Stochastic Enumeration and Splitting Methods for Counting Self-Avoiding Walks

Reuven Rubinstein

Faculty of Industrial Engineering and Management,

Technion, Israel Institute of Technology, Haifa, Israel

ierrr01@ie.technion.ac.il

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

Abstract

We present a new method for counting self-avoiding walks (SAW's), called *stochastic enumeration* (SE), a stochastic replica of the naive (computationally intractable) full enumeration method. Also presented is a new approach, called OSLA-SPLIT, a combination of classic splitting with importance sampling. The latter is based on the well-known method, one-step-look-ahead (OSLA) method. Polynomial convergence is proved for both SE and OSLA-SPLIT. Our simulation studies show that SE is able to counts reasonable fast SAW's of length up to $1,000$, while the current state of art is ≈ 100.

Contents

1 Introduction **2**

2 Counting SAW's **3**
2.1 Splitting Algorithm for SAW's 4
2.2 OSLA-SPLIT Algorithm for SAW's 7
2.3 Stochastic Enumeration Algorithm for SAW's 10

3 Convergence **13**

4 Numerical Results **15**
4.1 Splitting Algorithm 2.1 15
4.2 OSLA-SPLIT Algorithm 2.2 17
4.3 SE Algorithm 2.3 17

5 Concluding Remarks and Further Research **19**

6 Appendix **19**
6.1 OSLA for Self-Avoiding Walk 19
6.2 The Splitting Method 20

1 Introduction

There are two well-known methods for rare-event simulation and counting. They are called *splitting* and *importance sampling* (IS).

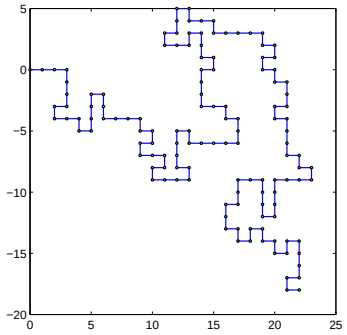
The splitting method dates back to Kahn and Harris [16]and Rosenbluth and Rosenbluth [26]. Since then hundreds of insightful papers have been written on that topic. We refer the reader to [4]-[19], and also to the excellent papers by Glasserman et al. [13], Cerou et al. [6] and Melas [22], which contain extensive valuable material as well as a detailed list of references. Recently, the connection between splitting for Markovian processes and *interacting particle methods* based on the Feynman-Kac model with a rigorous framework for mathematical analysis has been established in Del Morall's [7]) monograph.

Importance sampling is subject of almost any standard book on Monte Carlo simulation (see, for example, [30]). It is well known that a straight-forward application of importance sampling typically yields very poor approximations of the quantity of interest. It is shown in Gogate and Dechter [14], [15] that poorly chosen importance sampling in graphical models and in particular for satisfiability models generates many useless zero weight samples, which are often rejected yielding an inefficient sampling process. To address this problem which is called the problem of losing trajectories, the above authors propose a clever sample search method, the so-called *SampleSearch*, which is integrated into the importance sampling framework. They also derive a lower bound for the unknown counting quantity. Their empirical studies demonstrate the superiority of their method over its competitors.

In this work we present two new algorithms for fast counting *self-avoiding walks* (SAW's). One, called *stochastic enumeration* (SE), is a stochastic replica of the naive *full enumeration*. The other, called *OSLA-SPLIT*, is a combination of classic splitting with importance sampling, which in turn is based on the well-known *one-step-look-ahead* (OSLA) method (see Appendix).

A SAW of length $n = 130$ is given in Figure 1.

Figure 1: A SAW of length $n = 130$



We prove polynomial complexity of both SE and OSLA-SPLIT algorithms and present supportive numerical studies. In particular we show that SE is superior to OSLA-SPLIT and that it can handle problems of size $n = 1,000$, while the current state of the art [21] is $n \approx 100$.

The rest of the paper is organized as follows. In Section 2, which is the main one we present three algorithms; the splitting, OSLA-SPLIT and SE ones. Section 3 deals with the convergence issues of OSLA-SPLIT and the SE algorithms. In Section 4 we present supportive numerical results. Section 5 gives some conclusions and finally, in the Appendix we present some background on the OSLA and cloning methods.

2 Counting SAW's

In this section we present the splitting, OSLA-SPLIT and SE algorithms. The first two are similar to the cloning algorithm [28] (see Appendix) with a single major difference: they do not rely on the time consuming MCMC method and in particular on the Gibbs sampler. As in [28] we denote by N_t, ρ_t and $N_t^{(e)}$ the sample size, the adaptive rarity parameter and the number of elites at iteration t, respectively. Note that the number of elites equals $N_t^{(e)} = \lceil N_t \rho_t \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the largest integer. At iteration t we split each elite sample $\eta_t = \left\lceil \rho_t^{-1} \right\rceil$ times. By doing so we generate $\left\lceil \rho_t^{-1} N_t^{(e)} \right\rceil \approx N_t$ new samples for the next iteration $t+1$. The rationale is based on the fact that if all ρ_t are not small, say $\rho_t \geq 0.01$, we have enough elite samples $N_t^{(e)}$ to generate approximately N_t *new* stationary samples for the next level.

The final estimator of the number of SAW's, denoted by $|\mathcal{X}^*|$ is [28]

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0| \prod_{t=1}^{T} \widehat{c}_t, \tag{1}$$

where

$$\widehat{c}_t = \frac{N_t^{(e)}}{N_t} \tag{2}$$

and $|\mathcal{X}_0| = 4^n$.

2.1 Splitting Algorithm for SAW's

The splitting algorithm for SAW's, which is similar to [28], can be written as

Algorithm 2.1 (Splitting Algorithm for SAW) Given the initial parameter ρ_1, say $\rho_1 \in (0.01, 0.25)$ and the initial sample size N_1 execute the following steps:

1. **Generation of SAW's** Simulate randomly N_1 SAW's starting at 00 until each of them stops, that is, until each of the N_1 processes becomes *non-SAW*. Denote them by $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_1}$. Set a counter $t = 1$.

2. **Acceptance-Rejection** Let $\widetilde{\mathcal{X}}_1 = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_1^{(e)}}\}$ be the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_1}\}$, the elite samples for which $S(\boldsymbol{X}_i) \geq \widehat{m}_1$, where $\widehat{m}_1$ is the $(1-\rho_1)$ sample quantile of the ordered statistic values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_{N_1})$ and $S(\boldsymbol{X})$ denotes the length of a SAW. Take

$$\widehat{c}_1 = \widehat{\ell}(\widehat{m}_1) = \frac{1}{N_1} \sum_{i=1}^{N_1} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_1\}} = \frac{N_1^{(e)}}{N_1} \tag{3}$$

   as an *unbiased* estimator of c_1.

3. **Moving back** Move all elites, including those which have reached level $\widehat{m}_1$ and higher, to $\widehat{m}_1 - 1$. This step is necessary for all elite walks to become SAW and to start them from the same level $\widehat{m}_1 - 1$.

4. **Splitting** Let $\widetilde{\mathcal{X}}_{t-1} = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}\}$ be the elite sample of SAW's at iteration $(t-1)$, that is the subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_{t-1}}\}$ for which $S(\boldsymbol{X}_i) \geq \widehat{m}_{t-1}$. Reproduce $\eta_{t-1} = \lceil \rho_{t-1}^{-1} \rceil - 1$ times each SAW $\widetilde{\boldsymbol{X}}_k$ of the elite sample $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}\}$, that is take η_{t-1} identical copies of each vector $\widetilde{\boldsymbol{X}}_k$. Denote the entire new population, that is the new $\eta_{t-1} N_{t-1}^{(e)}$ splitting vectors plus the original elite sample $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}\}$, by $\mathcal{X}_{cl} = \{(\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_1), \ldots, (\widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}})\}$. Continue generating SAW's for each member of the population $\mathcal{X}_{cl}$. Denote by $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_t}\}$ the *entire new* population of SAW's.

5. **Estimating** c_t Take $\widehat{c}_t = \frac{N_t^{(e)}}{N_t}$ (see (2)) as an estimator of c_t.

6. **Stopping rule** If $m_t = n$, go to step 7, otherwise set $t = t+1$ and repeat from step 2.

7. **Final Estimator** Deliver (1), that is

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0| \prod_{t=1}^{T} \widehat{c}_t = |\mathcal{X}_0| \prod_{t=1}^{T} \frac{N_t^{(e)}}{N_t} \tag{4}$$

as an estimator of $|\mathcal{X}^*|$.

Remark 2.1 To speed up Algorithm 2.1 we can define the levels $\widehat{m}_t$ based on a small pilot run, denoted by $N_t^{(p)}$ and such that $N_t^{(p)} << N_t$. By doing so Step 3 **Moving back** becomes redundant.

Remark 2.2 Instead of sampling uniformly N parallel processes from the origin 00, one can adopt the following policy

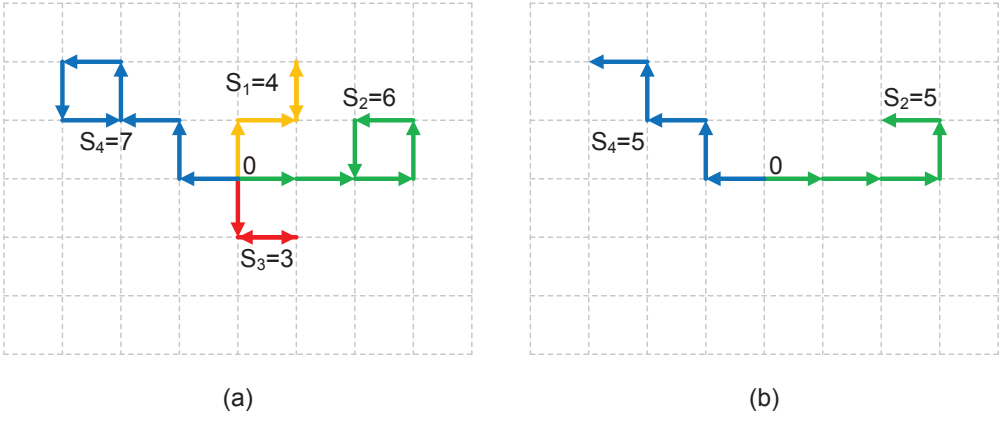
1. Choose a small number m_0, say $m_0 = 4$ and find via full enumeration all different SAW's starting from the origin 00 . Denote them by $|\mathcal{X}_{m_0}^*|$ and select them as the elite samples, that is set $N_0^{(e)} = |\mathcal{X}_{m_0}^*|$. For example, for $m_0 = 4$ the number of elites $N_0^{(e)} = |\mathcal{X}_{m_0}^*| = 100$.

2. Set the zero level to m_0 and proceed with Algorithm 2.1.

We now demonstrate Algorithm 2.1 for $N = 4$ and $\rho = 1/2$, using Figures 2 and 3 showing the first and second iterations, respectively.

1. **First iteration** Following Algorithm 2.1 we start at point (0,0) and generate $N = 4$ random trajectories. Assume that the results are $S_1 = 4, S_2 = 6, S_3 = 3, S_4 = 7$, where S_i denotes the length of the i-th SAW including the last unsuccessful movement. This corresponds to part (a) of Figure 2.

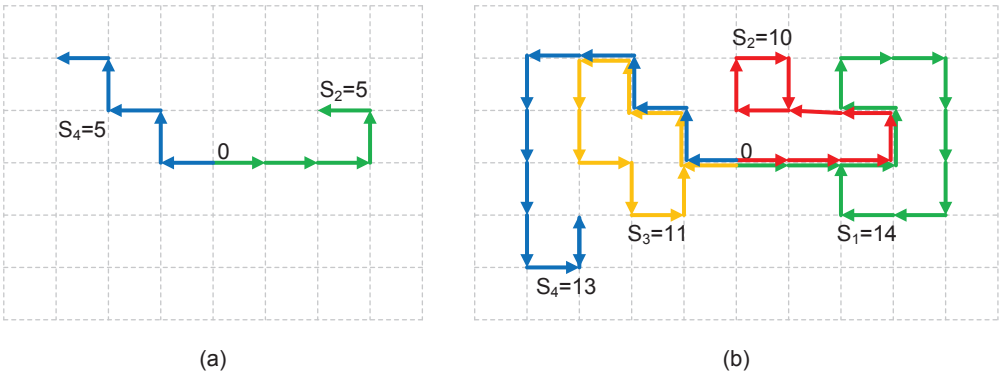
Figure 2: First Iteration of Algorithm 2.1



Clearly, $S_1 = 4$ and $S_3 = 3$ must be discarded, while $S_2 = 6$, and $S_4 = 7$ represent elite samples and must be kept. The first level reached is clearly $m_1 = 5$ (not $m_1 = 6$, because of the unsuccessful last movement). Part (b) of Figure 2 represents exactly those remaining two trajectories S_2 and S_4, each truncated to length 5.

2. **Second Iteration** Following Algorithm 2.1 we generate $N = 4$ random trajectories, proceeding from the trajectories S_2 and S_4, each of which is split by a factor of two. Note that for convenience the trajectories S_2 and S_4 are sown again in part (a) of Figure 3. The resulting 4 trajectories with length $S_1 = 14, S_2 = 10, S_3 = 11, S_4 = 13$ are shown in part (b). Clearly, to start the third trajectory we should set the second level $m_2 = 12$. This implies that we should retain trajectories $S_1 = 14$ and $S_4 = 13$ and discard S_2 and S_3.

Figure 3: The Second Iteration of Algorithm 2.1



Remark 2.3 As an alternative to the (deterministic) splitting step 4 in Algorithm 2.1 one can consider the following random one:

At iteration t, $t = 1, \ldots, T$ take a sample of size $N - N_t^{(e)}$ from the discrete uniform random $\mathcal{U}(1, \ldots, N_t^{(e)})$, $t = 1, \ldots, T$. Let R_j be the number of times for the subscript j, $j = 1, \ldots, N_t^{(e)}$. Split R_j times each elite trajectory j.

Note that

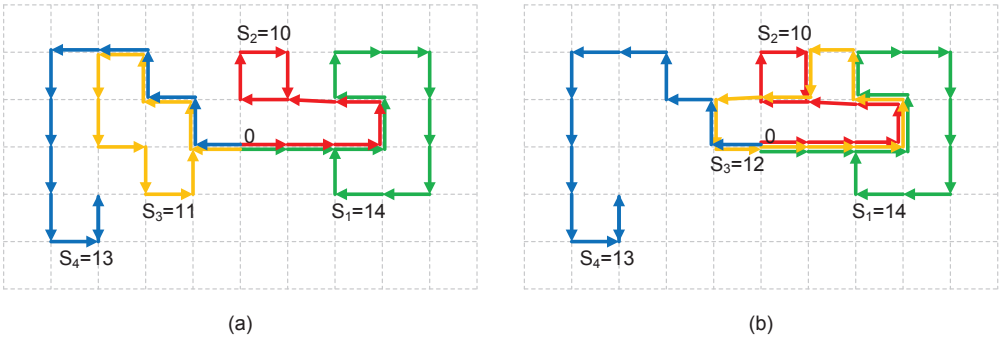
$$\sum_{j=1}^{N_t^{(e)}} R_j = N - N_t^{(e)}. \tag{5}$$

Note also that this alternative assumes that $N_t = N, \forall t = 1, \ldots, T$, that is coincides with (1). We shall refer to the original splitting estimator (1) and the one involving (5), as the *systematic* and *random* ones, respectively. If not stated otherwise we assume systematic splitting, since we found numerically that it is superior to the random one.

In part (b) of Figure 4 we present 4 trajectories for the second iteration of the splitting algorithm with random splitting. In particular, we split the first trajectory 3 times and the second only once. For convenience we reproduce

here (in part (a)) from Figure 3 the 4 trajectories corresponding to systematic splitting.

Figure 4: 4 Trajectories for Second Iteration with Systematic Splitting (part (a)) and Random Splitting (part (b))



Note that if we set in (1) the number of levels equal to n (non-adaptive updating) we obtain the famous Feynman-Kac splitting model extensively treated in the works of Cerou, Del Moral, Doucet and others [4]-[8]. In their terminology, the original acceptance-rejection Step 2 of Algorithm 2.1 is called the *selection* step, while the modified splitting step 4 (based on (1) and (5)) is called the *mutation* step. For nice surveys on the Feynman-Kac model, particle filters and sequential Monte Carlo, see [3], [8] and [9].

Our numerical results below show that the systematic splitting Algorithm 2.1 is slightly faster and more accurate than its random counterpart.

Remark 2.4 An alternative splitting estimator for rare event estimation is given in [6]. It is

$$\widetilde{\ell} = rc^T, \tag{6}$$

where $c = N^{(e)}/N$, r is the proportion of trajectories (elites), which reached the desired level n at iteration $T+1$, N denotes the number of samples which are simulated, and $N^{(e)}$ denotes the number of elites that are kept from one step to another. Note that

- The sample size N and the number of elites $N^{(e)}$ (and thus $c = N^{(e)}/N$) are fixed in advance.
- The elites with length of trajectories exceeding the threshold m_t are not moved back to the level m_t as in Algorithm 2.1, but are retained (as competetors) for the next iteration corresponding to the level m_{t+1}.

2.2 OSLA-SPLIT Algorithm for SAW's

To speed up the splitting Algorithm 2.1, we shall combine it with IS, and in particular with the OSLA (one-step-look-ahead) Algorithm 6.1 given in Section

6.1 of the Appendix. We call such enhanced algorithm, the *OSLA-SPLIT* algorithm. Note that the OSLA Algorithm 6.1 is able to handle (in manageable time) problems of small sizes, say up to $n = 100$, while OSLA-SPLIT is capable handles problems of $n = 500$.

Regarding OSLA-SPLIT the immediate temptation would be to replace the original splitting estimator $\widehat{c}_t = \frac{N_t^{(e)}}{N_t}$ (see (2)) by its OSLA counterpart

$$\widetilde{c}_t = \frac{1}{N_t^{(o)}} \sum_{i=1}^{N_t^{(o)}} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_t\}} w(\boldsymbol{X}_i) = \frac{1}{N_t^{(o)}} \sum_{i=1}^{N_t^{(oe)}} w(\boldsymbol{X}_i), \tag{7}$$

while all other data remaining unchanged. Here $N_t^{(oe)}$ and $N_t^{(o)}$ denote the respective number of elites and actual sample size in the OSLA estimator, and $w(\boldsymbol{X}_i)$ represents the weight factor associated with the i-th elite trajectory of OSLA and is given in (26) in the Appendix. Note that superscripts (oe) and (o) serve to distinguish the values from $N_t^{(e)}$ and N_t in the Algorithm 2.1.

Unfortunately, such an estimator of $|\mathcal{X}^*|$ (see (1)) with $\widehat{c}_t$ replaced by $\widetilde{c}_t$ is bound to fail mainly because - unlike Algorithm 2.1, where all $N_t^{(e)}$ elite trajectories are *uniformly* distributed in the corresponding subspace $\mathcal{X}_t$ - the $N_t^{(oe)}$ elites are no longer so.

To overcome this difficulty we suggest to run the original and the enhanced versions in parallel. Both version must be synchronized in terms of *the number of elites and the level- crossing value* m_t. More specifically:

- We require that $N_t^{(oe)} = N_t^{(e)}$ for all t. This can be achieved by making sure first that $N_t^{(e)} \geq N_t^{(oe)}$ and then discarding randomly $\Delta_t = N_t^{(e)} - N_t^{(oe)}$ trajectories from $N_t^{(e)}$. We need not to assume, however, $N_t^{(o)} = N_t$. In fact, we take $N_t^{(o)} > N_t$.

- We accept the estimator $\widetilde{c}_t$ as per (7) with a single major difference: we start each iteration t in OSLA-SPLIT with $N_{t-1}^{(e)}$ elites from Algorithm 2.1, rather than with $N_{t-1}^{(oe)}$ ones from OSLA. This is the same as to say that at the end of each iteration we dispense with $N_t^{(oe)}$ elites replacing them with $N_t^{(e)}$ ones to proceed to iteration $t + 1$.

The OSLA-SPLIT, algorithm can be written as follows:

Algorithm 2.2 (OSLA-SPLIT Algorithm for SAW's) Given the initial parameter ρ_1, say $\rho_1 \in (0.01, 0.25)$ and the sample size $N = N_1^{(o)}$, execute the following steps:

1. **Generation of SAW's** Using OSLA Algorithm 6.1, simulate a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_1}$ of SAW's starting at 00 until each of them stops (stacked at some intermediate state), that is until each of the $N_1^{(o)}$ processes becomes *non-SAW*. Set a counter $t = 1$.

2. **Acceptance-Rejection** Let $\widetilde{\mathcal{X}}_1 = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_1^{(oe)}}\}$ be the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_1^{(oe)}}\}$, the elite samples for which $S(\boldsymbol{X}_i) \geq \widehat{m}_1$, where $\widehat{m}_1$ is the $(1-\rho_1)$ sample quantile of the ordered statistics values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_{N_1^{(o)}})$ and $S(\boldsymbol{X})$ denotes the length of a SAW. Take

$$\widetilde{c}_1 = \frac{1}{N_1^{(o)}} \sum_{i=1}^{N_1^{(o)}} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_1\}} w(\boldsymbol{X}_i) = \frac{1}{N_1^{(o)}} \sum_{i=1}^{N_1^{(oe)}} w(\boldsymbol{X}_i) \tag{8}$$

as an *unbiased* estimator of c_1. Here $w(\boldsymbol{X}_i)$ (see (26)) is the weight factor associated with the i-th elite trajectory, each of length $\widehat{m}_1 - 2$, that is

$$w(\boldsymbol{X}_i) = d_{1i}, \ldots, d_{(\widehat{m}_1 - 2)i} \,. \tag{9}$$

3. **Moving back** Move all elites, including the one which reached level $\widehat{m}_1$ and higher to $\widehat{m}_1 - 2$. This step is necessary for all elite walks to become SAW and to start them from the same level $\widehat{m}_1 - 2$.

4. **Splitting** Remove the $N_{t-1}^{(oe)}$ elites from OSLA and replace them with $N_{t-1}^{(e)}$ ones from Algorithm 2.1 such that $N_{t-1}^{(e)} = N_{t-1}^{(oe)}$. Note that the elites $N_{t-1}^{(e)}$ and $N_{t-1}^{(oe)}$ are synchronized for each level $\widehat{m}_{t-1}$. As in the **Splitting** step of Algorithm 2.1, reproduce $\eta_{t-1} = \lceil \rho_{t-1}^{-1} \rceil - 1$ times each elite $N_{t-1}^{(e)}$ trajectory. Continue generating SAW's for each member of the new $\eta_{t-1} = \lceil \rho_{t-1}^{-1} \rceil$ population using OSLA until all of them become non-self-avoiding. Denote by $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_t^{(o)}}\}$ the *entire new* population of SAW's.

5. **Estimation of c_t** Take $\widetilde{c}_t$ in (7) as an *unbiased* estimator of c_t. Note again that $w(\boldsymbol{X}_i)$ in (7) is the weight factor associated with the i-th elite SAW trajectory, each of length $\widehat{m}_t - \widehat{m}_{t-1}$, that is

$$w(\boldsymbol{X}_i) = d_{ti}, \ldots, d_{(\widehat{m}_t - \widehat{m}_{t-1}),i} \,. \tag{10}$$

6. **Stopping rule** (the same as in Algorithm 2.1).

7. **Final Estimator** Deliver

$$\widetilde{|\mathcal{X}^*|} = \prod_{t=1}^{T} \widetilde{c}_t \tag{11}$$

as an estimator of $|\mathcal{X}^*|$, where $\widetilde{c}_t$ is as per (7).

Instead of the adaptive Algorithm 2.1 one can use a non-adaptive one, that is, where the number of levels T is fixed and equal to n. We found that the adaptive algorithm is only a little faster than the non-adaptive one, mainly because in the adaptive case at each iteration t we have to move backward all elite trajectories exceeding m_t back to the level m_t, while in the non-adaptive

we do not need to do so. Thus, there exists a trade-off in Algorithm 2.1 between ρ and N. Clearly, if we set $\rho \approx N^{-1}$ then there is a single elite and there is no need to move backward. But then we have to split the single elite N times. We found that good performance is obtained for $0.1 \leq \rho \leq 0.25$.

Remark 2.5 Note that in contrast to the estimator $\widehat{|\mathcal{X}^*|}$ in (1) its counterpart $\widetilde{|\mathcal{X}^*|}$ in (11) does not contain the term $|\mathcal{X}_0|$. Note also that if we use fixed levels m_t instead of adaptive ones, Step 3 (**Moving Back**) is redundant. As a result we can use $\widehat{m}_1$ in (9) instead of $\widehat{m}_1 - 2$ and, in general, $\widehat{m}_t$ instead of $\widehat{m}_t - 2$. The resulting $w(\boldsymbol{X}_i)$ is thus

$$w(\boldsymbol{X}_i) = d_{ti}, \ldots, d_{\widehat{m}_t i} \,. \tag{12}$$

2.3 Stochastic Enumeration Algorithm for SAW's

This method can be viewed as a dual to OSLA-SPLIT, in the sense that we first derive the number of SAW elites $N_{t-1}^{(e)}$ (at iteration $t-1$) and only then the sample size N_t for iteration t and the corresponding SAW's. This is done by *full enumeration* between the $N_{t-1}^{(e)}$ elites that have reached the threshold level m_{t-1} and all possible candidates for the next level $m_t > m_{t-1}$. Typically one might keep $N_t^{(e)}$ fixed, while N_t is varied from iteration to iteration. It is crucial to note that in contrast to the OSLA-SPLIT Algorithm 2.2 - the *stochastic enumeration* (SE) below is not stacked at any intermediate state $m_t < n$.

Algorithm 2.3 (Stochastic Enumeration for SAW's)

1. **Full Enumeration** Select a small number m_1, say $m_1 = 4$ (see Remark 2.2) and find via *full enumeration* all different SAW's starting from the origin 00. Denote the number of elites by $N_1^{(e)}$ and call them the *elite sample*. For example, for $m_1 = 4$ the number of elites $N_1^{(e)} = 100$. Set the first level to m_1. Proceed with the $N_1^{(e)}$ elites from level m_1 to the next one $m_2 = m_1 + r$, where r is a small integer (typically $r = 1$ or $r = 2$) and find again all different SAW's. Denote their number for m_2 by N_1 and call the associated SAW's *sample size*. For example, for $m_2 = 5$ we have $N_1 = 284$ different SAW's.

2. **Calculation of the First Weight Factor** Calculate

$$\widehat{\nu}_1 = \frac{N_1}{N_1^{(e)}} \tag{13}$$

   and call it the first *weight factor*.

3. **Stochastic Enumeration** Proceed with the $N_{t-1}^{(e)}$ elites from the level m_{t-1} to the next one $m_t = m_{t-1} + r$ and derive (via full enumeration) all SAW's for iteration t. Denote by N_t the resulting number of SAW's.

4. **Calculation of the t-th Weight Factor** Select randomly $N_t^{(e)}$ SAW's from the set of N_t ones and calculate

$$\widehat{\nu}_t = \frac{N_t}{N_t^{(e)}}. \tag{14}$$

Call it the t-th *weight factor*.

5. **Stopping Rule** Proceed with steps 3-4 until convergence and deliver

$$\widehat{|\mathcal{X}^*|} = N_1^{(e)} \prod_{t=1}^{n} \widehat{\nu}_t \tag{15}$$

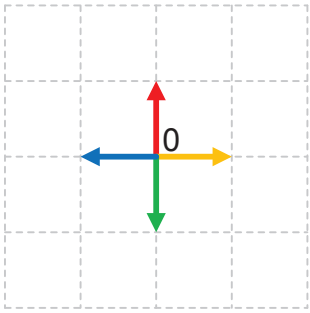
as an unbiased estimator of $|\mathcal{X}^*|$.

Remark 2.6 In analogy to OSLA-SPLIT, we can construct an OSLA-SE algorithm, where the SE part plays a role similar to that of the splitting algorithm in the parallel run.

Figures 5, 6 and 7 depict the following three steps: (i) full enumeration starting from the origin 00, (ii) the first iteration, (iii) second iterations of Algorithm 2.3 for $N = 4$.

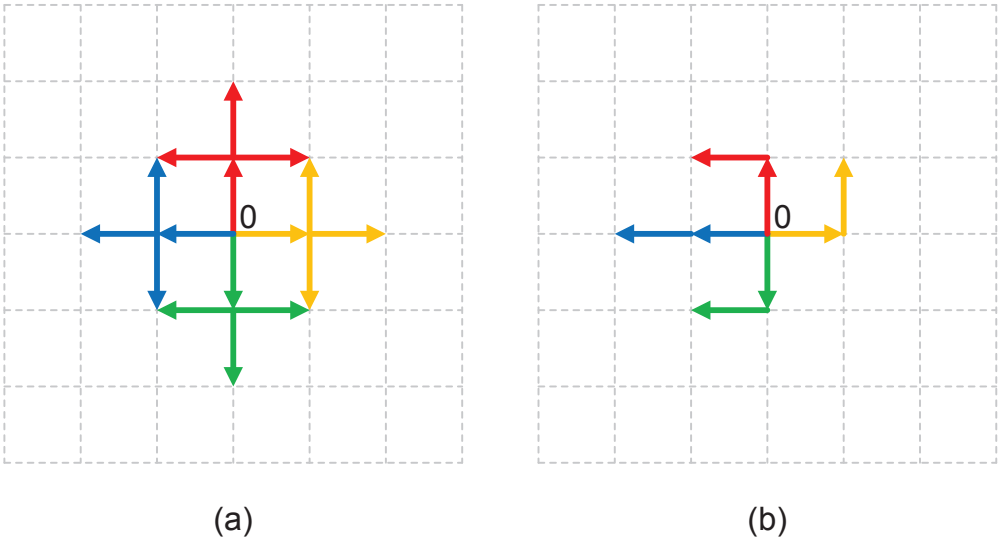
1. **Full Enumeration** Following Algorithm 2.3 we set $m_1 = 1$ and find (via full enumeration) all different SAW's of length m_1 starting from the origin 00.

Figure 5: Full Enumeration Step of Algorithm 2.3



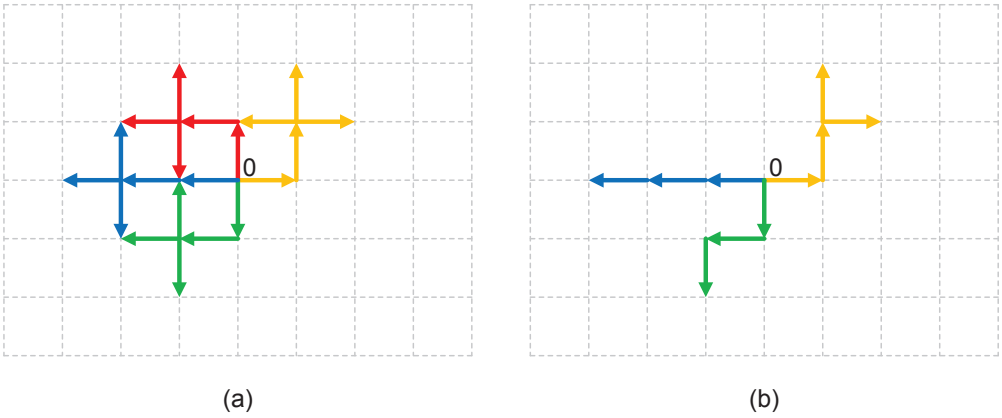
2. **First Iteration** We proceed by deriving from the above 4 SAW's (again via full enumeration) all SAW's of length $m_2 = 2$ (there are $N_1 = 12$ of them, see part (a)) and calculate the first Weight Factor $\widehat{\nu}_1 = \frac{N_1}{N_1^{(e)}} = \frac{12}{4} = 3$. Note that $N_1^{(e)} = 4$ (see part (b)).

Figure 6: First Iteration of Algorithm 2.3



3. **Second Iteration** we proceed by deriving from the above 4 elites (via full enumeration) all SAW's of length $m_3 = 3$ (again there are $N_2 = 12$ of them, see part (a)) and calculate the second Weight Factor $\widehat{\nu}_2 = \frac{N_2}{N_2^{(e)}} = \frac{12}{4} = 3$. Note that $N_2^{(e)} = 4$ (see part (b)).

Figure 7: Second Iteration of Algorithm 2.3



It is readily seen that the final estimator (15) of Algorithm 2.3 resembles the final one (1) of Algorithm 2.1 in that in (15) $\widehat{\nu}_t = \widehat{c}_t^{-1}$ and $N_1^{(e)}$ stands instead of $|\mathcal{X}_0|$.

We call Algorithm 2.3 - *stochastic enumeration* because of its main step 3, where at each level m_t we perform, full enumeration between the randomly selected $N_t^{(e)}$ elites and all possible candidates for the next level $m_{t+1} = m_t + r$.

Note that the estimator $\widehat{|\mathcal{X}^*|}$ in (15) corresponds to a *single run*. Averaging over M such independent runs, we obtain

$$\widetilde{|\mathcal{X}|^*} = \frac{1}{M}\sum_{i=k}^{M} \widehat{|\mathcal{X}_k^*|}\ . \tag{16}$$

The sample variance of $\widehat{|\mathcal{X}^*|}$ is

$$S^2(\widehat{|\mathcal{X}^*|}) = \frac{1}{M-1}\sum_{k=1}^{M}(\widehat{|\mathcal{X}_k^*|} - \widetilde{|\mathcal{X}|^*})^2 \tag{17}$$

and the relative error is

$$RE(\widehat{|\mathcal{X}^*|}) = \frac{S(\widehat{|\mathcal{X}^*|})}{\widetilde{|\mathcal{X}|^*}}. \tag{18}$$

It is not difficult to see that

- Explicitly the SE Algorithm 2.3 involves neither splitting nor OSLA. One might surmise, however, that they are embodied in the algorithm and in particular in Step 3.
- In contrast to splitting algorithm, the SE Algorithm is not stacked at any intermediate level $m_t < n$, and thus all generated walks are SAW's.
- The advantage of the splitting Algorithm 2.1 over its SE counterpart is that the former can be adaptive while the latter is not.

3 Convergence

We shall deal with the convergence properties of the splitting estimator $\widehat{|\mathcal{X}^*|}$ in (4), that is with

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0|\prod_{t=1}^{T}\widehat{c}_t = |\mathcal{X}_0|\prod_{t=1}^{T}\frac{N_t^{(e)}}{N_t}$$

for the splitting Algorithm 2.1. Convergence for OSLA-SPLIT and SE algorithms can be derived similarly.

We write (4) as

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0|\widehat{\ell}, \tag{19}$$

where

$$\widehat{\ell} = \prod_{k=1}^{n}\widehat{c}_k. \tag{20}$$

Since $|\mathcal{X}_0|$ is a constant we can proceed with the convergence of $\widehat{\ell}$ to $\ell = \prod_{k=1}^{n} c_k$. We shall make the following two requirements.

1.

(i) Each $\widehat{c}_k,\ k = 1,\ldots,n$ must represents an unbiased estimator of $c_k,\ k = 1,\ldots,n$ and each c_k is *not* a rare event probability.

(ii) The variance of each $\widehat{c}_k,\ k = 1, \ldots, n$ must be bounded, that is $\boldsymbol{Var}\ \{\widehat{c}_k\} < Const,\ k = 1, \ldots, n.$

We next justify all the assumptions.

(i) Each $\widehat{c}_k,\ k = 1, \ldots, n$ must represent an unbiased estimator of $c_k,\ k = 1, \ldots, n$ and each c_k is not a rare event probability

This is straightforward. Unbiasedness follows directly from the definition of c_k and the second requirement is assured by the fact that $\rho < c$, say $c \geq 10^{-2}$.

(ii) The variance of each $\widehat{c}_k,\ k = 1, \ldots, n$ must be bounded, that is

$$\mathbf{Var}\ \{\widehat{c}_k = \frac{N_k^{(e)}}{N_k}\} = \sigma_k^2 < Const, \forall k = 1, \ldots, n.$$

This is so since $\forall k = 1, \ldots, n$ both sample sizes $N_k^{(e)}$ and N_k are finite.

With the above two assumptions there are several nice convergence proves for splitting algorithms of type of Algorithm 2.1 (see for example [13], [18]).

Below we present a prove for a slightly modified version of Algorithm 2.1. Namely, instead of *dependent* random variables $\widehat{c}_t,\ t = 1, \ldots, n$ we modify Algorithm 2.1 such that they become *independent*. The modification is straightforward: instead of a single sequence $\widehat{c}_t,\ t = 1, \ldots, n$ we run *two independent* processes of $\widehat{c}_t$'s in parallel denoting them by $\widehat{c}_t^{(1)}$ and $\widehat{c}_t^{(2)}$, respectively. We next mix them as follows:

$$\widehat{c}_1^{(1)}, \widehat{c}_2^{(2)}, \ldots, \widehat{c}_{n-1}^{(1)}, \widehat{c}_n^{(2)} \tag{21}$$

and

$$\widehat{c}_1^{(2)}, \widehat{c}_2^{(1)}, \ldots, \widehat{c}_{n-1}^{(2)}.\widehat{c}_n^{(1)} \tag{22}$$

and then estimate ℓ as

$$\widehat{\ell} = 1/2(\widehat{\ell}^{(1)} + \widehat{\ell}^{(2)}),$$

where $\widehat{\ell}^{(1)}$ and $\widehat{\ell}^{(2)}$ correspond to the mixed sequences (21) and (22), respectively. It is readily seen that by doing so the dependence in either sequences between the random variables $\widehat{c}_t$ and $\widehat{\ell}$.

With the above assumptions satisfied we can prove the *polynomial in n* complexity of the estimator $\widehat{\ell}$ by citing some results from pp 438-439 of [10].

Since $\widehat{\ell} = \prod_{k=1}^n \widehat{c}_k$, is based on the product of n independent random variables $\widehat{c}_k$, each with bounded variance σ_k^2, the following result (formula (157) in [10]) holds

$$n\zeta^2/N \leq \ell^{-2}\, \boldsymbol{Var}\ \widehat{\ell} \leq (1 + \zeta^2/N)^n - 1, \tag{23}$$

where

$$\zeta^2 = \frac{1}{n}\sum_{k=1}^n \frac{\sigma_k^2}{c_k^2}.$$

Moreover, using Chebyshev's inequality we have (formula (159) in [10]) the following relative accuracy criterion

$$\mathbb{P}(|\widehat{\ell} - \ell| \leq \ell\varepsilon) \geq 1 - \delta,\ 0 < \varepsilon, \delta < 1. \tag{24}$$

This is valid

$$\forall N \geq \frac{n\zeta^2}{\ln(1+\delta\varepsilon^2)}.$$

Therefore, for fixed ε and δ, achieving (24) takes $O(n^2\zeta^2)$ time as $n \to \infty$. It is also shown in [10] that for independent Bernoulli random variables the time bound $O(n^2\zeta^2)$ becomes $O[n^2(\eta-1)]$, where $\eta = \frac{1}{n}\sum_{k=1}^{n}\frac{\sigma_k}{c_k}$ and that the bound (23) is *tight*.

4 Numerical Results

In this Section we present numerical results with Algorithms 2.1, 2.2 and 2.3.

4.1 Splitting Algorithm 2.1

Table 1 presents the performance of the systematic splitting algorithm for $n = 70$ with $\rho = 0.25$ and $N = 10,000$, where the exact cardinality $|\mathcal{X}^*{}_{70}| = 158078467825057188201748024363 6 \approx 1.5808E + 030$.

Table 1: Performance of Algorithm 2.1 for $n = 70$ with $\rho = 0.25$ and $N = 10,000$

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU (sec.)
1	23	1.534E+30	0.030	11.02
2	23	1.578E+30	0.002	11.03
3	23	1.653E+30	0.046	11.13
4	23	1.570E+30	0.007	11.08
5	23	1.539E+30	0.026	11.13
6	23	1.546E+30	0.022	11.10
7	23	1.512E+30	0.043	11.02
8	23	1.595E+30	0.009	11.02
9	23	1.508E+30	0.046	11.08
10	23	1.555E+30	0.017	11.04
Average	23	1.559E+30	0.025	11.07

Table 2 presents data similar to Table 1 for $n = 500$ with $\rho = 0.2$ and $N = 50,000$

Table 2: Performance of Algorithm 2.1 for $n = 500$ with $\rho = 0.2$ and $N = 50,000$

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU (sec.)
1	165	3.854E+211	0.147	523.58
2	165	4.921E+211	0.089	523.01
3	165	4.680E+211	0.035	523.60
4	165	4.078E+211	0.098	524.11
5	165	5.027E+211	0.112	523.76
6	165	4.388E+211	0.029	524.40
7	165	5.343E+211	0.182	525.13
8	165	4.454E+211	0.015	523.43
9	165	4.769E+211	0.055	524.55
10	164	4.474E+211	0.010	521.15
Average	164.9	4.599E+211	0.077	523.67

Table 3 presents the dynamic of the systematic splitting algorithm for a run of Table 1.

Table 3: Dynamics of a run of Algorithm 2.1 for SAW with $n = 70$

t	m_{*t}	m_t^*	$N_t^{(e)}$	N_t	$\widehat{c}_t$
1	1	24	2711	10000	0.271
2	5	28	3312	10000	0.331
3	8	28	3142	10000	0.314
4	11	39	3089	10000	0.309
5	14	37	3037	10000	0.304
11	32	54	3005	10000	0.301
18	53	77	3014	10000	0.301
19	56	81	2845	10000	0.285
20	59	84	2830	10000	0.283
21	62	89	2900	10000	0.290
22	65	91	2938	10000	0.294

Here we used the following notations

1. $N_t^{(e)}$ denotes the actual number of elites.
2. m_t^* and m_{*t} denote the upper and lower elite levels reached, respectively.
3. $\rho_t = N_t^{(e)}/N_t$ denotes the adaptive rarity parameter.

4.2 OSLA-SPLIT Algorithm 2.2

We ran the OSLA part of Algorithm 2.2 using $N = 10,000$ and $\rho = 0.1$ in parallel to its splitting part using $N = 400$ and $\rho = 0.25$. We found that OSLA-SPLIT algorithm typically outperforms Algorithm 2.1 (by about 50%) in terms of the relative error. We also found that the SE Algorithm 2.3 typically outperforms OSLA-SPLIT.

4.3 SE Algorithm 2.3

Tables 4 and 5 present data similar to Tables 1 and 3 for the splitting algorithm for $n = 500$ with $r = 2$, $N_1 = 284$, and $M = 20$ (see (16)). We started our simulation from the initial value $n = 4$. This corresponds to $N_t^{(e)} = 100$ (see also iteration $t = 0$ in Table 5 with $N_0^{(e)} = N_0 = 100$).

Table 4: Performance of SE Algorithm 2.3 for SAW with $n = 500$

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU (sec.)
1	248	4.799E+211	0.060	130.55
2	248	4.731E+211	0.045	130.49
3	248	4.462E+211	0.014	132.36
4	248	4.302E+211	0.050	136.22
5	248	5.025E+211	0.110	132.19
6	248	5.032E+211	0.112	131.79
7	248	4.397E+211	0.029	132.18
8	248	4.102E+211	0.094	131.60
9	248	4.820E+211	0.065	131.98
10	248	4.258E+211	0.059	131.73
Average	248	4.593E+211	0.064	132.11

Comparing the results of Tables 2 and 4 it follows that the SE Algorithm 2.3 is approximately 4 times faster than its counterpart splitting Algorithm 2.1.

Table 5: Dynamics of a run of SE Algorithm 2.3 for SAW with $n = 500$

t	m_t	$N_t^{(e)}$	N_t	$\widehat{\nu}_t$	$\lvert\widetilde{\mathcal{X}}_t^*\rvert$
0	4	100	100	1	100
1	6	100	780	7.8	780
2	8	100	759	7.59	5.920E+03
3	10	100	746	7.46	4.416E+04
4	12	100	731	7.31	3.228E+05
5	14	100	733	7.33	2.366E+06
50	104	100	699	6.99	4.528E+44
100	204	100	695	6.95	7.347E+86
150	304	100	699	6.99	1.266E+129
200	404	100	694	6.94	1.809E+171
244	492	100	693	6.93	2.027E+208
245	494	100	693	6.93	1.405E+209
246	496	100	696	6.96	9.780E+209
247	498	100	701	7.01	6.856E+210
248	500	100	700	7	4.799E+211

Table 6 present data similar to Table 4 for $n = 1,000$, with $r = 2$, $M = 20$ and $N_1 = 780$. The results are self explanatory.

Table 6: Performance of of the SE Algorithm 2.3 for SAW with $n = 1,000$

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	RE of $\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU (sec.)
1	497	2.514E+422	0.042	4008
2	497	2.629E+422	0.089	3992
3	497	2.757E+422	0.142	3980
4	497	2.354E+422	0.024	3975
5	497	2.200E+422	0.089	3991
6	497	2.113E+422	0.125	3991
7	497	2.081E+422	0.138	3970
8	497	2.281E+422	0.055	3983
9	497	2.504E+422	0.037	3982
10	497	2.552E+422	0.057	3975
Average	497	2.399E+422	0.080	3985

We also ran the OSLA-SE algorithm and found that its performance is similar to that of OSLA-SPLIT.

5 Concluding Remarks and Further Research

We presented a new method for counting self-avoiding walks (SAW's), the so-called *stochastic enumeration* (SE). We showed that SE is a stochastic replica of the naive full enumeration method, which is computationally unfeasible since the counting sets in SAW's are huge. In SE this difficulty is obviated by using a manageable sample size. In addition we also derived a new method, called OSLA-SPLIT, which is a combination of classic splitting with importance sampling, based on the well known method one-step-look-ahead (OSLA). We discussed the convergence properties of both version with numerical studies demonstrating their superiority over classic splitting.

As further research, we intend to apply the above ideas and methods to a wide class of NP-hard counting problems, in particular to the satisfiability problem.

6 Appendix

6.1 OSLA for Self-Avoiding Walk

The self-avoiding random walk, or simply *self-avoiding walk* (SAW), is a basic mathematical model for polymer chains. For simplicity we shall deal only with the 2-dimensional case. Each SAW is represented by a path $\boldsymbol{x} = (x_1, x_2, \ldots, x_{n-1}, x_n)$, where x_i represents the 2-dimensional position of the $i-$th molecule of the polymer chain. The distance between adjacent molecules is fixed to 1, and the main requirement is that the chain does not self-intersect. For simplicity we always assume the walk starts at the origin.

One of the main questions regarding the SAW model is: How many SAWs are there of length n? Let $\mathcal{X}^*$ be the set of SAWs of length n. We wish to estimate $|\mathcal{X}^*|$ by employing a convenient IS pdf $g(\boldsymbol{x})$. This pdf is defined by the following *one-step-look-ahead* (OSLA) procedure:

Procedure (One-Step-Look-Ahead)

1. Let $X_0 = (0, 0)$. Set $t = 1$.
2. Let d_t be the number of neighbors of X_{t-1} that have not yet been visited. If $d_t > 0$, choose X_t with probability $1/d_t$ from its neighbors. If $d_t = 0$ stop generating the path.
3. Stop if $t = n$. Otherwise increase t by 1 and go to step 2.

Note that the procedure either generates a SAW $\boldsymbol{x}$ of fixed length n or the path gets value zero. Let $g(\boldsymbol{x})$ be the corresponding discrete pdf. Then, for any SAW $\boldsymbol{x}$, we have by the product rule,

$$g(\boldsymbol{x}) = \frac{1}{d_1}\frac{1}{d_2}\cdots\frac{1}{d_n} = \frac{1}{w(\boldsymbol{x})}, \tag{25}$$

where

$$w(\boldsymbol{x}) = d_1 \ldots d_n \; . \tag{26}$$

The SAW counting algorithm now follows:

Algorithm 6.1 *(Counting SAWs)*

1. *Generate independently N paths $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ via the OSLA procedure.*

2. *For each SAW $\boldsymbol{X}_k$ compute the corresponding $w(\boldsymbol{X}_k)$ as in* (26)*. For the other parts (which do not reach the value n) set $w(\boldsymbol{X}_k) = 0$.*

3. *Return*

$$\widehat{|\mathcal{X}^*|} = \frac{1}{N} \sum_{i=k}^{N} w(\boldsymbol{X}_k) \; . \tag{27}$$

The efficiency of the simple one-step-look-ahead method deteriorates rapidly as n becomes large. It becomes impractical to simulate walks of length more than 200. This is due to the fact that if at any one step t the point x_{t-1} does not have unoccupied neighbors ($d_t = 0$) then the "weight" $w(\boldsymbol{x})$ is zero and contributes nothing to the final estimate of $|\mathcal{X}^*|$.

6.2 The Splitting Method

As mentioned this work deals with the *sequential splitting* method, for rare-events, counting and optimization.

We present some background from [28] on the splitting method, also called in [28], the *cloning* method. It is closely related to some other splitting methods, in particular to these discussed in [2], [11], [12], [18], and also to what is often called *randomized algorithms* [24, 25].

The main idea of a splitting method is to design a sequential sampling plan, with a view to decomposing a "difficult" counting problem defined on some set $\mathcal{X}^*$ into a number of "easy" ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ and such that $\mathcal{X}_m = \mathcal{X}^*$. Typically, splitting algorithms explore the connection between counting and sampling problems 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 [30].

A typical splitting algorithm comprises the following steps:

1. Formulate the counting problem as that of estimating the cardinality $|\mathcal{X}^*|$ of some set $\mathcal{X}^*$.

2. Find a sequence of sets $\mathcal{X} = \mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ such that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$, $|\mathcal{X}_m| = |\mathcal{X}^*|$ and $|\mathcal{X}| = |\mathcal{X}_0|$ is known.

3. Write $|\mathcal{X}^*| = |\mathcal{X}_m|$ as

$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = \ell |\mathcal{X}_0|, \tag{28}$$

where $\ell = \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}$. Note that ℓ is typically very small, like $\ell = 10^{-100}$, while each ratio

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \tag{29}$$

should not be small, like $c_t = 10^{-2}$ or bigger. Clearly, estimating ℓ directly while sampling in $|\mathcal{X}_0|$ is meaningless, but estimating each c_t separately seems to be a good alternative.

4. Develop an efficient estimator $\widehat{c}_t = \frac{|\widehat{\mathcal{X}}_t|}{|\widehat{\mathcal{X}}_{t-1}|}$ for each $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}|$.

5. Estimate $|\mathcal{X}^*|$ by

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}| \prod_{t=1}^{m} \widehat{c}_t, \tag{30}$$

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

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

(i) Put the well known NP-hard counting problems into the framework (28) by making sure that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ and each c_t is not a rare-event probability.

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

To proceed note that ℓ can be also written as

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) \geq m\}} \right], \tag{31}$$

where $\boldsymbol{X} \sim f(\boldsymbol{x})$, $f(\boldsymbol{x})$ is a uniform distribution on the set of points of $\mathcal{X}$, and as before, m is a fixed parameter, like the total number of constraints in an integer program, and $S(\boldsymbol{X})$ is the sample performance, like the number of feasible solution generated by the constraints of the integer program. It can be also written (see(28)) as

$$\ell = \prod_{t=1}^{T} c_t, \tag{32}$$

where

$$c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}| = \mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X}) \geq m_{t-1}\}}]. \tag{33}$$

Here

$$g^*_{t-1} = g^*(\boldsymbol{x}, m_{t-1}) = \ell(m_{t-1})^{-1} f(\boldsymbol{x}) I_{\{S(\boldsymbol{x}) \geq m_{t-1}\}}, \tag{34}$$

$\ell(m_{t-1})^{-1}$ is the normalization constant and similar to (28) 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 (28) we use in (32) a product of T terms instead of m terms. Note that T might be a random variable. The later case is associated with adaptive choice of the

level sets $\{\widehat{m}_t\}_{t=0}^T$ resulting in $T \leq m$. Since for counting problems the pdf $f(\boldsymbol{x})$ should be *uniformly* distributed on $\mathcal{X}$, which we denote by $\mathcal{U}(\mathcal{X})$, it follows from (34) 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 $\mathcal{U}(\mathcal{X}_t)$.

Although the pdf $g_{t-1}^* = \mathcal{U}(\mathcal{X}_t)$ is typically not available analytically, it is shown in [27, 28] that one can sample from it by using the MCMC method and in particular the Gibbs sampler, and as the result to update the parameters c_t and m_t adaptively. This is one of the most crucial issues of the splitting method.

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

$$\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t = \frac{1}{N^T} \prod_{t=1}^{T} N_t, \tag{35}$$

where

$$\widehat{c}_t = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq m_{t-1}\}} = \frac{N_t}{N}, \tag{36}$$

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

We next show how to cast the problem of counting the number of feasible solutions of the set of integer programming constraints into the framework (31)-(34).

Example 6.1 Counting on the set of an integer programming constraints Consider the set $\mathcal{X}^*$ containing both equality and inequality constraints of an integer program, that is,

$$\begin{aligned}
&\textstyle\sum_{k=1}^{n} a_{ik} x_k = b_i,\ i = 1, \ldots, m_1, \\
&\textstyle\sum_{k=1}^{n} a_{jk} x_k \geq b_j,\ j = m_1 + 1, \ldots, m_1 + m_2, \\
&\boldsymbol{x} = (x_1, \ldots, x_n) \geq \mathbf{0}, \quad x_k \text{ is integer } \forall k = 1, \ldots, n.
\end{aligned} \tag{37}$$

Our goal is to count the number of feasible solutions (points) of the set (37). We assume that each component $x_k,\ k = 1, \ldots, m$ has d different values, labeled $1, \ldots, d$. Note that the SAT problem represents a particular case of (37) with inequality constraints and where $x_1, \ldots, x_m$ are binary components. If not stated otherwise we will bear in mind the counting problem on the set (37) and in particular counting the number of true (valid) assignments in a SAT problem.

It is shown in [28] that in order to count the number of points of the set (37) one can associate it with the following rare-event probability problem

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) = m\}} \right] = \mathbb{E}_f \left[I_{\{\sum_{i=1}^{m} C_i(\boldsymbol{X}) = m\}} \right], \tag{38}$$

where the first m_1 terms $C_i(\boldsymbol{X})$'s in (38) are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^{n} a_{ik} X_k = b_i\}},\ i = 1, \ldots, m_1, \tag{39}$$

while the remaining m_2 ones are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^n a_{ik} X_k \geq b_i\}},\ i = m_1 + 1, \ldots, m_1 + m_2 \tag{40}$$

and $S(\boldsymbol{X}) = \sum_{i=1}^m C_i(\boldsymbol{X})$. Thus, in order to count the number of feasible solutions on the set (37) one can consider an associated rare-event probability estimation problem (38) involving a *sum of dependent Bernoulli random variables* C_i $i = m_1 + 1, \ldots, m$, and then apply $\widehat{|\mathcal{X}^*|} = \widehat{\ell}|\mathcal{X}|$. In other words, in order to count on $\mathcal{X}^*$ one needs to estimate efficiently the rare event probability ℓ in (38). A rare-event probability estimation framework similar to (38) can be readily established for many NP-hard counting problems [28].

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

$$\{(m_0, g^*(\boldsymbol{x}, m_{-1})),\ (m_1, g^*(\boldsymbol{x}, m_0)),\ (m_2, g^*(\boldsymbol{x}, m_1)), \ldots, (m_T, g^*(\boldsymbol{x}, m_{T-1}))\} \tag{41}$$

Here as before $g^*(\boldsymbol{x}, m_{-1}) = f(\boldsymbol{x}) = \mathcal{U}(\mathcal{X})$, $g^*(\boldsymbol{x}, m_t) = \mathcal{U}(\mathcal{X}_t)$, and m_t is obtained from the solution of the following non-linear equation

$$\mathbb{E}_{g^*_{t-1}} I_{\{S(\boldsymbol{X}) \geq m_t\}} = \rho, \tag{42}$$

where ρ is called the *rarity* parameter [29]. Typically one sets $0.1 \leq \rho \leq 0.01$.

Note that in contrast to the classic cross-entropy (CE) method [29], where one generates a sequence of tuples

$$\{(m_0, \boldsymbol{v}_0),\ (m_1, \boldsymbol{v}_1), \ldots, (m_T, \boldsymbol{v}_T)\}, \tag{43}$$

and, where $\{\boldsymbol{v}_t,\ t = 1, \ldots, T\}$ is a sequence of parameters in the parametric family of distributions $f(\boldsymbol{x}, \boldsymbol{v}_t)$, here in (41), $\{g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1},\ t = 0, 1, \ldots, T\}$ *is a sequence of non-parametric IS distributions.* Otherwise, the CE and the splitting algorithm are very similar.

Note that the splitting algorithm is also suitable for optimization. Here we also use the same sequence of tuples (41), but *without involving the product of the estimators* $\widehat{c}_t,\ t = 1, \ldots, T$.

Algorithm 6.2 (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 $\widetilde{\mathcal{X}}_0 = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0^{(e)}}\}$ 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

$$\widehat{c}_0 = \widehat{\ell}(\widehat{m}_0) = \frac{1}{N} \sum_{i=1}^N I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_0\}} = \frac{N_0^{(e)}}{N} \tag{44}$$

   as an *unbiased* estimator of c_0. Note that $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_0^{(e)}} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$, 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 $\widetilde{\mathcal{X}}_{t-1} = \{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}\}$ 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 $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}\}$, that is take η_{t-1} identical copies of each vector $\widetilde{\boldsymbol{X}}_k$. Denote the entire new population ($\eta_{t-1} N_{t-1}^{(e)}$ cloned vectors plus the original elite sample $\{\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}\}$) by $\mathcal{X}_{cl} = \{(\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_1), \ldots, (\widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}}, \ldots, \widetilde{\boldsymbol{X}}_{N_{t-1}^{(e)}})\}$. 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 $\widehat{c}_t = \frac{N_t^{(e)}}{N}$ (see (36)) as an estimator of c_t in (34). Note again that each vector of $\widetilde{\boldsymbol{X}}_1, \ldots, \widetilde{\boldsymbol{X}}_{N_t^{(e)}}$ 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 6, otherwise set $t = t+1$ and repeat from step 2.

5. **Final Estimator** Deliver $\widehat{\ell}(m)$ given in (35) as an estimator of $\ell(m)$ and $|\widehat{\mathcal{X}}^*| = \widehat{\ell}(m)|\mathcal{X}|$ as an estimator of $|\mathcal{X}^*|$.

Acknowledgments

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

References

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

[2] 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.

[3] Cappe, O. , Godsill, S. J. and Moulines, E. (2007) An overview of existing methods and recent advances in sequential Monte Carlo. IEEE Proceedings, 95(5):899-924.

[4] F. Cerou, P. Del Moral, T. Furon and A. Guyader. "Rare Event Simulation for Static Distribution". Technical Report, Inria, November 2009.

[5] F. Cerou, P. Del Moral, F. Le Gland, and P. Lezaud. "Genetic genealogical models in rare event analysis. Latin American Journal of Probability and Mathematical Statistics", 1, 2006.

[6] F. Cerou and A. Guyader. "Adaptive multilevel splitting for rare event analysis. Stoch. Anal. Appl.", 25(2):417443, 2007.

[7] P. Del Moral. Feynman-Kac formulae, *Genealogical and interacting particle systems with applications. Probability and its Applications. Springer-Verlag*, New York, 2004.

[8] P. Del Moral and A. Doucet. "Particle Methods: An Introduction with Applications" Technical Report No 6991, Inria, November 2009.

[9] A. Doucet and A. M. Johansen. "A Tutorial on Particle Filtering and Smoothing: Fifteen years later", 2009.

[10] G. Fishman *Monte Carlo*. Springer, 1995.

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

[12] Garvels M.J.J. and R.Y. Rubinstein "A Combined Splitting - Cross Entropy Method for Rare Event Probability Estimation of Single Queues and ATM Networks". Unpublished Manuscript.

[13] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. "Multilevel splitting for estimating rare event probabilities. Oper. Res.", 47(4):585600, 1999.

[14] Vibhav Gogate V,, and R. Dechter, "Approximate Counting by Sampling the Backtrack-free Search Space", American Association for Artificial Intelligence, 2007.

[15] Vibhav Gogate V,, and R. Dechter, "SampleSearch:Importance sampling in presence of Determinism", 2009

[16] H. Kahn and T.E. Harris. "Estimation of particle transmission by random sampling". National Bureau of Standards Appl. Math. Series, 12:27 30, 1951.

[17] A. Lagnoux. "Rare event simulation. Probability in the Engineering and Informational Sciences", 20(1):4566, 2006.

[18] A. Lagnoux-Renaudie *A two-steps branching splitting model under cost constraint*. Journal of Applied Probability, 2009.

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

[20] P. LEcuyer, J. Blanchet, B. Tuffin, and P.W. Glynn. "Asymptotic robustness of estimators in rare-event simulation". ACM Transactions on Modeling and Computer Simulation, 18(3):12691283, 2008.

[21] J. S. Lui. *Monte Carlo Strategies in Scientific Computing*. Springer, 2001.

[22] Melas V. B. "On the efficiency of the splitting and roulette approach for sensitivity analysis". Winter Simulation Conference, Atlanta, Georgia, pp. 269 - 274, 1997.

[23] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. "Equation of state calculations by fast computing machines". The Journal of Chemical Physics, 21(6):10871092, 1953.

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

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

[26] M. N. Rosenbluth and A. W. Rosenbluth. "Monte Carlo calculation of the average extension of molecular chains". Journal of Chemical Physics, 23(2), fev 1955.

[27] R. Y. Rubinstein. "The Gibbs Cloner for Combinatorial Optimization, Counting and Sampling", To be published in *Methodology and Computing in Applied Probability*, 2009.

[28] R. Y. Rubinstein. "Randomized Algorithms with Splitting: Why the Classic Randomized Algorithms do not Work and how to Make them Work" *Methodology and Computing in Applied Probability*, 12(1), 1-41, 2010.

[29] 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.

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