1314	0.021	258.144

Table 15: Performance of Split3 algorithm for 4-coloring graph problem with $n = 40$ nodes.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	66	1984.300	0.479	1338	0.003	720.096
2	66	1999.600	0.490	1327	0.011	722.244
3	66	1694.100	0.262	1336	0.004	723.914
4	66	1774.620	0.322	1326	0.012	718.257
5	66	1891.560	0.410	1332	0.007	700.993
6	66	1873.700	0.396	1332	0.007	701.613
7	66	2028.890	0.512	1334	0.006	700.402
8	66	1745.470	0.301	1328	0.010	700.883
9	66	1839.110	0.370	1328	0.010	701.685
10	66	1767.310	0.317	1336	0.004	700.933
Average	66	1859.86	0.385	1331.7	0.008	709.102

Table 16: Performance of Split2 for 4-coloring problem with $n = 40$ nodes.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	28	153.6	0.885	6	0.995	1937.2
2	26	195.8	0.853	12	0.991	1741.8
3	24	3394.9	1.549	50	0.962	1517.8
4	26	47.3	0.965	4	0.997	1652.2
5	24	2816.9	1.115	800	0.399	1588.5
6	25	1915.2	0.438	350	0.737	1584.3
7	25	418.5	0.686	52	0.961	1604.5
8	26	1771.6	0.330	250	0.812	1652.0
9	24	5220.0	2.919	114	0.914	1541.3
10	26	220.0	0.835	14	0.989	1691.5
Average	25.4	1615.4	1.057	165.2	0.876	1651.1

We also ran RAN1 and RAN2. As expected, both are trapped in a local extremum, delivering heavily biased estimators of $|\mathcal{X}^*|$.

Table 17 present the dynamics for one of the runs of the splitting Algorithm 3.1 for the same model.

Table 17: Dynamics of Algorithm 3.1 for 4-coloring graph with $n = 40$ nodes.

t	$\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	7.97E+22	0	6591	6591	-40	-64	6.59E-02
2	6.94E+21	0	8704	8704	-32	-54	8.70E-02
3	4.43E+20	0	6384	6384	-28	-46	6.38E-02
4	3.23E+19	0	7299	7299	-26	-40	7.30E-02
5	4.32E+18	0	13363	13363	-22	-36	1.34E-01
6	4.49E+17	0	10397	10397	-20	-32	1.04E-01
7	3.52E+16	0	7835	7835	-16	-28	7.84E-02
8	2.13E+15	0	6045	6045	-14	-24	6.05E-02
9	4.58E+14	0	21532	21531	-12	-22	2.15E-01
10	8.96E+13	0	19569	19569	-12	-20	1.96E-01
11	1.62E+13	0	18092	18092	-10	-18	1.81E-01
12	2.66E+12	0	16378	16378	-8	-16	1.64E-01
13	3.80E+11	0	14294	14291	-6	-14	1.43E-01
14	4.74E+10	0	12479	12472	-6	-12	1.25E-01
15	5.06E+09	0	10685	10675	-4	-10	1.07E-01
16	4.47E+08	0	8842	8825	-2	-8	8.84E-02
17	3.15E+07	0	7047	7003	-2	-6	7.05E-02
18	1.69E+06	1	5372	5261	0	-4	5.37E-02
19	5.99E+04	85	3537	3241	0	-2	3.54E-02
20	1.38E+03	1316	100000	1316	0	0	1.00E+00
21	1.38E+03	1326	100000	1326	0	0	1.00E+00
22	1.38E+03	1326	100000	1326	0	0	1.00E+00
23	1.38E+03	1326	100000	1326	0	0	1.00E+00
24	1.38E+03	1326	100000	1326	0	0	1.00E+00

Tables 18 and 19 present data similar to Tables 7 and 8 for RAN, Split2- Split3 and the splitting (cloning) Algorithm 3.1 for 40 node vertex coloring problem.

Table 18: Comparative studies of RAN, Split2- Split3 and cloning Algorithm 3.1 for 40 node vertex coloring problem using product estimator $|\bar{\mathcal{X}}^*|$.

RAN		Split2		Split3		Cloning	
$\widehat{\|\mathcal{X}^*\|}$	$\widehat{RE}$	$\widehat{\|\mathcal{X}^*\|}$	$\widehat{RE}$	$\widehat{\|\mathcal{X}^*\|}$	$\widehat{RE}$	$\widehat{\|\mathcal{X}^*\|}$	$\widehat{RE}$
NaN	NaN	1615.4	1.057	1859.86	0.385	1400.7	0.090

Table 19: Comparative studies of RAN, Split2- Split3 and cloning Algorithm 3.1 for 40 node vertex coloring problem using the direct estimator $|\bar{\mathcal{X}}^*|_{dir}$.

RAN		Split2		Split3		Cloning	
$\widehat{\|\mathcal{X}^*_{dir}\|}$	$\widehat{RE}$	$\widehat{\|\mathcal{X}^*_{dir}\|}$	$\widehat{RE}$	$\widehat{\|\mathcal{X}^*_{dir}\|}$	$\widehat{RE}$	$\widehat{RE}$	$\widehat{\|\mathcal{X}^*_{dir}\|}$
NaN	NaN	165.2	0.876	1331.7	0.008	1314	0.021

We performed extensive simulations with many other benchmark coloring models and observed consistently that reduction of the number of colors q always severely affects the RAN, Split1 and Split2 algorithms, but not the other two, Split3 and splitting Algorithm 3.1.

4.3 Permanent

To apply the Gibbs sampler for permanent we adopt the concept of "neighboring" elements, (see, chapter 10 of Ross, (2002) and chapter 6 of Rubinstein and Kroese, (2007)). In the latter reference it is called *2-opt* heuristics. Given a point (tour) $\boldsymbol{x}$ of length m_t generated by the pdf $g_t = g(\boldsymbol{x}, m_t)$, the conditional Gibbs sampling updates the existing tour $\boldsymbol{x}$ to $\widetilde{\boldsymbol{x}}$, where $\widetilde{\boldsymbol{x}}$ is generated from $g(\widetilde{\boldsymbol{x}}, m_t)$ and where $\widetilde{\boldsymbol{x}}$ is the same as $\boldsymbol{x}$ with one exception that the points x_i and x_j in $\widetilde{\boldsymbol{x}}$ are reversed. We accept the tour $\widetilde{\boldsymbol{x}}$ with probability $I_{\{S(\widetilde{\boldsymbol{X}}) \leq m_t\}}$, otherwise we leave the tour $\boldsymbol{x}$ the same. If $\widetilde{\boldsymbol{x}}$ is accepted we update the cost function $S(\boldsymbol{x})$ (for $j > i$) accordingly. In order for the tours $\boldsymbol{X}$ to be distributed approximately uniformly (at each sub-space $\mathcal{X}_t$) we use $n(n-1)/2$ trials.

We consider the following $A = 30 \times 30$ matrix as our benchmark for the permanent problem.

$$
\begin{array}{cccccccccccccccccccccccccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0\\
0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1
\end{array}
$$

Table 20 presents the performance of splitting Algorithm 3.1 for the permanent with $A = 30 \times 30$ matrix. We chose $N = 100,000$, $\rho = 0.1$ and, as usual $b = 1$.

Table 20: Performance of splitting Algorithm 3.1 for permanent with $A = 30 \times 30$ matrix.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	21	261.14	0.018	266	0	115.68
2	21	254.45	0.043	266	0	115.98
3	21	268.04	0.008	266	0	115.65
4	21	272.20	0.023	266	0	117.68
5	21	261.50	0.017	266	0	118.38
6	21	255.03	0.041	266	0	117.10
7	21	261.36	0.017	266	0	116.58
8	21	266.82	0.003	266	0	115.82
9	21	264.76	0.005	266	0	115.84
10	21	254.13	0.045	266	0	116.13
Average	21	261.94	0.022	266	0	116.48

We dispense with a comparison of Algorithm 3.1 with other less efficient algorithms, like RAN1-RAN2 and Split1-Split3, since the results are similar to those for the **second model** of $3 - SAT$ and the **second model** for coloring, respectively.

4.4 The Hamiltonian Cycles

We solve the Hamiltonian cycles problem by applying again the 2-opt heuristic used for the permanent. While counting the Hamiltonian cycles we simulated, in fact, an associated permanent problem by making use of the following observations:

- The set of Hamiltonian cycles of length n presents a subset of the associated permanent trajectories set.
- The latter set is formed from cycles of length $\leq n$.

It follows from the above that in order to count the number of Hamiltonian cycles for a given matrix A one can use do the following simple procedure:

1. Run Algorithm 3.2 and calculate the estimator of $|\mathcal{X}^*|$ of the associated permanent using the product formula (10). Denote such permanent estimator by $|\widehat{\mathcal{X}^*}_p|$.
2. Proceed with one more iteration of Algorithm 3.2 and calculate the ratio of the number of screened Hamiltonian elite cycles (cycles of length n) to the number of the screened elite samples (samples of length $\leq n$) in the permanent. Denote the ratio as ζ.
3. Deliver $|\widehat{\mathcal{X}^*}_H| = \zeta|\widehat{\mathcal{X}^*}_p|$ as the estimator of the number $|\mathcal{X}^*|$ of Hamiltonian cycles.

Below we present simulation results for three 30×30 models.
First model The matrix $A = 30 \times 30$ is

$$
\begin{array}{cccccccccccccccccccccccccccccc}
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&1&0&0&0&0&0\\
0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0
\end{array}
$$

$$
\begin{array}{cccccccccccccccccccccccccccccc}
0&0&0&0&0&0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0
$$

Table 21 presents the performance of splitting Algorithm 3.2 for the Hamiltonian cycle problem with $A = 30 \times 30$ matrix. We chose $N = 100,000$, $\rho = 0.1$ and $b = \eta$.

Table 21: Performance of splitting Algorithm 3.2 for Hamiltonian cycle problem with $A = 30 \times 30$ matrix.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	28	11.729	0.023	12	0	116.86
2	28	11.366	0.053	12	0	117.29
3	28	13.955	0.163	12	0	117.02
4	28	11.180	0.068	12	0	117.13
5	28	11.988	0.001	12	0	116.94
6	28	12.142	0.012	12	0	117.14
7	28	13.525	0.127	12	0	117.53
8	28	12.189	0.016	12	0	117.39
9	28	13.251	0.104	12	0	116.66
10	28	10.531	0.122	12	0	116.57
Average	28	12.185608	0.069	12	0	117.05

Second model The matrix $A = 30 \times 30$ is

$$
\begin{array}{cccccccccccccccccccccccccccccc}
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
1&0&0&0&0&0&1&1&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1\\
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0
$$

Table 22 presents the performance of splitting Algorithm 3.2 for the Hamiltonian cycle problem with $A = 30 \times 30$ matrix. We chose $N = 100,000$, $\rho = 0.1$ and $b = \eta$.

Table 22: Performance of splitting Algorithm 3.2 for Hamiltonian cycle problem with $A = 30 \times 30$ matrix.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	27	17.204	0.044	18	0	117.52
2	27	18.004	0.000	18	0	116.22
3	27	20.164	0.120	18	0	116.17
4	27	20.469	0.137	18	0	117.18
5	27	18.175	0.010	18	0	118.44
6	27	18.182	0.010	18	0	118.09
7	27	19.753	0.097	18	0	117.44
8	27	17.081	0.051	18	0	116.45
9	27	17.091	0.051	18	0	116.81
10	27	18.189	0.011	18	0	116.59
Average	27	18.431	0.053	18	0	117.09

Third model The matrix $A = 30 \times 30$ is

$$
\begin{array}{cccccccccccccccccccccccccccccc}
0&1&0&1&0&0&1&0&1&1&1&0&0&1&0&1&0&1&0&0&1&1&1&0&0&1&0&1&1&1\\
1&0&1&1&1&0&1&0&0&0&1&0&1&1&0&1&0&0&0&0&0&1&0&1&1&1&1&0&0&0\\
0&1&0&1&1&0&1&1&1&1&0&0&0&1&1&1&0&1&1&0&0&1&0&1&0&0&1&0&1&\\
1&1&1&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&1&0&1&1&\\
0&1&1&1&0&1&1&1&0&1&0&0&1&1&1&1&1&0&1&1&1&1&1&0&1&0&1&1&0&1\\
0&0&0&0&1&0&1&1&0&0&0&0&0&1&1&0&0&1&1&1&0&0&0&0&0&0&1&1&0&\\
1&1&1&0&1&1&0&1&1&0&0&1&1&0&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0&\\
0&0&1&0&1&1&1&0&1&1&1&0&0&1&0&1&0&1&0&0&0&1&1&1&0&1&0&0&0&\\
1&0&1&0&0&0&1&1&0&1&0&0&0&0&1&0&1&1&1&1&1&1&0&1&1&0&0&0&0&\\
1&0&1&0&1&0&0&1&1&0&1&1&0&0&1&0&1&1&1&0&0&1&0&1&0&0&0&1&1&\\
1&1&1&0&0&0&0&1&0&1&0&1&1&0&1&1&0&0&0&1&0&1&1&1&1&0&0&0&0&\\
0&0&0&0&0&0&1&1&0&1&1&0&1&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&\\
0&1&0&0&1&0&1&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&\\
1&1&0&0&1&1&0&0&0&0&0&1&1&0&1&1&0&0&0&0&0&1&1&0&1&1&0&1&0&0\\
0&0&1&1&1&1&0&1&1&1&1&0&1&1&0&1&1&1&0&0&0&0&1&0&1&1&0&0&1&1\\
1&1&1&0&1&0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&0&0&0&0&0&0&1&0&0&\\
0&0&1&0&1&0&0&1&1&1&0&0&0&0&1&1&0&1&1&1&0&0&1&0&0&0&0&0&0&1\\
1&0&0&0&0&1&0&0&1&1&0&0&0&0&1&0&1&0&1&1&1&0&1&0&0&0&0&1&0&1\\
0&0&1&0&1&1&0&1&1&1&0&1&0&0&0&0&1&1&0&1&0&1&1&0&1&1&1&0&0&0\\
0&0&1&1&1&1&0&0&1&0&1&0&0&0&0&0&1&1&1&0&1&1&0&1&0&0&0&1&1&0\\
1&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&1&0&1&0&1\\
1&1&0&0&1&0&0&0&1&1&1&0&0&1&0&0&0&0&1&1&1&0&1&0&1&0&1&1&1&0\\
1&0&1&0&1&0&1&1&0&0&1&0&1&1&1&0&1&1&1&0&0&1&0&1&1&0&1&0&1&0\\
0&1&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&1&0&0&1&0&1&0&0&1&1&1\\
0&1&1&0&1&0&0&1&1&0&1&0&0&1&1&0&0&0&1&0&0&1&1&1&0&1&1&1&0&1\\
1&1&0&0&0&0&1&0&0&0&0&0&0&1&1&0&0&0&1&0&1&0&0&0&1&0&1&1&1&1\\
0&1&0&1&1&0&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&1&1&0&1&1&0&1&1&0\\
1&0&1&0&1&1&0&0&0&1&0&0&0&1&0&1&0&1&0&1&1&1&0&1&1&1&1&0&1&1\\
1&0&0&1&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&1&1&0&1&1&1&0&1\\
1&0&1&1&1&0&0&0&0&1&0&1&0&0&1&0&1&1&0&0&1&0&0&1&1&1&0&1&1&0.
\end{array}
$$

Table 23 presents the performance of splitting Algorithm 3.2 for the Hamiltonian cycle problem with the above $A = 30 \times 30$ matrix. As before we set $N = 100,000$, $\rho = 0.1$ and $b = \eta$. Note that in this case the direct estimator is no longer available since the matrix A is not sparse.

Table 23: Performance of splitting Algorithm 3.2 for Hamiltonian cycle problem with $A = 30 \times 30$ matrix.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	14	2.38E+20	0.046	63.295
2	14	2.36E+20	0.033	62.809
3	14	2.28E+20	0.000	62.791
4	14	2.17E+20	0.049	62.714
5	14	2.35E+20	0.029	62.806
6	14	2.18E+20	0.046	62.951
7	14	2.10E+20	0.078	64.009
8	14	2.35E+20	0.032	62.790
9	14	2.23E+20	0.024	62.816
10	14	2.41E+20	0.057	62.614
Average	14	2.28E+20	0.039	62.959

4.5 Volume of a Polytope

Here we present numerical results with RAN and the splitting algorithms, on the volume of a polytope defined as $\boldsymbol{Ax} \leq \boldsymbol{b}$. We assume in addition that all components $x_k \in (0,1),\ k = 1,\ldots,n$. By doing so the polytope $\boldsymbol{Ax} \leq \boldsymbol{b}$ will be inside the unit n-dimensional cube and therefore its volume very small (rare-event probability) relative to the volume of the unit cube. In general, however, before implementing the Gibbs sampler one has to find the the minimal n-dimensional box, which contains the polytope, that is for each $i,\ i = 1,\ldots,n$, one has to find the values (c_i, d_i) of the minimal box satisfying $c_i \leq x_i \leq d_i,\ i = 1,\ldots,n$. This can be done by solving for each $i,\ i = 1,\ldots,n$ the following two linear programs

$$\min\ x_i \tag{26}$$

$$\text{s.t. } \boldsymbol{Ax} \leq \boldsymbol{b} \tag{27}$$

and

$$\max\ x_i \tag{28}$$

$$\text{s.t. } \boldsymbol{Ax} \leq \boldsymbol{b}. \tag{29}$$

We shall see that in this case all our algorithms perform reasonably well, since the polytope is a convex body, thus a single Markov trajectory (no splitting) will typically converge to $\mathcal{X}^*$. We consider three models.

First Model We take

$$\boldsymbol{A} = \begin{matrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{matrix} \quad \text{and } \boldsymbol{b} = \begin{matrix} 0.7 \\ 0.7 \\ 0.7 \\ 1.4 \\ 0.7 \\ 1.4 \\ 0.7 \\ 0.7 \\ 0.7 \\ 1.4\ . \end{matrix} \tag{30}$$

Table 24 presents the performance of the splitting Algorithm 3.1 for $\boldsymbol{A}$ and $\boldsymbol{b}$ as in (30) with $N = 10,000$ and $\rho = 0.3$ for 10 independent runs. We also present the results of the CMC, for which we took a sample of $N = 10^6$.

Table 24: Performance of splitting Algorithm 3.1 for $\boldsymbol{A}$ and $\boldsymbol{b}$ as in (30) with $N = 10,000$ and $\rho = 0.3$.

Run number	$\lvert\widetilde{\mathcal{X}^*}\rvert$	CMC
1	0.0019	0.0018
2	0.0019	0.0019
3	0.0018	0.0018
4	0.0017	0.0019
5	0.0018	0.0018
6	0.0018	0.0018
7	0.0018	0.0018
8	0.0018	0.0019
9	0.0018	0.0018
10	0.0018	0.0018
Average	0.0018	0.0018

Table 25 presents the dynamics of one of the runs of the splitting Algorithm 3.1. The results are self explanatory.

Table 25: Dynamics of a run of splitting Algorithm 3.1.

t	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	2377	2377	10	5	0.2377
2	3770	3770	10	7	0.2643
3	1485	1485	10	9	0.1313
4	2654	2654	10	10	0.2234
5	10616	10616	10	10	1
6	10616	10616	10	10	1
7	10616	10616	10	10	1
8	10616	10616	10	10	1

Note that the other RAN versions, RAN1 and RAN2, and the other splitting versions, Split1-Split3, also perform quite well, that is, similar to the splitting Algorithm 3.1.

Second Model The following table presents a 41 × 40 matrix, where its first part, (40 × 40) corresponds to the first 40 column vectors of matrix $\boldsymbol{A}$ with 0-1 entries while the second part, the vector $\boldsymbol{b}$, corresponds to the last column.

0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	1	2.24
0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1.60
1	1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	2.24
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2.36
0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1.98
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	2.87
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1.60
0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	1.22
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	2.36
1	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	2.24
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	1.98
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	2.49
0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	2.36
0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1.98
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1.73
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	1	0	0	1	1	2.49
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1.73
0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	3.13

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1.85
0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1.85
0	1	1	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2.62
0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2.11
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	3.25
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0	3.00
0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2.75
0	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1.60
1	1	0	0	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	2.87
1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	1.47
0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	0	1.85
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1.73
1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1.35
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1	0	0	0	0	0	2.36
0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1.35
0	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1.98
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1.35
0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1.98
0	1	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	2.11
0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	2.49
1	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	2.49
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	1	1	2.49

We took again $N = 10,000$ and $\rho = 0.3$. The number of unit elements in each row of the matrix $\boldsymbol{A}$ was chosen randomly on the interval $2, \ldots, 20$ and the vector $\boldsymbol{b}$ was chosen by trail and error, insuring that every constraint is active and that the estimator of the volume $|\mathcal{X}^*|$ is very small (rare-event).

Running the splitting Algorithm 3.1 for 10 independent runs, we obtained the following values of the product estimator $|\widetilde{\mathcal{X}^*}|$.

$7.8E-19,\ 7.8E-19,\ 9.5E-19,\ 9.9E-19,\ 7.5E-19,\ 7.5E-19,\ 8.1E-19,\ 7.9E-19,\ 9.1E-19,\ 9.2E-19,\ 8.8E-19.$

Table 26 presents the dynamics of one of the runs of the splitting Algorithm 3.1 for the $\boldsymbol{A} = (40 \times 40)$ matrix.

Table 26: Dynamics of one of the runs of splitting Algorithm 3.1 for $\boldsymbol{A} = (40 \times 40)$ matrix.

t	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	2661	2661	14	4	0.2661
2	2545	2545	18	6	0.2391
3	2077	2077	16	8	0.2040
4	2338	2338	20	10	0.1876
5	2627	2627	23	12	0.1873
6	1921	1921	23	14	0.1828
7	2120	2120	28	16	0.1839
8	2368	2368	29	18	0.1862
9	2575	2575	28	20	0.1812
10	1744	1744	31	22	0.1693
11	1768	1768	33	24	0.1690
12	1640	1640	34	26	0.1546
13	1905	1905	35	28	0.1452
14	1388	1388	36	30	0.1214
15	1261	1261	37	32	0.1136
16	3055	3055	38	33	0.3028
17	3260	3260	38	34	0.2668
18	3154	3154	39	35	0.2419
19	2602	2602	39	36	0.2062
20	1743	1743	39	37	0.1675
21	1239	1239	40	38	0.1185
22	836	836	40	39	0.0675
23	355	355	40	40	0.0354
24	10650	10650	40	40	1.0000

As mentioned, we found that all RAN and all splitting versions perform well for estimating the volume of convex bodies. For example, RAN2 in 10 independent runs yielded the following values of the product estimator $|\widetilde{\mathcal{X}^*}|$:

$9.6E-19,\ 7.5E-19,\ 8.5E-19,\ 8.6E-19,\ 7.8E-19,\ 7.7E-19,\ 7.6E-19,\ 7.9E-19,\ 10.1E-19,\ 10.2E-19,\ 8.6E-19.$

which is quite similar to the splitting Algorithm 3.1.

Table 27 presents dynamics of one of the runs of RAN2.

Table 27: Dynamics of a run of RAN2 for $\boldsymbol{A} = (40 \times 40)$ matrix.

t	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	2616	2616	17	4	0.2616
2	2354	2354	14	6	0.2354
3	2047	2047	17	8	0.2047
4	1893	1893	21	10	0.1893
5	1882	1882	22	12	0.1882
6	1895	1895	24	14	0.1895
7	1915	1915	26	16	0.1915
8	1830	1830	29	18	0.1830
9	1838	1838	32	20	0.1838
10	1831	1831	31	22	0.1831
11	1629	1629	31	24	0.1629
12	1590	1590	33	26	0.1590
13	1473	1473	34	28	0.1473
14	1243	1243	37	30	0.1243
15	3293	3293	37	31	0.3293
16	3087	3087	37	32	0.3087
17	2918	2918	38	33	0.2918
18	2715	2715	38	34	0.2715
19	2453	2453	38	35	0.2453
20	2037	2037	39	36	0.2037
21	1592	1592	40	37	0.1592
22	1217	1217	40	38	0.1217
23	701	701	40	39	0.0701
24	369	369	40	40	0.0369
25	10000	10000	40	40	1

Third Model This model is a little larger than the **Second Model**. The following table presents a 61×30 matrix, where its first part, (60×30) corresponds to the first 60 column vectors of matrix $\boldsymbol{A}$ with 0-1 entries while the second part, the vector $\boldsymbol{b}$, corresponds to the last 61-th column.

0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	1	2.46
0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	1.76
1	1	1	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	2.46
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	2.60
0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	2.18
0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	3.16
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	1.76
0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1.34
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	2.60
1	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	2.46
0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	1	0	2.18
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	2.74
0	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	1	0	1	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	0	0	2.60
0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	2.18
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1.90
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	1	1	1	0	1	0	0	1	1	1	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	2.74
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1.90
0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1	1	0	0	1	1	1	0	0	1	1	1	0	0	0	0	1	0	1	0	0	3.44
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	2.04
0	1	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	2.04

0 1 1 0 1 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 2.88
0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 2.32
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 3.58
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1 1 3.30
0 1 0 0 0 1 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 3.02
0 0 0 0 0 1 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 0 1 0 0 1.76
1 1 0 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 3.16
1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1.62
0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 2.04
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1.90.

Recall again that we use here only a single (randomly chosen) elite.

Running the splitting Algorithm 3.1 for 10 independent runs, each with $N = 10,000$ and $\rho = 0.3$ we obtained the following values of the product estimator $|\widetilde{\mathcal{X}}^*|$.

$$1.40E-29,\ 9.89E-30,\ 1.06E-29,\ 1.34E-29,\ 1.23E-29,\ 9.81E-30,\ 1.25E-29,\ 1.10E-29,\ 8.67E-30,\ 1.17E-29.$$

Table 28 presents dynamics of one of the runs of the splitting Algorithm 3.1 for the $\boldsymbol{A} = (60 \times 30)$ matrix.

Table 28: Dynamics of one of the runs of splitting Algorithm 3.1 for $\boldsymbol{A} = (60 \times 30)$ matrix.

t	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	910	910	3	1	0.091
2	369	369	4	2	0.0369
3	392	392	5	3	0.0392
4	681	681	6	4	0.0681
5	870	870	8	5	0.087
6	830	830	10	6	0.083
7	964	964	10	7	0.0964
8	1030	1030	11	8	0.103
9	1122	1122	14	9	0.1122
10	1329	1329	14	10	0.1329
11	1525	1525	16	11	0.1525
12	1590	1590	17	12	0.159
13	1735	1735	17	13	0.1735
14	1831	1831	18	14	0.1831
15	1957	1957	20	15	0.1957
16	1830	1830	20	16	0.183
17	1859	1859	23	17	0.1859
18	1844	1844	22	18	0.1844
19	1819	1819	23	19	0.1819
20	1738	1738	24	20	0.1738
21	1633	1633	24	21	0.1633
22	1606	1606	25	22	0.1606
23	1434	1434	26	23	0.1434
24	1314	1314	27	24	0.1314
25	1223	1223	28	25	0.1223
26	1055	1055	29	26	0.1055
27	877	877	30	27	0.0877
28	775	775	29	28	0.0775
29	535	535	30	29	0.0535
30	210	210	30	30	0.021
31	10000	10000	30	30	1

5 Conclusions and Further Research

We showed that the performance of the classic *randomized* (RAN) algorithms for counting NP-hard problems, like counting the number of satisfiability assignments in a SAT problem, counting the number of feasible colorings in a graph and calculating the permanent, can be dramatically improved by combining them with the classic *splitting* method. We presented 4 combined algorithms, which we called splitting algorithms. The most advanced splitting version is Algorithm 3.1, called also the *cloning algorithm*, suggested in (Rubinstein, 2008). We showed numerically that

- In the sequence

  RAN $\rightarrow$ Split1 $\rightarrow$ Split2 $\rightarrow$ Split3 $\rightarrow$ cloning

  the statistical properties of the counting estimators of $|\mathcal{X}^*|$ substantially improve. Two main breakthroughs occur when one moves from from RAN to Split1 and then from Split2 to Split3. The first is associated with the introduced *splitting* while second one is due to removing labeling at each sub-space $\mathcal{X}_j$. We showed that without splitting, as in RAN the algorithm is typically stacked at some intermediate level m_t, while enforcing labeling (even with splitting), as in Split1 and Split2, the algorithm is typically trapped in a local extremum. In short, both splitting and the not labeling issues must adopted simultaneously.

- For most of our case studies the original RAN algorithm typically fails; it either does not converge at all [is stacked at some intermediate level m_t, $(m_t < m)$], or is heavily biased (converges to a local extremum). Exceptions are convex counting problems, like estimating the volume of a convex plytope. Split3 and the splitting Algorithm 3.1 converge to the true counting quantity. In particular, within the limit of, say CPU = 0.5 hour, each of the parameters n and m should not exceed the value 100.

- As compared to the randomized algorithms, the proposed splitting algorithms require very little warm-up time when running the Gibbs sampler from iteration to iteration, since the underlying Markov chains are already in steady-state from the beginning. The only purpose remaining for them is fine tuning-to keep the Markov chains in steady-state while moving from iteration to iteration. The fine tuning is performed by introducing the splitting and burn-in parameters η and b, respectively. As result we obtain dramatic variance reduction and thus, dramatic speedup as compared to the randomized algorithms.

As for further research we intend to

- Establish solid mathematical grounding based on the splitting method (Garvels, 2000) and on use of the Feynman-Kac formulae and on interacting particle systems (Del Moral, 2004). We argue that since the splitting algorithms and in particular the basic Algorithm 3.1 need only fine tuning (in order for the associated Markov processes to be kept in the steady-state from iteration to iteration), the prove of its convergence and (polynomial) speed of convergence for quite general counting problems could be established by using arguments similar to these we used in Appendix under simplifying assumptions.

- Find the total sample size and the size of the elites at each iteration of the splitting Algorithm 3.1, while estimating the rare-event probability (13), that is

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

  which involves a sum of *dependent* Ber(1/2) random variables $C_i(\boldsymbol{X})$, $i = 1, \ldots, m$. Recall that the goal of estimator of ℓ is to insure approximate

uniformity of the sample at each sub-region $\mathcal{X}_t$ via formula $|\mathcal{X}^*| = \ell|\mathcal{X}|$, where ℓ is given in (7)-(9). Note that both, the CE and the exponential change of measure have limited applications in this case.

- Investigate the convergence properties of same alternatives to the Gibs splitting method, like the discrete hit-and-run (Baumert et al, 2008) combined with splitting.

6 Appendix

6.1 Complexity of Randomized Algorithms

A randomized algorithm is said to give an (ε, δ)-*approximation* for a parameter z if its output Z satisfies

$$\mathbb{P}(|Z - z| \leq \varepsilon z) \geq 1 - \delta, \tag{31}$$

that is, the "relative error" $|Z - z|/z$ of the approximation Z lies with high probability ($> 1 - \delta$) below some small number ε.

One of the main tools in proving (31) for various randomized algorithms is the so-called *Chernoff bound*, which states that for any random variable Y and any number a

$$\mathbb{P}(Y \leq a) \leq \min_{\theta > 0} \mathrm{e}^{\theta a}\, \mathbb{E}[\mathrm{e}^{-\theta Y}]\ . \tag{32}$$

Namely, for any fixed a and $\theta > 0$ define the functions $H_1(z) = I_{\{z \leq a\}}$ and $H_2(z) = \mathrm{e}^{\theta(a-z)}$. Then, clearly $H_1(z) \leq H_2(z)$ for all z. As a consequence, for any θ,

$$\mathbb{P}(Y \leq a) = \mathbb{E}[H_1(Y)] \leq \mathbb{E}[H_2(Y)] = \mathrm{e}^{\theta a}\, \mathbb{E}[\mathrm{e}^{-\theta Y}]\ .$$

The bound (32) now follows by taking the smallest such θ.

An important application is the following.

Theorem 6.1 *Let $X_1, \ldots, X_n$ be iid* $\mathsf{Ber}(\mathsf{p})$ *random variables, then their sample mean provides an (ε, δ)-approximation for p, that is,*

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - p\right| \leq \varepsilon p\right) \geq 1 - \delta, \tag{33}$$

provided $n \geq 3\ln(2/\delta)/(p\varepsilon^2)$.

For prove see, for example, (Rubinstein and Kroese, 2007).

Definition 6.1 (FPRAS) A randomized algorithm is said to provide a *fully polynomial randomized approximation scheme (FPRAS)* if for any input vector $\boldsymbol{x}$ and any parameters $\varepsilon > 0$ and $0 < \delta < 1$ the algorithm outputs an (ε, δ)-approximation to the desired quantity $z(\boldsymbol{x})$ in time that is, polynomial in $\varepsilon^{-1}, \ln \delta^{-1}$ and the size n of the input vector $\boldsymbol{x}$.

Note that the sample mean in Theorem 6.1 provides a FPRAS for estimating p. Note also that the input vector $\boldsymbol{x}$ consists of the Bernoulli variables $X_1, \ldots, X_n$.

There exists a fundamental connection between the ability to sample *uniformly* from some set $\mathcal{X}$ and counting the number of elements of interest. Since exact uniform sampling is not always feasible, MCMC techniques are often used to sample *approximately* from a uniform distribution.

Definition 6.2 **ε- uniform sample** *Let Z be a random output of a sampling algorithm for a finite sample space $\mathcal{X}$. We say that the sampling algorithm generates an ε-*uniform sample *from $\mathcal{X}$ if, for any $\mathcal{Y} \subset \mathcal{X}$*

$$\left|\mathbb{P}(Z \in \mathcal{Y}) - \frac{|\mathcal{Y}|}{|\mathcal{X}|}\right| \leq \varepsilon. \tag{34}$$

Definition 6.3 **Variation distance** *The variation distance between two distributions F_1 and F_2 on a countable space $\mathcal{X}$ is defined as*

$$||F_1 - F_2|| = \frac{1}{2}\sum_{\boldsymbol{x}\in\mathcal{X}} |F_1(\boldsymbol{x}) - F_2(\boldsymbol{x})|. \tag{35}$$

It is well-known, (Mitzenmacher and Upfal, 2005) that the definition of variation distance coincides with that of an ε- uniform sample in the sense that a sampling algorithm returns an ε- uniform sample on $\mathcal{X}$ if and only if the variation distance between its output distribution F and the uniform distribution $\mathcal{U}$ satisfies

$$||F - \mathcal{U}|| \leq \varepsilon.$$

Bounding the variation distance between the uniform distribution and the empirical distribution of the MC obtained after some warm-up period is a crucial issue while establishing the foundations of RA since with a bounded variation distance one can produce efficient approximation for $|\mathcal{X}^*|$.

Definition 6.4 (FPAUS) A sampling algorithm is called a *fully polynomial almost uniform sampler (FPAUS)* if, given an input vector $\boldsymbol{x}$ and a parameter $\varepsilon > 0$, the algorithm generates an ε-uniform sample from $\mathcal{X}(\boldsymbol{x})$ and runs in a time that is, polynomial in $\ln \varepsilon^{-1}$ and the size of the input vector $\boldsymbol{x}$.

An important issue is to prove that given an FPAUS for a combinatorial problem, one can construct a corresponding FPRAS.

Example 6.1 (FPAUS and FPRAS for Independent Sets) An FPAUS for independent sets takes as input a graph $G = (V, E)$ and a parameter $\varepsilon > 0$. The sample space $\mathcal{X}$ consists of all independent sets in G with the output being an ε-uniform sample from $\mathcal{X}$. The time to produce such an ε-uniform sample should be polynomial in the size of the graph and $\ln \varepsilon^{-1}$. Based on the product formula (1) Mitzenmacher and Upfal, (2005) prove that given an FPAUS one can construct a corresponding FPRAS.

We present next a theorem (based on the coupling arguments of MC), taken from Mitzenmacher and Upfal, (2005).

Theorem 6.2 *Given an n-vertex graph with maximum degree Δ (maximal number of neighbors for any $v \in V$), the mixing time $\tau(\varepsilon)$ of the graph coloring Algorithm 2.3 satisfies*

$$\tau(\varepsilon) \leq \frac{nq}{q - 2\Delta} \ln(\frac{n}{\varepsilon}),$$

provided $q \geq 2\Delta + 1$.

Recently (Hayes, Vera and Vigoda, 2007) improved the results of Theorem 6.2 for coloring of planar graphs. In particular they proved that one can still get polynomial mixing time of the MCMC by taking the number of colors q less than the maximum degree Δ.

6.2 Complexity of Splitting Method under Simplifying Assumptions

As before, we denote

ℓ- the rare-event probability.

m - number of levels. $m_t,\ t = 0, 1, \ldots, T$ - intermediate levels ($m_0 = 0, m_T = m$).

c_t - probability of hitting m_t, starting from m_{t-1}.

N_t - number of successful hits of level m_t (elite sample).

$N^{(t)}$ - total sample from level m_t (here, unlike (10)-(11) we assume that the sample size depends on t).

Our main goal is to present a calculation for $\mathrm{Var}\widehat{\ell}$, where $\widehat{\ell}$ is given in (10)-(11) (with N replaced by $N^{(t)}$) and to show how much variance can be obtained with splitting versus the *crude Monte Carlo* (CMC). To do so we shall cite some basic results from (Garvels 2000).

Let us write N_t (the number of elite samples) as

$$N_t = \sum_{i=1}^{N^{(t)}} I_i^{(t)},$$

where each $I_i^{(t)},\ i = 1, \ldots, N^{(t)};\ t = 1, \ldots, T$ represents the indicator of successes at the t-th stage, that is, $I_i^{(t)}$ is the indicator that the process reaches the level m_t from level m_{t-1}. We assume that

1. The indicators $I_i^{(t)},\ i = 1, \ldots, N^{(t)};\ t = 1, \ldots, T$ are generated using splitting Algorithm 3.1 or the enhanced one, Algorithm 3.2.

2. For fixed m_t the indicators $I_i^{(t)}$ are iid and $\mathbb{E}I_i^{(t)} = c_t$. In addition, we assume that for all combinations (i, j) and for $t \neq k$, they are also independent level-wise, that is, $\mathbb{E}I_i^{(t)} I_j^{(k)} = c_t c_k$. Clearly, this is a simplified assumption, since in practice we generate only approximately uniform samples at each sub-space $\mathcal{X}_t$. As a result, each estimator of c_t might be slightly biased. Recall that $c_t = \mathbb{P}(\mathcal{X}_t | \mathcal{X}_{t-1})$, that is, it represents the conditional probability of the process (particle) reaching the sub-space (event) $\mathcal{X}_t$ from $\mathcal{X}_{t-1}$, or in other words of it reaching the level m_t from level m_{t-1}. Moreover, the indicators $I_i^{(t)}$ and $I_j^{(k)}$ might be slightly correlated as well, in particular when k is close to t, say $k = t \pm 1$. We use the phrases "slightly biased" and "slightly correlated" being aware that our elites samples remain in "near" *steady-state (warm-up)* position in each sub-space $\mathcal{X}_t$. Recall that we do so by introducing the splitting and the burn-in parameters, b and η, respectively.

With this on hand, we have

$$\mathrm{Var}\widehat{\ell} = \mathbb{E}\widehat{\ell}^2 - \ell^2 = \mathbb{E}\prod_{t=1}^{T} \widehat{c}_t^2 - \ell^2. \tag{36}$$

Taking into account that

$$\mathrm{Var}\widehat{c}_t = \frac{1}{N^{(t)}} c_t(1 - c_t), \tag{37}$$

we obtain

$$\mathrm{Var}\widehat{\ell} = \prod_{t=1}^{T} \left\{ \frac{c_t(1 - c_t)}{N^{(t)}} + c_t^2 \right\} - \ell^2 = \ell^2 \left(\prod_{t=1}^{T} \left\{ \frac{1 - c_t}{c_t N^{(t)}} + 1 \right\} - 1 \right). \tag{38}$$

As a simple example consider estimation of ℓ in (24), that is estimation

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

where the X_i's are iid, each $X_i \sim$ Ber(1/2). Assume that $T = m$ and $N_t = N,\ t = 1, \ldots, m$. It is readily seen that in this case $c_t = 1/2$ and thus (38) reduces to

$$\mathrm{Var}\widehat{\ell} = \ell^2 \left(1 + \frac{1}{N}\right)^m - \ell^2. \tag{39}$$

For large m and $N = m$ we obtain that

$$\mathrm{Var}\widehat{\ell} \approx \ell^2 (e - 1). \tag{40}$$

We proceed next with (38). To find the optimal parameters T, and $(N_1, \ldots, N_T)$ in (38) we solve the following minimization problem

$$\min \mathrm{Var}\widehat{\ell} = \min \ell^2 \left(\prod_{t=1}^{T} \left\{ \frac{(1 - c_t)}{c_t N^{(t)}} + 1 \right\} - 1 \right) \tag{41}$$

with respect to T, and $(N_1, \ldots, N_T)$, subject to the following constraint

$$\sum_{t=1}^{T} N^{(t)} = M. \tag{42}$$

It is not difficult to show that for fixed T, the solution of (41)-(42) (see Garvels, 2000) is $c_t = c = \ell^{\frac{1}{T}}$ and $N^{(t)} = \frac{M}{T},\ \forall\ k = 1, \ldots, T$. Mote that we assumed that $\sum_{t=1}^{T} N^{(t)}$ is large and in solving (41)-(42) we used a continuous approximation for the true discrete program (41)-(42). It follows that for fixed T, the minimal variance (under $c_t = c = \ell^{\frac{1}{T}}$ and $N^{(t)} = \frac{M}{T},\ \forall\ k = 1, \ldots, T$) equals

$$\left\{ \frac{\ell^2 T^2 (1 - \ell^{\frac{1}{T}})}{\ell^{\frac{1}{T}} M} \right\}.$$

It remains to solve

$$\min_T \mathrm{Var}\widehat{\ell} = \min_T \left\{ \frac{\ell^2 T^2 (1 - \ell^{\frac{1}{T}})}{\ell^{\frac{1}{T}} M} \right\}. \tag{43}$$

It is readily seen that under the above simplifying assumptions the optimal values of T and c, the minimal variance, optimal squared relative error and optimal efficiency, denoted $T_r,\ c_r,\ \mathrm{Var}_r\widehat{\ell},\ \kappa_r^2,$ and ε_r are

$$T_r = -\frac{\ln \ell}{2}, \tag{44}$$

$$c_r = e^{-2}, \tag{45}$$

$$\mathrm{Var}_r\widehat{\ell} = \frac{(e\ell \ln \ell)^2}{4M}, \tag{46}$$

$$\kappa_r^2 = \frac{M^{-1}\mathrm{Var}_r\widehat{\ell}}{\ell^2} \approx \frac{(e \ln \ell)^2}{4}, \tag{47}$$

and

$$\varepsilon_r = \frac{M^{-1}\mathrm{Var}_r\widehat{\ell}}{\ell(1 - \ell)} \approx \frac{\ell (e \ln \ell)^2}{4}, \tag{48}$$

respectively. Confidence intervals and central limit theorems can be also readily established.

Even though the assumptions are indeed simplifying, they provide good insight into the polynomial complexity of splitting Algorithm 3.1 and in particular into the relative error κ_r^2. Note that for general counting problem, our numerical data in Section 4 are in agreement with these results and in particular with κ_r^2 in (47). Note finally, that for the Bernoulli model with $\mathrm{Var}\widehat{\ell} \approx \ell^2(e-1)$ (see (40)) we obtain (under the above simplifying assumptions) that $\kappa_r^2 = e - 1$ and thus, bounded relative error.

Acknowledgments

I would like to thank Radislav Weissman, and Alexander Libster for performing the computational part of the paper.

References

[1] Asmussen S. and P.W. Glynn, *Stochastic Simulation: Algorithms and Analyses*, Springer, 2007.

[2] S. Baumert, A. Ghate, S. Kiatsupaibul, Y. Shen, R. L. Smith and Z. B. Zabinsky, "Discrete Hit-and-Run for Sampling Points from Arbitrary Distributions over Subsets of Integer Hyper-rectangles," Operations Research, 2009.

[3] Z. I. Botev and D. P. Kroese, "An Efficient Algorithm for Rare-event Probability Estimation, Combinatorial Optimization, and Counting". *Methodology and Computing in Applied Probability*, 2008.

[4] Del Moral, P., *Feynman-Kac formulae, genealogical and interacting particle systems with applications*. New York: Springer, 2004.

[5] M.J.J. Garvels, The splitting method in rare-event simulation, Ph.D. thesis, University of Twente, 2000.

[6] P. Del Moral, *Feynman-Kac Formulae Genealogical and Interacting Particle Systems*, Springer, 2004.

[7] T. Hayes, J. Vera and E. Vigoda. "Randomly coloring planar graphs with fewer colors than the maximum degree". STOC 2007.

[8] Kahn H and T.E. Harris, Estimation of Particle Transmission by Random Sampling, Mational Bureau of Standards Applied Mathematics Series, 1951.

[9] P. L'Ecuyer, V. Demers, and B. Tuffin, "Rare-Events, Splitting, and Quasi-Monte Carlo", ACM Transactions on Modeling and Computer Simulation, 17, 2, 2007.

[10] M. Mitzenmacher and E. Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York (NY), 2005.

[11] R. Motwani and R. Raghavan. *Randomized Algorithms* Cambridge University Press, 1997.

[12] S. M. Ross *Simulation*, Academic Press, 4rd Edition, 2006.

[13] R. Y. Rubinstein. " The Gibbs Cloner for Combinatorial Optimization, Counting and Sampling" *Methodology and Computing in Applied Probability*, (to appear), 2008.

[14] R. Y. Rubinstein. "Entropy and Cloning Methods for Combinatorial Optimization, Sampling and Counting Using the Gibbs Sampler". (To be published in *Information Theory and Statistical Learning*, Springer, 2008a).

[15] R. Y. Rubinstein and D. P. Kroese, *The Cross-Entropy Method: a Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*, Springer, 2004.

[16] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo Method; Second Edition*, Wiley, 2007.