

Stochastic Enumeration Method for Counting NP-hard Problems

Reuven Rubinfeld

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

ierrr01@ie.technion.ac.il

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

July 2, 2011

Abstract

We present a new generic sequential importance sampling algorithm, called *stochastic enumeration* (SE) for counting #P-complete problems, such as the number of satisfiability assignments and the number of perfect matchings (permanent). We show that SE presents a natural generalization of the classic *one-step-look-ahead* algorithm in the sense that it

- Runs in parallel multiple trajectories instead of a single one.
- Employs a polynomial time decision making oracle, which can be viewed as an *n-step-look-ahead* algorithm, where n is the size of the problem.

Our simulation studies indicate good performance of SE as compared with the well-known splitting and SampleSearch methods.

Keywords. Counting, MCMC, Rare-Event, Self-Avoiding Walks, Satisfiability, Sequential Importance Sampling, Splitting.

^{0†} This research was supported by the BSF (Binational Science Foundation, grant No 2008482)

Contents

1	Introduction	3
2	The OSLA Method	6
3	Extension of OSLA: nSLA Method	8
4	Extension of OSLA: SE-OSLA Method for SAW's	11
4.1	SE-OSLA Algorithm for SAW's	12
5	SE Method	17
5.1	SE Algorithm	17
6	Applications of SE	25
6.1	Counting the Number of Trajectories in a Network	26
6.1.1	SE for Estimation of Probabilities in a Network	29
6.2	Counting the Number of Perfect Matching (Permanent) in a Graph	30
6.3	Counting SAT's	33
7	Choosing a Good Number $N^{(e)}$ of Elites	34
8	Numerical Results	35
8.1	Counting SAW's	35
8.2	Counting the Number of Trajectories in a Network	37
8.3	Counting the Number of Perfect Matching (Permanent) in a Graph	38
8.4	Counting SAT's	40
9	Concluding Remarks and Further Research	44
10	Appendices	45
10.1	SIS Method	45
10.2	DPLL Algorithm from Wikipedia	47
10.3	Random Graphs Generation	47

1 Introduction

Recently, rare-event and counting problems have attracted a wide range of interest of computer scientists and applied probabilists.

The most popular methods for handling rare-event simulation and counting are the classic *splitting* and *importance sampling* (IS).

The splitting method dates back to Kahn and Harris [21] and Metropolis et al [32]. Since then hundreds of insightful papers have been written on that topic. We refer the reader to [2, 3], [5]-[7], [12]-[16], [25, 27] and also to the papers by Glasserman et al. [16], Cerou et al. [7] and Melas [31], 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 Moral's [12] monograph.

Importance sampling, and in particular its sequential version, forms part of almost any standard book on Monte Carlo simulation. It is well known, however (see for example [43]) that its straightforward application may typically yield very poor approximations of the quantity of interest. For example, it is shown by Gogate and Dechter [17], [18] that in graphical models, and in particular in satisfiability models, it may generate many useless zero-weight samples which are often rejected yielding an inefficient sampling process.

In this paper we introduce a new generic *sequential importance sampling* (SIS) algorithm, called *stochastic enumeration* (SE) for counting #P-complete problems. SE represents a natural generalization of the classic *one-step-look-ahead* (OSLA) algorithm in the following sense:

- It runs multiple trajectories in parallel, instead of a single one.
- It employs polynomial time decision making oracles, which can be viewed as *n-step-look-ahead* algorithms, *n* being the size of the problem.

We shall also show that

1. SE reduces a difficult counting problem to a set of simple ones, applying at each step an oracle.
2. In contrast to the conventional splitting algorithm [40] there is very little randomness involved in SE. As a result, it is typically faster than splitting.

Note finally that use of fast decision making algorithms (oracles) for solving NP-hard problems is very common in Monte-Carlo methods, see, for example, the insightful monograph of Gertsbakh and Spungin [15] in which Kruskal's spanning trees algorithm is used for estimating network reliability.

More examples of "hard" (#P-complete) counting problems and "easy" (polynomial) decision making include:

1. How many different variable assignments will satisfy a given DNF formula?
2. How many different variable assignments will satisfy a given 2-SAT formula?

3. How many perfect matchings are there for a given bipartite graph?

It was known quite a long time ago that finding a perfect matching for a given bipartite graph $G(V, E)$ (decision making problem) can be solved in polynomial $O(|V||E|)$ time, while the corresponding problem "How many perfect matchings does the given bipartite graph have?" is already #P-complete. Note also that the problem of counting the number of perfect matchings (or in directed graphs: the number of vertex cycle covers) is known to be equivalent to the computation of the permanent of a matrix [44].

Similarly, there is a trivial algorithm for determining if a DNF form is satisfiable. Indeed, in this case examine each clause, and if one is found that does not contain a variable and its negation, then it is satisfiable, otherwise it is not. The counting version of this problem is #P-complete.

Many #P-complete problems have a *fully-polynomial-time randomized approximation scheme* (FPRAS), which, informally, will produce with high probability an approximation to an arbitrary degree of accuracy, in time that is polynomial with respect to both the size of the problem and the degree of accuracy required [33]. Jerrum, Valiant and Vazirani [20] showed that every #P-complete problem either has an FPRAS, or is essentially impossible to approximate; if there is any polynomial-time algorithm which consistently produces an approximation of a #P-complete problem which is within a polynomial ratio in the size of the input of the exact answer, then that algorithm can be used to construct an FPRAS.

Our main strategy in this work is as in [15]: *use fast polynomial decision making oracles to solve #P-complete problems*. In particular we show how to incorporate in SE

- Breadth first search (BFS) or Dijkstra's shortest path algorithm [9] for counting the number of path in the networks.
- Hungarian decision making assignment problem method [23] for counting the number of perfect matchings.
- Chinese postman(Eulerian cycle) decision making algorithm (finding the shortest tour in a graph) for counting the total number of such shortest tours. Recall that in an Eulerian cycle shortest tour problem one has to visit each edge at least once.
- Davis et al [10, 11] decision making algorithm for counting the number of valid assignments in 2-SAT.

As mentioned SE represents a natural extension of the classic *one-step-look-ahead* OSLA. We shall show that OSLA-type algorithms (see Algorithm 2.1) have the following two drawbacks:

(i) Its pdf $g(\mathbf{x})$ is typically non-uniformly distributed on the set of its valid trajectories, that is

$$g(\mathbf{x}) \neq \frac{1}{|\mathcal{X}^*|},$$

where \mathcal{X}^* is desired counting set and $|\mathcal{X}^*|$ is its cardinality.

(ii) It typically runs into very many zeros even for moderate size n of the problem. It is well known [29] that for self-avoiding walk most of its trajectories of length $n \geq 100$ in OSLA Algorithm 2.1 become not self-avoiding. A similar pattern was observed in other counting problems including satisfiability.

The exception is the OSLA algorithm of Rasmussen [37] for counting the permanent. Rasmussen [37] proofs that if the a_{ij} entries of the permanent matrix \mathbf{A} are Bernoulli outcomes, each generated randomly with probability $p = 1/2$, then using the corresponding OSLA procedure one gets a FPRAS estimator. This is quite a remarkable result!

In this paper we show that in contrast to OSLA the proposed SE method handles both issues (i) and (ii) successfully. In particular, to overcome

1. *Non-uniformity of $g(\mathbf{x})$* , SE runs in parallel multiple trajectories (see Algorithm 5.1 below) instead of a single one.
2. *Generation of many zeros* (trajectories of zero length) it employs fast (polynomial time) decision making oracles, which is equivalent of using an *n-step-look-ahead* algorithm instead of OSLA. For details see Section 5 below.

The rest of the paper is organized as follows. Section 2 presents background on the OSLA method. Section 3 deals with the simplest extension of OSLA, called *nSLA*, which uses an *n-step-look-ahead* strategy (based on an oracle) instead of OSLA. Its advantage is that it does not lose trajectories; its main drawback is that its estimators have high variance. Section 4 deals with another extension of OSLA, called *SE-OSLA*, highlighting the following points

1. The immediate advantage of *SE-OSLA* versus OSLA is that even for a moderate number of multiple trajectories $N^{(e)}$, called *elite samples* the variance is substantially reduced and the generated trajectories are nearly uniformly distributed.
2. The *SE-OSLA* algorithm still has a major drawback in that it loses most of the trajectories even for moderate n .

Section 5 is our main one; it deals with the SE method, which extends both, the *nSLA* and *SE-OSLA* ones. We show that

- SE combines the nice features of both *nSLA* and *SE-OSLA*; as the former it does not lose trajectories and as the latter its estimators have much smaller variance as compared to the *nSLA* ones. it is shown that
- The only difference between *SE-OSLA* and SE is that the former is still based on OSLA, and the latter - on a polynomial time decision making oracle (algorithm), which is typically available for many problems, such as finding a path in a network, a perfect matching in a graph, etc.

Section 6 deals with application of SE to counting of #P-complete problems, such as counting the number of trajectories in a network, number of satisfiability assignments in a SAT problem and calculating the permanent. Section

7 discuss how to choose the main parameter, the number of elites samples in the SE algorithm. In Section 8 we present numerical results for SE-OSLA and SE. Here we show that SE outperforms the well known splitting and Sample-Search methods. Our explanation for that is: SE is based on SIS (sequential importance sampling), while its two counterparts are based merely on regular IS (importance sampling). Section 9 presents some conclusions. Finally, the Appendix is devoted to some auxiliary material.

2 The OSLA Method

Let $|\mathcal{X}^*|$ be our counting quantity, such as the number of satisfiability assignments in a SAT problem with n literals, the number of perfect matchings (permanent) in a bipartite graph with n nodes and the number of SAW's of length n . We wish to estimate $|\mathcal{X}^*|$ by employing a SIS pdf $g(\mathbf{x})$ defined in Section 10.1 of Appendix . To do so we use the following *one-step-look-ahead* (OSLA) procedure due to Rosenbluth and Rosenbluth [39], which was originally introduced for SAW's:

Procedure 1: One-Step-Look-Ahead

1. **Initial step** Start from the origin point \mathbf{y}_0 . For example, in a two-dimensional SAW, $\mathbf{y}_0 = (0, 0)$ and in a network with source s and sink t , $\mathbf{y}_0 = s$. Set $t = 1$.
2. **Main step** Let ν_t be the number of neighbors of \mathbf{y}_{t-1} that have not yet been visited. If $\nu_t > 0$, choose \mathbf{y}_t with probability $1/\nu_t$ from its neighbors. If $\nu_t = 0$ stop generating the path and deliver an estimate $|\widehat{\mathcal{X}^*}| = 0$.
3. **Stopping rule** Stop if $t = n$. Otherwise increase t by 1 and go to step 2.
4. **The estimator** Return

$$g(\mathbf{x}) = \frac{1}{\nu_1} \frac{1}{\nu_2} \dots \frac{1}{\nu_n} = \frac{1}{|\widehat{\mathcal{X}^*}|(\mathbf{x})} \quad (1)$$

as a SIS pdf. Here

$$|\widehat{\mathcal{X}^*}| = \nu_1 \dots \nu_n . \quad (2)$$

The procedure either generates a path \mathbf{x} of fixed length n or the path gets value zero (in case of $\nu_t = 0$, for which $g(\mathbf{x}) = \infty$). The key assumption in OSLA is that one random walk on a graph is a good representative of the entire graph.

The OSLA counting algorithm now follows:

Algorithm 2.1 (OSLA Algorithm)

1. Generate independently M paths $\mathbf{X}_1, \dots, \mathbf{X}_M$ from SIS pdf $g(\mathbf{x})$ via the above OSLA procedure.

2. For each path \mathbf{X}_k compute the corresponding $|\widehat{\mathcal{X}}_k^*|$ as in (2). For the other parts (which do not reach the value n) set $|\widehat{\mathcal{X}}_k^*| = 0$.
3. Return

$$|\bar{\mathcal{X}}^*| = \frac{1}{M} \sum_{i=k}^M |\widehat{\mathcal{X}}_k^*|. \quad (3)$$

The efficiency of the one-step-look-ahead method deteriorates rapidly as n becomes large. For example, it becomes impractical to simulate SAW's of length more than 200 and similar for the other counting problems. This is due to the fact that if at any step t the point \mathbf{y}_{t-1} does not have unoccupied neighbors ($\nu_t = 0$) then $|\widehat{\mathcal{X}}^*|$ is zero and it contributes nothing to the final estimate of $|\mathcal{X}^*|$.

Figure 1 depicts a SAW (with arrows) trapped after 15 iterations. One can easily find from it the corresponding values ν_t , $t = 1, \dots, 15$ (using the corresponding short lines without arrows) of each of the 15 points. They are

$$\begin{aligned} \nu_1 = 4, \nu_2 = 3, \nu_3 = 3, \nu_4 = 3, \nu_5 = 3, \nu_6 = 3, \nu_7 = 2, \nu_8 = 3, \\ \nu_9 = 3, \nu_{10} = 3, \nu_{11} = 2, \nu_{12} = 3, \nu_{13} = 2, \nu_{14} = 1, \nu_{15} = 0. \end{aligned}$$

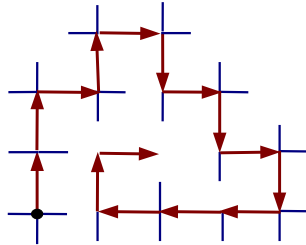


Figure 1: SAW trapped after 15 iterations

As for another situation where OSLA can be readily trapped consider a directed graph in Figure 2 with source s and sink t .

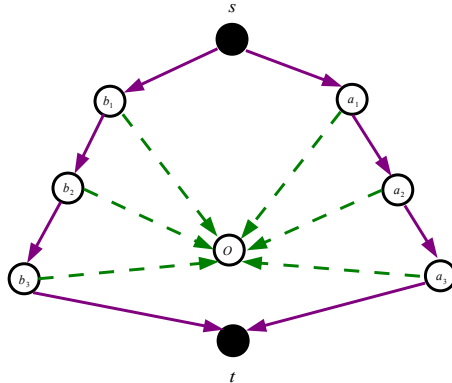


Figure 2: Directed graph

There are two ways from s to t , one via nodes a_1, \dots, a_n and another via nodes b_1, \dots, b_n . Figure 2 corresponds to $n = 3$. Note that all nodes beside s and t are directly connected to a central node o , which has no connection with t . Clearly in this case with probability $1 - (1/2)^n$ a random walk will be trapped at node o .

3 Extension of OSLA: n SLA Method

A natural extension of the one-step-look-ahead (OSLA) procedure is the k -step-look-ahead, where $2 \leq k \leq n$ one. It is crucial to note that for $k = n$ *no path will be ever lost*. This in turn implies that that $\nu_t > 0, \forall t$.

Consider, for example, the SAW in Figure 1. Note that if we could use three-step-look-ahead policy instead of OSLA we would rather move after step number 13 (corresponding to point \mathbf{y}_{13}) down instead of up. By doing so we would prevent the SAW being trapped after $n = 15$ iterations.

For many problems including SAW's the n -step-look-ahead policy requires additional memory and CPU and for that reason has limited applications. There exist, however a set of problems where the n -step-look-ahead (n SLA) policy can be still easily implemented using polynomial algorithms (oracles). As mentioned, relevant examples are counting perfect matchings (permanent) in a graph, counting the number of paths in a network and SAT counting.

Note that n SLA algorithm is identical to the OSLA Algorithm 2.1. Its corresponding procedure is similar to OSLA **Procedure 1**. For completeness we present it below.

Procedure 2: n -Step-Look-Ahead

1. **Initial step** (Same as in **Procedure 1**).
2. **Main step** Employ an oracle and find $\nu_t \geq 1$ the number of neighbors of \mathbf{y}_{t-1} that have not yet been visited. Choose \mathbf{y}_t with probability $1/\nu_t$ from its neighbors.

3. **Stopping rule** (Same as in **Procedure 1**).

4. **The Estimator** (Same as in **Procedure 1**).

To see how n SLA works in practice consider a simple example following the main steps of the OSLA Algorithm 2.1.

Example 3.1 Let $\mathbf{x} = (x_1, x_2, x_3)$ be a three dimensional vector with binary components and let $\{000, 001, 100, 110, 111\}$ be a set of its valid combinations (paths). We have $n = 3$ and $|\mathcal{X}^*| = 5$. Note that since we use an n SLA oracle instead of just OSLA we will have (in the former case) that $\nu_i \in \{1, 2\}$, $i = 1, 2, 3$ instead of $\nu_i \in \{0, 1, 2\}$, $i = 1, 2, 3$ (in the latter).

Figure 3 presents a tree corresponding to the set $\{000, 001, 100, 110, 111\}$.

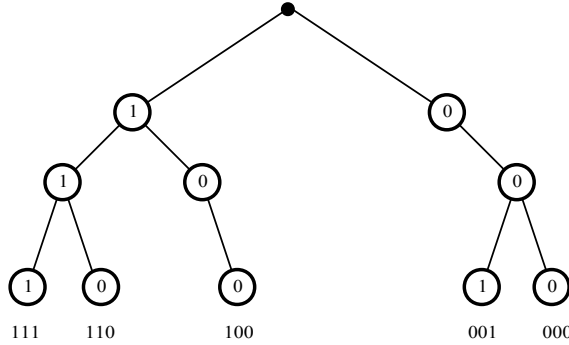


Figure 3: Tree corresponding to the set $\{000, 001, 100, 110, 111\}$.

According to n SLA **Procedure 2** we start with the first binary variable x_1 . Since $x_1 \in \{0, 1\}$ we employ the oracle two times; once for $x_1 = 0$, (by considering the triplet $0x_2x_3$), and once for $x_1 = 1$ (by considering the triplet $1x_2x_3$). The roll of the oracle is merely to provide a YES-NO answer for each triplet $0x_2x_3$ and $x_1 = 1$, that is to check whether or not there exist a valid assignment for $x_1x_2x_3$ separately for $x_1 = 0$ and $x_1 = 1$. Clearly, in our case the answer is YES in both cases, since an example of $0x_2x_3$ is, say 000 and an example of $1x_2x_3$ is, say 110. We have therefore $\nu_1 = 2$. Following **Procedure 2** we next flip a symmetric coin. Assume that the outcome is 0. This means that we will proceed to path $0x_2x_3$ and will discard the path $1x_2x_3$ (see Figure 4).

Consider next $x_2 \in \{0, 1\}$. As for x_1 we employ the oracle two times; once for $x_2 = 0$, (by considering the triplet $00x_3$), and once for $x_2 = 1$ (by considering the triplet $01x_3$). In this case the answer is YES for the case $00x_3$ (an example of $00x_3$ is, as before, 000) and NO for the case $01x_3$ (there is no valid assignment in the set $\{000, 001, 100, 110, 111\}$ for $01x_3$ with $x_3 \in \{0, 1\}$). We have therefore $\nu_2 = 1$.

We finally proceed with the oracle to x_3 by employing it two times; once for $x_3 = 0$ and once for $x_3 = 1$. In this case the answer is YES for both cases, since we automatically obtain 000 and 001. We have $\nu_3 = 2$.

The resulting estimator is therefore

$$|\widehat{\mathcal{X}^*}| = \nu_1 \nu_2 \nu_3 = 2 \cdot 1 \cdot 2 = 4.$$

Figure 4 presents the sub-trees $\{000, 001\}$ (in bold) generated by n SLA using the oracle.

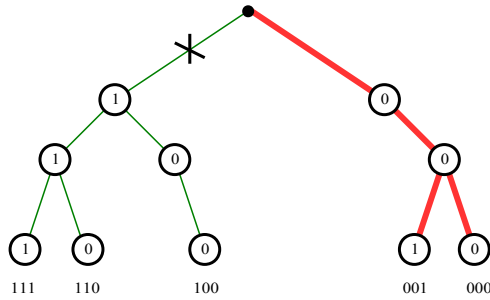


Figure 4: The sub-trees $\{000, 001\}$ (in bold) generated by n SLA using the oracle

It is readily seen that similar to the above randomly generated trajectories 000 and 001, the one 100 results in to $|\widehat{\mathcal{X}^*}| = 4$, while the trajectories 110, 111 result in to $|\widehat{\mathcal{X}^*}| = 8$. Noting that the corresponding probabilities for $|\widehat{\mathcal{X}^*}| = 4$ and $|\widehat{\mathcal{X}^*}| = 8$ are $1/4$ and $1/8$ and averaging over all 5 cases we obtain the desired results $|\mathcal{X}^*| = 5$. The variance of $|\widehat{\mathcal{X}^*}|$ is

$$\mathbf{Var} |\widehat{\mathcal{X}^*}| = 1/5\{3(4 - 5)^2 + 2(8 - 5)^2\} = 21/2.$$

The main drawback of n SLA **Procedure 2** is that its SIS pdf $g(\mathbf{x})$ is model dependent. Typically it is far away from the ideal uniform SIS pdf $g^*(\mathbf{x})$, that is

$$g(\mathbf{x}) \neq \frac{1}{|\mathcal{X}^*|}.$$

As result, the estimator of $|\mathcal{X}^*|$ has a large variance. Note that $g^*(\mathbf{x}) = \frac{1}{|\mathcal{X}^*|}$ corresponds to zero variance SIS.

To see the non-uniformity of $g(\mathbf{x})$ consider a 2-SAT model with clauses $C_1 \wedge C_2 \wedge \dots \wedge C_n$, where $C_i = x_i \vee \bar{x}_{i+1} \geq 1$, $i = 1, \dots, n$. Figure 5 presents the corresponding graph with 4 literals and $|\mathcal{X}^*| = 5$.

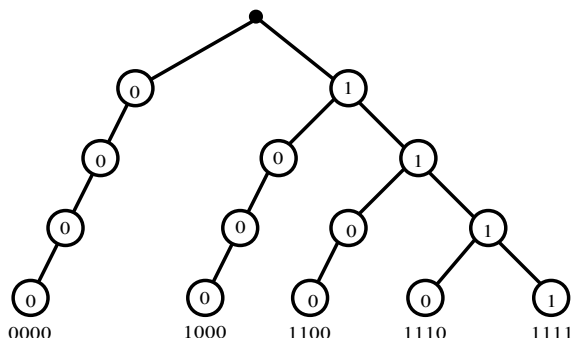


Figure 5: A graph with 4 clauses and $|\mathcal{X}^*| = 5$.

In this case $|\mathcal{X}^*| = n+1$, the ideal zero variance pdf $g^*(\mathbf{x}) = \frac{1}{|\mathcal{X}^*|} = 1/(n+1)$, while the SIS pdf is

$$g(\mathbf{x}) = \begin{cases} 1/2, & \text{for } (00, \dots, 00), \\ 1/2^2, & \text{for } (10, \dots, 00), \\ 1/2^3, & \text{for } (11, \dots, 00), \\ 1/2^n, & \text{for } (11, \dots, 10) \text{ and } (11, \dots, 11), \end{cases}$$

which is highly non-uniform.

To improve the non-uniformity of $g(\mathbf{x})$ we will develop in Section 5 a modified version of n SLA, called *stochastic enumeration* (SE) algorithm. In contrast to n SLA we will run in SE *multiple* trajectories instead of a single one. Before doing so we present below for convenience a multiple trajectory version of the OSLA Algorithm 2.1 for SAW's.

4 Extension of OSLA: SE-OSLA Method for SAW's

In this section we extend the OSLA method to multiple trajectories and present the corresponding algorithm, called *SE-OSLA*. For clarity of representation the SE-OSLA algorithm is given for self-avoiding walks (SAW's). Its adaptation for other counting problems is straightforward. In particular, we show that for SAW's

- OSLA represents a particular case of SE-OSLA, when the number of multiple trajectories, denoted by $N_t^{(e)}$ and called the *elite samples*, is $N_t^{(e)} = 1, \forall t$.
- In contrast to OSLA, which loses most of its trajectories (even for modest n), with SE-OSLA we can reach quite large levels, say $n = 10,000$, provided the number $N_t^{(e)}$ of elite samples is not too small, say $N_t^{(e)} = 100, \forall t$. As result one can generate very long SAW's with SE-OSLA.

4.1 SE-OSLA Algorithm for SAW's

A self-avoiding walk (SAW) is a sequence of moves on a lattice that does not visit the same point more than once. As such, SAWs are often used to model the real-life behavior of chain-like entities such as polymers, whose physical volume prohibits multiple occupation of the same spatial point. They also play a central role in modeling of the topological and knot-theoretic behavior of molecules such as proteins.

One of the main questions regarding the SAW model is: How many SAWs are there of length n exactly? There is currently no known formula for determining the number of self-avoiding walks, although there are rigorous methods for approximating them. Finding the number of such paths is conjectured to be an NP-hard problem.

The most advanced algorithms for SAW's are the pivot ones. They can handle SAW's of size 10^7 , see [8], [30]. For a nice recent survey see [19].

Although empirically the pivot algorithm [8] specially designed for SAW's outperforms the SE-OSLA algorithm, we nevertheless present it below in the interests of clarity of representation and motivation purposes.

For simplicity we assume that the walk starts at the origin and confine ourselves to the 2-dimensional case. Each SAW is represented by a path $\mathbf{x} = (x_1, x_2, \dots, x_{n-1}, x_n)$, where x_i denotes the 2-dimensional position of the i -th molecule of the polymer chain. The distance between adjacent molecules is taken as 1.

A SAW of length $n = 121$ is given in Figure 6.

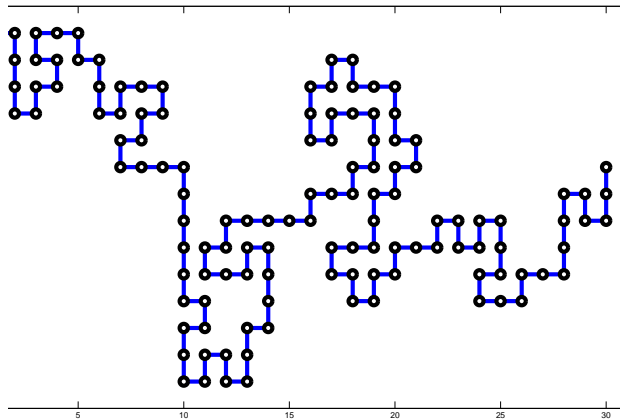


Figure 6: A SAW of length $n = 121$.

Algorithm 4.1 (SE-OSLA for SAW's)

1. Iteration one

- **Full Enumeration** Select a small number n_0 , say $n_0 = 4$ and count via full enumeration all different SAW's of size n_0 starting from the

origin 00. Denote the total number of these SAW's by $N_1^{(e)}$ and call them the *elite sample*. For example, for $n_0 = 4$ the number of elites $N_1^{(e)} = 100$. Set the first level to n_0 . Proceed with the $N_1^{(e)}$ elites from level n_0 to the next one $n_1 = n_0 + r$, where r is a small integer (typically $r = 1$ or $r = 2$) and count via *full enumeration* all different SAW's at level $n_1 = n_0 + r$. Denote the total number of such SAW's by N_1 . For example, for $n_1 = 5$ there are $N_1 = 284$ different SAW's.

- **Calculation of the First Weight Factor** Calculate

$$\nu_1 = \frac{N_1}{N_1^{(e)}} \quad (4)$$

and call it the first *weight factor*.

2. Iteration t , ($t \geq 2$)

- **Full Enumeration** Proceed with $N_{t-1}^{(e)}$ elites from iteration $t - 1$ to the next level $n_{t-1} = n_{t-2} + r$ and derive for iteration t via full enumeration all SAW's at level $n_{t-1} = n_{t-2} + r$, that is of all those SAW's that continue the $N_{t-1}^{(e)}$ paths resulted from the previous iteration. Denote by N_t the resulting number of such SAW's.
- **Stochastic Enumeration** Select randomly (*without replacement*) $N_t^{(e)}$ SAW's from the set of N_t ones and call this step *stochastic enumeration*.
- **Calculation of the t -th Weight Factor.** Calculate

$$\nu_t = \frac{N_t}{N_t^{(e)}} \quad (5)$$

and call it the t -th *weight factor*. It is important to note that for $N_t^{(e)} = 1$ we have OSLA. Note that here as in OSLA ν_t can be both ≥ 1 and $= 0$, and if $\nu_t = 0$, we stop and deliver $|\widehat{\mathcal{X}^*}| = 0$.

3. Stopping Rule

Proceed with iteration t , $t = 1, \dots, \frac{n-n_0}{r}$ and calculate

$$|\widehat{\mathcal{X}^*}| = N_1^{(e)} \prod_{t=1}^{\frac{n-n_0}{r}} \nu_t. \quad (6)$$

Call $|\widehat{\mathcal{X}^*}|$ the *point estimator* of $|\mathcal{X}^*|$. Note that for $N_t^{(e)} = 1$ and $r = 1$ we have that $|\widehat{\mathcal{X}^*}| = w$, where w is defined in (2). Note that since the number of levels is fixed in advance, $|\widehat{\mathcal{X}^*}|$ presents an unbiased estimator of $|\mathcal{X}^*|$ (see [2]).

4. Final Estimator

Run SE-OSLA for M independent replications and deliver

$$|\widetilde{\mathcal{X}^*}| = \frac{1}{M} \sum_{k=1}^M |\widehat{\mathcal{X}_k^*}| \quad (7)$$

as an unbiased estimator of $|\mathcal{X}^*|$. Call $|\widetilde{\mathcal{X}^*}|$ the *sample estimator* of $|\mathcal{X}^*|$. Note that for $N_t^{(e)} = 1$ we have that $|\widetilde{\mathcal{X}^*}| = |\bar{\mathcal{X}^*}|$, where $|\bar{\mathcal{X}^*}|$ is defined in (3).

Typically one keeps in SE-OSLA the number of elites $N_t^{(e)}$ fixed, say $N_t^{(e)} = N^{(e)} = 100$ while N_t varies from iteration to iteration.

Note that the sample variance of $|\widetilde{\mathcal{X}^*}|$ is

$$S^2(|\widetilde{\mathcal{X}^*}|) = \frac{1}{M-1} \sum_{k=1}^M (|\widetilde{\mathcal{X}_k^*}| - |\widetilde{\mathcal{X}^*}|)^2 \quad (8)$$

and the relative error is

$$RE(|\widetilde{\mathcal{X}^*}|) = \frac{|\widetilde{\mathcal{X}^*}|}{S(|\widetilde{\mathcal{X}^*}|)}. \quad (9)$$

The first two iterations of Algorithm 4.1 for $n_0 = 1$, $r = 1$ and $N_t^{(e)} = N^{(e)} = 4$ are given below.

Example 4.1 *Iteration one*

- **Full Enumeration** We set $n_0 = 1$ and count (via full enumeration) all different SAW's of length n_0 starting from the origin 00. We have $N_1^{(e)} = 4$ (see Figure 7).

We proceed (again via full enumeration) to derive from the $N_1^{(e)} = 4$ elites all SAW's of length $n_1 = 2$ (there are $N_1 = 12$ of them, see part (a) of Figure 8).

- Calculation of the first weight factor. We have $\nu_1 = \frac{N_1}{N_1^{(e)}} = \frac{12}{4} = 3$.

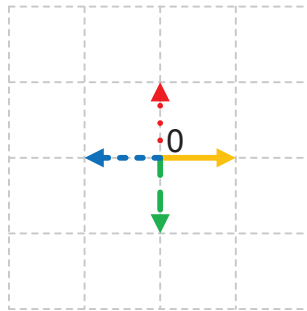


Figure 7: The First Four Elites $N_1^{(e)} = N^{(e)} = 4$

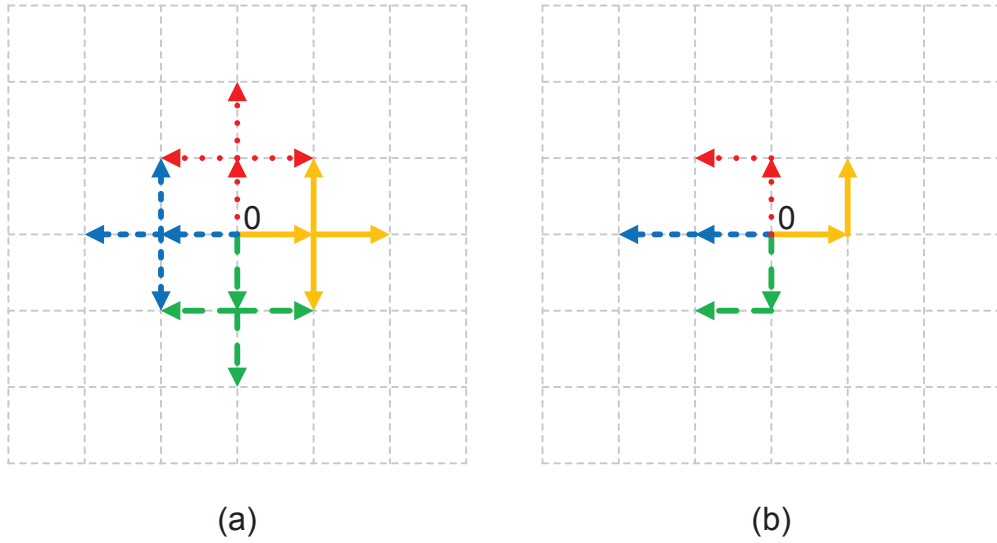


Figure 8: First Iteration of Algorithm 4.1

Iteration two

- **Stochastic Enumeration** Select randomly *without replacement* $N_2^{(e)} = 4$ elites from the set of $N_1 = 12$ ones (see part (b) of Figure 8).
- **Full Enumeration** Proceed (via full enumeration) deriving from the above $N_2^{(e)} = 4$ elites all SAW's of length $n_2 = 3$ (again there are $N_2 = 12$ of them, see part (a) of Figure 9).
- **Calculation of the second weight factor.** We have $\nu_2 = \frac{N_2}{N_2^{(e)}} = \frac{12}{4} = 3$.

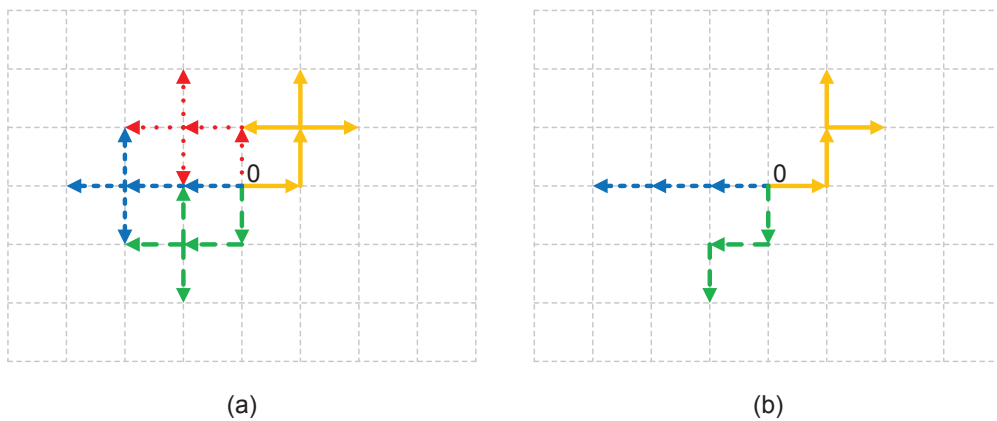


Figure 9: Second Iteration of Algorithm 4.1

We call Algorithm 4.1 SE-OSLA because it presents an extension of OSLA in the sense that at each level n_t we combine full enumeration with stochastic one

such that for the next level $n_{t+1} = n_t + r$ we perform full enumeration between the randomly selected $N_t^{(e)}$ elites and all possible candidates.

Remark 4.1 It follows from Algorithm 4.1 that for $r = 1$, $n_0 = 0$ and $N_t^{(e)} = 1$, $\forall t = 1, \dots, n$ the SE-OSLA estimator $|\widehat{\mathcal{X}^*}|$ in (6) reduces to the OSLA one. Thus, OSLA can be viewed as particular case of SE-OSLA with $N_t^{(e)} = 1$. The crucial difference between OSLA and SE-OSLA is that the former loses most of its trajectories after some modest number of steps, while with SE-OSLA we can practically reach any desired level, provided the number of elites $N_t^{(e)}$ is of moderate size, say $N_t^{(e)} \leq 100$, $\forall t$. As result one can generate with SE-OSLA a very long SAW, say of length $n = 10,000$ (see below) and count the number of SAW's up to several thousands in manageable CPU time (see the numerical results below). Note finally, that since OSLA presents an importance sampling estimator, so does SE-OSLA.

Remark 4.2 We next support empirically Remark 4.1 stating that SE-OSLA can generate very long SAW's. To do so, we run the SE-OSLA algorithm for several fixed values of $N_t^{(e)} = N^{(e)}$ and $r = 1$. In particular Table 1 shows the length of the SAW trajectories, denoted by R , for several $N^{(e)}$ scenarios, namely for $N^{(e)} = 1, 2, 5, 15, 25, 35$. Each scenario is based on 20 independent replications. Note that the values R_{ki} , $k = 1, \dots, 20; i = 1, \dots, 6$, where the index i corresponds to $N_1^{(e)} = 1$, $N_2^{(e)} = 2$, $N_3^{(e)} = 5$, $N_4^{(e)} = 15$, $N_5^{(e)} = 25$, $N_6^{(e)} = 35$ are arranged in Table 1 in the increasing order.

It readily follows from the results of Table 1 that

- The average $R(N^{(e)})$ increases faster than $N^{(e)}$, that is $R(N^{(e)}) > N^{(e)}$.
- The average length R_{k1} of SAW's for OSLA ($N^{(e)} = 1$) is about 65. We found empirically that in this case only 1% of SAW's reaches $R = 200$.

Table 1: The length of the SAW's R_{ki} , $k = 1, \dots, 20; i = 1, \dots, 6$ for $N^{(e)}=1, 2, 5, 15, 25, 35$

k	R_{k1}	R_{k2}	R_{k3}	R_{k4}	R_{k5}	R_{k6}
1	10	26	73	322	791	1185
5	33	62	193	539	1757	2057
10	44	178	306	1205	3185	2519
15	79	299	436	1849	4099	5537
20	198	644	804	4760	6495	9531
Average	64.2	216.25	356.6	1503.25	3343	4080.6

Below we present the sequence $(N_t^{(e)}, N_t)$, $t = 1, \dots, R$ for one of the runs

with $N_1^{(e)} = 2$ with the outcome $R = 30$.

$$\begin{aligned}
& (2, 6), (2, 6), (2, 6), (2, 6), (2, 6), (2, 6), (2, 5), (2, 5), (2, 5), (2, 4), \\
& (2, 4), (2, 6), (2, 5), (2, 6), (2, 6), (2, 6), (2, 5), (2, 4), (2, 6), (2, 6), \\
& (2, 6), (2, 5), (2, 4), (2, 4), (2, 3), (2, 4), (2, 3), (2, 2), (2, 2), (2, 0).
\end{aligned} \tag{10}$$

It is crucial to note that for general counting problems, like SAT's, Remark 4.1 does not apply in the sense that even for a relatively small model size the SE-OSLA Algorithm 4.1 typically generates many zeros (see for example the graph in Figure 2). For this reason it has limited applications.

As for another example of poor performance of SE-OSLA consider a 3-SAT model with an adjacency matrix $\mathbf{A} = (20 \times 80)$ and $|\mathcal{X}^*| = 15$. Running the SE-OSLA Algorithm 4.1 with $N^{(e)} = 2$ we observed that it finds the 15 valid assignments (satisfies all 80 clauses) only with probability $1.4 \cdot 10^{-4}$. We found that as the size of the models increases the percentage of valid assignments goes fast to zero.

5 SE Method

Here we present our main algorithm, called *stochastic enumeration* (SE), which extends both, the *nSLA* and the SE-OSLA Algorithms as follows.

- SE extends *nSLA* in the sense that it uses multiple trajectories instead of a single one.
- SE extends SE-OSLA Algorithm 4.1 in the sense that it uses a polynomial time decision making *n*-step-look-ahead oracle (algorithm) instead of OSLA. The oracle will be incorporated into the **Full Enumeration** step of SE-OSLA Algorithm 4.1.

We present SE for the models where the components of the vector $\mathbf{x} = (x_1, \dots, x_n)$ are assumed to be binary variables, such as SAT's, and where fast decision making oracles are available (see [10, 11] for SAT's). Its modification to arbitrary discrete variables is straightforward.

5.1 SE Algorithm

Consider SE-OSLA Algorithm 4.1 and assume for simplicity that $r = 1$ and that at iteration $t - 1$ the number of elites is, say, $N_{t-1}^{(e)} = 100$. In this case it is readily seen that in order to implement an oracle in the **Full Enumeration** step at iteration t of SE-OSLA we have to run it $2N_{t-1}^{(e)} = 200$ times: 100 times for $x_t = 0$ and 100 more times for $x_t = 1$. Note that for each *fixed* combination of (x_1, \dots, x_t) the oracle can be viewed as an $(n - t + 1)$ -step look-ahead algorithm in the sense that it

- Sets first $x_{t+1} = 0$ and then makes a decision (YES-NO path) for the remaining $n - t + 1$ variables (x_{t+2}, \dots, x_n) .
- Sets next $x_{t+1} = 1$ and then again makes a similar (YES-NO) decision for the same set (x_{t+2}, \dots, x_n) .

Algorithm 5.1 (SE Algorithm)

1. Iteration 1

- **Full Enumeration** (Similar to Algorithm 4.1). Let n_0 be the number corresponding to the first n_0 variables x_1, \dots, x_{n_0} . Count via full enumeration all different paths (valid assignments in SAT) of size n_0 . Denote the total number of these paths (assignments) by $N_1^{(e)}$ and call them the *elite sample*. Proceed with the $N_1^{(e)}$ elites from level n_0 to the next one $n_1 = n_0 + r$, where r is a small integer (typically $r = 1$ or $r = 2$) and count via *full enumeration* all different paths (assignments) of size $n_1 = n_0 + r$. Denote the total number of such paths (assignments) by N_1 .
- **Calculation of the First Weight Factor** (Same as in Algorithm 4.1).

2. Iteration t , ($t \geq 2$)

- **Full Enumeration** (Same as in Algorithm 4.1, except that it is performed via the corresponding polynomial time oracle rather than OSLA).
Recall that for each *fixed* combination of (x_1, \dots, x_t) the oracle can be viewed as an $(n - t + 1)$ -step look-ahead algorithm in the sense that it
 - Sets first $x_{t+1} = 0$ and then makes a YES-NO decision for the path associated with the remaining $n - t + 1$ variables (x_{t+2}, \dots, x_n) .
 - Sets next $x_{t+1} = 1$ and then again makes a similar (YES-NO) decision.
- **Stochastic Enumeration** (Same as in Algorithm 4.1).
- **Calculation of the t -th Weight Factor**. (Same as in Algorithm 4.1). Recall that in contrast to SE-OSLA, where $\nu_t \geq 0$, here $\nu_t \geq 1$.

3. **Stopping Rule** Same as in Algorithm 4.1.

4. **Final Estimator** Same as in Algorithm 4.1.

It follows that if $N_t^{(e)} \geq |\mathcal{X}^*|$, $\forall t$, the SE Algorithm 5.1 will be exact.

Remark 5.1 The SE method can be viewed as a multi-level splitting with a specially chosen *importance function* [13]. Note that in the traditional splitting [2, 3], [40] one chooses quite an obvious importance function, say the number of satisfied clauses in a SAT problem or the length of a walk in SAW. Here we simply decompose the rare event into non rare events by introducing intermediate levels. In the proposed method we introduce a random walk on a graph, starting at some node on the graph and we try to find an alternative (better) importance function which can be computed in polynomial time, and which provides a reasonable estimate of the probability of the rare event. This is the

same as to say that we are choosing a specific dynamics on the graph, and we are trying to optimize the importance function for this precise dynamics.

It is well known that the best possible importance function in dynamical rare events environment driven by a non stationary Markov chain, is the *committor function* [34]. It is also well known that its computing is at least as difficult as the underlying rare event. In the SE approach, however we roughly approximate the committor function using an oracle in the sense that we can say now whether this probability is zero or not. For example, in the SAT problem with already assigned literals, we can compute whether or not they can lead to a valid solution. Note also that stating - the committor is z means the probability of hitting the rare event, given that the Markov chain starts at z . In particular, for SAT - the committor is 0 implies keeping 0, otherwise we take 1; for SAW, we may also have 1 on points for which the committor is 0.

Figure 10 presents the SE Algorithm 5.1 as a splitting one for the first 3 iterations of a model with n variables using $N^{(e)} = 1$. There are 3 valid paths (corresponding to the dashed lines) reaching the final level n (with the aid of oracle), and 1 invalid path (corresponding to the line depicted as $\cdot - x$). It follows from Figure 10 that the accumulated weights are $\nu_1 = 2, \nu_2 = 2, \nu_3 = 1$.

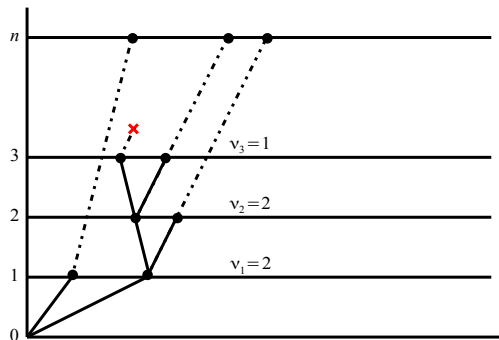


Figure 10: SE Algorithm 5.1 as a splitting one for the first 3 iterations

Note again that the SE estimator $|\widehat{\mathcal{X}^*}|$ is unbiased for the same reason as its SE-OSLA counterpart (6) is so; namely both can be viewed as multiple splitting methods with fixed (non-adaptive) levels [3].

Below we show how SE works for several toy examples

Example 5.1 Consider the SAT model $C_1 \wedge C_2 \wedge \dots \wedge C_n$, where $C_i = x_i \vee x_{i+1} \geq 1, i = 1, \dots, n$. Assume that $n = 4$. Suppose that we set $n_0 = 2, r = 1, N_t^{(e)} = 3$ and $M = 1$.

Iteration 1

- **Full Enumeration** Since $n_0 = 2$, we handle first the variables x_1 and x_2 . Using SAT solver we obtain the following three trajectories $(01x_3x_4, 10x_3x_4,$

$11x_3x_4$), which can be extended to valid solutions. (Note that $00x_3x_4$ cannot be extended to a valid solution and is discarded by the oracle). We have therefore $N_1^{(e)} = 3$, that is still within the allowed budget $N_t^{(e)} = 3$.

We proceed with oracle to x_3 , that is derive from the $N_1^{(e)} = 3$ elites all SAT assignments of length $m_1 = 3$. By full enumeration we obtain the trajectories $(011x_4, 010x_4, 101x_4, 110x_4, 111x_4)$. (Note that $100x_3x_4$ cannot be extended to a valid solution and is discarded by the oracle). We have therefore $N_1 = 5$.

It is readily seen that for this model we have in general $N_1^{(e)} = F_{n_0-2}$ and $N_1 = F_{n_0-1}$, where n_0 denotes the number of literals in the first level and F_n denotes the Fibonacci sequence. In particular, if $n_0 = 12$ we have $N_1^{(e)} = F_{10} = 88$ and $N_1 = F_{11} = 133$ different SAT assignments,

- **Calculation of the first weight factor.** We have $\nu_1 = \frac{N_1}{N_1^{(e)}} = \frac{5}{3}$.

Iteration two

- **Stochastic Enumeration** Since $N_1 > N_t^{(e)} = 3$ we resort to sampling by selecting randomly *without replacement* $N_2^{(e)} = 3$ trajectories from the set of $N_1 = 5$. Suppose we pick $(010x_4, 101x_4, 111x_4)$. These will be our working trajectories at the next step.
- **Full Enumeration** We proceed with oracle to handle x_4 , that is, derive from the $N_1^{(e)} = 3$ elites all valid SAT assignments of length $n_2 = 4$. By full enumeration we obtain the trajectories $(0101, 1010, 1011, 1110, 1111)$. We have therefore again $N_2 = 5$.
- **Calculation of the second weight factor.** We have $\nu_2 = \frac{N_2}{N_2^{(e)}} = \frac{5}{3}$.

The SE estimator of the true $|\mathcal{X}^*| = 8$ based on the above two iterations is $|\widehat{\mathcal{X}^*}| = 3 \cdot 5/3 \cdot 5/3 = 25/3$.

It is readily seen that if we set $N_t^{(e)} = 5$ instead of $N_t^{(e)} = 3$ we would get the exact result, that is $|\widehat{\mathcal{X}^*}| = 3 \cdot 5/3 \cdot 8/5 = 8$.

Example 5.2 Consider the SAT model

$$(x_1 \vee x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4).$$

Suppose again that $n_0 = 2$, $r = 1$, $N_t^{(e)} = 3$ and $M = 1$.

Iteration 1

The same as in Example 5.1.

Iteration two

- **Stochastic Enumeration** We select randomly *without replacement* $N_2^{(e)} = 3$ elites from the set of $N_1 = 5$. Suppose we pick $(010x_4, 110x_4, 111x_4)$.

- **Full Enumeration** We proceed with oracle to handle x_4 , that is derive from the $N_1^{(e)} = 3$ elites all SAT assignments of length $n_2 = 4$. By full enumeration we obtain the trajectories (0100, 0101, 1100, 1101, 1110, 1111)). We have therefore $N_2 = 6$.
- **Calculation of the second weight factor.** We have $\nu_2 = \frac{N_2}{N_2^{(e)}} = 2$.

The estimator of the true $|\mathcal{X}^*| = 9$ based on the above two iterations is $|\widehat{\mathcal{X}^*}| = 3 \cdot 5/3 \cdot 2 = 10$.

It is readily seen that if we set again $N_t^{(e)} = 5$ instead of $N_t^{(e)} = 3$ we would get the exact result, that is $|\mathcal{X}^*| = 3 \cdot 5/3 \cdot 9/5 = 9$.

It is also not difficult to see that as $N^{(e)}$ increases the variance of the SE estimator $|\widehat{\mathcal{X}^*}|$ in (6) decreases and for $N^{(e)} \geq |\mathcal{X}^*|$ we have $\mathbf{Var} |\widehat{\mathcal{X}^*}| = 0$.

Example 5.3 Example 3.1 continued Let again $\mathbf{x} = (x_1, x_2, x_3)$ be a three dimensional vector with $\{000, 001, 100, 110, 111\}$ being the set of its valid combinations. As before, we have $n = 3$, $|\mathcal{X}_0| = 2^3 = 8$ and $|\mathcal{X}^*| = 5$. In contrast to Example 3.1, where $N^{(e)} = 1$ we assume that $N^{(e)} = 2$.

We have $N_1 = 3$ and $\nu_1 = 3/2$. Since $N_k^{(e)} = N^{(e)} = 2$, $k = 1, 2, 3$ we proceed with the following 3 pairs (00, 10), (00, 11), (10, 11). They result into (000, 001, 100), (000, 001, 110, 111), (100, 110, 111), respectively with their corresponding pairs ($N_2 = 3, \nu_2 = 3/2$), ($N_2 = 4, \nu_2 = 2$), ($N_2 = 3, \nu_2 = 3/2$).

It is readily seen that the estimator of $|\mathcal{X}^*|$ based on (000, 001, 100) and (100, 110, 111) equals $|\widehat{\mathcal{X}^*}| = 2 \cdot 3/2 \cdot 3/2 = 9/2$, and the one based on (000, 001, 110, 111) is $|\widehat{\mathcal{X}^*}| = 6$, respectively. Noting that their probabilities are equal $1/3$ and averaging over all 3 cases we obtain the desired results $|\mathcal{X}^*| = 5$. The variance of $|\widehat{\mathcal{X}^*}|$ is

$$\mathbf{Var} |\widehat{\mathcal{X}^*}| = 1/3\{2(9/2 - 5)^2 + 2(6 - 5)^2\} = 1/2.$$

It follows from the above that by increasing $N^{(e)}$ from 1 to 2, the variance decreases 21 times.

Figure 11 presents the sub-trees $\{100, 000, 001\}$ (in bold) of the original tree (based on the set $\{000, 001, 100, 110, 111\}$), generated using the oracle with $N^{(e)} = 2$.

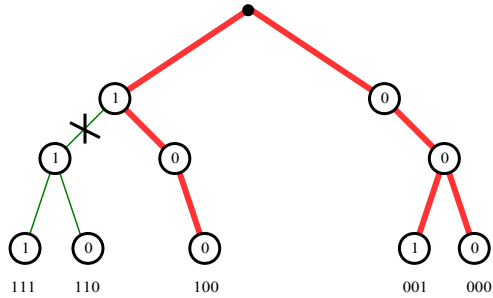


Figure 11: The sub-trees $\{100, 000, 001\}$ (in bold) corresponding to $N^{(e)} = 2$.

Figures 12, 13 and 14 present

- A tree with 5 variables.
- Sub-trees (in bold) corresponding to $N^{(e)} = 1$.
- Sub-trees (in bold) corresponding to $N^{(e)} = 2$,

respectively.

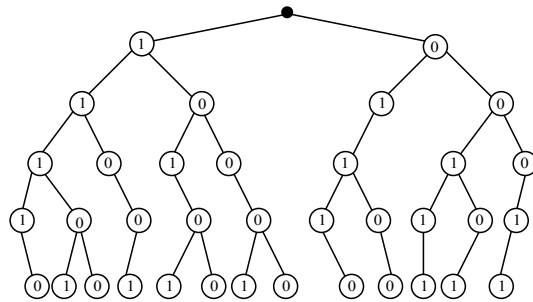


Figure 12: A tree with 5 variables.

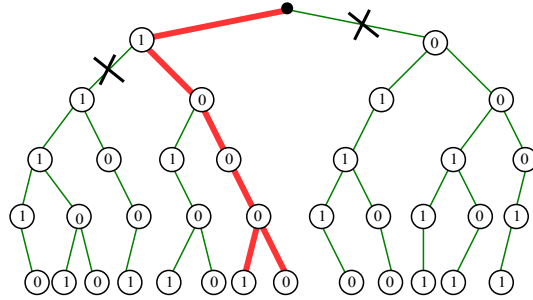


Figure 13: Sub-trees (in bold) corresponding to $N^{(e)} = 1$.

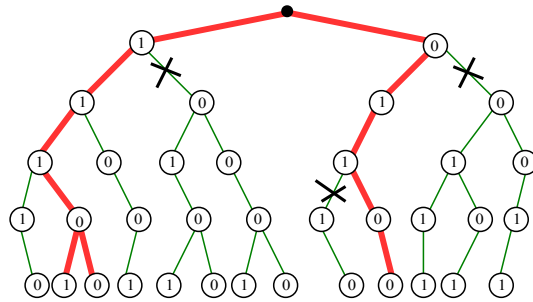


Figure 14: Sub-trees (in bold) corresponding to $N^{(e)} = 2$.

As mentioned earlier the major advantage of SE Algorithm 5.1 versus its n SLA counterpart is that the uniformity of $g(\mathbf{x})$ in the former increases in $N^{(e)}$. In other words $g(\mathbf{x})$ becomes "closer" to the ideal pdf $g^*(\mathbf{x})$. We next demonstrate numerically this phenomenon while considering the following two simple models:

(i) A 2-SAT model with clauses $C_1 \wedge C_2 \wedge \dots \wedge C_n$, where $C_i = x_i \vee \bar{x}_{i+1} \geq 1$, $i = 1, \dots, n$ (see Figure 5 for 4 literals and $|\mathcal{X}^*| = 5$).

Straightforward calculation yield that for this particular case ($|\mathcal{X}^*| = 5$) variance reduction obtained from using $N^{(e)} = 2$ instead of $N^{(e)} = 1$ is about 150 times.

(i) A graph having $|\mathcal{X}^*| = m$ paths and such that the length of its k -th path, $k = 1, \dots, m$ - is k . Figure 15 presents such a graph with $|\mathcal{X}^*| = 5$ paths and $s - t$ being the source and sink, respectively.

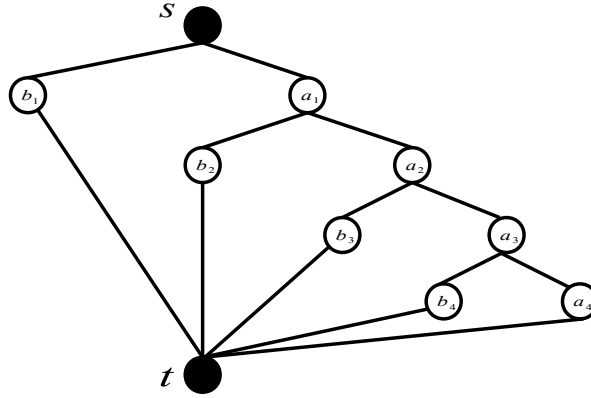


Figure 15: A graph with $|\mathcal{X}^*| = 5$ paths.

(i) Table 2 presents the efficiency of the SE Algorithm 5.1 for the 2-SAT model with $|\mathcal{X}^*| = 100$ for different values of $N^{(e)}$ and M . The comparison was done for

$$\begin{aligned}
 & (N^{(e)} = 1, M = 500), (N^{(e)} = 5, M = 100), (N^{(e)} = 10, M = 50), \\
 & (N^{(e)} = 25, M = 20), (N^{(e)} = 50, M = 10), (N^{(e)} = 75, M = 5), \quad (11) \\
 & (N^{(e)} = 100, M = 1).
 \end{aligned}$$

Table 2: The efficiencies of the SE Algorithm 5.1 for the 2-SAT model with $|\mathcal{X}^*| = 100$

$(N^{(e)}, M)$	$ \widetilde{\mathcal{X}^*} $	RE
$(N^{(e)} = 1, M = 500)$	11.110	0.296
$(N^{(e)} = 5, M = 100)$	38.962	0.215
$(N^{(e)} = 10, M = 50)$	69.854	0.175
$(N^{(e)} = 25, M = 20)$	102.75	0.079
$(N^{(e)} = 50, M = 10)$	101.11	0.032
$(N^{(e)} = 75, M = 5)$	100.45	0.012
$(N^{(e)} = 100, M = 1)$	100.00	0.000

Note also that the relative error RE was calculated as

$$RE = \frac{(1/10 \sum_{i=1}^{10} (|\widetilde{\mathcal{X}_i^*}| - 100)^2)^{1/2}}{100}.$$

Similar results were obtained for the graph in Figure 15 with $|\mathcal{X}^*| = 100$ paths.

As expected for small $N_t^{(e)}$ ($1 \leq N_t^{(e)} \leq 10$) the relative error RE of the estimator $|\widehat{\mathcal{X}^*}|$ is large and it underestimates $|\mathcal{X}^*|$. Starting from $N_t^{(e)} = 25$ the estimator stabilizes. Note that for $N_t^{(e)} = 100$ the estimator is exact since $|\mathcal{X}^*| = 100$. Recall that the optimal zero variance SIS pdf over the following $n + 1$ paths

$$(00 \cdots 00), (10 \cdots 00), (11 \cdots 00), \dots, (11 \cdots 10), (11 \cdots 11)$$

is $g^*(\mathbf{x}) = 1/(n + 1)$.

The variance of the SE method can be reduced by using the so-called *empirical length-distribution* method due to [38], which is described below.

Remark 5.2 Empirical length- distribution method. We demonstrate it for counting the number of paths in a network. Its modification for some other counting and rare-event estimation models is simple.

Denote the length of a path \mathbf{x} by $|\mathbf{x}|$. Note that this is not the length of the vector, but rather less than it by one. We first simulate a pilot run of N_0 samples using the SE Algorithm 5.1 to find an estimate of the length-distribution $\mathbf{r} = (r_1, \dots, r_{n-1})$, where

$$r_k = \frac{\text{number of paths of length } k}{\text{number of paths of length } > k}. \quad (12)$$

We can visualize r_k in the following way: suppose a path \mathbf{x} chosen at random from \mathcal{X}^* , and suppose that for some k we would then expect x_{k+1} to be the vertex n with probability r_k . We estimate \mathbf{r} as

$$\widehat{r}_k = \frac{\sum_{j=1}^{N_0} I\{|\mathbf{X}_j| = k\} I\{\mathbf{X}_j \in \mathcal{X}^*\} / f(\mathbf{X}_j)}{\sum_{j=1}^{N_0} I\{|\mathbf{X}_j| \geq k\} I\{\mathbf{X}_j \in \mathcal{X}^*\} / f(\mathbf{X}_j)}, \quad (13)$$

where $\mathbf{X} \sim f(\mathbf{x})$. We then use $\widehat{\mathbf{r}} = (\widehat{r}_1, \dots, \widehat{r}_{n-1})$ to generate paths similarly to the SE Algorithm 5.1 except that at each step t where we choose the next vertex to be n with probability \widehat{r}_t , and then choose a different random available vertex with probability $1 - \widehat{r}_t$. If there are no other available vertices we just choose the next vertex to be n . To ensure that $f(x) > 0$ for all $\mathbf{x} \in \mathcal{X}^*$, it is sufficient that $0 < \widehat{r}_k < 1$, for all k . If we obtain $r_k = 0$ or 1 for some k , then we just replace it with $1/|\widehat{\mathcal{X}^*}|$ or $1 - 1/|\widehat{\mathcal{X}^*}|$, respectively, where $|\widehat{\mathcal{X}^*}|$ is the estimate of $|\mathcal{X}^*|$ from the SE pilot run.

The drawback of $\widehat{\mathbf{r}}$ in (13) is that it represents an empirical unconditional marginal IS distribution rather than a empirical conditional one.

Our simulation studies show that with the *empirical length-distribution* method one can get variance reduction by a factor of about 5.

6 Applications of SE

Below we present several possible applications of SE. As usual we assume that there exist an associated polynomial time decision making oracle.

6.1 Counting the Number of Trajectories in a Network

We show how to use SE for counting the number of trajectories (paths) in a network with a fixed source and sink. We demonstrate this for $N^{(e)} = 1$. The modification to $N^{(e)} > 1$ is straightforward.

Consider the following two toy examples.

Example 6.1 Bridge Network Consider the undirected graph in Figure 16.

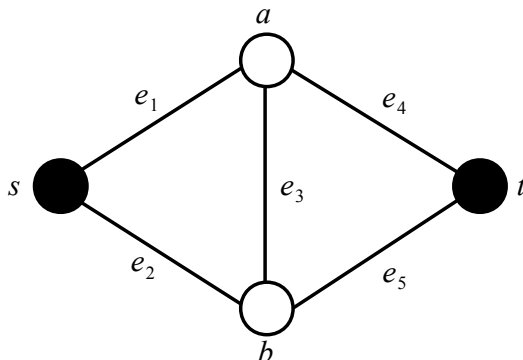


Figure 16: Bridge network: number of path from A to B.

Suppose we wish to count the 4 trajectories

$$(e_1, e_4), (e_1, e_3, e_5), (e_2, e_3, e_4), (e_2, e_5) \quad (14)$$

between the source node s and sink node t .

1. **Iteration 1** Starting from s we have two nodes e_1 and e_2 and the associated vector (x_1, x_2) . Since each x is binary we have the following four combination (00), (01), (10), (11). Note that only the trajectories (01), (10) are relevant here since (00) can not be extended to node t , while the trajectory (11) is redundant given (01), (10). We have thus $N_1 = 2$ and $\nu_1 = 2$.
2. **Iteration 2** Assume that we selected randomly the path (01) among the two, (01) and (10). By doing so we arrive at the node b containing the edges (e_2, e_3, e_5) . According to SE only the edges e_3 and e_5 are relevant. As before, their possible combinations are (00), (01), (10), (11). Arguing similarly to **Iteration 1** we have that $N_2 = 2$ and $\nu_2 = 2$. Consider separately the 2 possibilities associated with edges e_5 and e_3 .
 - (i) Edge e_5 is selected. In this case we can go directly to the sink node t and thus deliver an exact estimator $|\widehat{\mathcal{X}^*}| = \nu_1 \nu_2 = 2 \cdot 2 = 4$. The resulting path is (e_2, e_5) .

3. Iteration 3

(ii) Edge e_3 is selected. In this case we go to t via the edge e_4 . It is readily seen that $N_3 = 1$, $\nu_3 = 1$. We have again $|\widehat{\mathcal{X}^*}| = \nu_1\nu_2\nu_3 = 2 \cdot 2 \cdot 1 = 4$. The resulting path is (e_2, e_3, e_4) .

Note that if we select the combination (10) instead of (01) we would get again $N_2 = 2$ and $\nu_2 = 2$, and thus again an exact estimator $|\widehat{\mathcal{X}^*}| = 4$.

If instead of (14) we would have a directed graph with the following 3 trajectories

$$(e_1, e_4), (e_1, e_3, e_5), (e_2, e_5), \quad (15)$$

then we obtain (with probability $1/2$) an estimator $|\widehat{\mathcal{X}^*}| = 2$ for the path (e_2, e_5) and (with probability $1/4$) an estimator $|\widehat{\mathcal{X}^*}| = 4$, for the paths (e_1, e_4) and (e_1, e_3, e_5) , respectively.

Example 6.2 Extended Bridge

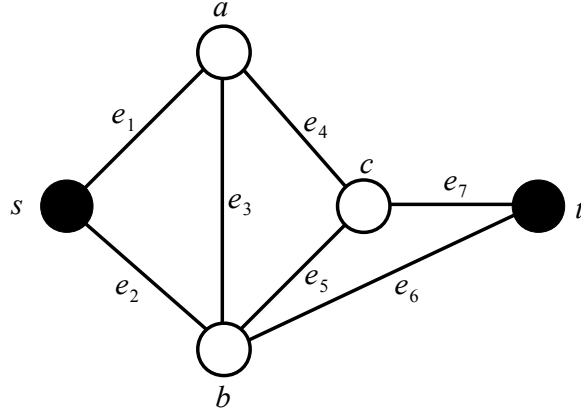


Figure 17: Extended bridge

We have the following 7 trajectories

$$\begin{aligned} &(e_1, e_4, e_7), (e_1, e_3, e_6), (e_1, e_3, e_5, e_7), (e_1, e_4, e_5, e_6), \\ &(e_2, e_6) (e_2, e_5, e_7), (e_2, e_3, e_4, e_7). \end{aligned} \quad (16)$$

1. **Iteration 1** This iteration coincides with **Iteration 1** of Example 6.1. We have $N_1 = 2$ and $\nu_1 = 2$.
2. **Iteration 2** Assume that we selected randomly the combination (01) from the two, (01) and, (10). By doing so we arrive at node b containing the edges (e_2, e_3, e_5, e_6) . According to SE only (e_3, e_5, e_6) are relevant. Of the 7 combinations, only (001), (010), (100) are relevant; there is no path through (000), and the remaining ones are redundant since they result

into the same trajectories as the above three. Thus we have $N_2 = 3$ and $\nu_2 = 3$. Consider separately the 3 possibilities associated with edges e_6, e_5 and e_3 .

(i) Edge e_6 is selected. In this case we can go directly to the sink node t and thus deliver $|\widehat{\mathcal{X}^*}| = \nu_1\nu_2 = 2 \cdot 3 = 6$. The resulting path is (e_2, e_6) . Note that if we select either e_5 or e_3 , there being no direct access to the sink t .

3. Iteration 3

(ii) Edge e_5 is selected. In this case we go to t via the edge e_7 . It is readily seen that $N_3 = 1$, $\nu_3 = 1$. We have again $|\widehat{\mathcal{X}^*}| = \nu_1\nu_2\nu_3 = 2 \cdot 3 \cdot 1 = 6$. The resulting path is (e_2, e_5, e_7) .

(iii) Edge e_3 is selected. By doing so we arrive at node a (the intersection of (e_1, e_3, e_4)). The only relevant edge is e_4 . We have $N_3 = 1$, $\nu_3 = 1$.

4. **Iteration 4** We proceed with the path (e_2, e_3, e_4) , which arrived to point c , the intersection of (e_4, e_5, e_7) . The only relevant edge among (e_4, e_5, e_7) is e_7 . We have $N_4 = 1$, $\nu_4 = 1$ and $|\widehat{\mathcal{X}^*}| = \nu_1\nu_2\nu_3\nu_4 = 2 \cdot 3 \cdot 1 \cdot 1 = 6$. The resulting path is (e_2, e_3, e_4, e_7) .

Figure 18 presents the sub-tree (in bold) corresponding to the path (e_2, e_3, e_4, e_7) for the extended bridge in Figure 17.

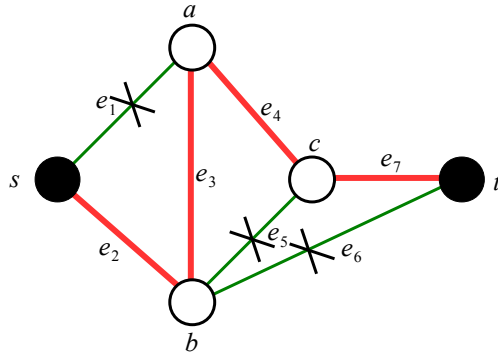


Figure 18: Sub-tree (in bold) corresponding to the path (e_2, e_3, e_4, e_7) .

It is readily seen that if we choose the combination (10) instead of (01) we obtain $|\widehat{\mathcal{X}^*}| = 8$ for all four remaining cases.

Below we summarize all 7 cases.

Path	Probability	Estimator
(e_1, e_4, e_7)	$1/2 \cdot 1/2 \cdot 1/2$	8
(e_1, e_3, e_6)	$1/2 \cdot 1/2 \cdot 1/2$	8
(e_1, e_3, e_5, e_7)	$1/2 \cdot 1/2 \cdot 1/2 \cdot 1$	8
(e_1, e_4, e_5, e_6)	$1/2 \cdot 1/2 \cdot 1/2 \cdot 1$	8
(e_2, e_6)	$1/2 \cdot 1/3$	6
(e_2, e_5, e_7)	$1/2 \cdot 1/3 \cdot 1$	6
(e_2, e_3, e_4, e_7)	$1/2 \cdot 1/3 \cdot 1 \cdot 1$	6

Averaging over the all cases we obtain $|\widehat{\mathcal{X}^*}| = 4 \cdot 1/8 \cdot 8 + 3 \cdot 1/6 \cdot 6 = 7$ and thus, the exact value.

Consider now the case $N^{(e)} > 1$. In particular consider the graph in Figure 17 and assume that $N^{(e)} = 2$. In this case at iteration 1 of SE Algorithm 5.1 both edges e_1 and e_2 will be selected. At iteration 2 we have to chose randomly 2 nodes out of the four ones e_3, e_4, e_5, e_6 . Assume that e_4 and e_5 are selected. Note that by selecting e_5 we completed an entire path which path se_2e_6t . Note, however, that the second path which goes through the edges e_3, e_4 will be not yet completed, since finally it can be either $se_3e_4e_7t$ or $se_3e_4e_5e_6t$. In both cases the shorter path se_2e_6t must be synchronized (length-wise) with the longer ones $se_3e_4e_7t$ or $se_3e_4e_5e_6t$ in the sense that depending on weather $se_3e_4e_5e_6t$ or $se_3e_4e_7t$ is selected we have to insert into se_2e_6t either one auxiliary edge from e_6 to t (denoted $e_6e_6^{(1)}$) or two auxiliary ones from e_6 to t (denoted by $e_6e_6^{(1)}$ and $e_6^{(1)}e_6^{(2)}$). The resulting path (with auxiliary edges) will be either $se_2e_6e_6^{(1)}t$ or $se_2e_6e_6^{(1)}e_6^{(2)}t$.

It follows from above that while adopting Algorithm 5.1 with $N^{(e)} > 1$ for counting the number of paths in a general network one will have to synchronize on-line all its paths with the longest one by adding some auxiliary edges until all paths will match length-wise with the longest one.

6.1.1 SE for Estimation of Probabilities in a Network

Algorithm 5.1 can be readily modified for the estimation of different probabilities in a network, such as the probability that the length $S(\mathbf{X})$ of a randomly chosen path \mathbf{X} is greater than a fixed number γ , i.e.

$$\ell = \mathbb{P}\{S(\mathbf{X}) \geq \gamma\}.$$

Assume first that the length of each edge equals unity. Then the length $S(\mathbf{x})$ of a particular path \mathbf{x} equals to the number of edges from s to t on that path. The corresponding number of iterations is $\frac{S(\mathbf{x})-n_0}{r}$ and the corresponding probability is

$$\ell = \mathbb{P}\left\{\frac{S(\mathbf{X}) - n_0}{r} \geq \gamma\right\} = \mathbb{E}\{I_{\frac{S(\mathbf{X})-n_0}{r} \geq \gamma}\}.$$

Clearly, there is no need to calculate here the weights ν_t . One needs to trace the length $S(\mathbf{x}_j)$ of each path \mathbf{x}_j , $j = 1, \dots, N^{(e)}$.

In case where the lengths of the edges are different from one, $S(\mathbf{x})$ presents the sum of the length of edges associated with that path \mathbf{x} .

Algorithm 6.1 (SE for Estimation Probabilities)

1. Iteration 1

- **Full Enumeration** (Same as in Algorithm 5.1).
- **Calculation of the First Weight Factor** (Redundant). Instead, store the lengths of the corresponding edges $\eta_{1,1}, \dots, \eta_{1,N^{(e)}}$.

2. Iteration t , ($t \geq 2$)

- **Full Enumeration** (Same as in Algorithm 5.1).
- **Stochastic Enumeration** (Same as in Algorithm 5.1).
- **Calculation of the t -th Weight Factor**. (Redundant). Instead, store the lengths of the corresponding edges $\eta_{t,1}, \dots, \eta_{t,N^{(e)}}$.

3. Stopping Rule Proceed with iteration t , $t = 1, \dots, \frac{n-n_0}{r}$ and calculate

$$I_j = I_{\left\{\frac{S(\mathbf{x}_j) - n_0}{r} \geq \gamma\right\}}, \quad j = 1, \dots, N^{(e)}, \quad (17)$$

where as before, $S(\mathbf{x})$ is the length of path \mathbf{x} presenting the sum of the length of the edges associated with \mathbf{x} .

4. Final Estimator Run Algorithm 6.1 for M independent replications and deliver

$$\tilde{\ell} = \frac{1}{MN^{(e)}} \sum_{k=1}^M \sum_{j=1}^{N^{(e)}} I_{\left\{\frac{S(\mathbf{x}_{jk}) - n_0}{r} \geq \gamma\right\}} \quad (18)$$

as an unbiased estimator of ℓ .

6.2 Counting the Number of Perfect Matching (Permanent) in a Graph

Here we deal with application of SE to calculation of the number of matchings in a graph with particular emphasis on the number of *perfect matchings* in a bipartite graph. It is well known [35] that the latter number coincides with the permanent of the corresponding 0 – 1 matrix \mathbf{A} . More precisely, let Q_i denote the set of matchings of size i in the graph G . Assume that Q_n is non-empty, so that

- G has a perfect matching of vertices V_1 and V_2 .
- The number of perfect matchings $|Q_n|$ in G equals the permanent \mathbf{A} , that is, $|Q_n| = \text{per}(\mathbf{A})$, defined as

$$\text{per}(\mathbf{A}) = |\mathcal{X}^*| = \sum_{\mathbf{x} \in \mathcal{X}} \prod_{i=1}^n a_{ix_i}, \quad (19)$$

\mathcal{X} is the set of all permutations $\mathbf{x} = (x_1, \dots, x_n)$ of $(1, \dots, n)$ and the elements a_{ij} can be written as

$$a_{ij} = \begin{cases} 1, & \text{if the nodes } v_{1i} \text{ and } v_{2j} \text{ are in } E \\ 0, & \text{otherwise.} \end{cases}$$

Remark 6.1 Recall that a graph $G = (V, E)$ is bipartite if it has no circuits of odd length. It has the following property: the set of vertices V can be partitioned in two independent sets, $V_1 = (v_{11}, \dots, v_{1n})$ and $V_2 = (v_{21}, \dots, v_{2n})$ such that each edge in E has one vertex in V_1 and one in V_2 . A matching of a graph $G = (V, E)$ is a subset of the edges with the property that no two edges share the same node. In other words, a matching is a collection of edges $M \subseteq E$ such that each vertex occurs at most once in M . A perfect matching is a matching of size n .

Example 6.3 Consider the following adjacency matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \quad (20)$$

The associated bipartite graph is given in Figure 19.

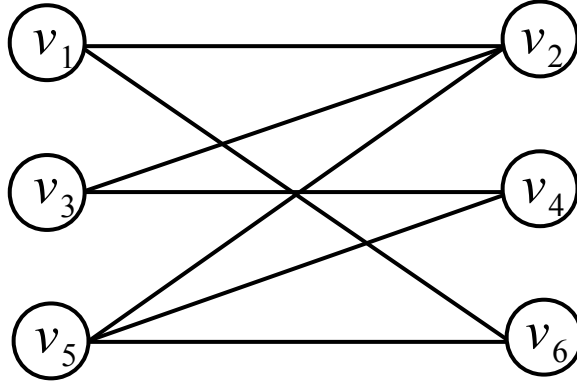


Figure 19: The bipartite graph.

It has the following three perfect matchings

$$\begin{aligned} M_1 &= [(v_1, v_2), (v_3, v_4), (v_5, v_6)], \\ M_2 &= [(v_1, v_6), (v_5, v_4), (v_3, v_2)], \\ M_3 &= [(v_1, v_6), (v_2, v_5), (v_3, v_4)]. \end{aligned} \quad (21)$$

We shall show how SE works for $N^{(e)} = 1$. Its extension to $N^{(e)} > 1$ is simple.

We say that an edge is *active* if the outcome of the corresponding variable is 1 and *passive* otherwise.

- (a) **Iteration 1** Let us start from node 1. Its degree is 2, the corresponding edges are (v_1, v_2) and (v_1, v_6) . Possible outcomes of the two associated Bernoulli ($p = 1/2$) random variables are (00), (01), (10), (11). Note that only (01) and (10) are relevant since neither (00) nor (11) define a perfect matching. Note also that (10) means that edge (v_1, v_2) is active and (v_1, v_6) is passive, while (01) means the other way around. Since (employing the oracle) we obtain that each of the combinations (01) and (10) is valid and since starting from (v_1, v_2) and (v_1, v_6) we generate two different perfect matchings (see (21)), we have that $N_1 = 2, \nu_1 = 2$.

We next proceed separately with the outcomes (10) and (01).

• **Outcome (10)**

- (b) **Iteration 2.** Recall that the outcome (10) means that (v_1, v_2) is active. This automatically implies that all three neighboring edges $(v_2, v_6), (v_2, v_5), (v_2, v_3)$ must be passive. Using the perfect matching oracle we will arrive at the next active edge, which is (v_3, v_4) . Since the degree of node 3 is two and since (v_2, v_3) must be passive we have that $N_2 = 1, \nu_2 = 1$.

- (c) **Iteration 3** Since (v_3, v_4) is active (v_4, v_5) must be passive. The degree of node 5 is three, but since (v_4, v_5) and (v_5, v_2) are passive (v_4, v_5) must be the only available active edge. This implies that $N_3 = 1, \nu_3 = 1$. The final estimator of $|\mathcal{X}^*| = 3$ is $\widehat{|\mathcal{X}^*|} = 2 \cdot 1 \cdot 1 = 2$.

• **Outcome (01)**

- (b) **Iteration 2** Since (01) means that (v_1, v_6) is active we automatically set the neighboring edges $(v_6, v_2), (v_6, v_5)$ as passive. Using an oracle we shall arrive at node 5 which has degree three. Since (v_1, v_2) is passive it is easily seen that each edge, (v_5, v_2) and (v_5, v_4) , will become active with probability $1/2$. This means that with (v_5, v_2) and (v_5, v_4) we are in a similar situation (in the sense of active-passive edges) to that of (v_1, v_2) , and (v_1, v_6) . We thus have for each case $N_2 = 2, \nu_2 = 2$.

- (c) **Iteration 3** It is readily seen that both cases (v_5, v_2) , and (v_5, v_4) lead to $N_3 = 1, \nu_3 = 1$. The resulting perfect matchings (see (21)) and the estimator of $|\mathcal{X}^*|$ are

$$\begin{aligned} M_3 &= [(v_6, v_1), (v_2, v_5), (v_3, v_4)], \\ M_2 &= [(v_1, v_6), (v_5, v_4), (v_3, v_2)] \end{aligned} \tag{22}$$

and

$$\widehat{|\mathcal{X}^*|} = 2 \cdot 2 \cdot 1 = 4,$$

respectively.

Since each initial edge (v_1, v_2) and (v_1, v_6) at **Iteration 1** is chosen with probability $1/2$, by averaging over both cases we obtain the exact result, namely $|\mathcal{X}^*| = 3$.

It is not difficult to see that

1. If we select $N^{(e)} = 2$ instead of $N^{(e)} = 1$ we obtain $|\widehat{\mathcal{X}^*}| = 3/2 \cdot 2 \cdot 1 = 3$, that is the exact value $|\mathcal{X}^*| = 3$.
2. Since $|\mathcal{X}^*| = 3$, the optimal zero variance importance sampling pdf $g^*(\mathbf{x}) = 1/3$. Its corresponding conditional probabilities (for $N^{(e)} = 1$, starting at node 1) are given below:

$$g^* = \begin{pmatrix} 0 & 1/3 & 0 & 0 & 0 & \mathbf{2/3} \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1, \mathbf{1}) & 0 \\ 0 & \mathbf{1/2} & 0 & \mathbf{1/2} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \quad (23)$$

The plain and bold numbers in (23) correspond to the trajectories starting at edges (v_1, v_2) and (v_1, v_6) , respectively. Also, the notation $(1, \mathbf{1})$ means that the two perfect matching trajectories (one starting at (v_1, v_2) and one at (v_1, v_6) , each with probability 1) pass through node (v_3, v_4) .

6.3 Counting SAT's

The most common SAT problem comprises the following two components:

- A set of n Boolean variables $\{x_1, \dots, x_n\}$, representing statements that can either be **TRUE** ($=1$) or **FALSE** ($=0$). The negation (logical NOT) of a variable x is denoted by \bar{x} . For example, $\overline{\text{TRUE}} = \text{FALSE}$.
- A set of m distinct *clauses* $\{C_1, C_2, \dots, C_m\}$ of the form $C_j = z_{j_1} \vee z_{j_2} \vee \dots \vee z_{j_q}$, where the z 's are literals and \vee denotes the logical **OR** operator. For example, $0 \vee 1 = 1$.

The binary vector $\mathbf{x} = (x_1, \dots, x_n)$ is called a *truth assignment*, or simply an *assignment*. Thus, $x_i = 1$ assigns truth to x_i and $x_i = 0$ assigns truth to \bar{x}_i , for each $i = 1, \dots, n$.

Denoting the logical **AND** operator by \wedge , we can represent the above SAT problem via a single *formula* as

$$F = C_1 \wedge C_2 \wedge \dots \wedge C_m,$$

where the C_j 's consist of literals connected with only \vee operators. The SAT formula is then said to be in *conjunctive normal form* (CNF).

Note that although for general SAT's the decision making is an NP-hard problem, there are several very fast heuristics for this purpose. The most popular one is the famous DPLL solver (oracle) [10], on which two main heuristic algorithms are based for approximate counting with emphasis on SAT's. The first, called **ApproxCount** and introduced by Wei and Selman in [45], is a *local search* method that uses Markov Chain Monte Carlo (MCMC) sampling to approximate the true counting quantity. It is fast and has been shown to yield good estimates for feasible solution counts, but there are no guarantees as to uniformity of the MCMC samples. Gogate and Dechter [17], [18] recently proposed an alternative counting technique called **SampleMinisat**, based on sampling from the so-called backtrack-free search space of a Boolean formula through **SampleSearch**. An approximation of the search tree thus found is used as the *importance sampling* density instead of a uniform distribution over all solutions. They also derived a lower bound for the unknown counting quantity. Their empirical studies demonstrate the superiority of their method over its competitors.

7 Choosing a Good Number $N^{(e)}$ of Elites

Assume that we have a fixed budget $K = N^{(e)} \times M$ to estimate $|\mathcal{X}^*|$. We want to choose M such that the variance of $|\widetilde{\mathcal{X}^*}|(M)$ (with respect to M) is minimized. One would be tempted to choose $M = 1$, since it is indeed optimal for the above 2-SAT model with clauses $C_i = x_i \vee \bar{x}_{i+1} \geq 1$, $i = 1, \dots, n$ and $|\mathcal{X}^*| = n + 1$ (see Table 2). Recall that by setting $M = 1$ and choosing $N^{(e)} = n + 1$ we obtained $|\widetilde{\mathcal{X}^*}| = 0$. If the budget $K < n + 1$, we would still choose $N^{(e)} = K$ and obtain maximum variance reduction (see Table 2). Indeed, we found numerically that choosing $M = 1$ is also typically the best policy for randomly generated SAT's models. For other randomly generated counting problems, such as counting the number of perfect matchings (permanent) and the number of paths in a network, we found, however, that $M = 1$ is not necessarily the best choice. In fact, we found that the best M can vary from 1 till K ($N^{(e)} = 1$). The reason is that in contrast to the case $N^{(e)} = 1$, where all generated paths are independent, they are typically not so for the case $N^{(e)} > 1$. The dependent paths might be positively correlated and as result the variance can blow up as compared to the independent case $N^{(e)} = 1$. To see this consider Figure 17 of the extended bridge and assume that $N^{(e)} = 2$. Since $N^{(e)} = 2$ we arrive at iteration 1 to both nodes a and b . We have here $N_1 = 5$ (edges e_3, e_4 for node a and edges e_3, e_5, e_6 for node b). At iteration 2 we have to choose randomly $N^{(e)} = 2$ elites out of $N_1 = 5$. Since the edge e_3 is common for both paths, that is for paths going through edges e_1e_3 and e_2e_3 , respectively, there is a positive probability that e_3 will be common in both paths. Clearly, these two paths will be positively correlated.

Although we found numerically that for randomly generated graphs the relative error fluctuates only 2-3 times by varying M from 1 till K we suggest to perform several small pilot runs with several different values of M and selected the best one.

8 Numerical Results

Here we present numerical results with the SE-OSLA and SE algorithms. In particular, we use SE-OSLA Algorithm 4.1 for counting SAW's. The reason for doing so is that we are not aware of any polynomial decision making algorithm (oracle) for SAW's. For the remaining problems we use the SE Algorithm 5.1, since polynomial decision making algorithms (oracles) are available for them. If not stated otherwise we use $r = 1$.

To achieve high performance of SE Algorithm 5.1 we set M as suggested to follow Section 7. In particular

1. For SAT models to set $M = 1$. Note that by doing so, $N^{(e)} = K$, where K is the allowed budget. Also in this case $N^{(e)}$ is the only parameter of SE. For the instances, where we occasionally obtained (after simulation) an exact $|\mathcal{X}^*|$, that is where $|\mathcal{X}^*|$ is relatively small and where we originally set $N^{(e)} \geq |\mathcal{X}^*|$, we purposely reset $N^{(e)}$ (to be fair with the other methods) by choosing a new $N_*^{(e)}$, satisfying $N_*^{(e)} < |\mathcal{X}^*|$ and run SE again. By doing so we prevent SE from being an ideal (zero variance) estimator.
2. For other counting problems, such as counting the number of perfect matchings (permanent) and the number of paths in a network, to perform small pilot runs with several values of M and select the best one.

8.1 Counting SAW's

Note that since SAW is symmetric, we can confine ourselves to a single octant and thus save CPU time.

Tables 3 and 4 present the performance of the SE-OSLA Algorithm 4.1 for SAW for $n = 500$ and $n = 1,000$ respectively, with $r = 2$, $n_0 = 4$ and $M = 20$, (see (7)). This corresponds to the initial values $N_0^{(e)} = 100$ and $N_1 = 780$, (see also iteration $t = 0$ in Table 5).

Table 3: Performance of SE-OSLA Algorithm 4.1 for SAW with $n = 500$

Run N_0	Iterations	$ \widetilde{\mathcal{X}^*} $	RE of $ \widetilde{\mathcal{X}^*} $	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

Table 4: Performance of of the SE-OSLA Algorithm 4.1 for SAW for $n = 1,000$

Run N_0	Iterations	$ \widetilde{\mathcal{X}}^* $	RE of $ \widetilde{\mathcal{X}}^* $	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

Table 5 presents dynamics of one of the runs of the SE-OSLA Algorithm 4.1 for $n = 500$. We used the following notations

1. $N_t^{(e)}$ denotes the number of elites at iteration t .
2. n_t denote the level reached at iteration t .
3. $\nu_t = N_t^{(e)}/N_t$.

Table 5: Dynamics of a run of the SE-OSLA Algorithm 4.1 for $n = 500$

t	n_t	$N_t^{(e)}$	N_t	$\widehat{\nu}_t$	$ \widehat{\mathcal{X}}_t^* $
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

8.2 Counting the Number of Trajectories in a Network

Consider the dodecahedron graph in Figure 20.

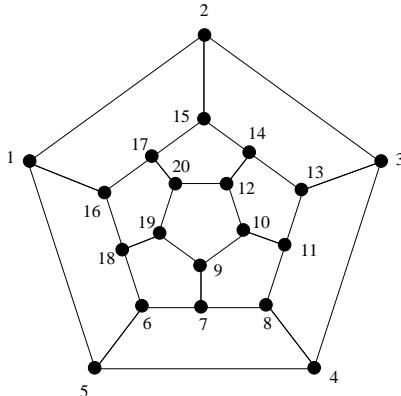


Figure 20: The dodecahedron graph

The goal is to count the number of paths $|\mathcal{X}^*|$ from node 1 to node 20. Using full enumeration, we obtained $|\mathcal{X}^*| = 1338$. Clearly if we set $N_*^{(e)} = |\mathcal{X}^*| = 1338$, SE will be exact. We purposely worsen the situation. We set $N_t^{(e)} = 10$ and $M = 10$.

Table 6 presents the performance of the SE Algorithm 5.1 for the dodecahedron graph with $N_t^{(e)} = 10$ and $M = 10$.

Table 6: Performance of SE Algorithm 4.1 for the dodecahedron graph with $N_t^{(e)} = 100$.

Run N_0	Iterations	$ \widetilde{\mathcal{X}^*} $	CPU
1	15	1567.3	3.467
2	17	1644.8	3.252
3	15	1220.3	2.956
4	15	1364.4	2.992
5	17	1567.4	3.134
6	16	1342.9	3.029
7	18	1148.7	3.033
8	14	1237.0	2.956
9	18	1322.5	3.083
10	16	1357.7	3.141
Average	16.1	1377.3	3.104

Based on 100 runs, we found that $RE = 0.0121$.

We counted in addition the number of paths in randomly generated graphs using Erdos-Renyi approach (see Section 10.3). We generated the graphs with $p = \ln n/n$ (see Section 10.3) and run all models with $N^{(e)} = 1$ and $M = 30,000$.

We found that SE performs reliable ($RE \leq 0.05$) for $n \leq 200$, provided the CPU time is limited by 5-10 minutes.

8.3 Counting the Number of Perfect Matching (Permanent) in a Graph

Consider the adjacency matrix \mathbf{A} with $|V| = 20$ and $|E| = 78$ given as

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad (24)$$

The true number of perfect matchings (permanent) is $|\mathcal{X}^*| = 255,112$, obtained using full enumeration.

Table 7 presents the performance of the SE Algorithm 5.1 for the matrix \mathbf{A} in (24) with $N_t^{(e)} = 10$ and $M = 100$.

Table 7: Performance of SE Algorithm 5.1 for the for the matrix \mathbf{A} in (24) with $N_t^{(e)} = 10$ and $M = 100$.

Run N_0	Iterations	$ \widetilde{\mathcal{X}^*} $	CPU
1	10	2.59E+05	1.911
2	10	2.48E+05	1.882
3	10	2.67E+05	1.889
4	10	2.44E+05	1.887
5	10	2.53E+05	1.889
6	10	2.50E+05	1.880
7	10	2.54E+05	1.883
8	10	2.57E+05	1.886
9	10	2.48E+05	1.883
10	10	2.51E+05	1.879
Average	10	2.53E+05	1.887

We found that the relative error is near 0.0275.

Consider the following $\mathbf{A} = 30 \times 30$ matrix with true number of perfect matchings (permanent) is $|\mathcal{X}^*| = 266$ obtained using full enumeration.

Table 9: Performance of the splitting algorithm with $N = 15,000$ and $\rho = 0.1$.

Run N_0	Iterations	$ \widetilde{\mathcal{X}}^* $	CPU
1	27	240.45	2.628
2	27	299.84	2.596
3	27	201.41	2.611
4	27	428.98	2.589
5	27	279.92	2.613
6	27	225.10	2.604
7	27	172.88	2.606
8	27	213.04	2.588
9	27	239.31	2.599
10	27	251.52	2.584
Average	27	255.24	2.602

We found that the relative error is near 0.2711. It follows that SE is about 100 times faster than splitting.

We also applied the SE algorithm for counting arbitrary matchings in a bipartite graph in the graphs of sizes up to several hundreds. We managed to obtain relative error of about 0.02 within reasonable time.

8.4 Counting SAT's

Here we present numerical results with SE for several 3-SAT models. We set $r = 1$ in for all models. Recall that in this case instead of a polynomial decision making oracle [10], (available for 2-SAT only) we use a heuristic one, based on the same DPLL solver [11, 10] as in Gogate and Dechter [17], [18]. Note also that the SE Algorithm 5.1 remains the same, provided the polynomial decision making oracle is replaced by the above mentioned heuristic DPLL solver.

Table 10 presents the performance of the SE Algorithm 5.1 for the 3-SAT 75×325 model with exactly 2258 solutions. We set $N_t^{(e)} = 1000$.

Table 10: Performance of SE Algorithm 5.1 for the 3-SAT 75×325 model.

Run N_0	Iterations	$ \widetilde{\mathcal{X}}^* $	RE of $ \widetilde{\mathcal{X}}^* $	CPU
1	75	2359.780	0.045	2.74
2	75	2389.660	0.058	2.77
3	75	2082.430	0.078	2.79
4	75	2157.850	0.044	2.85
5	75	2338.100	0.035	2.88
6	75	2238.940	0.008	2.75
7	75	2128.920	0.057	2.82
8	75	2313.390	0.025	3.04
9	75	2285.910	0.012	2.81
10	75	2175.790	0.036	2.85
Average	75	2247.077	0.040	2.83

Table 11 presents performance of the SE Algorithm 5.1 for the 3-SAT 75×270 model. We set $N_t^{(e)} = 2,000$. The exact solution for this instance is $\mathcal{X}^* = 1,346,963$ and is obtained via full enumeration using the backward method. It is interesting to note that for this instance (with relatively small $|\mathcal{X}^*|$) the CPU time of the exact backward method is 332 seconds (compare with average 16.8 second time of SE in Table 11).

Table 11: Performance of SE Algorithm 5.1 for the 3-SAT 75×270 model.

Run N_0	Iterations	$ \widetilde{\mathcal{X}}^* $	RE of $ \widetilde{\mathcal{X}}^* $	CPU
1	75	1.42E+06	0.057	16.88
2	75	1.37E+06	0.021	16.91
3	75	1.31E+06	0.025	17.24
4	75	1.35E+06	0.002	17.38
5	75	1.31E+06	0.027	16.75
6	75	1.32E+06	0.018	16.81
7	75	1.32E+06	0.019	16.51
8	75	1.25E+06	0.069	16.27
9	75	1.45E+06	0.077	16.3
10	75	1.33E+06	0.011	16.95
Average	75	1.35E+06	0.033	16.80

Table 12 presents the performance of the SE Algorithm 5.1 for a 3-SAT 300×1080 model. We set $N_t^{(e)} = 5,000$.

Table 12: Performance of SE Algorithm 5.1 for SAT 300×1080 model with $N_t^{(e)} = 5,000$ and $r = 1$.

Run N_0	Iterations	$ \widetilde{\mathcal{X}^*} $	RE of $ \widetilde{\mathcal{X}^*} $	CPU
1	300	3.30E+24	2.61E-03	2010.6
2	300	3.46E+24	5.10E-02	2271.8
3	300	3.40E+24	3.22E-02	2036.8
4	300	3.42E+24	4.00E-02	2275.8
5	300	3.39E+24	2.83E-02	2022.4
6	300	3.35E+24	1.67E-02	2267.8
7	300	3.34E+24	1.46E-02	2019.6
8	300	3.34E+24	1.39E-02	2255.4
9	300	3.32E+24	9.13E-03	2031.7
10	300	3.33E+24	1.21E-02	2149.6
Average	300	3.36E+24	2.21E-02	2134.1

Note that in this case the estimator of $|\mathcal{X}^*|$ is very large and, thus full enumeration is impossible. We made, however, the exact solution $|\mathcal{X}^*|$ available as well. It is $|\mathcal{X}^*| = (1,346,963)^4 = 3.297E + 24$, and is obtained using a specially designed procedure (see Remark 8.1 below) for SAT instances generation. In particular the instance matrix 300×1080 was generated from the previous one 75×270 , for which $|\mathcal{X}^*| = 1,346,963$.

Remark 8.1 *Generating an instance with an available solution* We shall show that given a small instance with a known solution $|\mathcal{X}^*|$ we can generate an associated instance of any size and still obtain its exact solution. Suppose without loss of generality that we have k instances (of relatively small sizes) with known $|\mathcal{X}_i^*|$, $i = 1, \dots, k$. Denote those instances by I_1, \dots, I_k , their dimensionality by $(n_1, m_1), \dots, (n_k, m_k)$ and their corresponding solutions by $|\mathcal{X}_1^*|, \dots, |\mathcal{X}_k^*|$. We will show how to construct a new SAT instance that will have a size $(\sum_{i=1}^k n_i, \sum_{i=1}^k m_i)$ and its exact solution will be equal to $\prod_{i=1}^k |\mathcal{X}_i^*|$. The idea is very simple. Indeed, denote the variables of I_1 by x_1, \dots, x_{n_1} . Now take the second instance and rename it's variables from x_1, \dots, x_{n_2} to $x_{n_1+1}, \dots, x_{n_2+n_1}$, i.e to each variable index of I_2 we add n_1 new variables. Continue in the same manner with the rest of the instances. It should be clear that we have now an instance of size $(\sum_{i=1}^k n_i, \sum_{i=1}^k m_i)$. Let us look next at some particular solution $X_1, \dots, X_{n_1}, X_{n_1+1}, \dots, X_{n_1+n_2}, \dots, X_{\sum_{i=1}^k n_i}$ of this instance. This solution consists of independent components of sizes n_1, \dots, n_k , and it is straight forward to see that the total number of those solutions is $\prod_{i=1}^k |\mathcal{X}_i^*|$. It follows therefore that one can easily construct a large SAT instance from a set of small ones and still have an exact solution for it. Note that the above 300×1080 instance with exactly $(1,346,963)^4$ solutions was obtained from the 4 identical instances of size 75×270 , each with exactly 1,346,963 solutions.

We also performed experiments with different values of r . Table 13 summarizes the results. In particular it presents the relative error (RE) for $r_1 = 1$, $r_3 = 3$ and $r_5 = 5$ with SE run for a predefined time period for each instance. We can see that changing r does not affect the relative error.

Table 13: The relative errors as function of r .

Instance	$r = 1$	$r = 2$	$r = 3$
20x80, $N_t^{(e)}=10$	2.284E-02	1.945E-02	2.146E-02
75x325, $N_t^{(e)} = 2, 000$	5.057E-02	4.587E-02	5.614E-02
75x270, $N_t^{(e)} = 10, 000$	4.449E-02	4.745E-02	4.056E-02

At the rest of this section we compare SE with splitting and SampleSearch for several 3-SAT instances. Before doing so we make the following remarks.

Remark 8.2 SE versus splitting Note that

1. In the splitting method [40] $|\mathcal{X}^*|$ is presented as $|\mathcal{X}^*| = \ell|\mathcal{X}_0|$, where $|\mathcal{X}_0|$ is the cardinality of the entire sample space, such as $|\mathcal{X}_0| = 2^n$ and the rare event probability ℓ is presented as

$$\ell = \mathbb{E} \left[I_{\{\sum_{j=1}^m C_j(\mathbf{X})=m\}} \right], \quad (25)$$

where \mathbf{X} has a uniform distribution on a finite n -dimensional set \mathcal{X}_0 , and m is the number of clauses C_j , $j = 1, \dots, m$. To estimate \mathcal{X}^* the splitting algorithm generates an adaptive sequence of pairs

$$\{(m_0, g^*(\mathbf{x}, m_0)), (m_1, g^*(\mathbf{x}, m_1)), (m_2, g^*(\mathbf{x}, m_2)), \dots, (m_T, g^*(\mathbf{x}, m_T))\}, \quad (26)$$

where $g^*(\mathbf{x}, m_t)$, $t = 0, 1, \dots, T$ is uniformly distributed on the set \mathcal{X}_t and such that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \dots \supset \mathcal{X}_T = \mathcal{X}^*$.

2. In contrast to splitting which samples from a sequence of n -dimensional pdf's $g^*(\mathbf{x}, m_t)$ (see (26)) sampling in the SE Algorithm 5.1 is minimal; it resort to sampling n times only. In terms of urn models it is the same as saying that with SE we draw (without replacing) $N_t^{(e)}$ balls from an urn containing $N_t \geq N_t^{(e)}$, $t = 1, \dots, n$ ones.
3. Splitting relies on the time-consuming MCMC method and in particular on the Gibbs sampler, while SE dispenses with them and is thus substantially simpler and faster. For more details on splitting see [40].
4. The limitation of SE relative to splitting is that the former is suitable for counting, where only fast (polynomial) decision making oracles are available, while splitting dispenses with them. In addition it is not suitable for optimization and rare-events as splitting does.

Remark 8.3 SE versus SampleSearch

The main difference between "SampleSearch" [17, 18] and SE is that the former approximates an entire #P-complete counting problem like SAT by incorporating IS in the oracle. As a result, it is only asymptotically unbiased. Wei and Selman's [45] `ApproxCount` is similar to "SampleSearch" in that it uses an MCMC sampler instead of IS. In contrast to both the above, SE reduces the difficult counting problem to a set of simple ones, applying the oracle each time directly (via the **Full Enumeration** step) to the elite trajectories of size $N_t^{(e)}$. Note also that unlike both the above, there is no randomness involved in SE as far as the oracle is concerned in the sense that once the elite trajectories are given, the oracle generates (via **Full Enumeration**) a *deterministic* sequence of new trajectories of size N_t . As a result, SE is unbiased for any $N_t^{(e)} \geq 1$ and is typically more accurate than "SampleSearch". Our numerical results below confirm this.

Table 14 presents a comparison of the efficiencies of SE (at $r = 1$) with those of `SampleSearch` and splitting (SaS and Split columns respectively) for several SAT instances. We ran all three methods for the same amount of time (Time column).

Table 14: Comparison of the efficiencies of SE, `SampleSearch` and standard splitting

Instance	Time	SaS	SaS RE	SE	SE RE	Split	Split RE
20x80	1 sec	14.881	7.95E-03	15.0158	5.51E-03	14.97	3.96E-02
75x325	137 sec	2212	2.04E-02	2248.8	9.31E-03	2264.3	6.55E-02
75x270	122 sec	1.32E+06	2.00E-02	1.34E+06	1.49E-02	1.37E+06	3.68E-02
300x1080	1600 sec	1.69E+23	9.49E-01	3.32E+24	3.17E-02	3.27E+24	2.39E-01

It is readily seen that in terms of speed (which equals to $(RE)^2$) SE is faster than `SampleSearch`, by about 2-10 times and than the standard splitting in [40] by about 20-50 times. Similar comparison results were obtained for other models including perfect matching. Our explanation is that SE is an SIS method, while `SampleSearch` and splitting are not - in the sense that SE samples sequentially with respect to coordinates x_1, \dots, x_n , while the other two sample (random vectors \mathbf{X} from the IS pdf $g(\mathbf{x})$) in the entire n -dimensional space.

We also ran the models of Table 14 with the (exact) backward method. We found that for the first 3 models the CPU times are 0.003, 0.984 and 332 seconds, respectively. As mentioned before the backward method cannot handle the last model in reasonable time, because $|\mathcal{X}^*|$ is very large.

9 Concluding Remarks and Further Research

In this work we introduced a new generic *sequential importance sampling* (SIS) algorithm, called *stochastic enumeration* (SE) for counting #P-complete prob-

lems such as the number of satisfiability assignments, the number of paths in a network and the number of perfect matchings in a graph (permanent). We showed that SE presents a natural generalization of the classic *one-step-look-ahead* (OSLA) algorithm in the sense that it

- Runs in parallel multiple trajectories instead of a single one.
- Employs a polynomial time decision making oracle, which can be viewed as an *n-step-look-ahead* algorithm, n being the size of the problem rather than OSLA.

The presented extensive simulation studies indicate good performance of SE Algorithm 4.1 as compared with the well-known algorithms and in particular over the splitting [40], and SampleSearch [17] methods.

As for further research, we are planning to

- Find the set of optimal parameters $\{\widetilde{N}_t^{(e)}, r, \widetilde{M}\}$ which, for fixed n , minimizes the variance of the estimator $|\mathcal{X}|^* = |\mathcal{X}|^*(\widetilde{N}_t^{(e)}, r, \widetilde{M})$ in (7) as function of $N_t^{(e)}, r, M$, provided $N_t^{(e)} = N^{(e)}, \forall t$.
- Apply the SE method to a wide class of NP-hard counting and rare-event problems, such as estimating the reliability of a network.
- Establish a mathematical foundation for SE and to investigate its complexity properties. In particular to extend Rasmussen [37] FPRAS result for counting permanent with OSLA to counting some other graphs quantities using n SLA and SE.

10 Appendices

10.1 SIS Method

Sequential importance sampling (SIS), also called *dynamic importance sampling*, is simply importance sampling carried out in a sequential manner. To explain the SIS procedure, consider the expected performance

$$\ell = \mathbb{E}_f[S(\mathbf{X})] = \int S(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} , \quad (27)$$

where H is the sample performance and f is the probability density of \mathbf{X} .

Let g be another probability density such that $H f$ is *dominated* by g . That is, $g(\mathbf{x}) = 0 \Rightarrow S(\mathbf{x}) f(\mathbf{x}) = 0$. Using the density g we can represent ℓ as

$$\ell = \int S(\mathbf{x}) \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x} = \mathbb{E}_g \left[S(\mathbf{X}) \frac{f(\mathbf{X})}{g(\mathbf{X})} \right] , \quad (28)$$

where the subscript g means that the expectation is taken with respect to g . Such a density is called the *importance sampling* density, Consequently, if

$\mathbf{X}_1, \dots, \mathbf{X}_N$ is a *random sample* from g , that is, $\mathbf{X}_1, \dots, \mathbf{X}_N$ are iid random vectors with density g , then

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^N S(\mathbf{X}_k) \frac{f(\mathbf{X}_k)}{g(\mathbf{X}_k)} \quad (29)$$

is an unbiased estimator of ℓ . This estimator is called the *importance sampling estimator*. The ratio of densities,

$$W(\mathbf{x}) = \frac{f(\mathbf{x})}{g(\mathbf{x})}, \quad (30)$$

is called the *likelihood ratio*. Suppose that (a) \mathbf{X} is decomposable, that is, it can be written as a vector $\mathbf{X} = (X_1, \dots, X_n)$, where each of the X_i may be multi-dimensional, and (b) it is easy to sample from $g(\mathbf{x})$ sequentially. Specifically, suppose that $g(\mathbf{x})$ is of the form

$$g(\mathbf{x}) = g_1(x_1) g_2(x_2 | x_1) \cdots g_n(x_n | x_1, \dots, x_{n-1}), \quad (31)$$

where it is easy to generate X_1 from density $g_1(x_1)$, and sequential on $X_1 = x_1$, the second component from density $g_2(x_2 | x_1)$, and so on, until one obtains a single random vector \mathbf{X} from $g(\mathbf{x})$. Repeating this independently N times, each time sampling from $g(\mathbf{x})$, one obtains a random sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from $g(\mathbf{x})$ and estimates ℓ according to (29). To further simplify the notation, we abbreviate (x_1, \dots, x_t) to $\mathbf{x}_{1:t}$ for all t . In particular, $\mathbf{x}_{1:n} = \mathbf{x}$. Typically, t can be viewed as a (discrete) time parameter and $\mathbf{x}_{1:t}$ as a path or trajectory. By the product rule of probability, the target pdf $f(\mathbf{x})$ can also be written sequentially, that is,

$$f(\mathbf{x}) = f(x_1) f(x_2 | x_1) \cdots f(x_n | \mathbf{x}_{1:n-1}). \quad (32)$$

From (31) and (32) it follows that we can write the likelihood ratio in product form as

$$W(\mathbf{x}) = \frac{f(x_1) f(x_2 | x_1) \cdots f(x_n | \mathbf{x}_{1:n-1})}{g_1(x_1) g_2(x_2 | x_1) \cdots g_n(x_n | \mathbf{x}_{1:n-1})} \quad (33)$$

or recursively as

$$w_t(\mathbf{x}_{1:t}) = u_t w_{t-1}(\mathbf{x}_{1:t-1}), \quad t = 1, \dots, n, \quad (34)$$

where $w_t(\mathbf{x}_{1:t})$ denotes the likelihood ratio up to time t , $w_0(\mathbf{x}_{1:0}) = 1$ is the initial weight, and $u_1 = f(x_1)/g_1(x_1)$ and

$$u_t = \frac{f(x_t | \mathbf{x}_{1:t-1})}{g_t(x_t | \mathbf{x}_{1:t-1})} = \frac{f(\mathbf{x}_{1:t})}{f(\mathbf{x}_{1:t-1}) g_t(x_t | \mathbf{x}_{1:t-1})}, \quad t = 2, \dots, n \quad (35)$$

are *incremental weights*.

Remark 10.1 Note that the incremental weights u_t only need to be defined up to a constant, say c_t , for each t . In this case the likelihood ratio $W(\mathbf{x})$ is known up to a constant as well, say $W(\mathbf{x}) = C V(\mathbf{x})$, where $1/C = \mathbb{E}_g[V(\mathbf{X})]$

can be estimated via the corresponding sample mean. In other words, when the normalization constant is unknown, one can still estimate ℓ using the the following *weighted sample estimator*:

$$\widehat{\ell} = \frac{\sum_{k=1}^N S(\mathbf{X}_k) W_k}{\sum_{k=1}^N W_k}. \quad (36)$$

rather than the likelihood ratio estimator (30). Here the $\{W_k\}$, with $W_k = W(\mathbf{X}_k)$, are interpreted as *weights* of the random sample $\{\mathbf{X}_k\}$, and the sequence $\{(\mathbf{X}_k, W_k)\}$ is called a *weighted (random) sample* from $g(\mathbf{x})$.

Summarizing, the SIS method can be written as follows.

Algorithm 10.1 (SIS Algorithm)

1. For each finite $t = 1, \dots, n$, sample X_t from $g_t(x_t | \mathbf{x}_{1:t-1})$.
2. Compute $w_t = u_t w_{t-1}$, where $w_0 = 1$ and u_t is given in (35).
3. Repeat N times and estimate ℓ via $\widehat{\ell}$ in (29) or $\widehat{\ell}$ in (36).

Note that $\widehat{\ell}$ is an unbiased estimator of ℓ , while $\widehat{\ell}_w$ is an asymptotically consistent estimator of ℓ .

10.2 DPLL Algorithm from Wikipedia

Davis-Putnam-Logemann-Loveland (DPLL) algorithm [10] is a complete, backtracking-based algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. for solving the CNF-SAT problem. DPLL is a highly efficient procedure and after more than 40 years still forms the basis for most efficient complete SAT solvers, as well as for many theorem provers for fragments of first-order logic.

The basic backtracking algorithm runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value. This is known as the splitting rule, as it splits the problem into two simpler sub-problems. The simplification step essentially removes all clauses which become true under the assignment from the formula, and all literals that become false from the remaining clauses.

10.3 Random Graphs Generation

This section is taken almost verbatim from <http://en.wikipedia.org/wiki/Erd>

Random graphs generation is associated with the *ErdosRenyi* model, named for Paul Erdos and Alfred Renyi. There are two closely related variants of the ErdosRenyi random graph model, the so-called (i) $G(n, p)$ and (ii) $G(n, M)$ model.

(i) $G(n, p)$ model In the $G(n, p)$ model, a graph is constructed by connecting nodes randomly. Each edge is included in the graph with probability p independent from every other edge. Equivalently, all graphs with n nodes and m edges have equal probability of

$$p^m(1-p)^{\binom{n}{2}-m}.$$

The distribution of the degree of any particular vertex v is binomial:

$$\mathbb{P}(\text{deg}(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k},$$

where n is the total number of vertices in the graph.

A simple way to generate a random graph in $G(n, p)$ is to consider each of the possible $\binom{n}{2}$ edges in some order and then independently add each edge to the graph with probability p .

The expected number of edges in $G(n, p)$ is $p\binom{n}{2}$, and each vertex has expected degree $p(n-1)$. Since every thing is distributed independently their corresponding variances are $p(1-p)\binom{n}{2}^2$, and $p(1-p)(n-1)^2$.

The parameter p in this model can be thought of as a weighting function; as p increases from 0 to 1, the model becomes more and more likely to include graphs with more edges and less and less likely to include graphs with fewer edges. In particular, $p = 0.5$ corresponds to the case where all $2^{\binom{n}{2}}$ graphs on n vertices are chosen with equal probability. The behavior of random graphs are often studied asymptotically in n , that is when the number of vertices tends to infinity. For example, the statement "almost every graph in is connected" means "as n tends to infinity, the probability that a graph on n vertices with edge probability is connected tends to 1".

(ii) $G(n, M)$ model In the $G(n, M)$ model, a graph is chosen uniformly at random from the collection of all graphs, which have n nodes and M edges.

In practice, the $G(n, p)$ model is the one more commonly used today, in part due to the ease of analysis allowed by the independence of the edges.

Erdos and Renyi described the behavior of $G(n, p)$ very precisely for various values of p . In particular

1. If $np < 1$, then a graph in $G(n, p)$ will almost surely have no connected components of size larger than $O(\ln n)$.
2. If $np = 1$, then a graph in $G(n, p)$ will almost surely have a largest component whose size is of order $n^{2/3}$.
3. If np tends to a constant $c > 1$, then a graph in $G(n, p)$ will almost surely have a unique giant component containing a positive fraction of the vertices. No other component will contain more than $O(\ln n)$ vertices.
4. If $p < \frac{(1-\varepsilon)\ln n}{n}$, then a graph in $G(n, p)$ will almost surely contain isolated vertices, and thus be disconnected.
5. If $p > \frac{(1+\varepsilon)\ln n}{n}$, then a graph in $G(n, p)$ will almost surely be connected.

Thus $\frac{\ln n}{n}$ is a sharp threshold for the connectedness of $G(n, p)$.

The transition at $np = 1$ from giant component to small component can be regarded as a phase transition studied by percolation theory.

In our numerical results we shall always deal with connected graphs, that is when $p > \frac{(1+\varepsilon)\ln n}{n}$.

Acknowledgments

I would like to thank Ad Ridder and Dirk Kroese for valuable comments on the earlier draft, Fred Cerou for introducing me to the committor function and providing Remark 5.1 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] Botev Z. I. and D. P. Kroese. "An Efficient Algorithm for Rare-event Probability Estimation, Combinatorial Optimization, and Counting". *Methodology and Computing in Applied Probability*, 10 (4), 471-505, 2008.
- [3] Botev, Z.I. and D. P. Kroese. "Efficient Monte Carlo simulation via the generalized splitting method", *Statistics and Computing*, 2011.
- [4] Cappe, O. , Godsill, S. J. and E. Moulines. An overview of existing methods and recent advances in sequential Monte Carlo. *IEEE Proceedings*, 95(5):899-924, 2007.
- [5] Cerou F., Del Moral P., Furon T. and A. Guyader. "Sequential Monte Carlo for rare event estimation" to appear in *Statistics and Computing*, 2011.
- [6] Cerou F., Del Moral P., Le Gland F. and P. Lezaud. "Genetic genealogical models in rare event analysis. *Latin American Journal of Probability and Mathematical Statistics*", 1, 2006.
- [7] Cerou F. and A. Guyader. "Adaptive multilevel splitting for rare event analysis. *Stoch. Anal. Appl.*", 25(2):417443, 2007.
- [8] N. Clisby. "Efficient implementation of the pivot algorithm for selfavoiding walks", arXiv:1005.1444, *J. Stat. Phys.* 140:349392, 2010.
- [9] Cormen T. H., Leiserson C. E., Rivest R. L., and S. Clifford. "Section 24.3: Dijkstra's algorithm". *Introduction to Algorithms (Second ed.)*. MIT Press and McGraw-Hill. pp. 595,601, (2001).
- [10] Davis, M., Logemann, G. and D. Loveland. "A machine program for theorem proving". *Communications of the ACM* 5:394397, 1962.
- [11] Davis M. and H. Putnam. "A Computing Procedure for Quantification Theory". *Journal of the ACM* 7: 201-215, 1960.

- [12] P. Del Moral. *Feynman-Kac formulae, Genealogical and interacting particle systems with applications. Probability and its Applications.* Springer-Verlag, New York, 2004.
- [13] M.J.J. Garvels. *The splitting method in rare-event simulation*, Ph.D. thesis, University of Twente, 2000.
- [14] 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, 1998.
- [15] Gertsbakh I.B. and Y. Spungin. *Models of Reliability: Analysis, Combinatorics and Monte Carlo*, CRC Press, 2010.
- [16] Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. ”Multilevel splitting for estimating rare event probabilities”. *Oper. Res.*, 47(4):585600, 1999.
- [17] Gogate, V. and R. Dechter, . “Approximate counting by sampling the backtrack-free search space”, in *Proceedings 22-nd Conference on Artificial Intelligence*, pp. 198-203, 2007.
- [18] Gogate V. and R. Dechter. “SampleSearch:Importance sampling in presence of Determinism”, Manuscript, 2011.
- [19] Janse van Rensburg E. J. “ Monte Carlo methods for the self-avoiding walk” *J. Phys. A: Math. Theor.* 42, pp. 1-97, 2009.
- [20] Jerrum M, R., Valiant L. G., and V. V. Vazirani. ”Random Generation of Combinatorial Structures from a Uniform Distribution”. *Theoretical Computer Science (Elsevier)* 32: 169188, 1986.
- [21] Kahn H. and T.E. Harris. ”Estimation of particle transmission by random sampling”. *National Bureau of Standards Appl. Math. Series*, 12:27 30, 1951.
- [22] Kroese D.P. and K.P. Hui. ”Applications of the Cross-Entropy Method in Reliability”. Chapter 3 in *Computational Intelligence in Reliability Engineering* G. Levitin, ed., Springer-Verlag, 2006.
- [23] Kuhn H. W. ”The Hungarian Method for the assignment problem”, *Naval Research Logistics Quarterly*, 2:8397, 1955.
- [24] A. Lagnoux. ”Rare event simulation. Probability in the Engineering and Informational Sciences”, 20(1):4566, 2006.
- [25] A. Lagnoux-Renaudie. *A two-steps branching splitting model under cost constraint.* *Journal of Applied Probability*, vol 46, 2, 429-452, 2009.
- [26] L’Ecuyer P., Demers V. and B. Tuffin. “Rare-Events, Cloning, and Quasi-Monte Carlo”, *ACM Transactions on Modeling and Computer Simulation*, 17, 2, 2007.

- [27] LEcuyer P., Blanchet J., Tuffin B. and P.W. Glynn. "Asymptotic robustness of estimators in rare-event simulation". *ACM Transactions on Modeling and Computer Simulation*, 18(3):12691283, 2008.
- [28] Li R., Zhou D. and D. Du. "Satisfiability and Integer Programming as Complementary Tools" *Proceedings of the 2004 Asia and South Pacific Design Automation Conference IEEE Press Piscataway, NJ, USA* 879-882, 2004.
- [29] J. S. Lui. *Monte Carlo Strategies in Scientific Computing*. Springer, 2001.
- [30] Madras N. and A. D. Sokal. "The Pivot Algorithm: A Highly Efficient Monte Carlo Method for the Self-Avoiding Walk" *Journal of Statistical Physics*, Vol. 50, No. 1/2, 1988.
- [31] V. B. Melas. "On the efficiency of the splitting and roulette approach for sensitivity analysis". *Winter Simulation Conference, Atlanta, Georgia*, pp. 269 - 274, 1997.
- [32] Metropolis N., Rosenbluth M. N., Rosenbluth A. H., Teller H., and E. Teller. "Equation of state calculations by fast computing machines". *The Journal of Chemical Physics*, 21(6):10871092, 1953.
- [33] Mitzenmacher M. and E. Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York (NY), 2005.
- [34] Metzner, P., Ch. Schtte and E. Vanden-Eijnden. "Illustration of transition path theory on a collection of simple examples" *The Journal of Chemical Physics*, 125, 084110, 2006.
- [35] Motwani R. and R. Raghavan, *Randomized Algorithms* Cambridge University Press, 1997.
- [36] Murray L., Cancela H., and G. Rubino. "A Splitting Algorithm for Network Reliability Estimation", 2011.
- [37] L. E. Rasmussen, *Approximating the Permanent: a Simple Approach*, *Random Structures & Algorithms* 5, 349361, 1994.
- [38] Roberts, B. and D.P. Kroese. "Estimating the number of s-t paths in a graph". *Journal of Graph Algorithms and Applications* 11 (1), 195-214, 2007.
- [39] Rosenbluth M. N. and A. W. Rosenbluth. "Monte Carlo calculation of the average extension of molecular chains". *Journal of Chemical Physics*, 23(2), 1955.
- [40] R. Y. Rubinstein. "The Gibbs Cloner for Combinatorial Optimization, Counting and Sampling", *Methodology and Computing in Applied Probability*, 11 (2), 491-549, 2009.

- [41] 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.
- [42] Rubinstein R. Y. and D. P. Kroese. *The Cross-Entropy Method: a Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*, Springer, 2004.
- [43] Rubinstein R. Y. and D. P. Kroese. *Simulation and the Monte Carlo Method; Second Edition*, Wiley, 2007.
- [44] L. G. Valiant. "The Complexity of Computing the Permanent". *Theoretical Computer Science (Elsevier)* 8: 189201, 1979.
- [45] Wei, W. and B. Selman. "A new approach to model counting". In *Proceedings of SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing*, volume 3569 of "Lecture Notes in Computer Science", St. Andrews, U.K, pp. 324-339, 2005.