Stochastic Enumeration Methods for Counting, Rare-Events and Optimization

Radislav Vaisman

Stochastic Enumeration Method for Counting, Rare-Events and Optimization

Research Thesis

In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy.

Radislav Vaisman

Submitted to the Senate of the Technion - Israel Institute of Technology

Tishrei 5774 Haifa September 2013

This research thesis was done under the supervision of Professor Reuven Y. Rubinstein in the Faculty of Industrial Engineering and Management. I would like to express my deepest gratitude to Prof. Rubinstein for his devoted and professional guidance and for his financial support.

I also want to thank Professor Ofer Strichman. This thesis would never be completed without his help and guidance.

The generous financial help of the Technion is gratefully acknowledged.

Thanks to all my family members that were patient and encouraged, pushed and stood by me during the entire period of my studies.

This thesis is dedicated to Prof. Reuven Y. Rubinstein, the best teacher I had.

PUBLICATIONS

Books:

- R. Y. Rubinstein, A. Ridder, and R. Vaisman. Fast Sequential Monte Carlo Methods for Counting and Optimization. John Wiley & Sons, New York, 2013. To appear.

Journal Publications:

- P. Glynn, A. Dolgin, R. Y. Rubinstein, and R. Vaisman. How to Generate Uniform Samples on Discrete Sets Using the Splitting Method. *Prob. in Eng. and Inf. Sciences, vol. 24, 3, pp. 405-422, 2010.*
- R. Y. Rubinstein, A. Dolgin, and R. Vaisman. The Splitting Method for Decision Making. *Journal of Statistical Planning and Inference, 2011.*
- F. C'erou, A. Guyader, R. Y. Rubinstein, and R. Vaisman. Smoothed Splitting Method for Counting. *Stochastic Models, 2012.*
- P. Dupuis, B. Kaynar, A. Ridder, R. Y. Rubinstein, and R. Vaisman. Counting with Combined Splitting and Capture - Recapture Methods. *Stochastic Models, 2012.*

Accepted for Publication:

- I. Gertsbach, R. Y. Rubinstein, Y. Shpungin, and R. Vaisman. Permutational Methods for Performance Analysis of Stochastic Flow Networks. *Prob. in Eng. and Inf. Sciences (accepted at May, 2013)*
- I. Gertsbach, E. Newman, and R. Vaisman. Monte Carlo for estimating exponential convolution. *Communications in Statistics - Simulation and Computation (accepted at August, 2013)*

Technical reports:

- R. Y. Rubinstein, Z. Botev, and R. Vaisman. Hanging Edges for Fast Reliability Estimation. *Technical report, Technion 2012.*

Submitted for publication:

- I. Gertsbach, Y. Shpungin, and R. Vaisman. New Version of Network Evolution Monte Carlo With Nodes Subject to Failure.
- R. Vaisman Z. Botev and A. Ridder. Sequential Monte Carlo Method for counting vertex covers in general graphs. *Journal of the American Statistical Association.*
- R. Vaisman O. Strichman and I. Gertsbach. Counting monotone cnf formulas with Spectra method. *INFORMS, Journal on Computing.*

In Preparation:

- Z. Botev, R. Vaisman and R. Rubinstein. Monte Carlo splitting for Stochastic Flow Networks.
- I. Gertsbakh, Y. Shpungin and R. Vaisman. D-spectrum and Reliability of Binary System With Ternary Components.

Contents

List of Tables

List of Figures

Abstract

In this research thesis we present a series of papers that deals with problems under a *rare events* settings. Main results obtained are:

1. A generic randomized algorithm, called the *splitting* is presented. The later is used for combinatorial optimization, counting and sampling uniformly on complex sets, such as the set defined by the constraints of an integer program. We adopt the classical randomized algorithm scheme that uses a sequential sampling plan to decompose a "difficult" problem into a sequence of "easy" ones. The *splitting* combines Markov Chain Monte Carlo (MCMC) sampler and specially designed splitting mechanism. Our algorithm runs in parallel multiple Markov chains by making sure that all of them run in steady-state at each iteration.

   In the first article we introduce a simple modification of the splitting method based on Gibbs sampler and show that it can be efficiently used for decision making.

   In the second article we show that the classic MCMC used as a universal tool for generating samples on complex sets can fail and in particular miss some points in the region of interest. On the other hand, the *splitting* algorithm can be efficiently used for generating uniform samples on those sets.

   In the third article we present an entirely new *splitting* algorithm called the *smoothed splitting method* (SSM). The main difference between SSM and splitting is that it works with an auxiliary sequence of continuous sets instead of the original discrete ones. Operating in the continuous space have an added value in sense of flexibility and robustness.

   In our last article dedicated to *splitting* we present an enhanced version of the splitting method based on the capture-recapture technique. This modification allows to achieve a variance and computation time reductions. Moreover, we show how to apply the algorithm to count graphs with prescribed degrees, and binary contingency tables.

2. We present an adaptation of the well known *Permutation Monte Carlo* (PMC) method for estimation rare events in flow networks. (PMC) was originally developed for reliability networks, but as we show in our our last article can be successfully adapted for stochastic flow networks, and in particular for estimation of the probability that the maximal flow in such a network is above some fixed level, called the *threshold*.

3. Finally, we present a new generic *stochastic enumeration* (SE) algorithm 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 sequential algorithm in the sense that it runs in parallel multiple trajectories instead of a single one and employs a polynomial time decision making oracle to prevent the exploration of empty trajectories (dead ends) and thus overcomes the difficulty of the rare events.

List of Acronyms

BFS	Breadth first search
BRE	Bounded Relative Error
CAP-RECAP	Capture Recapture Method
CE	Cross Entropy
CLT	Central Limit Theorem
CMC	Crude Monte Carlo
CNF	Conjunctive Normal Form
DOWN	Network Down State
FPAUS	Fully Polynomial Almost Uniform Sampler
FPRAS	Fully Polynomial Time Randomized Approximation Scheme
ID	Internal Distribution
IS	Importance Sampling
ISD	Importance Sampling Distribution
MCMC	Markov Chain Monte Carlo
nSLA	n Step Look Ahead
OSLA	One Step Look Ahead
PDF	Probability Density Function
PMC	Permutation Monte Carlo
RE	Relative Error
SAT	Satisfiability Problem
SAW	Self Avoiding Walk
SE	Stochastic Enumeration
SIS	Sequential Importance Sampling
SSM	Smoothed Splitting Method
UP	Network Up State

Chapter 1

Introduction

Counting and optimization are not only common and natural human activities but also an important issue in computer science and engineering. The majority of practical problems in those fields belongs to the NP-Complete or even more difficult complexity classes. In other words, they probably cannot be solved efficiently according to the modern computation theory. As an example, consider an efficient manufacture of microchips made possible by solving the well known traveling salesman problem (TSP). Many real and abstract counting and optimization problems become quickly too hard to solve exactly so we consider randomized algorithms that return estimates of the exact solution. Those methods are based on computer simulation, also called Monte Carlo methods and even if the rigorous proofs are not always available they are extensively used in today industry with the hope to obtain optimal solutions.

The Monte Carlo method was pioneered in the 1940s by John von Neumann, Stanislaw Ulam and Nicholas Metropolis, while they were working at nuclear weapon Manhattan Project in the Los Alamo National Laboratory. It was named after the Monte Carlo casino, a famous casino where Ulam's uncle often gambled away his money. In general, Monte Carlo algorithms are easy to implement and they can deliver reliable estimates even for very hard problems. On the other hand, there are settings for which the simple algorithm can quickly become unusable so some more advanced techniques should be introduced.

In this work we concern mainely with counting problems under a rare event settings. Most counting problems of interest belong to the class of so-called #P-complete problems which is related to the familiar class of NP-hard problems. See [69] for details. The area of counting, and in particular a definition of #P-complete class, introduced by Valiant [92] received much attention in the Computer Science community. An efficient Algorithms were found for some problems. For example, Karp and Lubby [51] introduced a *FPRAS* (A fully polynomial randomized approximation scheme) for counting the solutions of *DNF* satisfiability formula. Similar results were obtained for Knapsack and Permanent problems, see [47, 23]. On the other hand, there are many "negative" results. For example, Dyer, Frieze and Jerrum showed that there is no FPRAS for $\#IS$ (Independent Set) if the maximum degree of the graph is 25 unless RP = NP [24]. Counting the number of vertex covers remains hard even when restricted to planar bipartite graphs of bounded degree or regular graphs of constant degree, see [91] for details. We can conclude that to date, very little is known about how to construct efficient algorithms for solving various #P-complete problems.

Next, we present some examples of #P-complete problems:

- *The Hamiltonian cycle problem.*
  How many Hamiltonian cycles does a graph have? That is, how many tours contains a graph in which every node is visited exactly once (except for the beginning/end node)?

- *The permanent problem.*
  Calculate the permanent of a matrix $\boldsymbol{A}$, or equivalently, the number of perfect matchings in a bipartite balanced graph with $\boldsymbol{A}$ as its biadjacency matrix.

- *The self-avoiding walk problem.*
  How many self-avoiding random walks of length n exist, when we are allowed to move at each grid point in any neighboring direction with equal probability?

- *The connectivity problem.*
  Given two different nodes in a directed or undirected graph, say v and w, how many paths exist from v to w that do not traverse the same edge more than once?

- *The satisfiability problem.*
  Let $\mathcal{X}$ be a collection of all subsets of n Boolean variables $\{x_1, \dots, x_n\}$. Thus, $\mathcal{X}$ has cardinality $|\mathcal{X}| = 2^n$. Let $\mathcal{C}$ be a set of m Boolean disjunctive clauses. Examples of such clauses are $C_1 = x_1 \vee \bar{x}_2 \vee x_4$, $C_2 = \bar{x}_2 \vee \bar{x}_3$, etc. How many (if any) satisfying truth assignments for $\mathcal{C}$ exist, that is, how many ways are there to set the variables $x_1, \dots, x_n$ either true or false so that all clauses $C_i \in \mathcal{C}$ are true?

- *The k-coloring problem.*
  Given $k \geq 3$ distinct colors, in how many different ways can one color the nodes (or the edges) of a graph, so that each two adjacent nodes (edges, respectively) in the graph have different colors?

- *The spanning tree problem.*
  How many unlabeled spanning trees has a graph G? Note that this counting problem is easy for labeled graphs [10].

- *The isomorphism problem.*
  How many isomorphisms exist between two given graphs G and H? In other words, in an isomorphism problem one needs to find all mappings ϕ between the nodes of G and H such that (v, w) is an edge of G if and only if $(\phi(v), \phi(w))$ is an edge of H.

- *The clique problem.*
  How many cliques of fixed size k exist in a graph G? Recall that a clique is a complete subgraph of G.

The decision versions of #P problems generally belong to the NP-complete class so counting all feasible solutions, denoted by #P, is even a harder problem. Nevertheless, we encounter many examples where the counting problem is hard to solve, while the associated decision problem is easy. As an example, finding a perfect matching in bipartite graph is easy while counting all such matchings is hard. Generally, the complexity class #P consists of the counting problems associated with the decision problems in NP. Completeness is defined similarly to the decision problems: a problem is #P-complete if it is in #P, and if every #P problem can be reduced to it via parsimonious reduction. Hence, the counting problems that we presented above are all #P-complete. For more details we refer to the classic monograph [71].

There are two major approaches that proved useful when trying to tackle counting problems. The first is MCMC (Markov Chain Monte Carlo). The second is Importance Sampling and in particular a sequential one. The reason for those is as follows; it was shown earlier by Jerrum at. al. [48] that counting is equivalent to uniform sampling in the space of interest. Obviously, it is possible to sample in such a region by constructing an ergodic Markov Chain and sampling using MCMC methods. The pitfall is in the speed of convergence of such a chain to the stationary distribution. If the chain "mixes" rapidly, on in other words the chain reaches stationary distribution in number of steps bounded by some polynomial, then an efficient algorithm for counting exists. On the other hand, Importance Sampling can achieve the same results but usually much faster. In general, counting algorithms based on Importance Sampling report very good performance. A previously mentioned DNF satisfiability Algorithm by Karp and Lubby is one such example. For success stories using SIS Algorithm, see [84] and [14, 5]. Unfortunately, it is not easy to provide rigorous proofs in those cases [4]. In this work, we employ the MCMC approach in our Splitting method presented in chapters 2,3,4,5 and the Importance Sampling is a basis for the Stochastic Enumeration presented in chapter 7. While MCMC and IS are definitely the most common methods, a different approach called the *Evolution* method, is presented in chapter 6. The later provides a general framework for tackling counting problems with monotonic properties, like monotone CNF and graph covers.

Next, we introduce some general overview of the techniques used in this thesis.

- The splitting method dates back to Kahn and Harris [50] and Rosenbluth and Rosenbluth [77]. The main idea is to partition the state-space of a system into a series of nested subsets and to consider the rare event as the intersection of a nested sequence of events. When a given subset is entered by a sample trajectory during the simulation, numerous random retrials are generated with the initial state for each retrial being the state of the system at the entry point. By doing so, the system trajectory is split into a number of new sub-trajectories, hence the name splitting. Since then hundreds of papers have been written on that topic, both from a theoretical and a practical point of view. Applications of the splitting method arise in particle transmission (Kahn and Harris [50]), queueing systems (Garvels [28], Garvels and Kroese [29], Garvels et al. [30]), and reliability (L'Ecuyer et al. [57]). The method has been given new impetus by the RESTART (Repetitive Simulation Trials After Reaching Thresholds) method in the sequence of papers by Villén-Altimirano and Villén-Altimirano [95, 96, 97]. A fundamental theory of the splitting method was developed by Melas [64], Glasserman et al. [39, 40], and Dean and Dupuis [21, 22]. Recent developments include the adaptive selection of the splitting levels in Cérou and Guyader [11], the use of splitting in reliability networks [53, 80], to quasi-Monte Carlo estimators in L'Ecuyer et al. [58], and the connection between splitting for Markovian processes and interacting particle methods based on the Feynman-Kac model in Del Morall [68]. In this work we further developed the *splitting* method combined with Gibbs sampler. This combination proved itself to be a powerful tool for solving hard combinatorial optimization and counting problems.

- Importance sampling is a well-known variance reduction technique in stochastic simulation studies. The idea behind importance sampling is that certain values of the input random variables have a bigger impact on the output parameters than others. If these "important" values are sampled more frequently, the variance of the output estimator can be reduced. However, such direct use of importance sampling distributions will result in a biased estimator. In order to eliminate the bias the simulation outputs must be modified (weighted) by using a likelihood ratio factor, also called the Radon Nikodym derivative [86]. The fundamental issue in implementing importance sampling is the choice of the importance sampling distribution.

  In case of counting problems, it is well known that a straightforward application of importance sampling typically yields very poor approximations of the quantity of interest. In particular, Gogate and Dechter [41, 42] show that poorly chosen importance sampling in graphical models such as satisfiability models generates many useless zero-weight samples, which are often rejected yielding an inefficient sampling process. To address this problem, which is called the problem of losing trajectories, the above authors propose a clever sample search method, which is integrated into the importance sampling framework.

  Concerning probability problems, a wide range of applications of importance sampling have been reported successfully in the literature over the last decades. Siegmund [89] was the first to argue that, using an exponential change of measure, asymptotically efficient importance sampling schemes can be built for estimating gambler's ruin probabilities. His analysis is related to the theory of large deviations, which has since become an important tool for the design of efficient Monte Carlo experiments. Importance sampling is now a subject of almost any standard book on Monte Carlo simulation (see, for example, [1, 86]). We shall use importance sampling in chapter 7.

The rest of this work is organized as follows. Sections 1.0.1 and 1.0.2 of this chapter is dedicated to preliminaries for the methods used in this work. In Chapter 2 we show how the *splitting* method combined with some simple modifications can be used for decision making. In Chapter 3 we consider a crucial problem of uniform sampling in complex sets. We show numerically, that the *splitting* algorithm combined with Gibbs sampler provides good uniformity results. In Chapter 4 we present a new version of *splitting* algorithm that operates in the continuous domain. In Chapter 5 we present an enhanced version of the splitting method based on the capture-recapture estimator. The later allows us to achieve a considerable variance reduction. In Chapter 6 we present an adaptation of Permutation Monte Carlo technique for solving stochastic flow networks threshold problem. In Chapter 7 we present a new sequential counting algorithm for solving self reducible problems. In Chapter 8 we present a discussion,concluding remarks and some direction for future research. Finally, in the Appendix we consider an efficiency of Monte Carlo estimators, then, we discuss the complexity of randomized algorithms and in particular we give a formal definition of *FPRAS* and *FPAUS* using [67]. In the end we introduce some mathematical background on the Gibbs sampler - the method of Diaconis-Holmes-Ross (DHR), from which we adopted some basic ideas.

1.0.1 Randomized algorithms for counting

Below we present some background on randomized algorithms. The main idea of *randomized algorithms* for counting [67, 69] is to design a sequential sampling plan, with a view to decomposing a "difficult" counting problem defined on the set $\mathcal{X}^*$ into a number of "easy" ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ and such that $\mathcal{X}_m = \mathcal{X}^*$. Typically, randomized algorithms explore the connection between counting and sampling problems and in particular the reduction from approximate counting of a discrete set to approximate sampling of elements of this set, where the sampling is performed by the classic MCMC method [86]. A typical randomized algorithm comprises the following steps:

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

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

3. Write $|\mathcal{X}^*| = |\mathcal{X}_m|$ as
$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}. \tag{1.1}$$

Note that the quantity

$$\ell = \frac{|\mathcal{X}^*|}{|\mathcal{X}_0|}$$

is very small, like $\ell = 10^{-100}$, while each ratio

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = \mathbb{P}(x \in \mathcal{X}_t | x \in \mathcal{X}_{t-1}) \tag{1.2}$$

should not be small, like $c_t = 10^{-2}$ or greater. As we shall see below in typical applications such c_t will be available. Clearly, estimating ℓ directly while sampling in $\mathcal{X}_0$ is meaningless, but estimating each c_t separately seems to be a good alternative.

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

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

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0| \prod_{t=1}^{m} \widehat{c}_t = \prod_{t=1}^{m} \frac{|\widehat{\mathcal{X}}_t|}{|\widehat{\mathcal{X}}_{t-1}|}, \tag{1.3}$$

where $\widehat{c}_t = \frac{|\widehat{\mathcal{X}}_t|}{|\widehat{\mathcal{X}}_{t-1}|}$ and $|\widehat{\mathcal{X}}_t|,\ t = 1, \ldots, m$ is an estimator of $|\mathcal{X}_t|$.

Algorithms based on the sequential Monte Carlo sampling estimator (1.3) are called in computer literature [67, 69], *randomized algorithms.* We shall call them simply the *RAN* algorithms.

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

(i) Put the well known NP-hard counting problems into the framework (1.1) by making sure that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ and each c_t is not a rare-event probability.

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

We shall see below that task (i) is not difficult and shall proceed with it in this section. Task (ii) is quite complicated and is associated with uniform sampling separately at each sub-region $\mathcal{X}_t$. This will be done by combining the Gibbs sampler with the classic splitting method [28] and will be considered in the subsequent chapters.

It readily follows that as soon as both tasks (i) and (ii) are resolved one can obtain an efficient estimators for each c_t, and, thus a low variance estimator $\widehat{|\mathcal{X}^*|}$ in (1.3). We therefore proceed with task (i) by considering several well-known NP-hard counting problems.

Example 1.0.1 (Independent Sets)**.** Consider a graph $G = (V, E)$ with m edges and n vertices. Our goal is to count the number of independent node (vertex) sets of this graph. A node set is called *independent* if no two nodes are connected by an edge, that is, no two nodes are adjacent, see Figure 1.1 for an illustration of this concept.

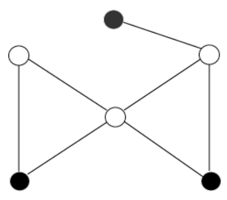


Figure 1.1: The black nodes form an independent set since they are not adjacent to each other.

Consider an arbitrary ordering of the edges. Let E_j be the set of the first j edges and let $G_j = (V, E_j) = (V, \{e_1, \ldots, e_j\})$ be the associated sub-graph. Note that $G_m = G(V, E_m) = G(V, E) = G$, and that G_{j+1} is obtained from G_j by adding the edge e_{j+1}, which is not in G_j. Denoting by $\mathcal{X}_j$ the set of independent sets of G_i we can write $|\mathcal{X}^*| = |\mathcal{X}_m|$ in the form (1.1). Here $|\mathcal{X}_0| = 2^n$, since G_0 has no edges and thus every subset of V is an independent set, including the empty set. Note that here $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$.

Example 1.0.2 (Vertex Coloring)**.** Given a graph $G = (V, E)$ with m edges and n vertices, color the vertices of V with given q colors, such that for each edge $(i, j) \in E$, vertices i and j have different colors. The procedure for vertex coloring while applying the randomized algorithm is the same as for independent sets. Indeed, we again consider an arbitrary ordering of the edges. Let E_j be the set of the first j edges and let $G_j = (V, E_j)$ be the associated sub-graph. Note that $G_m = G$, and that G_{j+1} is obtained from G_j by adding the edge e_{j+1}. Denoting by $|\mathcal{X}_i|$ the cardinality of the set $\mathcal{X}_i$ corresponding to G_i we can write again $|\mathcal{X}^*| = |\mathcal{X}_m|$ in the form (1.1), where $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$. Here $|\mathcal{X}_0| = q^n$, since G_0 has no edges.

Example 1.0.3 (Hamiltonian Cycles)**.** Given a graph $G = (V, E)$ with k edges and n vertices, find all Hamiltonian cycles, that is, those corresponding to the tours of length n.

Figure 1.2 presents a graph with 9 nodes and several Hamiltonian cycles, one of which is marked in bold lines.

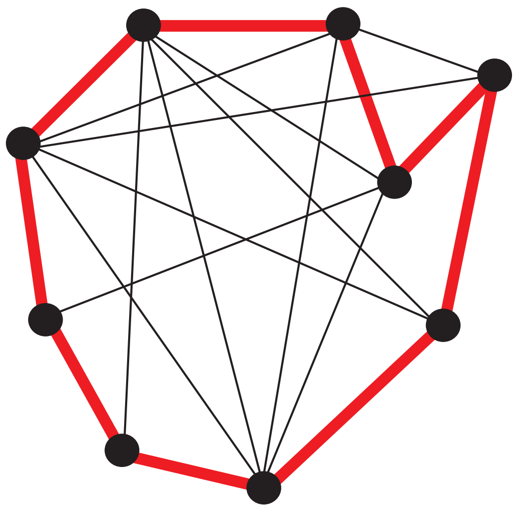


Figure 1.2: A Hamiltonian graph. The bold edges form a Hamiltonian cycle.

The procedure for Hamiltonian cycles is similar to independent sets. Consider now all possible graph edges $e_1, e_2, \cdots, e_{\binom{n}{2}}$ and suppose that $k \leq \binom{n}{2}$ and that $E = \{e_{\binom{n}{2}-k+1}, \cdots, e_{\binom{n}{2}}\}$. Define a graph $G_j = (V, E_j)$, where $E_j = \{e_{j+1}, \cdots, e_{\binom{n}{2}}\}$ for $j = 0, 1, 2, \cdots, m = \binom{n}{2} - k$. Note that $G_m = G_{\binom{n}{2}-k} = G(V, E_{\binom{n}{2}-k+1}) = G(V, E) = G$ and G_{j+1} is obtained from G_j by removing an edge e_{j+1}. Denoting by $\mathcal{X}_i$ the set of all Hamiltonian cycles of length n corresponding to the graph G_i we can write again $|\mathcal{X}^*| = |\mathcal{X}_m|$. Finally, having in mind that G_0 is a complete graph, we conclude that $|\mathcal{X}_0| = (n-1)!$ and the $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ setting of framework (1.1) is obtained.

Example 1.0.4 (Knapsack Problem)**.** Given items of sizes $a_1, \ldots, a_m > 0$ and a positive integer $b \geq \min_i a_i$, find the number of vectors $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$, such that

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

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

To put the knapsack problem into the framework (1.1) Jerrum and Sinclair [46], assume without loss of generality that $a_1 \leq a_2 \leq \cdots \leq a_n$ and define $b_j = min\{b, \sum_{i=1}^{j} a_i\}$, with $b_0 = 0$. Denote by $\mathcal{X}_j$ the set of vectors $\boldsymbol{x}$ that satisfy $\sum_{i=1}^{n} a_i\, x_i \leq b_j$, and let m be the largest integer such that $b_m \leq b$. Clearly, $\mathcal{X}_m = \mathcal{X}^*$. Thus, (1.1) is established again.

Example 1.0.5 (Counting the Permanent)**.** The permanent of a general $n \times n$ binary matrix $A = (a_{ij})$ is defined as

$$\mathrm{per}(A) = |\mathcal{X}^*| = \sum_{\boldsymbol{x} \in \mathcal{X}} \prod_{i=1}^{n} a_{ix_i}, \tag{1.4}$$

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

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

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}. \tag{1.5}$$

The corresponding bipartite graph is given in Figure 1.3. The graph has 3 perfect matchings, one of which is displayed in the figure. These correspond to all permutations $\boldsymbol{x}$ for which the product $\prod_{i=1}^{n} a_{ix_i} = 1$.

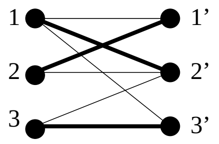


Figure 1.3: A bipartite graph. The bold edges form a perfect matching.

For a general binary matrix A let $\mathcal{X}_i$ denote the set of matchings of size i in the corresponding bipartite graph G. Assume that $\mathcal{X}_n$ is non-empty, so that G has a perfect matching of nodes V_1 and V_2. We are interested in calculating $|\mathcal{X}_n| = \mathrm{per}(A)$. Taking into account that $|\mathcal{X}_0| = |E|$ we obtain (1.1).

1.0.2 The Evolution model

In order to understand the material presented in chapter 6, we give below a brief introduction to the Evolution model for network reliability. The traditional network reliability problem, as it has been stated in the recently published Handbook of Monte Carlo Methods [53] , is formulated as follows. Let $G(V, E, K)$ be

an undirected graph (network) with V being the set of n nodes (or vertices), E being the set of m edges (or links), and $K \subseteq V$, $|K| = s$, being a set of special nodes called *terminals*.
Associated with each edge $e \in E$ is a Bernoulli random variable $X(e)$ such that $X(e) = 1$ corresponds to the event that the edge is operational *(up)* and $X(e) = 0$ corresponds to the event that the edge has failed (is *down*). The edge failures are assumed to be independent events.
Based on this model, the network reliability $R = \mathbb{P}(UP)$ and unreliability (probability to be $DOWN$) $Q = \mathbb{P}(DOWN) = 1 - R$ are defined as the probability that a set of terminal nodes K is connected (not connected) in the sub-graph containing all of the nodes in V. When $s = 2$, the model describes so-called $s-t$ terminal connectivity. When $s = n$, we have all-node connectivity.
This model, although very simple, has been employed in a wide number of application settings. Among other cases, many examples can be found in the reliability evaluation and topology design of communication networks, mobile ad hoc and tactical radio networks, evaluation of transport and road networks, see [16, 35, 37, 45, 59, 63, 70].

One of the most computationally efficient methods for calculating network reliability for the above model is based on so-called evolution or creation process first suggested in [26]. It works as follows. Initially, at $t = 0$ all edges e are down. At some random moment $\xi(e)$, edge e is born, independently of other edges, and remains in state up forever. $\xi(e)$ is assumed to be exponentially distributed with parameter $\lambda(e)$:

$$\mathbb{P}(\xi(e) \leq t) = 1 - e^{\lambda(e)t}, e \in E \tag{1.6}$$

Fix an arbitrary moment $t = t_0$, for example, $t_0 = 1$. Choose for each e its birth rate $\lambda(e)$ so that the following condition holds:

$$\mathbb{P}(\xi(e) > t_0) = e^{\lambda(e)t_0} = 1 - p(e) \tag{1.7}$$

This formula means that at time t_0 the edge e has already born (is *up*) with probability $p(e)$. The crucial observation is that the snap shot of the whole network taken at t_0 gives the picture of the whole network which probabilistically coincides with its true state as if we generate the state of each edge with static probability $p(e)$ of being *up* and $1 - p(e)$ of being *down*.

The Monte Carlo procedure for estimating network UP probability R is implemented by generating so-called trajectories which imitate the development in time of the evolution process. The best way to explain how this method works is to demonstrate it on a simple example, see figure 1.4.

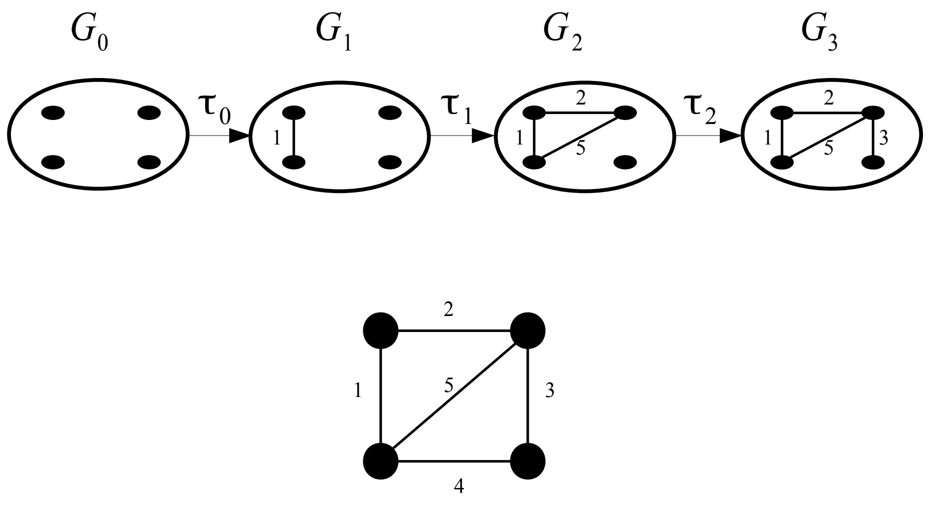


Figure 1.4: Evolution process. Edges are born in the sequence: $1 \to 2 \to 3$

We have a four node network with five edges. The network is *UP* if all its nodes are connected to each other. The initial state without edges at $t = 0$ is denoted as σ_0. The network stays in it during random time τ_0 which is exponentially distributed with parameter $\Lambda_0 = \sum_{i=1}^{5} \lambda_i$. Suppose the edge 1 is born first. By the properties of exponential distribution, this happens with probability $\frac{\lambda_1}{\Lambda}$. Then the system goes into its next state σ_1. Now in this state the system spends random exponentially distributed time $\tau_1 \sim Exp(\Lambda_1 = \sum_{i=2}^{5} \lambda_i)$. Suppose that the next edge born is 2. This happens with probability $\frac{\lambda_1}{\sum_{i=2}^{5} \lambda_i}$. This transfers the system into state σ_2. Now note that at this stage of the evolution process we can add edge 5 to already born edges and exclude it from the further evolution process because the existence or nonexistence of this edge does not affect the already formed component of three nodes a, b, c created by edges 1 and 2. This operation was originally called by its inventor M. Lomonosov as closure [26].

After adding edge 5 to already born edges, there remains only two unborn edges 3 and 4. The system spends in σ_2 random time $\tau_2 \sim Exp(\lambda_3+\lambda_4)$. Suppose edge 3 is born first, which happens with probability $\frac{\lambda_3}{\lambda_3+\lambda_4}$. Then the system enters the state σ_3 which is, by definition, the network *UP* state. Note that the random times τ_0, τ_1, τ_2 are independent, and the trajectory $\omega = \{\sigma_1 \to \sigma_2 \to \sigma_3\}$ takes place with probability

$$\mathbb{P}(\omega) = \frac{\lambda_1}{\Lambda_0} \cdot \frac{\lambda_2}{\Lambda_1} \cdot \frac{\lambda_3}{\Lambda_2} \tag{1.8}$$

Finally, let us find out the probability $\mathbb{P}(UP;\omega)$ that the network will be *UP* given that the evolution goes along this trajectory:

$$\mathbb{P}(UP;\omega) = \mathbb{P}(\tau_0 + \tau_1 + \tau_2 \leq t_0;\omega) \tag{1.9}$$

This probability can be found in a closed form using well-known *hypoexponential* distribution, see [79], page 299. It is worth to present the corresponding formula in its general form.

Let $\tau_i \sim Exp(\Lambda_i)$, $i = 0, 1, 2, \cdots, r-1$ be independent random variables and suppose that

$$\Lambda_0 > \Lambda_1 > \cdots > \Lambda_{r-1}$$

Then

$$\mathbb{P}(\sum_{i=0}^{r-1} \tau_i \leq t_0) = 1 - \sum_{i=0}^{r-1} e^{-\Lambda_i t_0} \prod_{j \neq i} \frac{\Lambda_j}{\Lambda_j - \Lambda_i} \tag{1.10}$$

Let us observe in more detail the above evolution (creation) process. It starts always when the number of components equals n, the number of nodes. Adding a newborn edge always leads to the decrease of the number of components by 1. So, σ_1 has three components, σ_2 - two components, and σ_3 - one component. Therefore, in case of all-terminal connectivity, each trajectory has the same length of n transitions. Without applying the closure operation, the trajectories would have been considerably longer since the number of edges is usually larger than the number of nodes, especially for dense graphs.

Now suppose that we have generated M trajectories $\omega_1, \cdots, \omega_M$ and for each ω_i we have calculated by (1.10) the corresponding convolution $\mathbb{P}(UP;\omega_i)$. The unbiased estimate of network *UP* probability is found as an average

$$\widehat{\mathbb{P}}(UP) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{P}(UP;\omega_i) \tag{1.11}$$

A detailed description of the above evolution process and its properties is given in Chapter 9 of [35].

Chapter 2

The Splitting Method for Decision Making

Reuven Rubinstein, Andrey Dolgin and Radislav Vaisman

Faculty of Industrial Engineering and Management,

Technion, Israel Institute of Technology, Haifa, Israel

ierrr01@ie.technion.ac.il

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

[0†] This research was supported by the BSF (Binational Science Foundation, grant No 2008482) and ISF (Israel Science Foundation) grant No 2012910

Abstract

We show how a simple modification of the splitting method based on Gibbs sampler can be efficiently used for decision making in the sense that one can efficiently decide whether or not a given set of integer program constraints has at least one feasible solution. We also show how to incorporate the classic *capture-recapture* method into the splitting algorithm in order to obtain a low variance estimator for the counting quantity representing, say the number of feasible solutions on the set of the constraints of an integer program. We finally present numerical with with both, the decision making and the capture-recapture estimators and show their superiority as compared to the conventional one, while solving quite general decision making and counting ones, like the satisfiability problems.

Keywords. Decision Making, Gibbs Sampler, Cross-Entropy, Rare-Event, Combinatorial Optimization, Counting, Splitting.

2.1 Introduction: The Splitting Method

In this work we show how a simple modification of the splitting method introduced in [82, 83] can be used for decision making. The goal of the decision making algorithm is to decide whether or not a discrete set, like the set defined by the integer programming constraints has a feasible solution.

Although there is a vast literature on the splitting method, (see [7], [28], [31], [55], [64], [58], [83]), we follow [83] and present some background on the splitting method, also called in [83] the *cloning* method. The main idea is to design a sequential sampling plan, with a view of decomposing a "difficult" counting problem defined on some set $\mathcal{X}^*$ into a number of "easy" ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ and such that $\mathcal{X}_m = \mathcal{X}^*$. Typically, splitting algorithms explore the connection between counting and sampling problems and in particular the reduction from approximate counting of a discrete set to approximate sampling of elements of this set, where the sampling is performed by the classic MCMC method [86].

A typical splitting algorithm comprises the following steps:

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

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

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

$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = \ell |\mathcal{X}_0|, \tag{2.1}$$

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

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \tag{2.2}$$

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

4. Develop an efficient estimator $\widehat{c}_t$ for each c_t and estimate $|\mathcal{X}^*|$ by

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0| \, \widehat{\ell} = |\mathcal{X}_0| \prod_{t=1}^{m} \widehat{c}_t, \tag{2.3}$$

   where $\widehat{\ell} = |\mathcal{X}_0| \prod_{t=1}^{m} \widehat{c}_t$.

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

(i) Put the well known NP-hard counting problems into the framework (2.1) by making sure that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ and each c_t is not a rare-event probability.

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

To proceed note that ℓ can be also written as

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) \geq m\}} \right], \tag{2.4}$$

where $\boldsymbol{X} \sim f(\boldsymbol{x})$, $f(\boldsymbol{x})$ is a uniform distribution on the set of points of $\mathcal{X} = \mathcal{X}_0$, m is a fixed parameter, like the total number of constraints in an integer program, and $S(\boldsymbol{X})$ is the sample performance, like the number of feasible solution generated by the constraints of the integer program. It can be also written (see(2.1)) as

$$\ell = \prod_{t=1}^{T} c_t, \tag{2.5}$$

where

$$c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}| = \mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X}) \geq m_{t-1}\}}]. \tag{2.6}$$

Here

$$g^*_{t-1} = g^*(\boldsymbol{x}, m_{t-1}) = \ell(m_{t-1})^{-1} f(\boldsymbol{x}) I_{\{S(\boldsymbol{x}) \geq m_{t-1}\}}, \tag{2.7}$$

$\ell(m_{t-1})^{-1}$ is the normalization constant and similar to (2.1) the sequence $m_t,\ t = 0, 1, \ldots, T$ represents a fixed grid satisfying $-\infty < m_0 < m_1 < \cdots < m_T = m$. Note that in contrast to (2.1) we use in (2.5) a product of T terms instead of a product of m terms. Note that T might be a random variable. The later case is associated with adaptive choice of the level sets $\{\widehat{m}_t\}_{t=0}^{T}$ resulting in $T \leq m$. Since for counting problems the pdf $f(\boldsymbol{x})$ should be *uniformly* distributed on $\mathcal{X}$, which we denote by $\mathcal{U}(\mathcal{X})$, it follows from (2.7) that the pdf $g^*(\boldsymbol{x}, m_{t-1})$ must be *uniformly* distributed on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_{t-1}\}$, that is, $g^*(\boldsymbol{x}, m_{t-1})$ must be equal to $\mathcal{U}(\mathcal{X}_t)$. Although the pdf $g^*_{t-1} = \mathcal{U}(\mathcal{X}_t)$ is typically not available analytically, it is shown in [82, 83] that one can sample from it by using the MCMC method and in particular the Gibbs sampler, and as the result to update the parameters c_t and m_t adaptively. This is one of the most crucial issues of the cloning method.

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

$$\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t = \frac{1}{N^T} \prod_{t=1}^{T} N_t, \tag{2.8}$$

where

$$\widehat{c}_t = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq m_{t-1}\}} = \frac{N_t}{N}, \tag{2.9}$$

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

We next show how to cast the problem of counting the number of feasible solutions of the set of integer programming constraints into the framework (2.4)-(2.7).

Example 2.1.1. Counting on the set of an integer programming constraints Consider the set $\mathcal{X}^*$ containing both equality and inequality constraints of an integer program, that is,

$$\begin{aligned} &\textstyle\sum_{k=1}^{n} a_{ik} x_k = b_i,\ i = 1, \ldots, m_1, \\ &\textstyle\sum_{k=1}^{n} a_{jk} x_k \geq b_j,\ j = m_1 + 1, \ldots, m_1 + m_2, \\ &\boldsymbol{x} = (x_1, \ldots, x_n) \geq \mathbf{0}, \quad x_k \text{ is integer } \forall k = 1, \ldots, n. \end{aligned} \tag{2.10}$$

Our goal is to count the number of feasible solutions (points) of the set (2.10). We assume that each component $x_k,\ k = 1, \ldots, n$ has d different values, labeled $1, \ldots, d$. Note that the SAT problem represents a particular case of (2.10) with inequality constraints and where $x_1, \ldots, x_n$ are binary components. If not stated otherwise we will bear in mind the counting problem on the set (2.10) and in particular counting the number of true (valid) assignments in a SAT problem.

It is shown in [83] that in order to count the number of points of the set (2.10) one can associate it with the following rare-event probability problem

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) = m\}} \right] = \mathbb{E}_f \left[I_{\{\sum_{i=1}^{m} C_i(\boldsymbol{X}) = m\}} \right], \tag{2.11}$$

where the first m_1 terms $C_i(\boldsymbol{X})$'s in (2.11) are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^{n} a_{ik} X_k = b_i\}},\ i = 1, \ldots, m_1, \tag{2.12}$$

while the remaining m_2 ones are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^{n} a_{ik} X_k \geq b_i\}},\ i = m_1 + 1, \ldots, m_1 + m_2 \tag{2.13}$$

and $S(\boldsymbol{X}) = \sum_{i=1}^{m} C_i(\boldsymbol{X})$. Thus, in order to count the number of feasible solutions on the set (2.10) one can consider an associated rare-event probability estimation problem (2.11) involving a *sum of dependent Bernoulli random variables* C_i $i = m_1 + 1, \dots, m$, and then apply $\widehat{|\mathcal{X}^*|} = \widehat{\ell}|\mathcal{X}|$. In other words, in order to count on $\mathcal{X}^*$ one needs to estimate efficiently the rare event probability ℓ in (2.11). A rare-event probability estimation framework similar to (2.11) can be readily established for many NP-hard counting problems [83].

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

$$\{(m_0, g^*(\boldsymbol{x}, m_{-1})),\ (m_1, g^*(\boldsymbol{x}, m_0)),\ (m_2, g^*(\boldsymbol{x}, m_1)), \dots, (m_T, g^*(\boldsymbol{x}, m_{T-1}))\} \tag{2.14}$$

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

$$\mathbb{E}_{g^*_{t-1}} I_{\{S(\boldsymbol{X}) \geq m_t\}} = \rho, \tag{2.15}$$

where ρ is called the *rarity* parameter [86]. Typically one sets $0.1 \leq \rho 0.01$. Note that in contrast to the classic cross-entropy (CE) method [81], [85], where one generates a sequence of tuples

$$\{(m_0, \boldsymbol{v}_0),\ (m_1, \boldsymbol{v}_1), \dots, (m_T, \boldsymbol{v}_T)\}, \tag{2.16}$$

and, where $\{\boldsymbol{v}_t,\ t = 1, \dots, T\}$ is a sequence of parameters in the parametric family of distributions $f(\boldsymbol{x}, \boldsymbol{v}_t)$, here in (2.14), $\{g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1},\ t = 0, 1, \dots, T\}$ *is a sequence of non-parametric IS distributions.* Otherwise, the CE and the splitting algorithms are very similar.

In Appendix (see Section 2.6), following [83], we present two versions of the splitting algorithm: the so-called *basic* version and the *enhanced* version having in mind Example 2.1.1. Here we also present what is called the *direct* estimator and an associated Algorithm 2.6.3, which can be viewed as an alternative to the conventional product estimator $|\widehat{\mathcal{X}^*}|$ generated by Algorithm 2.6.2. This estimator is based on *direct counting* of the number of samples obtained immediately after crossing the level m, that is without involving the product of $\widehat{c}_t$. The drawback of the direct Algorithm 2.6.3 is that it is able to count only if $|\mathcal{X}^*|$ is up to the order of thousands.

Note that the splitting algorithm in [83] is also suitable for optimization. Here we shal use the same sequence of tuples (2.14), but *without involving the product of the estimators* $\widehat{c}_t,\ t = 1, \dots, T$.

The rest of the paper is organized as follows. In Section 2.2 we present two heuristics for speeding up the direct splitting Algorithm 2.6.3. They are called (i) *local m_t updating* and (ii) *global m_t updating*, respectively and will be used for decision making. Recall that decision making here is merely to decide whether or not the set $\mathcal{X}^*$ of the integer programming constraints (2.10) has a feasible solution. Section 2.3 shows how to combine the well known *capture-recapture* (CAP-RECAP) method with splitting in order to obtain a low variance alternative to both the product estimator $\widehat{|\mathcal{X}^*|}$ in (2.3) and the direct estimator $\widehat{|\mathcal{X}^*_{dir}|}$ in (2.22). Note that the CAP-RECAP estimator (see (2.17) below) can be viewed as a generalization of the direct one $\widehat{|\mathcal{X}^*_{dir}|}$ in the sense that once $m_t = m$ it involves two Gibbs samples instead of one sample. In Section 2.4 supportive numerical results are presented. In particular we show that the CAP-RECAP estimator outperforms the product one $\widehat{|\mathcal{X}^*|}$. Finally, in Section 2.5 some concluding remarks are given.

2.2 Decision Making

Here we present two heuristics for speeding up the direct Algorithm 2.6.3, which will be used for decision making. They are called (i) *local m_t updating* and (ii) *global m_t updating*, respectively. It is important to note that they are applicable only for the direct estimator (see (2.22) below), but not for the product one (2.8).

Local m_t updating In this version, at each iteration t we replace the fixed m_t value in the Gibbs sampler with the elite sample values at that iteration. To clarify, consider for simplicity the sum of n Bernoulli random variables. Let for concreteness $n = 5$, $N = 100$ and $\rho = 0,01$. Assume that while taking the sample of size $N = 100$ we obtained the following sequence of elite vectors $\boldsymbol{X}_{t1} = (1,1,0,1,0)$, $\boldsymbol{X}_{t2} = (1,0,0,1,0)$, $\boldsymbol{X}_{t3} = (1,0,1,1,0)$, $\boldsymbol{X}_{t4} = (1,0,0,1,0)$, $\boldsymbol{X}_{t5} = (1,0,0,1,1)$. The corresponding elite sample values are $S(\boldsymbol{X}_{t1}) = 3$, $S(\boldsymbol{X}_{t2}) = 2$, $S(\boldsymbol{X}_{t3}) = 3$, $S(\boldsymbol{X}_{t4}) = 2$, $S(\boldsymbol{X}_{t5}) = 3$, and clearly $m_t = 2$. Thus, in this version we simple replace $m_t = 2$ by the corresponding $S(\boldsymbol{X})$ elite values $3,\ 2,\ 3, 2,\ = 3$, while all ather data in the direct Algorithm 2.6.3 remaining the same.

Global m_t updating In this version we want the sample performance $S(\boldsymbol{X})$ to be a non-decreasing function as Gibbs proceeds. To clarify, assume that at iteration t we have the same elite sample as before, that is $\boldsymbol{X}_{t1} = (1,1,0,1,0)$, $\boldsymbol{X}_{t2} = (1,0,0,1,0)$, $\boldsymbol{X}_{t3} = (1,0,1,1,0)$, $\boldsymbol{X}_{t4} = (1,0,0,1,0)$, $\boldsymbol{X}_{t5} = (1,0,0,1,1)$

with $m_t = 2$. Let us pick up one of the elites, say the first one corresponding to $\boldsymbol{X}_{t1} = (1,1,0,1,0)$ and let us apply to it the systematic Gibbs sampler. Noticing that $\boldsymbol{X}_{t1}$ has 3 unities, we simply replace the original level $m_t = 2$ by the current value $S(\boldsymbol{X}_{t1}) = 3$ and set a new sub-level $m_{t1} = S(\boldsymbol{X}_{t1}) = 3$. We then proceed from $\boldsymbol{X}_{t1} = (1,1,0,1,0)$ with $m_{t1} = 3$ to a new value denoted as $\boldsymbol{X}_{t1}^{(1)}$. Assume that while applying systematic Gibbs sampler to the first components we obtained $\boldsymbol{X}_{t1}^{(1)} = (1,1,0,1,0)$, that is $\boldsymbol{X}_{t1}^{(1)} = \boldsymbol{X}_{t1} = (1,1,0,1,0)$. We next proceed from $\boldsymbol{X}_{t1}^{(1)} = (1,1,0,1,0)$ (with $m_{t2} = 3$) to a new vector $\boldsymbol{X}_{t1}^{(2)}$ by applying systematic Gibbs sampler to the second components. Let $\boldsymbol{X}_{t1}^{(2)} = (1,1,1,1,0)$. Since $\boldsymbol{X}_{t1}^{(2)} = (1,1,1,1,0)$ contains 4 unities, we set the new sub-level $m_{t3} = 4$. Assume that proceeding further we obtain $m_{t5} = m_{t4} = m_{t3} = 4$ and let also $\boldsymbol{X}_{t1}^{(4)} = \boldsymbol{X}_{t1}^{(3)} = \boldsymbol{X}_{t1}^{(2)} = (1,1,1,1,0)$. The resulting sequence of sub-levels m_{ti} in the systematic Gibbs sampler starting at $\boldsymbol{X}_{t1} = (1,1,0,1,0)$ is therefore $(m_{t1}, m_{t2}, m_{t3}, m_{t4}, m_{t5}) = (3,3,4,4,4)$ and similar for the remaining four elite values $\boldsymbol{X}_{t2} = (1,0,0,1,0)$, $\boldsymbol{X}_{t3} = (1,0,1,1,0)$, $\boldsymbol{X}_{t4} = (1,0,0,1,0)$, $\boldsymbol{X}_{t5} = (1,0,0,1,1)$. After that we define a new common level m_{t+1} for iteration $t+1$. In Section 2.4 we present some numerical results with the above heuristics.

2.3 Counting with the Capture-Recapture Method

Here we show how the well known *capture-recapture* (CAP-RECAP) method can be used as alternative to the product estimator $\widehat{|\mathcal{X}^*|}$ in (2.3).

We consider two versions of (CAP-RECAP) (i) the classic one (ii) the proposed on-line one.

2.3.1 Application of the Classic Capture Recapture

Originally the capture-recapture method was used to estimate the size, say M, of unknown population, under the assumption that *two* independent samples are taken from that population.

To see how the CAP-RECAP method works consider an urn model model with a total of M identical balls. Denote by N_1 and N_2, the sample sizes taken at the first and the second draw. Assume in addition that

1. The second draw take place only after all N_1 balls are returned back to the urn.

2. Before returning the N_1 balls back we *mark* each of them, say we paint them in a different color.

Denote by R the number of balls from the first draw that also appear at the second one. Clearly that the estimate of M, denoted by $\widetilde{M}$ is

$$\widetilde{M} = \frac{N_1 N_2}{R}.$$

This is so since

$$\frac{N_2}{M} \approx \frac{R}{N_1}.$$

Note that the name *capture-recapture* comes from the name of model where one is interested to estimate the animal population size in a particular area, provided two visits are available to the area. In this case R denotes the number of animals captured on the first visit that were then recaptured on the second one.

It is well know that a slightly better unbiased estimate of M is

$$\widehat{M} = \frac{(N_1+1)(N_2+1)}{(R+1)} - 1. \tag{2.17}$$

The corresponding variance is

$$\mathbb{V}\text{ar}(\widehat{M}) = \frac{(N_1+1)(N_2+1)(N_1-R)(N_2-R)}{(R+1)(R+2)(R+3)}. \tag{2.18}$$

Application of The CAP-RECAP to counting problems is trivial. We set $|\mathcal{X}^*| = M$ and note that N_1 and N_2 correspond to the screened out Gibbs samples at the first and second draws, which are performed after Algorithm 2.6.2 reaches the desired level m.

As an example, assume that in both experiments (draws) we set originally $N = 10,000$ and then we obtained $N_1 = 5,000$, $N_2 = 5,010$ and $R = 10$. The capture-recapture (CAP-RECAP) estimator of $|\mathcal{X}^*|$, denoted by $\widehat{|\mathcal{X}^*|_{cap}}$ is therefore

$$\widehat{|\mathcal{X}^*|_{cap}} = 2,505,000.$$

Clearly, the direct estimator $\widehat{|\mathcal{X}^*|_{dir}}$ can not handle such big number.

Our numerical results below clearly indicate that the CAP-RECAP estimator $\widehat{|\mathcal{X}^*|_{cap}}$ is typically more accurate than the product one $\widehat{|\mathcal{X}^*|}$, that is

$$\mathbb{V}\text{ar}\widehat{|\mathcal{X}^*|} > \mathbb{V}\text{ar}\widehat{|\mathcal{X}^*|_{cap}},$$

provided that the the sample N is limited, say by 10,000 and if $|\mathcal{X}^*|$ is large but limited, say by 10^7. If, however, $|\mathcal{X}^*|$ is very large, say $|\mathcal{X}^*| > 10^7$, then $\widehat{|\mathcal{X}^*|_{cap}}$ might become meaningless, since with the budget of $N = 10,000$ we will obtain often $R = 0$, provided $|\mathcal{X}^*| > 10^7$. However, the latter case has limited application, since if $|\mathcal{X}^*|$ is very large we can estimate it with the crude Monte Carlo.

2.3.2 Application of the On-line Capture Recapture

To see how the CAP-RECAP method works on-line consider again the urn model with a total of M unknown identical balls. In this case we take only *one draw* of size N instead of two ones (of sizes N_1 and N_2, respectively as before) and proceed as follows:

1. We draw the N balls *one-by-one with replacement.*

2. Before returning each of the N balls back to the urn we *mark* it.

3. As in classic method we count the number of marked balls.

Note that the marking procedure here is different from the classic one. Also note that by drawing the N balls on-line (sequentially), each ball can be drawn with positive probability up to N times, while in the classic one each ball has a positive probability to be drawn only up to *two* times. Now having R marked balls at hand how can we find the on-line estimator of M, denoted by $\widehat{|\mathcal{X}^*_{on}|}$ (recall that in our case $M = |\mathcal{X}^*|$), its expected value and the associated variance. We proceed argue as follows.

The probability that the same ball will appear exactly N times is $(1/M)^N$; exactly 2 times is $N(1-1/M)(1/M)^{N-1}$, etc.

Proceeding we can readily obtain that the on-line estimator of M and the associated variance. Although this will be done some where else, it is intuitively clear that the on-line CAP-RECAP estimator $\widehat{|\mathcal{X}^*_{on}|}$ is more exact than the classic one $\widehat{|\mathcal{X}^*_{cap}|}$, provided $N = N_1 + N_2$. The reason is that the on-line one is based on conditioning and conditioning always reduces variance.

Remark 2.3.1. As a third alternative to both the classic and the on-line CAP-RECAP estimators we can use the direct estimator $\widehat{|\mathcal{X}^*_{dir}|}$ in (2.22) to estimate M by taking into account the number of screened out elements at the level m, which is equal to $N - \widehat{|\mathcal{X}^*_{dir}|}$. As an estimator of $\mathcal{X}^*$, denoted by $\widehat{|\mathcal{X}^*_{scr}|}$ we can take the following one.

$\widehat{|\mathcal{X}^*_{scr}|} = \widehat{|\mathcal{X}^*_{dir}|}$ if $\widehat{|\mathcal{X}^*_{dir}|} \le N/2$ and $\widehat{|\mathcal{X}^*_{scr}|} = \frac{N^2}{\widehat{|\mathcal{X}^*_{dir}|}}$, otherwise.

2.4 Numerical Results

Below we present numerical results with CAP-RECAP method for counting and with the global m_t heuristics for decision making.

2.4.1 Counting

Consider the random 3-SAT with the instance matrix $A = (75 \times 325)$ taken from www.satlib.org and its truncated version $A = (75 \times 305)$.

Table 2.1 presents comparison of the performance of the product estimator $\widehat{|\mathcal{X}^*|}$ and its counterpart $\widehat{|\mathcal{X}^*_{cap}|}$ using the enhanced splitting Algorithm 2.6.2 for $A = (75 \times 305)$. Table 2.2 presents similar data for $A = (75 \times 325)$. We set $N = 10,000$, $\rho = 0.1$ and $b = \eta$. Table 2.3 presents the dynamic of one of the runs of Algorithm 2.6.2 for $A = (75 \times 305)$. We found that the the average CPU time is about 6 minutes for each run.

Note that the sample N_1 was obtained as soon as soon as Algorithm 2.6.2 reaches the final level m, and N_2 was obtained while runing Algorithm 2.6.2 for one more iteration at the same level m. The actual sample sizes N_1 and N_2 were chosen according to the following rule: *sample until Algorithm 2.6.2 screens out 50% of the samples and then stop.* It follows from Tables 2.1 that for model $A = (75 \times 305)$ this corresponds to $N_1 \approx N_2 \approx 26,000$ and $R \approx 22,000$, while for model $A = (75 \times 325)$ it follows from Table 2.2 that $N_1 \approx N_2 \approx R \approx 2,200$. It also follows that for $A = (75 \times 305)$ and $A = (75 \times 325)$ the relative error of $\widehat{|\mathcal{X}^*_{cap}|}$ is about 10 and 100 times smaller as compared to $\widehat{|\mathcal{X}^*|}$. It is readily seen that by enlarging the samples N_1 and N_2 only at the last two iterations of Algorithm 2.6.2 the relative error of $\widehat{|\mathcal{X}^*_{cap}|}$ will further decrease.

Table 2.1: Comparison of the performance of the product estimator $\widehat{|\mathcal{X}^*|}$ with its counterpart $\widehat{|\mathcal{X}^*_{cap}|}$ for SAT (75×305) model.

Run N_0	Iterations	$\widehat{\lvert\mathcal{X}^*\rvert}$	RE of $\widehat{\lvert\mathcal{X}^*\rvert}$	$\widehat{\lvert\mathcal{X}^*_{cap}\rvert}$	RE of $\widehat{\lvert\mathcal{X}^*_{cap}\rvert}$	N_1	N_2	R
1	21	2.67E+04	1.39E-01	3.07E+04	4.49E-02	23993	23908	18681
2	21	4.10E+04	3.22E-01	3.27E+04	1.57E-02	27064	26945	22333
3	21	2.85E+04	8.08E-02	3.19E+04	7.33E-03	26638	26567	22176
4	21	2.96E+04	4.36E-02	3.09E+04	3.83E-02	23907	23993	18552
5	21	2.87E+04	7.29E-02	3.29E+04	2.41E-02	26967	27120	22214
6	21	3.63E+04	1.71E-01	3.23E+04	4.25E-03	26838	26762	22247
7	21	2.39E+04	2.28E-01	3.30E+04	2.64E-02	26719	26697	21618
8	21	4.10E+04	3.22E-01	3.29E+04	2.32E-02	26842	26878	21933
9	21	2.72E+04	1.23E-01	3.21E+04	1.44E-03	26645	26578	22060
10	21	2.70E+04	1.29E-01	3.21E+04	1.75E-03	26512	26588	21965
Average	21	3.10E+04	1.63E-01	3.21E+04	1.87E-02			
Variance	0	3.77E+07	9.71E-03	6.45E+05	2.34E-04			

Table 2.2: Comparison of the performance of the product estimator $\widehat{|\mathcal{X}^*|}$ and its counterpart $\widehat{|\mathcal{X}^*_{cap}|}$ for SAT (75×325) model.

Run N_0	Iterations	$\widehat{\lvert\mathcal{X}^*\rvert}$	RE of $\widehat{\lvert\mathcal{X}^*\rvert}$	$\widehat{\lvert\mathcal{X}^*_{cap}\rvert}$	RE of $\widehat{\lvert\mathcal{X}^*_{cap}\rvert}$	N_1	N_2	R
1	24	2.02E+03	1.03E-02	2.21E+03	6.55E-03	2201	2195	2191
2	24	1.94E+03	2.95E-02	2.20E+03	7.01E-03	2200	2202	2198
3	24	1.59E+03	2.03E-01	2.24E+03	7.86E-03	2234	2235	2232
4	24	2.34E+03	1.70E-01	2.23E+03	2.45E-03	2221	2223	2219
5	24	1.69E+03	1.54E-01	2.20E+03	1.11E-02	2194	2191	2190
6	24	2.38E+03	1.89E-01	2.24E+03	6.96E-03	2230	2230	2225
7	24	1.63E+03	1.86E-01	2.22E+03	6.53E-04	2215	2216	2210
8	24	2.38E+03	1.89E-01	2.23E+03	5.15E-03	2225	2229	2223
9	24	1.97E+03	1.66E-02	2.22E+03	1.55E-03	2217	2219	2213
10	24	2.12E+03	6.03E-02	2.21E+03	3.86E-03	2206	2208	2203
Average	24	2.01E+03	1.21E-01	2.22E+03	5.31E-03			
Variance	0	9.10E+04	6.55E-03	2.04E+02	1.03E-05			

Table 2.3: Dynamics of of one of the runs of the enhanced Algorithm for the random 3-SAT with matrix $A = (75 \times 305)$.

t	$\widehat{\lvert\mathcal{X}^*\rvert}$	$\widehat{\lvert\mathcal{X}^*_{cap}\rvert}$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	4.62E+21	–	1223	1223	285	274	0.122
2	6.88E+20	–	1490	1490	288	279	0.149
3	7.52E+19	–	1093	1093	291	283	0.109
4	8.62E+18	–	1146	1146	292	286	0.115
5	1.57E+18	–	1817	1817	293	288	0.182
6	2.33E+17	–	1489	1489	296	290	0.149
7	2.46E+16	–	1053	1053	296	292	0.105
8	7.34E+15	–	2987	2987	297	293	0.299
9	1.93E+15	–	2635	2635	298	294	0.264
10	4.74E+14	–	2454	2454	300	295	0.245
11	1.07E+14	–	2251	2251	299	296	0.225
12	2.09E+13	–	1960	1960	300	297	0.196
13	3.65E+12	–	1742	1742	302	298	0.174
14	5.66E+11	–	1551	1551	302	299	0.155
15	7.22E+10	–	1276	1276	303	300	0.128
16	8.34E+09	–	1155	1155	304	301	0.116
17	7.64E+08	–	917	917	304	302	0.092
18	5.10E+07	–	667	667	304	303	0.067
19	2.10E+06	–	412	412	305	304	0.041
20	3.28E+04	–	156	156	305	305	0.016
21	3.38E+04	3.21e+004	10000	8484	305	305	1.000

Here we used the following notations

1. N_t and $N_t^{(s)}$ denote the actual number of elites and the one after screening, respectively.

2. m_t^* and m_{*t} denote the upper and the lower elite levels reached, respectively.

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

2.4.2 Decision Making

Recall that the goal of the decision making algorithm is to decide whether or not the set $\mathcal{X}$ of the integer program constraints (2.10) has a feasible solution. The decision making algorithm presents a simple modification the direct Algorithm 2.6.3. In particular:

1. Instead of counting according to (2.22), that is

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

   we make decision in the sense that we need to decide whether $|\widehat{\mathcal{X}}^*_{dir}| > 0$ or $|\widehat{\mathcal{X}}^*_{dir}| = 0$.

2. Apply *global* m_t policy from instead of the standard splitting step (see Step 3. in Algorithm 2.6.3.

We call such modified Algorithm 2.6.3 as the *decision making* Algorithm.

We run the decision making Algorithm 2.6.3 for different SAT problems taken from www.satlib.org using the *global* m_t policy. In particular we took 40 instances out of more than 200 instances available, each presenting a random 3-SAT with the instance matrix $\boldsymbol{A}$ of size (250×1065) and with $|\widehat{\mathcal{X}}^*_{dir}| > 1$. We set the burn in parameter $b = 10$, $N = 10,000$ and $\rho = 0.5$. The CPU time was about 10 minutes. We always obtained that $|\widehat{\mathcal{X}}^*_{dir}| > 1$, that is we found that our algorithms works nicely. For $N = 1,000$ we found, however that our algorithm failed for some instances. The same was true for $N = 10,000$ and $\rho < 0.5$.

Table 2.4 presents the dynamics of one of such runs.

Table 2.4: Performance of the decision making Algorithm 2.6.3 with *global* m_t policy for the random 3-SAT with the clause matrix $A = (250 \times 1065)$, $N = 10,000$, $\rho = 0.5$ and $b = 10$

t	$\widehat{\lvert\mathcal{X}^*\rvert}$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	8.76e+074	0	5161	4841	972	933	0.48
2	4.27e+074	0	7436	7079	1050	1023	0.49
3	1.84e+074	0	7288	6106	1058	1043	0.43
4	7.49e+073	0	6601	4970	1062	1051	0.41
5	2.85e+073	0	8318	5665	1062	1056	0.38
6	8.43e+072	0	6383	3353	1064	1059	0.30
7	2.48e+072	0	6745	2965	1065	1061	0.29
8	3.65e+071	0	6090	1745	1065	1063	0.15
9	5.90e+070	0	10470	1690	1065	1064	0.16
10	1.09e+070	1871	10140	1873	1065	1065	0.18
11	1.09e+070	11238	11238	11238	1065	1065	1.00

Table 2.5 presents the dynamics of a run with the same *global* m_t policy Algorithm 2.6.3 for the random SAT with the instance matrix $A = (122 \times 663)$ and a single valid assignment, ($|\mathcal{X}^*| = 1$), taken from http://www.is.titech.ac.jp/ watanabe/gensat. We set $N = 50,000$, $\rho = 0.95$ and $b = 10$. The results are self-explanatory. Note that

1. For $\rho < 0.95$ Algorithm 2.6.3 is stacked some where before 663.
2. The CPU time is about 5 hours.
3. For this difficult model with a single valid assignment we found that the *global* m_t policy Algorithm 2.6.3 has approximately the same running time as the enhanced cloning Algorithm 2.6.2, for which we used the same $N = 50,000$, but $\rho = 0.1$ instead of $\rho = 0.95$.

Table 2.5: Performance of the *global* m_t policy Algorithm 2.6.3 for the random $3-4$-SAT with the instance matrix $A = (122 \times 663)$, $N = 50,000$, $\rho = 0.95$ and $b = 10$

t	$\widehat{\lvert\mathcal{X}^*\rvert}$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	ρ_t
1	5.03e+036	0	28777	28412	627	585	0.96
2	4.77e+036	0	54792	53926	650	622	0.96
3	4.43e+036	0	52042	50652	653	634	0.97
4	4.18e+036	0	49586	48082	655	639	0.98
5	3.90e+036	0	47238	45392	656	642	0.98
6	3.63e+036	0	44824	42724	657	644	0.99
7	3.15e+036	0	41357	37781	658	646	0.97
8	2.85e+036	0	37781	34972	658	647	1.00
9	2.62e+036	0	34972	31710	658	648	1.00
10	2.20e+036	0	31710	27547	658	649	1.00
11	2.03e+036	0	55094	45921	660	650	1.00
12	1.44e+036	0	45921	34996	660	651	1.00
13	5.36e+035	0	34996	23854	660	652	1.00
14	4.20e+035	0	47708	28573	660	653	1.00
15	4.19e+035	0	57146	28378	660	654	1.00
16	3.43e+035	0	56756	22473	661	655	1.00
17	1.07e+035	0	44946	13879	661	656	1.00
18	1.66e+032	0	41637	7867	661	657	1.00
19	1.08e+031	0	31468	3390	661	658	1.00
20	9.72e+030	0	30510	1126	663	659	1.00
21	6.98e+024	0	30402	204	663	660	1.00
22	1.99e+017	0	30600	3	663	661	1.00
23	1.10e+010	0	30600	1	663	662	1.00
24	1.10e+010	1	30600	1	663	663	1.00

2.5 Concluding Remarks

We showed how a simple modification of the splitting method based on Gibbs sampler can be efficiently used for decision making in the sense that one can efficiently decide whether or not a given set of integer program constraints has at least one feasible solution. Our decision making is based on what is called the *local* m_t and *global* m_t modification of the direct splitting Algorithm 2.6.3. We also show how to incorporate the classic *capture-recapture* method into the direct Algorithm 2.6.3 in order to obtain a low variance estimator for the counting quantity representing, say the number feasible solution on the set defined as by the constraints of an integer program. We finally present numerical with with both, the decision making and the capture-recapture estimators and show their superiority as compared to the conventional product one $|\widehat{\mathcal{X}^*}|$, while solving quite general decision making and counting ones, like the satisfiability problems.

2.6 Appendix: Splitting Algorithms

Below, following [83], we present two versions of the splitting algorithm: the so-called *basic* version and the *enhanced* version having in mind Example 2.1.1.

2.6.1 Basic Splitting Algorithm

Let N, ρ_t and N_t be the fixed sample size, the adaptive rarity parameter and the number of elite samples at iteration t, respectively (see [83] details). Recall [83] that the elite sample $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ corresponds to the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_t}\}$, for which $S(\boldsymbol{X}_i) \geq \widehat{m}_t$, that is $\widehat{m}_t$ is the $(1-\rho_t)$ sample quantile of of the ordered statistics values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$. It follows that the number of elites $N_t = \lceil N\rho_t \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the largest integer.

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

Algorithm 2.6.1 (Basic Splitting Algorithm for Counting)**.** Given the initial parameter ρ_0, say $\rho_0 \in (0.01, 0.25)$ and the sample size N, say $N = nm$, execute the following steps:

1. **Acceptance-Rejection** Set a counter $t = 1$. Generate a sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ uniformly on $\mathcal{X}_0$. Let $\widehat{\mathcal{X}}_0 = \{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_0}\}$ be the elite samples. Take
$$\widehat{c}_0 = \widehat{\ell}(\widehat{m}_0) = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_0\}} = \frac{N_0}{N} \tag{2.19}$$
as an *unbiased* estimator of c_0. Note that $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_0} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$, where $g^*(\boldsymbol{x}, \widehat{m}_0)$ is a *uniform distribution* on the set $\mathcal{X}_1 = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_0\}$.

2. **Splitting** Let $\widehat{\mathcal{X}}_{t-1} = \{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ be the elite sample at iteration $(t-1)$, that is the subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$ for which $S(\boldsymbol{X}_i) \geq \widehat{m}_{t-1}$. Reproduce $\eta_{t-1} = \lceil \rho_{t-1}^{-1} \rceil$ times each vector $\widehat{\boldsymbol{X}}_k =$

$(\widehat{X}_{1k}, \dots, \widehat{X}_{nk})$ of the elite sample $\{\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$, that is take η_{t-1} identical copies of each vector $\widehat{\boldsymbol{X}}_k$. Denote the entire new population ($\eta_{t-1}N_{t-1}$ cloned vectors plus the original elite sample $\{\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{cl} = \{(\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_1), \dots, \ (\widehat{\boldsymbol{X}}_{N_{t-1}}, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each of the cloned vectors of the population $\mathcal{X}_{cl}$ apply the MCMC (and in particular the random Gibbs sampler) for a single period (single burn-in). Denote the *new entire* population by $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_N\}$. Note that each vector in the sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ is distributed $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$, where $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$ has *approximately* a uniform distribution on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_{t-1}\}$.

3. **Estimating** c_t Take $\widehat{c}_t = \frac{N_t}{N}$ (see (2.9)) as an estimator of c_t in (2.7). Note again that each vector of $\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_t}$ of the elite sample is distributed $g^*(\boldsymbol{x}, \widehat{m}_t)$, where $g^*(\boldsymbol{x}, \widehat{m}_t)$ has approximately a uniform distribution on the set $\mathcal{X}_{t+1} = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_t\}$.

4. **Stopping rule** If $m_t = m$ go to step 5, otherwise set $t = t + 1$ and repeat from step 2.

5. **Final Estimator** Deliver $\widehat{\ell}$ given in (2.8) as an estimator of ℓ and $|\widehat{\mathcal{X}}^*| = \widehat{\ell}|\mathcal{X}|$ as an estimator of $|\mathcal{X}^*|$.

Note that at iteration t Algorithm 2.6.1 *splits* each elite sample $\eta_t = \lceil \rho_t^{-1} \rceil$ times. By doing it generates $\lceil \rho_t^{-1} N_t \rceil \approx N$ new samples for the next iteration $t+1$. The rationale is based on the fact that if all ρ_t are not small, say $\rho_t \geq 0.01$, we have at each iteration t enough *stationary* elite samples, and all what the Gibbs sampler has to do for the next iteration is to generate $N \approx \lceil \rho_t^{-1} N_t \rceil$ *new* samples uniformly distributed on $\mathcal{X}_{t+1}$.

Figure 2.1 presents a typical dynamics of the splitting algorithm, which terminates after two iterations. The set of points denoted $\star$ and $\bullet$ is associated with these two iterations. In particular the points marked by $\star$ are uniformly distributed on the sets $\mathcal{X}_0$ and $\mathcal{X}_1$. (Those, which are in $\mathcal{X}_1$ correspond to the elite samples). The points marked by $\bullet$ are approximately uniformly distributed on the sets $\mathcal{X}_1$ and $\mathcal{X}_2$. (Those, which are in $\mathcal{X}_2 = \mathcal{X}^*$ likewise correspond to the elite samples).

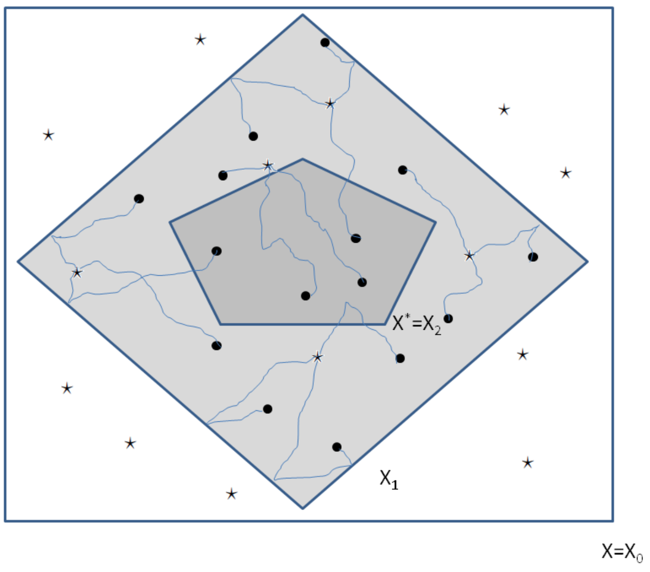


Figure 2.1: Dynamics of Algorithm 2.6.1

2.6.2 Enhanced Splitting Algorithm for Counting

Here we introduce an enhanced version of the basic splitting Algorithm 2.6.1, which contains (i) an enhanced splitting (splitting) step instead of the original one as in Algorithms 2.6.1 and a (ii) new screening step.

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

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

$$\eta_{t-1} = \left\lceil \frac{N}{bN_{t-1}} \right\rceil - 1 = \left\lceil \frac{N_{cl}}{N_{t-1}} \right\rceil - 1. \tag{2.20}$$

Here $N_{cl} = N/b$ is called the *cloned sample size*, and as before $N_{t-1} = \rho_{t-1}N$ denotes the number of elites and ρ_{t-1} is the adaptive rarety parameter at iteration $t-1$ [see [86] for details).

As an example, let $N = 1,000,\ \ b = 10$. Consider two cases: $N_{t-1} = 21$ and $N_{t-1} = 121$. We obtain $\eta_{t-1} = 4$ and $\eta_{t-1} = 0$ (no cloning), respectively.

As an alternative to (2.20) one can use the following heuristic strategy in defining b and η: find b_{t-1} and η_{t-1} from $b_{t-1}\eta_{t-1} \approx \frac{N}{N_{t-1}}$ and take $b_{t-1} \approx \eta_{t-1}$. In short, one can take

$$b_{t-1} \approx \eta_{t-1} \approx \left(\frac{N}{N_{t-1}}\right)^{1/2}. \tag{2.21}$$

Consider again the same two cases for N_{t-1} and N We have $b_{t-1} \approx \eta_{t-1} = 7$ and $b_{t-1} \approx \eta_{t-1} = 3$, respectively. We found numerically that both versions work well, but unless stated otherwise we shall use (2.21).

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

Algorithm 2.6.2 (Enhanced Splitting Algorithm for Counting)**.** Given the parameter ρ, say $\rho \in (0.01, 0.25)$ and the sample size N, say $N = nm$, execute the following steps:

1. **Acceptance-Rejection** - the same as in Algorithm 2.6.1.

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

3. **Splitting (Cloning)** Given the size N_{t-1} of the screened elites $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ at iteration $(t-1)$, find the splitting and the burn-in parameters η_{t-1} and b_{t-1} according to (2.21). Reproduce η_{t-1} times each vector $\widehat{\boldsymbol{X}}_k = (\widehat{X}_{1k}, \ldots, \widehat{X}_{nk})$ of the screened elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$, that is, take η_{t-1} identical copies of each vector $\widehat{\boldsymbol{X}}_k$ obtained at the $(t-1)$-th iteration. Denote the entire new population ($\eta_{t-1}N_{t-1}$ cloned vectors plus the original screened elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{cl} = \{(\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_1), \ldots, (\widehat{\boldsymbol{X}}_{N_{t-1}}, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each of the cloned vectors of the population $\mathcal{X}_{cl}$ apply the MCMC (and in particular the Gibbs sampler) for b_{t-1} burn-in periods. Denote the *new entire* population by $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$. Note that each vector in the sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is distributed approximately $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$, where $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$ is a uniform distribution on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_{t-1}\}$.

4. **Estimating** c_t - the same as in Algorithm 2.6.1.

5. **Stopping rule** -the same as in Algorithm 2.6.1.

6. **Final estimator** - the same as in Algorithm 2.6.1.

Note that the basic Algorithm 2.6.1 (with $b = 1$ and without screening) presents a particular case of the enhanced Algorithm 2.6.2.

2.6.3 Direct Splitting Algorithm

The direct estimator below can be viewed as an alternative to the estimator $|\widehat{\mathcal{X}^*}|$ obtained by Algorithm 2.6.1. This estimator is based on *direct counting* of the number of screened samples obtained immediately after crossing the level m. Such a counting estimator, denoted by $|\widehat{\mathcal{X}^*_{dir}}|$, is associated with the *empirical* distribution of the uniform distribution $g^*(\boldsymbol{x}, m)$. We found numerically that $|\widehat{\mathcal{X}^*_{dir}}|$ is extremely useful and very accurate. Note that it is applicable only for counting problems with $|\mathcal{X}^*|$ not too large. Specifically $|\mathcal{X}^*|$ should be less than the sample size N, that is $|\mathcal{X}^*| < N$. Note also that counting problems with values small relative to $|\mathcal{X}|$ are the most difficult ones and in many counting problems one is interested in the cases where $|\mathcal{X}^*|$ does not exceed some fixed quantity, say $\mathcal{N}$. Clearly, this is possible only if $N \geq \mathcal{N}$. It is important to note that $|\widehat{\mathcal{X}^*_{dir}}|$ is typically much more accurate than its counterpart, the standard estimator $|\widehat{\mathcal{X}^*}| = \widehat{\ell}|\mathcal{X}|$. The reason is that $|\widehat{\mathcal{X}^*_{dir}}|$ is obtained *directly* by counting all distinct values of $\boldsymbol{X}_i$, $i = 1, \ldots, N$ satisfying $S(\boldsymbol{X}_i) \geq m$, that is it can be written as

$$\widehat{|\mathcal{X}^*_{dir}|} = \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i^{(d)}) \geq m\}}, \tag{2.22}$$

where $\boldsymbol{X}_i^{(d)} = \boldsymbol{X}_i$, if $\boldsymbol{X}_i \neq \boldsymbol{X}_j$, $\forall j = 1, \ldots, i-1$ and $\boldsymbol{X}_i^{(d)} = 0$, otherwise. Note that we set in advance $\boldsymbol{X}_1^{(d)} = \boldsymbol{X}_1$. Note also that there is no need here to calculate $\widehat{c}_t$ at any step.

Algorithm 2.6.3 (Direct Algorithm for Counting)**.** Given the rarity parameter ρ, say $\rho = 0.1$, the parameters a_1 and a_2, say $a_1 = 0.01$ and $a_2 = 0.25$, such that $\rho \in (a_1, a_2)$, and the sample size N, execute the following steps:

1. **Acceptance-Rejection** - same as in Algorithm 2.6.2.

2. **Screening** - same as in Algorithm 2.6.2.

3. **Splitting** - same as in Algorithm 2.6.2.

4. **Stopping rule** - same as in Algorithm 2.6.2.
5. **Final Estimator** For $m_T = m$, take a sample of size N, and deliver $|\widehat{\mathcal{X}}^*_{dir}|$ in (2.22) as an estimator of $|\mathcal{X}^*|$.

Note that the counting Algorithm 2.6.3 can be readily modified for combinatorial optimization, since an optimization problem can be can be viewed as a particular case of counting, where the counting quantity $|\mathcal{X}^*| = 1$.

Chapter 3

How to Generate Uniform Samples on Discrete Sets Using the Splitting Method

Peter W. Glynn, Andrey Dolgin, Reuven Y. Rubinstein and Radislav Vaisman

Faculty of Industrial Engineering and Management,

Technion, Israel Institute of Technology, Haifa, Israel

ierrr01@ie.technion.ac.il

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

[0†] This research was supported by the BSF (Binational Science Foundation) grant No 2008482 and ISF (Israel Science Foundation) grant No 2012910

Abstract

The goal of this work is twofold. We show that

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

Keywords. Combinatorial Optimization, Counting, Cross-Entropy, Decision Making, Gibbs Sampler, MCMC, Rare-Event, Splitting.

3.1 Introduction: The Splitting Method

The goal of this work is to show that:

1. The classic MCMC (Markov Chain Monte Carlo) *fails* to generate points uniformly distributed on discrete sets, such as that defined by the constraints of integer programming with both equality and inequality constraints, that is

$$\begin{aligned}
&\textstyle\sum_{k=1}^{n} a_{ik}x_k = b_i,\ i = 1,\ldots,m_1,\\
&\textstyle\sum_{k=1}^{n} a_{jk}x_k \geq b_j,\ j = m_1+1,\ldots,m_1+m_2,\\
&\boldsymbol{x} = (x_1,\ldots,x_n) \geq \boldsymbol{0},\quad x_k \text{ is integer } \forall k = 1,\ldots,n.
\end{aligned} \tag{3.1}$$

   We demonstrate empirically that starting MCMC from any initial point in the desired set $\mathcal{X}^*$ given in (3.1) and running it for a very long time, not only it *fails* to generate uniform points on $\mathcal{X}^*$, but it samples only in some subset of $\mathcal{X}^*$, rather than in the entire set $\mathcal{X}^*$. We observed that this is the case even if $\mathcal{X}^*$ is very small containing only view points. Thus, in spite of the common consensus on MCMC as a universal tool for generating samples on complex sets, our empirical studies on discrete sets, like (3.1) have proved quite negative. Although we do not have a formal prove, but we guess (conjecture) that the main reason the classic MCMC is not working is that its resulting chain is not irreducible.

2. In contrast to MCMC, the *splitting* method, also called the *cloning* method, recently introduced in [82, 83] can be efficiently used for generating uniform samples on sets like (3.1). We provide valid statistical tests supporting the uniformity of generated samples on $\mathcal{X}^*$ and present supportive numerical results.

At first glance one might think that the classic MCMC [1, 78, 86, 90] should be a good alternative sets like (3.1). Indeed, MCMC has been successfully used for generating points on different complex regions. In all such cases, given an arbitrary initial point in $\mathcal{X}^*$, one runs MCMC for some time until it reaches steady-state, and then collects the necessary data. One of the most popular MCMC is the hit-and-run method [90] for generation of uniform points on continuous regions. Applications of hit-and-run for convex optimization are given in [72].

As mentioned, we shall show that this is not always the case: MCMC fails when one deals with discrete sets like (3.1). To emphasize this point, observe that most decision making, optimization and counting problems associated with the set (3.1) are NP-hard, thus one should not expect an easy way of generating points uniformly on (3.1) since it is shown in [82, 83]) that counting on $\mathcal{X}^*$ (which is NP-hard) is directly associated with uniform sampling on $\mathcal{X}^*$. It is also shown that the splitting method [82, 83] presents a combination of MCMC with a specially designed splitting mechanism. Again, without the appropriate splitting mechanism, MCMC fails.

Although this paper is mainly of empirical nature, we believe that it provides a good insight of the state of the art of generating uniform points on discrete sets $\mathcal{X}^*$ like (3.1), and that it will motivate further research.

We start by presenting some background on the splitting method following [82, 83]. For related references see [7], [28], [31], [55], [64], and [58].

Like the classic cross-entropy (CE) method [81], [85], the splitting one in [82, 83] was originally designed for counting and combinatorial optimization. As mentioned, the counting algorithm in [82, 83] assumes, in fact, uniform generation on that set $\mathcal{X}^*$. So, from that retrospective, one can view this generation as a nice "free“ by-product of this algorithm.

The rest of this section deals with the splitting method from [83] for counting. The main idea is to design a sequential sampling plan, with a view to decomposing a “difficult” counting problem defined on some set $\mathcal{X}^*$ into a number of “easy” ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ and such that $\mathcal{X}_m = \mathcal{X}^*$. Typically, splitting algorithms explore the connection between counting and sampling problems, in particular - reduction from approximate counting on a discrete set to approximate sampling of its elements by the classic MCMC method [86].

A typical splitting algorithm comprises the following steps:

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

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

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

$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = \ell |\mathcal{X}_0|, \tag{3.2}$$

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

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \tag{3.3}$$

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

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

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

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}| \prod_{t=1}^{m} \widehat{c}_t, \tag{3.4}$$

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

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

(i) Put the well known NP-hard counting problems into the framework (3.2) by making sure that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*$ and each c_t is not a rare-event probability.

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

While task (i) is typically not difficult [83], task (ii) is quite complicated and associated with generation of uniform samples at each sub-region $\mathcal{X}_t$ separately. As we shall see below, this can be done by combining the Gibbs sampler with a specially designed splitting mechanism. The resulting algorithm is called the *splitting* or *cloning* algorithm [83].

The main goal of this work is to show empirically that the *splitting* algorithm [83] is able to generate points uniformly distributed on different discrete sets.

To proceed, note that ℓ can be also written as

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X}) \geq m\}} \right], \tag{3.5}$$

where $\boldsymbol{X} \sim f(\boldsymbol{x})$, $f(\boldsymbol{x})$ is a uniform distribution on the entire set $\mathcal{X}$, as before; m is a fixed parameter, e.g. the total number of constraints in an integer program; and $S(\boldsymbol{X})$ is the sample performance, e.g. the number of feasible solutions generated by the above constraints. Alternatively (see(3.2)), ℓ can be written as

$$\ell = \prod_{t=1}^{T} c_t, \tag{3.6}$$

where

$$c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}| = \mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X})\geq m_{t-1}\}}]. \tag{3.7}$$

Here

$$g^*_{t-1} = g^*(\boldsymbol{x}, m_{t-1}) = \ell(m_{t-1})^{-1} f(\boldsymbol{x}) I_{\{S(\boldsymbol{x})\geq m_{t-1}\}}, \tag{3.8}$$

$\ell(m_{t-1})^{-1}$ is the normalization constant and similarly to (3.2) the sequence $m_t,\ t = 0, 1, \ldots, T$ represents a fixed grid satisfying $-\infty < m_0 < m_1 < \cdots < m_T = m$. Note that in contrast to (3.2) we use in (3.6) a product of T terms instead of m terms, where T may be a random variable. The latter case is associated with adaptive choice of the level sets $\{\widehat{m}_t\}_{t=0}^T$ resulting in $T \leq m$. Since for counting problems the pdf $f(\boldsymbol{x})$ should be *uniformly* distributed on $\mathcal{X}$, which we denote by $\mathcal{U}(\mathcal{X})$, it follows from (3.8) that the pdf $g^*(\boldsymbol{x}, m_{t-1})$ should be *uniformly* distributed on each set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_{t-1}\},\ t = 1, \ldots, T$, that is, $g^*(\boldsymbol{x}, m_{t-1})$ should be equal to $\mathcal{U}(\mathcal{X}_t)$. Recall that the goal of the paper is to show that this is indeed the case for $\mathcal{X}_T = \mathcal{X}^* = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq m_T\}$, where $m_T = m$.

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

$$\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t = \frac{1}{N^T} \prod_{t=1}^{T} N_t, \tag{3.9}$$

where

$$\widehat{c}_t = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i)\geq m_{t-1}\}} = \frac{N_t}{N}, \tag{3.10}$$

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

We next show how to put the counting problem of finding the number of feasible solutions of the set of integer programming constraints into the framework (3.5)- (3.8).

Example 3.1.1. The set of integer programming constraints Consider again the set $\mathcal{X}^*$ of integer programming constraints given in (3.1). Our goal is to generate points uniformly distributed of this set. We assume that each component $x_k,\ k = 1, \ldots, n$ has d different values, labeled $1, \ldots, d$. Note that the SAT problem represents a particular case of (3.1) with inequality constraints, and where $x_1, \ldots, x_n$ are binary components. Unless stated otherwise, we will bear in mind the counting problem on the set (3.1), in particular counting the true (valid) assignments in a SAT problem.

It is shown in [83] that in order to count the points of the set (3.1), one can associate it with the following rare-event probability problem

$$\ell = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X})=m\}} \right] = \mathbb{E}_f \left[I_{\{\sum_{i=1}^m C_i(\boldsymbol{X})=m\}} \right], \tag{3.11}$$

where the first m_1 terms $C_i(\boldsymbol{X})$'s in (3.11) are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^n a_{ik} X_k = b_i\}}, \; i = 1, \ldots, m_1, \tag{3.12}$$

while the remaining m_2 ones are

$$C_i(\boldsymbol{X}) = I_{\{\sum_{k=1}^n a_{ik} X_k \geq b_i\}}, \; i = m_1 + 1, \ldots, m_1 + m_2 \tag{3.13}$$

and $S(\boldsymbol{X}) = \sum_{i=1}^m C_i(\boldsymbol{X})$. Thus, in order to count the number of feasible solutions on the set (3.1) one can consider an associated rare-event probability estimation problem (3.11) involving a *sum of dependent Bernoulli random variables* C_i $i = m_1 + 1, \ldots, m$, and then apply $\widehat{|\mathcal{X}^*|} = \widehat{\ell}|\mathcal{X}|$. In other words, in order to count on $\mathcal{X}^*$ one needs to estimate efficiently the rare event probability ℓ in (3.11). A framework similar to (3.11) can be readily established for many NP-hard counting problems [83].

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

$$\{(m_0, g^*(\boldsymbol{x}, m_{-1})), \; (m_1, g^*(\boldsymbol{x}, m_0)), \; (m_2, g^*(\boldsymbol{x}, m_1)), \ldots, (m_T, g^*(\boldsymbol{x}, m_{T-1}))\}. \tag{3.14}$$

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

$$\mathbb{E}_{g^*_{t-1}} I_{\{S(\boldsymbol{X}) \geq m_t\}} = \rho, \tag{3.15}$$

where ρ is called the *rarity* parameter [83]. Typically one sets $0.1 \leq \rho \leq 0.01$. Note that in contrast to the CE method [81], [85], where one generates a sequence of tuples

$$\{(m_0, \boldsymbol{v}_0), \; (m_1, \boldsymbol{v}_1), \ldots, (m_T, \boldsymbol{v}_T)\}, \tag{3.16}$$

and where $\{\boldsymbol{v}_t, \; t = 1, \ldots, T\}$ is a sequence of parameters in the parametric family of distributions $f(\boldsymbol{x}, \boldsymbol{v}_t)$ here in (3.14) $\{g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1}, \; t = 0, 1, \ldots, T\}$ is a sequence of *non-parametric* IS distributions. Otherwise, the CE and the splitting algorithms are similar.

In the Appendix (see Section 3.5), following [83], we present two versions of the splitting algorithm for counting: the so-called *basic* Algorithm 3.5.1 and the *enhanced* one, Algorithm 3.5.2 bearing in mind Example 3.1.1. Recall that the crucial point is to ensure that the points generated from the pdf $g^*(\boldsymbol{x}, m_{t-1}) = g^*_{t-1}$ are uniformly distributed on the corresponding set $\mathcal{X}_t = \{S(\boldsymbol{X}) \geq m_t\},\ t = 1, \ldots, T$.

To understand that this is so, consider the enhanced Algorithm 3.5.2, bearing in mind that

1. The samples generated on the set $\mathcal{X}_1 = \{S(\boldsymbol{X}) \geq \widehat{m}_0\}$ from the pdf $g^*(\boldsymbol{x}, \widehat{m}_0) = g^*_0$ are *exactly* distributed uniformly, since the original distribution f is a uniform one on the entire space $\mathcal{X} = \mathcal{X}_0$ and since use of acceptance-rejection (see Step 1) yields uniform points on $\mathcal{X}_1$.

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

Note that the splitting algorithm in [83] is also suitable for optimization as well. Here we use the same sequence of tuples (3.14), but *without involving the product of the estimators* $\widehat{c}_t,\ t = 1, \ldots, T$.

The rest of our paper is organized as follows. Section 3.2 deals with the Gibbs sampler, which is an important element of the splitting algorithm. In particular, we show how to generate points uniformly on the set (3.1) avoiding acceptance-rejection. Section 3.3 presents supporting numerical results. In Section 3.4 conclusions and some directions for further research are given. Finally, in Appendix 3.5 the basic and the enhanced versions of the splitting algorithm are presented.

3.2 The Gibbs Sampler

In this Section we show how to use efficiently the Gibbs sampler to generate points uniformly on the set (3.1). We start with some background [86] on generation of points from a given joint pdf $g(x_1, \ldots, x_n)$. In the latter, instead of sampling directly from $g(x_1, \ldots, x_n)$, which might be very difficult, one samples

from the one-dimensional conditional pdfs
$g(x_i|X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_n),\ i=1,\ldots,n$, which is typically much simpler. Two basic versions of the Gibbs sampler are available: *systematic* and *random*. In the former the components of the vector $\boldsymbol{X}=(X_1,\ldots,X_n)$ are updated in a fixed, say increasing order, while in the latter they are chosen randomly according to a discrete uniform n-point pdf. Below we present the systematic Gibbs sampler algorithm. In the systematic version, for a given vector $\boldsymbol{X}=(X_1,\ldots,X_n)\sim g(\boldsymbol{x})$, one generates a *new* vector $\widetilde{\boldsymbol{X}}=(\widetilde{X}_1,\ldots,\widetilde{X}_n)$ with the same distribution $\sim g(\boldsymbol{x})$ as follows:

Algorithm 3.2.1 (Systematic Gibbs Sampler)**.**

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

Iterating with Algorithm 3.2.1, the Gibbs sampler generates (under some mild conditions [86]), a sample distributed $g(x_1,\ldots,x_n)$.

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

Next we present a random Gibbs sampler taken from [78] for estimating each $c_t=\mathbb{E}_{g^*_{t-1}}[I_{\{S(\boldsymbol{X})\geq m_{t-1}\}}],\ \ t=0,1,\ldots,T$ separately according to (3.10), that is,

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

Algorithm 3.2.2 (Ross' Acceptance-Rejection Algorithm for Estimating c_t)**.**

1. Set $N_t^{(e)}=N{=}0$.
2. Choose a vector $\boldsymbol{x}$ such that $S(\boldsymbol{x})\geq m_{t-1}$.
3. Generate a random number $U\sim U(0,1)$ and set $I{=}\mathrm{Int}(nU)+1$.
4. If $I=k$, generate Y_k from the conditional one-dimensional distribution $g(x_k|\boldsymbol{x}_{-k})$ (see Algorithm 3.2.1).
5. If $S(\widetilde{X}_1,\ldots,\widetilde{X}_{k-1},Y_k,X_{k+1},\ldots,X_n)<m_{t-1}$, return to 4.
6. Set $N=N+1$ and $Y_k=\widetilde{X}_k$.

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

8. Go to 3.

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

Note that Algorithm 3.2.2 (see Step 5) is based on the acceptance-rejection method. For many rare-event and counting problems, generation from the conditional pdf $g(x_i|\boldsymbol{x}_{-i})$ can be done directly, that is, skipping step 5 in it. This should clearly result in a speed-up.

Example 3.2.1. Sum of Independent Random Variables

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

$$\ell = \mathbb{E}_f \left[I_{\{\sum_{i=1}^{n} X_i \geq m\}} \right] \ . \tag{3.17}$$

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

$$g^*(x_i, m|\boldsymbol{x}_{-i}) = \propto f_i(x_i) I_{\{x_i \geq m - \sum_{j \neq i} x_j\}} \ , \tag{3.18}$$

where $\propto$ means proportional to.

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

Generating a Bernoulli random variable $\widetilde{X}_i$ from (3.18) with the Gibbs sampler can be done as follows. Generate $Y \sim$ Ber (p). If $I_{\{Y \geq m - \sum_{j \neq i} X_j\}}$, is unity, then set $\widetilde{X}_i = Y$, otherwise set $\widetilde{X}_i = 1 - Y$.

Example 3.2.2. Assume that all $X_i \sim f_i(x_i)$ are iid, and each $f_i(x_i)$ is a uniform d-point discrete pdf with mass equal to $1/d$ at points $1, 2, \ldots, d$, that is

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

We first apply for this example the original Algorithm 3.2.2, that is using acceptance-rejection, and then show how one can dispense with it.

Procedure 1: Acceptance-Rejection In this case, given a fixed point $\boldsymbol{X} = (X_1, \ldots, X_n)$, generating $\widetilde{X}_i$ from $g(x_i, m|\boldsymbol{x}_{-i})$ (see step 5 of Algorithm 3.2.2) can be done as follows:

Generate $Y \sim f_i(x_i)$. If $Y \geq m - \sum_{j \neq i} X_j$, then accept Y, that is set $\widetilde{X}_i = Y$, otherwise reject Y and try again.

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

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

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

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

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

1. *Generating* $\widetilde{X}_1$. Generate $Y \sim f_1(x_1)$. Let $Y = 2$. Check if $Y \geq m - \sum_{j \neq 1} X_j$ holds. We have $2 \geq 8 - 6 = 2$. This is true, so we set $\widetilde{X}_1 = Y = 2$.

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

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

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

Example 3.2.3. Sum of Independent Random Variables: Example 3.2.1 Continued

We now modify the above **Procedure 1** such that all Gibbs samples $\widetilde{\boldsymbol{X}} = (\widetilde{X}_1, \dots, \widetilde{X}_n)$ are accepted. The modified procedure takes into account availability of the quantity $m - \sum_{j \neq i} X_j$ and the fact that Y is a truncated version of X_i with the truncation point $m - \sum_{j \neq i} X_j$.

Define

$$r_i = \begin{cases} m - \sum_{j \neq i} X_j, & \text{if } m - \sum_{j \neq i} X_j \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.19}$$

Once $r_i, (r_i \geq 0)$ is available, we sample a point $Z \sim \mathcal{U}(\frac{1}{d - r_i + 1})$, instead of $Y \sim \mathcal{U}(\frac{1}{d})$. Recall that d is the number of different values taken by each random variable X_i. **Procedure 2: Without Acceptance-Rejection** *Generate Y from the truncated uniform distribution $\mathcal{U}(\frac{1}{d - r_i + 1})$, where r_i is an online parameter defined in (3.19).*

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

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

2. *Generating $\widetilde{X}_2$ without rejection.* Find $r_2 = m - \sum_{j \neq i} X_j$. We have $r_2 = 8 - 4 = 4$. So, the truncated distribution is uniform on the points $(4, 5, 6)$. Generate Y uniformly on these points. Let the result be $Y = 6$. Set $\widetilde{X}_2 = Y = 6$.

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

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

1. *Generating $\widetilde{X}_1$ without rejection.* Find $r_1 = m - \sum_{j \neq i} X_j$. We have $r_1 = 8 - 6 = 2$. So the truncated distribution is uniform over the points (2, 3, 4, 5, 6). Generate Y uniformly over the points (2, 3, 4, 5, 6). Let the outcome be $Y = 2$. Set $\widetilde{X}_1 = Y = 2$.

2. *Generating $\widetilde{X}_2$ without rejection.* Find $r_2 = m - \sum_{j \neq i} X_j$. We have $r_2 = 8 - 2 = 6$. So the truncated distribution is a degenerated one with the whole

mass at point 6 and we set automatically $Y = 6$. We deliver $\widetilde{X}_2 = Y = 6$. The generated point is therefore $(\widetilde{X}_1, \widetilde{X}_2) = (2, 6)$ with $S(\widetilde{X}_1, \widetilde{X}_2) = 8$.

Note that we deliberately made the results of Examples 3.2.1 and 3.2.3 identical.

Clearly, the above enhancement can be used for more complex models as in Example 3.1.1 for SAT and also for continuous pdfs. Our numerical results show that it can be typically achieve a speed-up by a factor of 2-3 as compared with the acceptance-rejection approach.

Example 3.2.4. Counting on the set of an integer program: Example 3.1.1 continued

Consider the set (3.1). It is readily seen (see also [82]) that in order to count on this set with given matrix $\boldsymbol{A} = \{a_{ij}\}$, one only needs to sample from the one-dimensional conditional pdfs

$$g^*(x_i, \widehat{m}_{t-1} | \boldsymbol{x}_{-i}) =\propto \mathcal{U}(1/d) I_{\{\sum_{r \in R_i} C_r(\boldsymbol{X}) \geq (\widehat{m}_{t-1} - c_{-i}) - \sum_{r \notin R_i} C_r(\boldsymbol{X})\}}\,, \tag{3.20}$$

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

Remark 3.2.1. The goal of the set R_i is to avoid calculation of every C_r. It is used mainly for speed-up, which can be significant for sparse matrices $\boldsymbol{A}$, where matrix calculations are performed in loops and when using low level programming languages, e.g. MatLab. The latter operates with matrices very fast and has its own inner optimizer.

Sampling a random variable $\widetilde{X}_i$ from (3.20) using the Gibbs sampler is simple. In particular for the Bernoulli case with $\boldsymbol{x} \in \{0, 1\}^n$ this can be done as follows. Generate $Y \sim$ Ber $(1/2)$. If

$$\sum_{r \in R_i} C_r(x_1, \ldots, x_{i-1}, Y, x_{i+1}, \ldots, x_n) \geq \widehat{m}_{t-1}, \tag{3.21}$$

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

3.3 Uniformity Results

Here we demonstrate empirically that

1. The original MCMC (without splitting) fails to generate points uniformly distributed on discrete sets of type (3.1). As mentioned, we shall demonstrate that not only does MCMC fail to generate uniform points on the set $\mathcal{X}^*$, but typically it samples only on some subset of $\mathcal{X}^*$ instead of the entire one.

2. The splitting Algorithm 3.5.2 handles successfully the uniformity problem in the sense that it generates uniform points on the set $\mathcal{X}^*$.

We consider both issues separately.

3.3.1 MCMC without Splitting

Our first model is the random 3-SAT model with the instance matrix ($A = 20 \times 80$) adapted from [82]. Table 3.1 presents the performance of the splitting Algorithm 3.5.1 based on 10 independent runs for the 3-SAT problem with with $N = 1,000$, rarity parameter $\rho = 0.1$ and burn-in parameter $b = 1$.

Here we use the following notations:

1. $N_t^{(e)}$ and $N_t^{(s)}$ denote the actual number of elites and the one after screening, respectively.

2. m_t^* and m_{*t} denote the upper and the lower elite levels reached, respectively.

3. $\rho_t = N_t^{(e)}/N$ denotes the adaptive proposal rarity parameter.

4. $\widehat{|\mathcal{X}^*|}$ and $\widehat{|\mathcal{X}^*_{dir}|}$ denote the product and what is called the *direct* estimator. The latter counts all distinct values of $\boldsymbol{X}_i,\ i = 1, \dots, N$ satisfying $S(\boldsymbol{X}_i) \geq m$, that is it can be written as

$$\widehat{|\mathcal{X}^*_{dir}|} = \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i^{(d)}) \geq m\}}, \tag{3.22}$$

where $\boldsymbol{X}_i^{(d)} = \boldsymbol{X}_i$, if $\boldsymbol{X}_i \neq \boldsymbol{X}_j,\ \forall j = 1, \dots, i-1$ and $\boldsymbol{X}_i^{(d)} = 0$, otherwise. For more details on $\widehat{|\mathcal{X}^*_{dir}|}$ see [83].

Table 3.1: Performance of splitting Algorithm 3.5.1 for SAT problem with instance matrix $\boldsymbol{A} = (20 \times 80)$.

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

Table 3.2 presents the dynamics for one run of the splitting Algorithm 3.5.1 for the same model.

Table 3.2: Dynamics of Algorithm 3.5.1 for SAT 20×80 model.

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

To demonstrate that the original Gibbs sampler (without splitting) fails to allocate all 15 valid SAT assignments as per Table 3.2, we took one of these 15 points and ran it for a very long time (allowing 1,000,000 Gibbs steps). We found that depending on the initial point $\boldsymbol{X} \in \mathcal{X}^*$ Gibbs sampler converges either to 6 or 9, that is it is able to find only 6 or 9 points out of the 15.

It is interesting to note that a similar phenomenon occurs with the splitting Algorithm 3.5.2 if instead of keeping all 15 elites $N_T^{(s)}$ for $m_T = m = 80$, we leave only one of them and then proceed with the Gibbs sampler for a long time. Clearly that setting $N_t^{(s)} = 1$ is exactly the same as dispensing with the splitting, that is staying with the original Gibbs sampler.

A similar phenomenon was observed with some other SAT models: starting Gibbs from a *single* point $\boldsymbol{X} \in \mathcal{X}^*$ it was able to generate points inside *only* of some subsets of $\mathcal{X}^*$, rather than in the entire $\mathcal{X}^*$. In other words, Gibbs sampler was stacked at some local minima.

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

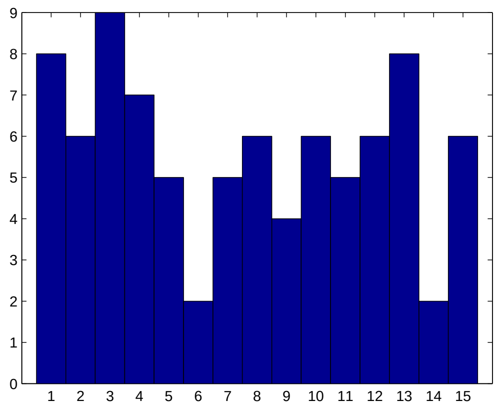
3.3.2 Uniformity of the Splitting Method

To get uniform samples on $\mathcal{X}^*$ we modify Algorithm 3.5.2 as follows. Once it reaches the desired level $m_T = m$ we perform several more steps (burn-in periods) with the corresponding elite samples. That is, we use the screening and splitting (cloning) steps exactly as we did for $m_t < m$. This will clearly only improve the uniformity of the samples of the desired set $\mathcal{X}^*$.

Observe also that the number of additional steps k needed for the resulting sample to pass the χ^2 test depends on the quality of the original elite sample at level m, which in turn depends on the values of ρ and b set in Algorithm 3.5.2. We found numerically that the closer ρ is to 1 the more uniform is the sample. But, clearly running the splitting Algorithm 3.5.2 at ρ close to 1 is time consuming. Thus, there exists a trade-off between the quality (uniformity) of the sample and the number of additional iterations k required.

Consider again the 3-SAT problem with the instance matrix $A = (20 \times 80)$ and $|\mathcal{X}^*| = 15$ (see Table 3.1). Figure 3.1 presents the histogram for $k = 0$, $N = 100$, $\rho = 0.05$ and $b = 1$. We found that the corresponding sample passes the χ^2 test with $\chi^2 = 12.8333$. With $\rho > 0.05$ instead of $\rho = 0.05$, and with $k > 0$ instead of $k = 0$ we found that χ^2 was even smaller as expected. In particular for for $\rho = 0.5$ and $k = 1$, we found that $\chi^2 = 9.7647$ and $\chi^2 = 10.3524$, respectively.

Figure 3.1: Histogram for the 3-SAT problem with the instance matrix $A = (20 \times 80)$ for $b = 1, k = 0,\ N = 100$ and $\rho = 0.05$.



Note again that using a single elite $\boldsymbol{X} \in \mathcal{X}^*$, that is using one of the 15 points (valid SAT assignments), the Gibbs sampler was unable to find all of them. More precisely, depending on the initial value of the point $\boldsymbol{X} \in \mathcal{X}^*$, Algorithm 3.5.2 was stacked at a local extremum, delivering as an estimator of $|\mathcal{X}^*|$ either 6 or 9 instead of the true value $|\mathcal{X}^*| = 15$.

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

3.4 Conclusion and Further Research

In this paper we showed empirically that

1. In spite of the common consensus on MCMC as n universal tool for generating samples on complex sets, we show that this is not the case when one needs to generate points uniformly distributed on discrete sets, such as on that defined in (3.1), that is one containing the constraints of integer programming. We have demonstrated empirically that not only does the original MCMC fail to generate uniform points on the set $\mathcal{X}^*$, but typically it generates points only at some subset of $\mathcal{X}^*$ instead on the entire set $\mathcal{X}^*$.

2. In contrast to the classic MCMC, the splitting Algorithm 3.5.2, which represents a combination of MCMC with a specially designed splitting

mechanism handles efficiently the uniformity problem in the sense that under some mild requirements the generated samples pass the χ^2 test.

Further Research

We intend to present rigorous statistical treatment of the splitting Algorithm 3.5.2, and in particular find the associated parameters N, ρ, b and η ensuring that generated samples are uniformly distributed on different discrete sets $\mathcal{X}^*$.

3.5 Appendix: Splitting Algorithms

Below, following [83], we present the two versions of the splitting algorithm: the so-called *basic* version and the *enhanced* version, bearing in mind Example 3.1.1.

3.5.1 Basic Splitting Algorithm

Let N, ρ_t and N_t be the fixed sample size, the adaptive rarity parameter and the number of elite samples at iteration t, respectively (see [83] for details). Recall that the elite sample $\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_t}$ corresponds to the largest subset of the population $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_t}\}$, for which $S(\boldsymbol{X}_i) \geq \widehat{m}_t$, that is $\widehat{m}_t$ is the $(1-\rho_t)$ sample quantile of of the ordered statistics values of $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$. It follows that the number of elites $N_t = \lceil N\rho_t \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the largest integer.

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

Algorithm 3.5.1 (Basic Splitting Algorithm for Counting)**.** Given the initial parameter ρ_0, say $\rho_0 \in (0.01, 0.25)$ and the sample size N, say $N = nm$, execute the following steps:

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

$$\widehat{c}_0 = \widehat{\ell}(\widehat{m}_0) = \frac{1}{N} \sum_{i=1}^{N} I_{\{S(\boldsymbol{X}_i) \geq \widehat{m}_0\}} = \frac{N_0}{N} \tag{3.23}$$

as an *unbiased* estimator of c_0. Note that $\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_0} \sim g^*(\boldsymbol{x}, \widehat{m}_0)$, where $g^*(\boldsymbol{x}, \widehat{m}_0)$ is a *uniform distribution* on the set $\mathcal{X}_1 = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_0\}$.

2. **Splitting** Let $\widehat{\mathcal{X}}_{t-1} = \{\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ be the elite sample at iteration $(t-1)$, that is the subset of the population $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_N\}$ for which $S(\boldsymbol{X}_i) \geq \widehat{m}_{t-1}$. Reproduce $\eta_{t-1} = \lceil \rho_{t-1}^{-1} \rceil$ times each vector $\widehat{\boldsymbol{X}}_k = (\widehat{X}_{1k}, \dots, \widehat{X}_{nk})$ of the elite sample $\{\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$, that is take η_{t-1} identical copies of each vector $\widehat{\boldsymbol{X}}_k$. Denote the entire new population ($\eta_{t-1} N_{t-1}$ cloned vectors plus the original elite sample $\{\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{cl} = \{(\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_1), \dots, (\widehat{\boldsymbol{X}}_{N_{t-1}}, \dots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each cloned vector of the population $\mathcal{X}_{cl}$ apply MCMC (and in particular the random Gibbs sampler) for a single period (single burn-in). Denote the *new entire* population by $\{\boldsymbol{X}_1, \dots, \boldsymbol{X}_N\}$. Note that each vector in the sample $\boldsymbol{X}_1, \dots, \boldsymbol{X}_N$ is distributed $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$, where $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$ has approximately a uniform distribution on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_{t-1}\}$.

3. **Estimating** c_t Take $\widehat{c}_t = \frac{N_t}{N}$ (see (3.10)) as an estimator of c_t in (3.8). Note again that each vector of $\widehat{\boldsymbol{X}}_1, \dots, \widehat{\boldsymbol{X}}_{N_t}$ of the elite sample is distributed $g^*(\boldsymbol{x}, \widehat{m}_t)$, where $g^*(\boldsymbol{x}, \widehat{m}_t)$ has approximately a uniform distribution on the set $\mathcal{X}_{t+1} = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_t\}$.

4. **Stopping rule** If $m_t = m$ go to step 5, otherwise set $t = t+1$ and repeat from step 2.

5. **Final Estimator** Deliver $\widehat{\ell}$ given in (3.9) as an estimator of ℓ and $|\widehat{\mathcal{X}}^*| = \widehat{\ell}|\mathcal{X}|$ as an estimator of $|\mathcal{X}^*|$.

Figure 3.2 presents a typical dynamics of the splitting algorithm, which terminates after two iterations. The set of points denoted $\star$ and $\bullet$ is associated with these two iterations. In particular the points marked $\star$ are uniformly distributed on the sets $\mathcal{X}_0$ and $\mathcal{X}_1$. (Those which are in $\mathcal{X}_1$ correspond to the elite samples). The points marked $\bullet$ are approximately uniformly distributed on the sets $\mathcal{X}_1$ and $\mathcal{X}_2$. (Those which are in $\mathcal{X}_2 = \mathcal{X}^*$ like wise correspond to the elite samples).

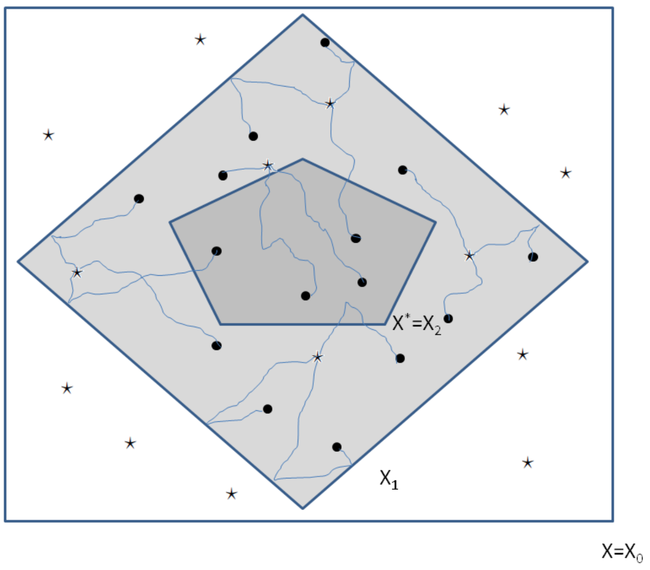


Figure 3.2: Dynamics of Algorithm 3.5.1

3.5.2 Enhanced Splitting Algorithm for Counting

The improved version of the basic splitting Algorithm 3.5.1, which contains (i) an enhanced splitting step instead of the original one as in Algorithms 3.5.1 and a (ii) new screening step.

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

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

$$\eta_{t-1} = \left\lceil \frac{N}{bN_{t-1}} \right\rceil - 1 = \left\lceil \frac{N_{cl}}{N_{t-1}} \right\rceil - 1. \tag{3.24}$$

Here $N_{cl} = N/b$ is called the *cloned sample size*, and as before $N_{t-1} = \rho_{t-1}N$ denotes the number of elites and ρ_{t-1} is the adaptive rarety parameter at iteration $t-1$ (see [86] for details).

As an example, let $N = 1,000$, $b = 10$. Consider two cases: $N_{t-1} = 21$ and $N_{t-1} = 121$. We obtain $\eta_{t-1} = 4$ and $\eta_{t-1} = 0$ (no splitting), respectively.

As an alternative to (3.24) one can use the following heuristic strategy in defining b and η: find b_{t-1} and η_{t-1} from $b_{t-1}\eta_{t-1} \approx \frac{N}{N_{t-1}}$ and take $b_{t-1} \approx \eta_{t-1}$. In short, one can take

$$b_{t-1} \approx \eta_{t-1} \approx \left(\frac{N}{N_{t-1}}\right)^{1/2}. \tag{3.25}$$

Consider again the same two cases for N_{t-1} and N We have $b_{t-1} \approx \eta_{t-1} = 7$ and $b_{t-1} \approx \eta_{t-1} = 3$, respectively. We found numerically that both versions work well, but unless stated otherwise we shall use (3.25).

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

Algorithm 3.5.2 (Enhanced Splitting Algorithm for Counting)**.** Given the parameter ρ, say $\rho \in (0.01, 0.25)$ and the sample size N, say $N = nm$, execute the following steps:

1. **Acceptance-Rejection** - same as in Algorithm 3.5.1.

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

3. **Splitting (Cloning)** Given the size N_{t-1} of the screened elites $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$ at iteration $(t-1)$, find the splitting and the burn-in parameters η_{t-1} and b_{t-1} according to (3.25). Reproduce η_{t-1} times each vector $\widehat{\boldsymbol{X}}_k = (\widehat{X}_{1k}, \ldots, \widehat{X}_{nk})$ of the screened elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$, that is, take η_{t-1} identical copies of each vector $\widehat{\boldsymbol{X}}_k$ obtained at the $(t-1)$-th iteration. Denote the entire new population ($\eta_{t-1}N_{t-1}$ cloned vectors plus the original screened elite sample $\{\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}}\}$) by $\mathcal{X}_{cl} = \{(\widehat{\boldsymbol{X}}_1, \ldots, \widehat{\boldsymbol{X}}_1), \ldots, (\widehat{\boldsymbol{X}}_{N_{t-1}}, \ldots, \widehat{\boldsymbol{X}}_{N_{t-1}})\}$. To each of the cloned vectors of the population $\mathcal{X}_{cl}$ apply the the Gibbs sampler for b_{t-1} burn-in periods. Denote the *new entire* population by $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$. Note that each vector in the sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ is distributed approximately $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$, where $g^*(\boldsymbol{x}, \widehat{m}_{t-1})$ is a uniform distribution on the set $\mathcal{X}_t = \{\boldsymbol{x} : S(\boldsymbol{x}) \geq \widehat{m}_{t-1}\}$.

4. **Estimating** c_t - same as in Algorithm 3.5.1.

5. **Stopping rule** - same as in Algorithm 3.5.1.

6. **Final estimator** - same as in Algorithm 3.5.1.

Chapter 4

On the Use of Smoothing to Improve the Performance of the Splitting Method

Frédéric Cérou[b], Arnaud Guyader[b,c], Reuven Rubinstein[a,1] and Radislav Vaisman[a]

[a] Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology, Haifa, Israel
ierrr01@ie.technion.ac.il,slvaisman@gmail.com

[b] INRIA Rennes Bretagne Atlantique
Aspi project-team
Campus de Beaulieu, 35042 Rennes Cedex, France
Frederic.Cerou@irisa.fr

[c] Université Rennes II – Haute Bretagne
Campus Villejean
Place du Recteur Henri Le Moal, CS 24307
35043 Rennes Cedex, France
arnaud.guyader@uhb.fr

[1]Corresponding author (http://iew3.technion.ac.il/Home/Users/ierrr01.phtml).

[1†] The research of Reuven Rubinstein was supported by the BSF (Binational Science Foundation, grant No 2008482).

Abstract

We present an enhanced version of the splitting method, called the *smoothed splitting method* (SSM), for counting associated with complex sets, such as the set defined by the constraints of an integer program and in particular for counting the number of satisfiability assignments. Like the conventional splitting algorithms, ours uses a sequential sampling plan to decompose a “difficult” problem into a sequence of “easy” ones. The main difference between SSM and splitting is that it works with an auxiliary sequence of continuous sets instead of the original discrete ones. The rationale of doing so is that continuous sets are easier to handle. We show that while the proposed method and its standard splitting counterpart are similar in their CPU time and variability, the former is more robust and more flexible than the latter.

Keywords. Combinatorial Optimization, Rare Event, Counting, Splitting.
Mathematical Subject Classification. Primary 65C05, 65C35; Secondary 68W20, 60C05.

4.1 Introduction: The Splitting Method

The goal of this work is to propose a novel and original way, called the *smoothed splitting method* (SSM), for counting on discrete sets associated with NP-hard discrete combinatorial problems and in particular counting the number of satisfiability assignments. The main idea of the SSM is to transform a combinatorial counting problem into a continuous integration problem using a type of "smoothing" of discrete indicator functions. Then we are in a position to apply a quite standard Sequential Monte Carlo/splitting method to this continuous integration problem. We show that although numerically the proposed method performs similar to the standard splitting one [82, 83] (in terms of CPU time and accuracy), the former one is more robust than the latter. In particular, tuning the parameters in SSM is simpler than in its standard splitting counterpart.

Before proceeding with SSM we present the splitting method for counting, following [82, 83]. For relevant references on the splitting method see [7], [12], [11], [28], [31], [55], [64], [58], 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 monograph [68].

The main idea of the splitting method for counting is to design a sequential sampling plan, with a view of decomposing a "difficult" counting problem defined on some set $\mathcal{X}^*$ into a number of "easy" ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \dots, \mathcal{X}_T$ and such that $\mathcal{X}_T = \mathcal{X}^*$. Similar to *randomized algorithms* [67], [69] splitting algorithms explore the connection between counting and sampling problems and in particular the reduction from approximate counting of a discrete set to approximate sampling of elements of this set, where the sampling is performed by the classic MCMC method [86]. Very recently, [8] discusses several splitting variants in a very similar setting, including a discussion on an empirical estimate of the variance of the rare event probability estimate.

A typical splitting algorithm comprises the following steps:

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

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

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

$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{T} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = |\mathcal{X}_0|\ell, \tag{4.1}$$

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

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \tag{4.2}$$

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

4. Develop an efficient estimator $\widehat{c}_t$ for each c_t and estimate $|\mathcal{X}^*|$ by

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}_0|\,\widehat{\ell} = |\mathcal{X}_0| \prod_{t=1}^{T} \widehat{c}_t, \tag{4.3}$$

   where $\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t$ is an estimator of $\ell = \prod_{t=1}^{T} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}$.

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

(i) Put the well known NP-hard counting problems into the framework (4.1) by making sure that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_T = \mathcal{X}^*$ and each c_t is not a rare-event probability.

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

In Section 4.2, we briefly recall the SAT problem, which we will focus on in order to present our new method. In Section 4.3, which is our main one, we show how to resolve problems (i) and (ii) for the SAT problem by using the smoothed splitting method (SSM), which presents an enhanced version of the splitting method [82, 83]. Section 4.4 is devoted to the theoretical analysis of SSM in an idealized version, which we call i.i.d. SSM. In Section 4.5 numerical results for both the SSM and splitting algorithm are presented. Their efficiencies are compared for several SAT instances.

4.2 Presentation of the SAT problem

The most common SAT problem comprises the following two components:

- A set of n Boolean variables $\{x_1, \ldots, x_n\}$, representing statements that can either be `TRUE` (=1) or `FALSE` (=0). The negation (the logical `NOT`) of a variable x is denoted by $\overline{x}$. For example, $\overline{\texttt{TRUE}} = \texttt{FALSE}$. A variable or its negation is called a *literal.*

- A set of m distinct *clauses* $\{S_1, S_2, \ldots, S_m\}$ of the form $S_j = z_{j_1} \vee z_{j_2} \vee \cdots \vee z_{j_q}$, where the z's are literals and the $\vee$ denotes the logical `OR` operator. For example, $0 \vee 1 = 1$.

The binary vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ 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 $\overline{x}_i$, for each $i = 1, \ldots, n$. The simplest SAT problem can now be formulated as: find a truth assignment $\boldsymbol{x}$ such that *all* clauses are true.

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

$$F = S_1 \wedge S_2 \wedge \cdots \wedge S_m,$$

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

The problem of deciding whether there *exists* a valid assignment, and, indeed, providing such a vector, is called the *SAT-assignment* problem.

Toy Example Let us consider the following toy SAT problem with two clauses and two variables: $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$. It is straightforward by considering all the four possible assignments, that this formula is satisfiable, with two valid assignments $x_1 = 1, x_2 = 0$ and $x_1 = 0, x_2 = 1$. If now we consider the three clauses formula $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_2)$, then it is clearly unsatisfiable.

It is shown in [86] that the SAT-assignment problem can be modeled via rare-events with ℓ given by

$$\ell = \mathbb{E}\left[\mathbb{1}_{\left\{\sum_{j=1}^{m} C_j(\boldsymbol{X}) = m\right\}}\right], \tag{4.4}$$

where $\boldsymbol{X}$ has a "uniform" distribution on the finite set $\{0,1\}^n$. It is important to note that here each $C_j(\boldsymbol{x}) = \mathbb{1}_{\{\sum_{k=1}^{n} a_{jk} x_k \geq b_j\}}$ can be also written alternatively as

$$C_j(\boldsymbol{x}) = \max_k \{0,\ (2\, x_k - 1)\, a_{jk}\}.$$

Here $C_j(\boldsymbol{x}) = 1$ if clause S_j is `TRUE` with truth assignment $\boldsymbol{x}$ and $C_j(\boldsymbol{x}) = 0$ if it is `FALSE`, $A = (a_{jk})$ is a given clause matrix that indicates if the literal corresponds to the variable (+1) , its negation (-1), or that neither appears in the clause (0). If for example $x_k = 0$ and $a_{jk} = -1$, then the literal $\overline{x}_j$ is `TRUE`. The entire clause is `TRUE` if it contains at least one true literal. In other words, ℓ in (4.4) is the probability that a uniformly generated SAT assignment (trajectory) $\boldsymbol{X}$ is valid, that is, all clauses are satisfied, that is

$$S(x) = \min_{1 \le j \le m} C_j(x) \ge 1,$$

which is typically very small.

4.3 Smoothed Splitting Method

Before presenting the SSM algorithm we shall discuss its main features having in mind a SAT problem.

To proceed, recall that the main idea of SSM is to work within a continuous space rather than a discrete one. As a result this involves a continuous random vector $\boldsymbol{Y}$ instead of the discrete random vector $\boldsymbol{X}$ with i.i.d. components with law Ber($p = 1/2$). For example for a SAT problem one needs to adopt the following steps:

1. Choose a random vector $\boldsymbol{Y}$ of the same size as $\boldsymbol{X}$, such that the components $Y_1, \ldots, Y_n$, are i.i.d. uniformly distributed on the interval $(0, 1)$. Clearly the Bernoulli components $X_1, \ldots, X_n$ can be written as $X_1 = \mathbb{1}_{\{Y_1 > 1/2\}}, \ldots, X_n = \mathbb{1}_{\{Y_n > 1/2\}}$.

2. Instead of the former $0 - 1$ variables x or $\bar{x}$ we will use for each clause a family of functions from $(0, 1)$ to $(0, 1)$. In particular, for each occurrence of x or $\bar{x}$, we consider two functions, say $g_\varepsilon(y)$ and $h_\varepsilon(y) = g_\varepsilon(1-y)$ indexed by $\varepsilon \ge 0$. These functions need to be increasing in ε, which means that

$$0 < \varepsilon \le \varepsilon' \;\Rightarrow\; g_\varepsilon(y) \le g_{\varepsilon'}(y), \; \forall y \in (0, 1). \tag{4.5}$$

and for $\varepsilon = 0$, $g_0(y) = \mathbb{1}_{\{y > 1/2\}}$, $h_0(y) = g_0(1 - y) = \mathbb{1}_{\{y \le 1/2\}}$. Possible choices of $g_\varepsilon(y)$ are:

$$g_\varepsilon(y) = (2y)^{1/\varepsilon} \mathbb{1}_{\{0 < y < \frac{1}{2}\}} + \mathbb{1}_{\{y > \frac{1}{2}\}} \tag{4.6}$$

or

$$g_\varepsilon(y) = \mathbb{1}_{\{\frac{1}{2} - \varepsilon < y < \frac{1}{2}\}} \left(\frac{y}{\varepsilon} + 1 - \frac{1}{2\varepsilon} \right) + \mathbb{1}_{\{y > \frac{1}{2}\}}. \tag{4.7}$$

or

$$g_\varepsilon(y) = I_{[1/2 - \varepsilon, 1]}(y). \tag{4.8}$$

3. For each clause C_j, we consider the approximate ε-clause $C_{j\varepsilon}$, where we replace x by $g_\varepsilon(y)$, $\bar{x}$ by $h_\varepsilon(y)$, and $\vee$ by $+$. Note also that the statement "C_j is true" is replaced in the new notations by $C_{j\varepsilon} \geq 1$.

4. **Nested sets**. For each $\varepsilon \geq 0$, consider the subset (or event) B_ε of $(0,1)^n$ defined as

   $$B_\varepsilon = \{\mathbf{y} \in (0,1)^n : \forall j \in \{1,\dots,m\}, C_{j\varepsilon}(\mathbf{y}) \geq 1\} = \{\mathbf{y} \in (0,1)^n : S_\varepsilon(\mathbf{y}) \geq 1\},$$

   where $S_\varepsilon(\mathbf{y}) = \min_{1 \leq j \leq m} C_{j\varepsilon}(\mathbf{y})$. Then it is clear from the above that for $\varepsilon_1 \geq \varepsilon_2 \geq 0$, we have the inclusions $B_0 \subset B_{\varepsilon_2} \subset B_{\varepsilon_1}$. Note that B_0 is the event for which *all* the *original* clauses are satisfied and B_ε is an event on which *all* the *approximate* ε-clauses are satisfied. Note also that $\varepsilon_t,\ t = 1,\dots,T$, should be a decreasing sequence, with T being the number of nested sets, and $\varepsilon_T = 0$. In our SSM algorithm below (see section 4.3.2), we shall choose the sequence $\varepsilon_t,\ t = 1,\dots,T$, adaptively, similar as the sequence $m_t,\ t = 1,\dots,T$, is chosen in the Basic Splitting Algorithm of [83].

4.3.1 The SSM Algorithm with fixed nested subsets

Below we outline the main steps of the SSM algorithm.

1. **Initialization** Generate N i.i.d. samples $\boldsymbol{Y}_1^1,\dots,\boldsymbol{Y}_N^1$ of distribution $\mathcal{U}((0,1)^n)$.

2. **Selection** Keep only those samples for which all the ε_1-clauses (constructed with g_{ε_1} and h_{ε_1}) are satisfied. Reindex them $1,\dots,N_1$. Set $\widehat{p}_1 = N_1/N$.

3. **Cloning** Draw $N - N_1$ clones from the previous sample (with equal probabilities). Together with it we have again a sample of size N.

4. **Mutation** For all $N - N_1$ new samples apply the Gibbs sampler (see subsection 4.3.3 below) one or several times.

5. **Selection/Cloning/Mutation** for $\varepsilon_2,\dots,\varepsilon_T$. This yields the estimates $\widehat{p}_2,\dots,\widehat{p}_{T-1}$.

6. **Final Estimator and solutions to the original SAT problem** Select the samples that satisfy all the original clauses. Let N_T be their number. Estimate $\widehat{p}_T = N_T/N$. From this last sample, construct a discrete sample $X_1,\dots,X_{N_T}$ by $X_{j,k} = I_{\{Y_{j,k}>1/2\}}$, $1 \leq k \leq n$, which is not independent,

but identically distributed on the instances of $\boldsymbol{x}$ that satisfy all the original clauses. An estimate of ℓ is given by $\widehat{\ell} = \prod_{t=1}^{T} \widehat{p}_t$, so that an estimate of $|\mathcal{X}^*|$ is given by $2^n \widehat{\ell} = 2^n \prod_{t=1}^{T} \widehat{p}_t$.

A crucial issue in this algorithm is to choose the successive levels $\varepsilon_1, \varepsilon_2$, etc., so that the variance of the estimator $\widehat{\ell}$ is as small as possible. The following subsection explains how to do it adaptively.

4.3.2 The SSM Algorithm with adaptive nested subsets

Say that we implemented the algorithm up to iteration t, and want to choose ε_{t+1}. Let $\boldsymbol{Y}_1^t, \ldots, \boldsymbol{Y}_N^t$ the current sample satisfying all the ε_t-clauses. Choose (as usual in adaptive rare-event simulation) a given rate of success ρ, with $0 < \rho < 1$. Then the appropriate choice for ε_{t+1} would be a value $\varepsilon > 0$ such that the number of replicas in the current sample $\boldsymbol{Y}_1^t, \ldots, \boldsymbol{Y}_N^t$ that satisfy all the ε-clauses is equal (close) to ρN. A simple way of doing this is to perform a binary search in the interval $[0, \varepsilon_t]$ bearing in mind that $\varepsilon_t \geq \varepsilon_{t+1}$.

The following algorithm summarizes the above.

Algorithm 4.3.1. [Adaptive Choice of ε_{t+1}] Given the parameters ρ and ε_t proceed as follows:

1. Set $\varepsilon_{low} = 0$, $\varepsilon_{high} = \varepsilon_t$ and $\varepsilon_{t+1} = \frac{\varepsilon_{high}}{2}$.
2. While the proportion of replicas in the current sample $\boldsymbol{Y}_1^t, \ldots, \boldsymbol{Y}_N^t$ that satisfy all ε_{t+1}-clauses is not close to ρ, do the following:
   (a) Calculate the ε_{t+1} performance $S_{\varepsilon_{t+1}}(\boldsymbol{Y})$ of the trajectories conveniently defined as the minimum over all $C_{\varepsilon_{t+1}}(\boldsymbol{Y})$ corresponding to the trajectory $\boldsymbol{Y}$. [Recall that by saying that $\boldsymbol{Y}$ is a satisfying trajectory, we mean that $S_{\varepsilon_{t+1}}(\boldsymbol{Y}) \geq 1$].
   (b) If the number of ε_{t+1} satisfying trajectories is larger than ρN set $\varepsilon_{high} = \varepsilon_{t+1}$.
   (c) If the number of ε_{t+1} satisfying trajectories is smaller than ρN set $\varepsilon_{low} = \varepsilon_{t+1}$.
   (d) Set $\varepsilon_{t+1} = \frac{\varepsilon_{low} + \varepsilon_{high}}{2}$.
3. Deliver ε_{t+1} as the new adaptive level.

We are now in a position to describe the adaptive smoothed splitting algorithm, which is the one that will be used in the simulations.

Algorithm 4.3.2. [SSM Algorithm for Counting]

Fix the parameter ρ, say $\rho \in (0.01, 0.5)$ and the sample size N such that $N_e = \rho N$ is an integer which denotes the size of the elite sample at each step. Choose also the function $g_\varepsilon(y)$, say the one given in (4.8), and ε_0 accordingly (e.g. $\varepsilon_0 = 1/2$ for (4.8)). Then execute the following steps:

1. **Acceptance-Rejection** Set a counter $t = 1$. Generate an i.i.d. sample $\boldsymbol{Y}_1^1, \dots, \boldsymbol{Y}_N^1$ each uniformly on $(0,1)^n$. Obtain the first $\widehat{\varepsilon}_1$ using Algorithm 4.3.1 and let $\widehat{\mathcal{Y}}^1 = \{\widehat{\boldsymbol{Y}}_1^1, \dots, \widehat{\boldsymbol{Y}}_{N_e}^1\}$ be the elite sample. Note that $\widehat{\boldsymbol{Y}}_1^1, \dots, \widehat{\boldsymbol{Y}}_{N_e}^1 \sim \mathcal{U}(B_{\widehat{\varepsilon}_1})$, the uniform distribution on $B_{\widehat{\varepsilon}_1}$.

2. **Splitting (Cloning)** Given the elite sample $\{\widehat{\boldsymbol{Y}}_1^t, \dots, \widehat{\boldsymbol{Y}}_{N_e}^t\}$ at iteration t, reproduce ρ^{-1} times each vector $\widehat{\boldsymbol{Y}}_i^t$. Denote the entire new population by
$$\mathcal{Y}_{cl} = \{(\widehat{\boldsymbol{Y}}_1^t, \dots, \widehat{\boldsymbol{Y}}_1^t), \dots, (\widehat{\boldsymbol{Y}}_{N_e}^t, \dots, \widehat{\boldsymbol{Y}}_{N_e}^t)\}.$$
To each of the cloned vectors of the population $\mathcal{Y}_{cl}$ apply the MCMC (and in particular the Gibbs sampler Algorithm 4.3.3) for b_t burn-in periods. Denote the *new entire* population by $\{\boldsymbol{Y}_1^{t+1}, \dots, \boldsymbol{Y}_N^{t+1}\}$. Note that each vector in the sample $\boldsymbol{Y}_1^{t+1}, \dots, \boldsymbol{Y}_N^{t+1}$ is distributed uniformly in $B_{\widehat{\varepsilon}_t}$.

3. **Adaptive choice** Obtain $\widehat{\varepsilon}_{t+1}$ using Algorithm 4.3.1. Note again that each vector of $\widehat{\boldsymbol{Y}}_1^{t+1}, \dots, \widehat{\boldsymbol{Y}}_{N_e}^{t+1}$ of the elite sample is distributed uniformly in $B_{\widehat{\varepsilon}_{t+1}}$.

4. **Stopping rule** If $\widehat{\varepsilon}_{t+1} = 0$ go to step 5, otherwise set $t = t + 1$ and repeat from step 2.

5. **Final Estimator** Denote $\widehat{T} + 1$ the current counter, and
$$\widehat{r} = \frac{|\{i \in \{1, \dots, N\} : S_0(\boldsymbol{Y}_i^{\widehat{T}+1}) \geq 1\}|}{N} > \rho,$$
and deliver $\widehat{\ell} = \widehat{r} \times \rho^{\widehat{T}}$ as an estimator of ℓ and $|\widehat{\mathcal{X}}^*| = 2^n\, \widehat{\ell}$ as an estimator of $|\mathcal{X}^*|$.

Remark: Differences between Basic Splitting and SSM Algorithms

1. SSM Algorithm 4.3.2 operates on a continuous space, namely $(0,1)^n$, while the Basic Splitting Algorithm of [83] operates on a discrete one, namely $\{0,1\}^n$. As a consequence their MCMC (Gibbs) samplers are different.

2. In the discrete case the performance function $S(\boldsymbol{X})$ represents the number of satisfied clauses, while in the continuous one it depends on both ε and the g_ε. It is crucial to note that in the discrete case all clauses are satisfied *at the last iteration only* while in the continuous case *each clause is ε_t-satisfied at each iteration t.*

3. The stopping rules in both algorithms are the same. In particular, at the last iteration the SSM Algorithm 4.3.2 transforms its vectors from the continuous space to the discrete one.

4.3.3 Gibbs Sampler

Starting from $\boldsymbol{Y} = (Y_1, \ldots, Y_n)$, which is uniformly distributed on

$$B_\varepsilon = \{\mathbf{y} \in (0,1)^n : \ \forall j \in \{1, \ldots, m\}, C_{j\varepsilon}(\mathbf{y}) \geq 1\} = \{\mathbf{y} \in (0,1)^n : \ S_\varepsilon(\mathbf{y}) \geq 1\},$$

a possible way to generate $\widetilde{\boldsymbol{Y}}$ with the same law as $\boldsymbol{Y}$ is to use the following general systematic Gibbs sampler:

Algorithm 4.3.3. [Systematic Gibbs Sampler]

1. Draw $\widetilde{Y}_1$ from the conditional pdf $g(y_1|y_2, \ldots, y_n)$.

2. Draw $\widetilde{Y}_k$ from the conditional pdf $g(y_k|\widetilde{y}_1, \ldots, \widetilde{y}_{k-1}, y_{k+1}, \ldots, y_n)$, $2 \leq k \leq n-1$.

3. Draw $\widetilde{Y}_n$ from the conditional pdf $g(y_n|\widetilde{y}_1, \ldots, \widetilde{y}_{n-1})$.

where g is the target distribution. In our case, g is the uniform distribution on B_ε, and the conditional distribution of the kth component given the others is simply the uniform distribution on some interval (r, R) given as explained below.

Toy Example Let us consider first a small example with four variables and two clauses: $(X_1 \vee X_2) \wedge (\bar{X}_2 \vee X_3 \vee \bar{X}_4)$. For a given $\varepsilon > 0$, this gives the two ε-clauses:

$$\begin{aligned} g_\varepsilon(Y_1) + g_\varepsilon(Y_2) &\geq 1 \\ h_\varepsilon(Y_2) + g_\varepsilon(Y_3) + h_\varepsilon(Y_4) &\geq 1. \end{aligned}$$

Let us say we want the distribution of Y_2 given Y_1, Y_3, Y_4. If we want the first one to be satisfied, we need $g_\varepsilon(Y_2) \geq 1 - g_\varepsilon(Y_1)$, that is $Y_2 \geq g_\varepsilon^{-1}(1 - g_\varepsilon(Y_1)) = r$. Similarly, the second clause gives $h_\varepsilon(Y_2) \geq 1 - g_\varepsilon(Y_3) - h_\varepsilon(Y_4)$, and because h_ε is decreasing, $Y_2 \leq h_\varepsilon^{-1}(1 - g_\varepsilon(Y_3) - h_\varepsilon(Y_4)) = 1 - g_\varepsilon^{-1}(1 - g_\varepsilon(Y_3) - h_\varepsilon(Y_4)) = R$. Thus the conditional distribution of Y_2 is uniform on the interval (r, R).

The generalization is straightforward, and is given below.

Algorithm 4.3.4. [Conditional sampling of $\widetilde{Y}_k$]

- Denote by $\mathcal{I}_k$ the set of ε-clauses $C_{j\varepsilon}$ in which $g_\varepsilon(Y_k)$ is involved.
- For all $j \in \mathcal{I}_k$, denote by $Z_1, \dots, Z_{q-1}$ the other $g_\varepsilon(Y_i)$'s or $h_\varepsilon(Y_i)$'s involved in clause $C_{j\varepsilon}$. Denote $r_j = g_\varepsilon^{-1}(1 - Z_1 - \cdots - Z_{q-1})$, and $r = \sup_{j \in \mathcal{I}_k} r_j$.
- Denote by $\mathcal{J}_k$ the set of ε-clauses $C_{j\varepsilon}$ in which $h_\varepsilon(Y_k) = g_\varepsilon(1 - Y_k)$ is involved.
- For all $j \in \mathcal{J}_k$, denote by $Z_1, \dots, Z_{q-1}$ the other $g_\varepsilon(Y_i)$'s or $h_\varepsilon(Y_i)$'s involved in clause $C_{j\varepsilon}$. Denote $R_j = 1 - g_\varepsilon^{-1}(1 - Z_1 - \cdots - Z_{q-1})$, and $R = \inf_{j \in \mathcal{I}_k} R_j$.
- Sample $\widetilde{Y}_k$ uniformly in the interval $[r, R]$.

Remark: It is readily seen that $r < R$ and $\widetilde{\boldsymbol{Y}} = (\widetilde{Y}_1, \dots, \widetilde{Y}_n)$ has the same distribution as $\boldsymbol{Y}$. This is so since the initial point $\boldsymbol{Y} = (Y_1, \dots, Y_n)$ belongs to and is uniformly distributed in B_ε. Note that our simulation results clearly indicate that one round of the Gibbs Algorithm 4.3.3 suffices for good experimental results. Nonetheless, if one wants the new vector $\widetilde{\boldsymbol{Y}}$ to be independent of its initial position $\boldsymbol{Y}$, then in theory the Gibbs sampler would have to be applied an infinite number of times. This is what we call the i.i.d. SSM in section 4.4, and this is the algorithm that we will analyze from a theoretical point of view.

4.4 Statistical Analysis of i.i.d. SSM

It is possible to obtain exact results about the estimator $\widehat{\ell}$ in an assumed situation (never encountered in practice) that each step begins with an N i.i.d. sample. We call this idealized version "the *i.i.d.* smoothed splitting algorithm" - *i.i.d. SSM.* This would typically correspond to the situation where at each step the Gibbs sampler is applied an infinite number of times, which is not realistic but will be our main hypothesis in Subsection 4.4.1. The following theoretical results do not exactly match the algorithm which is used in practice, but can be expected to provide insight.

4.4.1 Statistical Analysis of i.i.d. SSM

The aim of this subsection is to precise the statistical properties of the estimator $\widehat{\ell}$ obtained by the *i.i.d. SSM.*

Let us denote by s the number of solutions of the SAT problem at hand, and by $\mathcal{S}$ the union of s hypercubes (with edge length 1/2) which correspond to these solutions in the continuous version: this means that for all $\mathbf{y} = (y_1, \dots, y_n) \in (0,1)^n$, $\mathbf{y}$ belongs to $\mathcal{S}$ if and only if $\mathbf{x} = (I_{y_1 \geq 1/2}, \dots, I_{y_n \geq 1/2})$ is a solution of the SAT problem.

With these notations, the probability that we are trying to estimate is

$$\ell = \mathbb{P}(\mathbf{Y} \in \mathcal{S})$$

where $\mathbf{Y}$ is a uniform random vector in the hypercube $(0,1)^n$. Recall that for any $\varepsilon \geq 0$

$$B_\varepsilon = \{\mathbf{y} \in (0,1)^n : \ \forall j \in \{1, \dots, m\}, C_{j\varepsilon}(\mathbf{y}) \geq 1\} = \{\mathbf{y} \in (0,1)^n : \ S_j(\mathbf{y}) \geq \varepsilon\},$$

so that we have the following Bayes formula for the splitting algorithm

$$\ell = \mathbb{P}(B_0) = \mathbb{P}(B_0 | B_{\varepsilon_T}) \times \cdots \times \mathbb{P}(B_{\varepsilon_1} | B_{\varepsilon_0}),$$

where ε_0 is large enough (possibly infinite) so that $\mathbb{P}(B_{\varepsilon_0}) = 1$ (for example $\varepsilon_0 = 1/2$ when g_ε is defined by formula (4.8) and $\varepsilon_0 = +\infty$ when g_ε is defined by formula (4.6) or (4.7)).

Let us now describe briefly the smoothed splitting algorithm in this framework. As previously, ρ is the fixed proportion of the elite sample at each step. For simplicity, we will assume that ρN is an integer.

Starting with an N i.i.d. sample $(\mathbf{Y}_1^1, \dots, \mathbf{Y}_N^1)$, with $\mathbf{Y}_i^1$ uniformly distributed in $(0,1)^n$ for all $i \in \{1, \dots, N\}$, the first step consists in applying a binary search to find $\widehat{\varepsilon}_1$ such that

$$\frac{|\{i \in \{1, \dots, N\} : \mathbf{Y}_i^1 \in B_{\widehat{\varepsilon}_1}\}|}{N} = \rho.$$

Such an $\widehat{\varepsilon}_1$ is not unique, but this will not matter from the theoretical point of view, as will become clear in the proof of Theorem 4.4.1 below.

Knowing $\widehat{\varepsilon}_1$ and using a Gibbs sampler, the elite sample of size ρN allows ideally (which means: for the i.i.d. SSM) to draw an N i.i.d. sample $(\mathbf{Y}_1^2, \dots, \mathbf{Y}_N^2)$, with $\mathbf{Y}_i^2$ uniformly distributed in $B_{\widehat{\varepsilon}_1}$. Using a binary search , one can then find $\widehat{\varepsilon}_2$ such that

$$\frac{|\{i \in \{1, \dots, N\} : \mathbf{Y}_i^2 \in B_{\widehat{\varepsilon}_2}\}|}{N} = \rho,$$

and iterate the algorithm, with only the last step being different: the algorithm stops when for an N i.i.d. sample $(\mathbf{Y}_1^{\widehat{T}+1}, \dots, \mathbf{Y}_N^{\widehat{T}+1})$, with $\mathbf{Y}_i^{\widehat{T}+1}$ uniformly distributed in $B_{\widehat{\varepsilon}_{\widehat{T}}}$, the proportion of points which satisfy the SAT problem is larger than ρ:

$$\frac{|\{i \in \{1, \dots, N\} : \mathbf{Y}_i^{\widehat{T}+1} \in B_0\}|}{N} = \widehat{r} > \rho.$$

In summary, the "ideal" smoothed splitting estimator is defined as

$$\widehat{\ell} = \widehat{r}\ \rho^{\widehat{T}}, \text{with } \widehat{r} \in (\rho, 1],$$

whereas the true probability of the rare event may be decomposed as

$$\ell = r\ \rho^{T}, \text{with } T = \left\lfloor \frac{\log \ell}{\log \rho} \right\rfloor \text{ and } r = \ell \rho^{-T} \in (\rho, 1].$$

Let us summarize now the statistical properties of this "ideal" estimator.

Theorem 4.4.1. *The ideal estimator $\widehat{\ell}$ has the following properties:*

1. *Strong consistency:* $\widehat{\ell} \xrightarrow[N \to \infty]{a.s.} \ell$
2. *Number of steps:* $\mathbb{P}(\widehat{T} \neq T) \leq 2\ (T+1)\ e^{-2N\alpha^2}$ *where* $\alpha = \min(\rho - \ell^{\frac{1}{T}}, \ell^{\frac{1}{T+1}} - \rho)$.
3. *Asymptotic normality:* $\sqrt{N}\ \frac{\widehat{\ell}-\ell}{\ell} \xrightarrow[N \to \infty]{\mathcal{D}} \mathcal{N}(0, \sigma^2)$ *where* $\sigma^2 = T\ \frac{1-\rho}{\rho} + \frac{1-r}{r}$.
4. *Positive bias:* $N\ \frac{\mathbb{E}[\ \widehat{\ell}\ ]-\ell}{\ell} \xrightarrow[N \to \infty]{} T\ \frac{1-\rho}{\rho}$.

Proof. We first prove the strong consistency. Let us denote by $F(\varepsilon)$ the Lebesgue measure of B_ε: $\forall \varepsilon \in \mathbb{R}$, $F(\varepsilon) = \mathbb{P}(\mathbf{Y} \in B_\varepsilon)$. By convention, we will assume that $B_\varepsilon = \emptyset$ for $\varepsilon < 0$. One can readily see that $F(\varepsilon)$ has the following properties:

- $F(\varepsilon) = 0$ when $\varepsilon < 0$,
- $F(0) = \ell$,
- $F(\varepsilon) = 1$ when $\varepsilon \geq \varepsilon_0$, or $\lim_{\varepsilon \to +\infty} F(\varepsilon) = 1$ in the infinite case (cf. for example formulae (4.6) or (4.7)),
- F is a non decreasing and continuous function on $(0, \varepsilon_0)$.

We will also make use of the mapping $F(\varepsilon, \varepsilon')$, defined for $0 \leq \varepsilon' \leq \varepsilon \leq \varepsilon_0$ as

$$F(\varepsilon, \varepsilon') = \mathbb{P}(\mathbf{Y} \in B_{\varepsilon'} | \mathbf{Y} \in B_\varepsilon) = \frac{F(\varepsilon')}{F(\varepsilon)}.$$

With these notations, let us recall the following point: by construction and by assumption on the i.i.d. SSM, given $\widehat{\varepsilon}_{t-1}$, the random vectors $\mathbf{Y}_1^t, \dots, \mathbf{Y}_N^t$ are i.i.d. with uniform distribution in $B_{\widehat{\varepsilon}_{t-1}}$. For all $i = 1, \dots, N$, let us define

$$\varepsilon(\mathbf{Y}_i^t) = \inf\{\varepsilon \in [0, \widehat{\varepsilon}_{t-1}] : S_\varepsilon(\mathbf{Y}_i^t) \geq 1\}.$$

Then the random variables $D_1 = \varepsilon(\mathbf{Y}_1^t), \dots, D_N = \varepsilon(\mathbf{Y}_N^t)$ are i.i.d. with cdf $F(\widehat{\varepsilon}_{t-1}, .)$.

Thus, given $\widehat{\varepsilon}_{t-1}$, $\widehat{\varepsilon}_t$ is an empirical quantile of order ρ for the i.i.d. sample $(D_1, \dots, D_N)$. Denoting by $F_N(\widehat{\varepsilon}_{t-1}, .)$ the empirical cdf of F with this sample, we have

$$|F(\widehat{\varepsilon}_{t-1}, \widehat{\varepsilon}_t) - \rho| \leq |F(\widehat{\varepsilon}_{t-1}, \widehat{\varepsilon}_t) - F_N(\widehat{\varepsilon}_{t-1}, \widehat{\varepsilon}_t)| + |F_N(\widehat{\varepsilon}_{t-1}, \widehat{\varepsilon}_t) - \rho| .$$

By construction of $\widehat{\varepsilon}_t$, we know that the second term of this inequality is less than $1/N$, so that the almost sure convergence to 0 follows for it. For the first term, denoting by $\|f\|_\infty$ the supremum norm of f, and using the Dvoretsky-Kiefer-Wolfowitz inequality (see for example [93] p. 268), we know that for any $\eta > 0$

$$\mathbb{P}(\|F(\widehat{\varepsilon}_{t-1}, .) - F_N(\widehat{\varepsilon}_{t-1}, .)\|_\infty \geq \eta) \leq 2e^{-2N\eta^2},$$

which guarantees the almost sure convergence via the Borel-Cantelli Lemma. Thus we have proved that for all t

$$F(\widehat{\varepsilon}_{t-1}, \widehat{\varepsilon}_t) \xrightarrow[N\to\infty]{a.s.} \rho$$

Next, since the product of a finite and deterministic number of random variables will almost surely converge to the product of the limits, we conclude that for all t

$$\rho^t - \prod_{k=1}^{t} F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k) \xrightarrow[N\to\infty]{a.s.} 0.$$

Finally we have to proceed with the last step. We will only focus on the general case where $\log \ell / \log \rho$ is not an integer. Recall that $T = \lfloor \log l / \log \rho \rfloor$ is the "correct" (theoretical) number of steps *i.e.* the number of steps that "should" be done, whereas $\widehat{T}$ is the true and random number of steps of the algorithm. From the preceding results, we have that almost surely for N large enough

$$\prod_{k=1}^{T+1} F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k) < \ell < \prod_{k=1}^{T} F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k),$$

so that, almost surely for N large enough, the algorithm stops after $\widehat{T} = T$ steps. Therefore, in the following, we can assume that $\widehat{T} = T$.

Using the same reasoning as previously, we have

$$|F(\widehat{\varepsilon}_T, 0) - F_N(\widehat{\varepsilon}_T, 0)| \xrightarrow[N\to\infty]{a.s.} 0.$$

By definition, T satisfies

$$\prod_{k=1}^{T} F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k)\ F(\widehat{\varepsilon}_T, 0) = F(0) = \ell,$$

which implies

$$F(\widehat{\varepsilon}_T, 0) \xrightarrow[N\to\infty]{a.s.} \frac{\ell}{\rho^T},$$

and also

$$F_N(\widehat{\varepsilon}_T, 0) \xrightarrow[N\to\infty]{a.s.} \frac{\ell}{\rho^T}.$$

Putting all things together, we get

$$\widehat{\ell} = F_N(\widehat{\varepsilon}_T, 0) \times \rho^T \xrightarrow[N\to\infty]{a.s.} \frac{\ell}{\rho^T} \times \rho^T = \ell,$$

which concludes the proof of the consistency.

Let us prove now the exponential upper bound for the probability that $\widehat{T}$ differs from T. To this end, let us denote by $A = \{\widehat{T} = T\}$ the event for which the algorithm stops after the correct number of steps, and which can be written as follows

$$A = \{\widehat{\varepsilon}_{T+1} = 0 < \widehat{\varepsilon}_T\} = \left\{ \prod_{k=1}^{T+1} F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k) = \widehat{\ell} < \prod_{k=1}^{T} F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k) \right\}.$$

For all $k = 1, \ldots, T+1$, if we denote

$$A_k = \left\{ \ell^{\frac{1}{T}} - \rho < \rho - F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k) < \ell^{\frac{1}{T+1}} - \rho \right\},$$

we have

$$\mathbb{P}(A) \geq \mathbb{P}(A_1 \cap \cdots \cap A_{T+1}) \geq 1 - \sum_{k=1}^{T+1} (1 - \mathbb{P}(A_k)).$$

Denoting $\alpha = \min\left(\rho - \ell^{\frac{1}{T}}, \ell^{\frac{1}{T+1}} - \rho\right)$, the Dvoretsky-Kiefer-Wolfowitz inequality implies

$$1 - \mathbb{P}(A_k) \leq \mathbb{P}(|\rho - F(\widehat{\varepsilon}_{k-1}, \widehat{\varepsilon}_k)| > \alpha) \leq 2e^{-2N\alpha^2},$$

so that the result is proved

$$\mathbb{P}(A) = \mathbb{P}(\widehat{T} = T) \geq 1 - 2(T+1)e^{-2N\alpha^2}.$$

By the way, this is another method to see that $\widehat{T} \xrightarrow[N\to\infty]{a.s.} T$.

For the asymptotic normality and bias properties, we refer the reader to Theorem 1 and Proposition 4 of [13]: using the notations and tools of smoothed splitting, the proofs there can be adapted to yield the desired results.

4.4.2 Remarks and comments

Number of steps With an exponential probability, the number of steps of the algorithm is $T = \lfloor \log \ell / \log \rho \rfloor$.

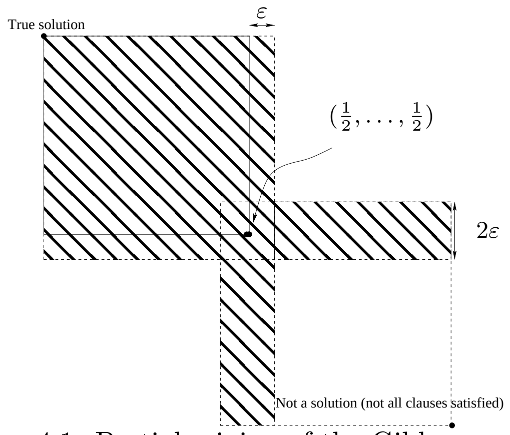
Bias The fact that this estimator is biased stems from the adaptive character of the algorithm. This is not the case with a sequence of fixed levels $(\varepsilon_1, \dots, \varepsilon_T)$. However, this bias is of order $1/N$, so that when N is large enough, it is clearly negligible relative to the standard deviation. Moreover, the explicit formula for this bias allows us to derive confidence intervals for ℓ which take this bias into account.

Estimate of the rare-event cardinality The previous discussion focused on the estimation of the rare-event probability, which in turn provides an estimate of the actual number of solutions to the original SAT problem by taking $|\widehat{\mathcal{X}}^*| = 2^n\,\widehat{\ell}$. In fact, the number of solutions may be small and thus can be determined by actual counting the different instances in the last sample of the algorithm.This estimator will be denoted by $|\widehat{\mathcal{X}}^*_{dir}|$. Typically it underestimates the true number of solutions $|\mathcal{X}^*|$, but at the same time it has a smaller (empirical) variance as compared to the product estimator. Even if we do not know its mathematical properties, this estimate can be useful. Firstly, it may be interesting for practical purposes to know the set (and the number) of all the different solutions that have been found for the original SAT problem. Secondly, it is also convenient when we compare our results with the ones given by the algorithm in [83], where a screening step (i.e. removal of the duplicates on the finite space) is involved.

Mixing properties Our purpose here is to explain why the Gibbs sampler used at each step of the algorithm is irreducible and globally reaching and hence has good mixing properties. For the sake of clarity, we will focus first on g_ε as per (4.8). With this function, for a given ε, we can split the region explored by the Gibbs sampler in several small (sub) hypercubes or hyperrectangles, as shown schematically in Figure 4.1. To each vertex of the whole hypercube $(0, 1)^n$ that represents a solution of the original SAT problem, corresponds a sub-hypercube of edge length $1/2+\varepsilon$, including the central point with coordinates $(1/2, \dots, 1/2)$. And around this point, we have a sub-hypercube of edge length 2ε, which is common to all those elements.

For the other parts of the domain, which do not correspond to a solution, things become a bit more complicated. It is a union of ε-thin "fingers" extending outwards in several directions (a subspace). The corresponding sub-domain being explored depends on the minimum number of variables that need to be taken in $(1/2-\varepsilon, 1/2+\varepsilon)$ in order to satisfy all the ε-clauses. The domain is then a rectangle of length $1/2+\varepsilon$ on the "free" variables, and of length 2ε in the other directions, that is on the $(1/2-\varepsilon, 1/2+\varepsilon)$ constrained variables. Again, all those rectangles include the small central sub-hypercube.

The union of all these sub-hypercubes/rectangles is the domain currently explored by the Gibbs sampler. The geometry of the whole domain is then quite complex.


True solution
ε
$(\frac{1}{2},\ldots,\frac{1}{2})$
2ε
Not a solution (not all clauses satisfied)
Figure 4.1: Partial mixing of the Gibbs sampler.

It is clear that starting with any one of these sub-hypercubes/rectangles we can reach any other point within it in one iteration of the Gibbs sampler. Moreover, as long as the Markov chain stays within the same sub-hypercube/rectangle, any other point is accessed with uniform probability. This means that the mixing properties of our Gibbs sampler are the best possible as long as *we are restricted to one sub-hypercube*. Actually this suffices to make the algorithm work.

For g_ε as per (4.6) or (4.7), the same picture mostly holds, but the mixing properties within each sub-hypercube is not that easy to analyze. This is somehow compensated by an ability to deal with the inter-variable relations: the geometry of the domain explored around the centre point reflects these constraints, and thus has a much more complicated shape. These g_ε functions work in practice better than (4.8).

4.5 Numerical Results

Below we present numerical results with both SSM Algorithm 4.3.2 and its counterpart Enhanced Cloner Algorithm [83] for several SAT instances. In particular we present data for three different SAT models: one of small size, another of moderate size and the third of large size. To study the variability in the solutions we run each problem 10 times and report the statistic.

To compare the efficiencies of both algorithms we run them on the same set of parameters ρ and b, where b is the number of cycles in the systematic Gibbs sampler. If not stated otherwise we set $b = 1$ and $\rho = 0.2$. From our numerical results follows that although both algorithms perform similarly (in terms of the CPU time and variability) the SSM is more robust than its splitting counterpart. In particular we shall see that SSM Algorithm 4.3.2 produces quite reliable estimator for a large set of b including $b = 1$, while its splitting counterpart Enhanced Cloner Algorithm is quite sensitive to b and thus, requires tuning.

Below we use the following notations:

1. $N_t^{(e)}$ and $N_t^{(s)}$ denote the actual number of elites and the one after screening, respectively.

2. ε_t denotes the adaptive ε parameter at iteration t.

3. $\rho_t = N_t^{(e)}/N$ denotes the adaptive proposal rarity parameter at iteration t.

4. RE denotes the relative error. Note that for our first, second and third model we used $|\mathcal{X}^*| = 15$, $|\mathcal{X}^*| = 2258$ and $|\mathcal{X}^*| = 1$, respectively. They were obtained by using the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$ with a very large sample, namely $N = 100,000$.

4.5.1 Smoothed Splitting Algorithm

In all our numerical results we use $g_\varepsilon(y)$ in (4.7).

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

Table 4.1 presents the performance of smoothed Algorithm 4.3.2 for the 3-SAT problem with an instance matrix $A = (20 \times 80)$ with $N = 1,000$, $\rho = 0.2$ and $b = 1$. Since the true number of solution is $|\mathcal{X}^*| = 15$, following the notations of

Section 4.4, we have that

$$\ell = \frac{15}{2^{20}} = r\ \rho^T, \text{with } T = \left\lfloor \frac{\log \ell}{\log \rho} \right\rfloor = \left\lfloor \frac{\log(15/2^{20})}{\log 0.2} \right\rfloor = 6$$

and

$$r = \ell \rho^{-T} = \frac{15}{2^{20}} 0.2^{-6} \approx 0.22.$$

Each run of the algorithm gives an estimator :

$$|\widehat{\mathcal{X}}^*| = 2^{20} \times \widehat{\ell} = 2^{20} \times (\widehat{r}\ \rho^{\widehat{T}}) = 2^{20} \times (\widehat{r}\ 0.2^{\widehat{T}}), \text{with } \widehat{r} \in (\rho, 1] = (0.2, 1].$$

In Table 4.1 , the column "Iterations" corresponds to $\widehat{T} + 1$ for each of the 10 runs (the theoretical value is thus $T + 1 = 7$). It is indeed 7 most of the time, but sometimes jumps to 8, which is not a surprise since $r = 0.22 \approx 0.2$.

Concerning the relative error of $|\widehat{\mathcal{X}}^*|$ (RE of $|\widehat{\mathcal{X}}^*|$), Theorem 4.4.1 states that it should be approximately equal to

$$\frac{1}{\sqrt{N}} \sqrt{T \frac{1-\rho}{\rho} + \frac{1-r}{r}} \approx 0.17,$$

while we find experimentally (see Table 4.1) a relative error of 0.228. There are two main reasons for this: first we performed only 10 runs, and second we set $b = 1$, while the analysis of the i.i.d. SSM suggests b to be large. Altogether, it gives the correct order of magnitude.

Concerning the relative bias of $|\widehat{\mathcal{X}}^*|$, Theorem 4.4.1 states that it should be approximately equal to

$$\frac{1}{N} \times \left(T \frac{1-\rho}{\rho} \right) \approx 0.024,$$

while experimentally (see Table 4.1) we find a relative bias of 0.018. The comments on the bias are the same as for the relative error above.

Table 4.1: Performance of smoothed Algorithm 4.3.2 for SAT 20 × 80 model.

Run N_0	Iterations	$\lvert\widehat{\mathcal{X}}^*\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	7	13.682	0.088	15	0	1.207
2	7	16.725	0.115	15	0	1.192
3	7	24.852	0.657	15	0	1.189
4	8	12.233	0.184	15	0	1.383
5	7	14.217	0.052	15	0	1.248
6	8	12.564	0.162	15	0	1.341
7	7	19.770	0.318	15	0	1.174
8	7	17.073	0.138	15	0	1.192
9	8	12.448	0.170	15	0	1.338
10	8	9.089	0.394	15	0	1.399
Average	7.4	15.265	0.228	15	0	1.266

In Figure 4.2, we give an illustration of the asymptotic normality, as given by theorem 4.4.1. The Figure compares the cdf of the limit Gaussian distribution, and the empirical distribution on 100 runs. Here $\rho = 1/2$.

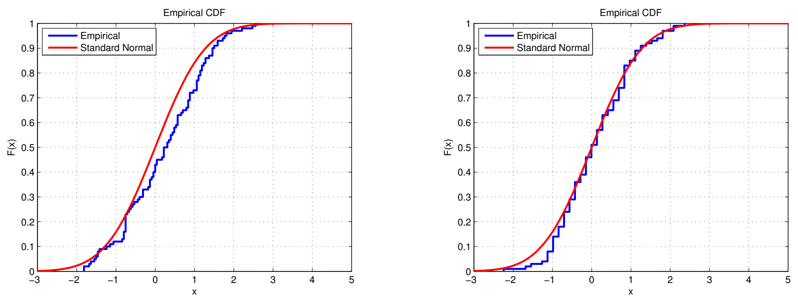


Figure 4.2: Asymptotic normality: empirical (100 runs) and limiting Gaussian cdf's, 1000 replicas (left) and $10,000$ (right).

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

This example is taken from `www.satlib.org`. Table 4.2 presents the performance of smoothed Algorithm 4.3.2. We set $N = 10,000$, $\rho = 0.2$ and $b = 1$ for all iterations.

Table 4.2: Performance of the smoothed Algorithm 4.3.2 for SAT 75×325 model.

Run N_0	Iterations	$\lvert\widehat{\mathcal{X}}^*\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	28	2210.2	0.021	2254	0.0018	519.7
2	28	2750.5	0.218	2232	0.0115	518.0
3	28	1826.1	0.191	2248	0.0044	523.6
4	28	2403.3	0.064	2254	0.0018	524.3
5	28	2189.6	0.030	2250	0.0035	519.3
6	28	1353.6	0.401	2254	0.0018	524.5
7	28	2572.8	0.139	2214	0.0195	528.6
8	28	2520.0	0.116	2246	0.0053	525.2
9	28	2049.2	0.092	2208	0.0221	521.8
10	28	2827.3	0.252	2244	0.0062	528.8
Average	28	2270.3	0.153	2240.4	0.0078	523.4

It follows from Table 4.2 that the average relative error of the product estimator $|\widehat{\mathcal{X}}^*|$ is RE $= 0.163$ and of the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$ is only RE $= 0.038$.

Third Model: Random $3-4$-SAT with matrix $A = (122 \times 663)$.

Our last model is the random 3-SAT with the instance matrix $A = (122 \times 663)$ and a single valid assignment, that is $|\mathcal{X}^*| = 1$, taken from `http://www.is.titech.ac.jp/~watanabe/gensat`. We set $N = 50,000$ and $\rho = 0.4$ for all iterations.

We found that the the average CPU time is about 3 hours for each run, the average relative error for the product estimator $|\widehat{\mathcal{X}}^*|$ is RE $= 0.15$, while for the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$ it is RE $= 0.1$. This means that in 9 out of 10 runs SSM finds the unique SAT assignment.

4.5.2 Splitting Algorithm

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

Table 4.3 presents the performance of the improved splitting Enhanced Cloner Algorithm [83] for the 3-SAT problem with an instance matrix $A = (20 \times 80)$ with $N = 1,000$, $\rho = 0.2$ and $b = 1$.

Table 4.3: Performance of Enhanced Cloner Algorithm [83] for SAT 20×80 model.

Run N_0	Iterations	$\|\widehat{\mathcal{X}}^*\|$	RE of $\|\widehat{\mathcal{X}}^*\|$	$\|\widehat{\mathcal{X}}^*_{dir}\|$	RE of $\|\widehat{\mathcal{X}}^*_{dir}\|$	CPU
1	10	17.316	0.154	15	0	0.641
2	10	15.143	0.010	15	0	0.640
3	10	12.709	0.153	15	0	0.645
4	9	16.931	0.129	15	0	0.566
5	10	13.678	0.088	15	0	0.644
6	9	15.090	0.006	15	0	0.565
7	9	10.681	0.288	15	0	0.558
8	10	13.753	0.083	15	0	0.661
9	10	14.022	0.065	15	0	0.646
10	10	13.445	0.104	15	0	0.651
Average	9.7	14.277	0.108	15	0	0.622

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

This example is taken from `www.satlib.org`. Table 4.4 presents the performance of the Enhanced Cloner Algorithm [83]. We set $N = 10,000$ and $\rho = 0.1$ and $b = \eta$ for all iterations until the Enhanced Cloner Algorithm reached the desired level 325, (recall that b is the number of Gibbs cycles and η is the number of splitting of each trajectory). After that, at the last iteration, we switched to $N = 100,000$.

Table 4.4: Performance of the Enhanced Cloner Algorithm [83] for SAT 75×325 model.

Run N_0	Iterations	$\lvert\widehat{\mathcal{X}}^*\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*\rvert$	$\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	RE of $\lvert\widehat{\mathcal{X}}^*_{dir}\rvert$	CPU
1	24	2458.8	0.089	2220	0.017	640.8
2	24	1927.8	0.146	2224	0.015	673.8
3	24	1964.6	0.130	2185	0.032	664.5
4	24	2218.9	0.017	2216	0.019	661.3
5	24	2396.9	0.062	2191	0.030	678.1
6	24	2271.8	0.006	2230	0.012	661
7	24	2446.1	0.083	2202	0.025	695
8	24	2090.5	0.074	2200	0.026	711.7
9	24	2147.7	0.049	2213	0.020	696.8
10	24	2395	0.061	2223	0.016	803.3
Average	24	2231.8	0.072	2210.4	0.021	688.6

It is interesting to note that if we set $b = 1$ instead of $b = \eta$, the average relative error of both the product and the direct estimators of Enhanced Cloner Algorithm [83] substantially increases. They become 0.27 and 0.16 instead of 0.072 and 0.021, respectively (see Table 4.4). This is in turn worse than 0.153 and 0.0078, the average relative errors of the product estimator of SSM Algorithm 4.3.2 (see Table 4.2). It is also important to note that by setting $b \neq 1$ in the SSM Algorithm 4.3.2, in particular setting $b = \eta$ we found that both relative errors remain basically close to these for $b = 1$. This means that one full cycle of the Gibbs sampler suffices for Algorithm 4.3.2, while the Basic Splitting Algorithm of [83] requires tuning of b. In other words, the SSM Algorithm 4.3.2 is robust with respect to b, while its counterpart Basic Splitting Algorithm is not.

Third Model: Random 3-4-SAT with matrix $A = (122 \times 663)$

Similar to SSM Algorithm 4.3.2 we set $N = 10,000$ and $\rho = 0.1$ for all iterations until Enhanced Cloner Algorithm [83] has reached the desired level 663. After that we switched to $N = 100,000$ for the last iteration. Again, as for the second model, we set here $b = \eta$ instead of $b = 1$ as for SSM Algorithm 4.3.2.

The the average CPU time is about 2 hours for each run, the average relative error for the product estimator $|\widehat{\mathcal{X}}^*|$ is RE $= 0.23$, while for the direct estimator $|\widehat{\mathcal{X}}^*_{dir}|$ it is RE $= 0.4$. This means that only in 6 out of 10 runs Enhanced Cloner Algorithm [83] finds the unique SAT assignment (compare this with 9 out of 10 runs for SSM Algorithm 4.3.2).

The above numerical results can be summarized as follows:

The proposed smoothed splitting method performs similarly to the standard splitting one (in terms of CPU time and variability).

The proposed method is robust, while the standard splitting is not, especially for the more difficult models, such as the Second and the Third Models. This means that parameters ρ and N in the former method can be chosen from a wide range, while in the latter they require careful tuning.

Chapter 5

Counting with Combined Splitting and Capture-Recapture Methods

Paul Dupuis[a]
Brown University, Providence, USA
Bahar Kaynar[b] , Ad Ridder[d]
Vrije University, Amsterdam, Netherlands
Reuven Rubinstein[c] , Radislav Vaisman
Technion, Haifa, Israel

[1]Research supported by AFOSR grant FA9550-09-0378.

[2]Research supported by NWO grant 400-06-044.

[3]Research supported by BSF(Binational Science Foundation) grant 2008482, and by NWO grant 040-11-168.

[4]Corresponding author. Department of Econometrics and Operations Research; Vrije University Amsterdam; Netherlands. Email address: ad.ridder@vu.nl

Abstract

We apply the splitting method to three well-known counting problems, namely 3-SAT, random graphs with prescribed degrees, and binary contingency tables. We present an enhanced version of the splitting method based on the capture-recapture technique, and show by experiments the superiority of this technique for SAT problems in terms of variance of the associated estimators, and speed of the algorithms.

Keywords. Counting, Gibbs Sampler, Capture-Recapture, Splitting.

5.1 Introduction

In this paper we apply the splitting method introduced in Botev and Kroese [8] to a variety of counting problems in the class #P-complete. The classes #P and #P-complete have been introduced by Valiant [92] in the following way. Given any decision problem in the class NP, one can formulate the corresponding counting problem which asks for the total number of solutions for a given instance of the problem. The set of all these counting problems determines the complexity class #P. Clearly, a #P problem is at least as hard as its corresponding NP problem. In this paper we consider #P-complete problems. Completeness is defined similarly as for the decision problems: a problem is #P-complete if it is in #P, and if every #P problem can be reduced to it in polynomial counting reduction. This means that exact solutions to these problems cannot be obtained in polynomial time, and accordingly, our study focuses on approximation algorithms.

As an example, the satisfiability problem—commonly abbreviated to SAT—is well-known to be NP-complete. Its associated counting problem is denoted by #SAT for which is proved that it is #P-complete [92]. For more background on the complexity theory of decision and counting problems we refer to Papadimitriou [71].

The proposed splitting algorithm for approximate counting is a randomized one. It is based on designing a sequential sampling plan, with a view to decomposing a "difficult" counting problem defined on some set $\mathcal{X}^*$ into a number of "easy" ones associated with a sequence of related sets $\mathcal{X}_0, \mathcal{X}_1, \ldots, \mathcal{X}_m$ and such that $\mathcal{X}_m = \mathcal{X}^*$. Splitting algorithms explore the connection between counting and sampling problems, in particular the reduction from approximate counting of a discrete set to approximate sampling of elements of this set, with the sampling performed, typically, by some Markov chain Monte Carlo method.

Recently, counting problems have attracted research interest, notably #SAT which is also called model counting in Gomes et al. [44]. Although it has been shown that many solution techniques for SAT problems can be adapted for these problems, yet due to the exponential increase in memory usage and running times of these methods, their application area in counting is limited. This drawback motivated the approximative approach mentioned earlier. There are two main heuristic algorithms for approximate counting methods in #SAT. The first one, called `ApproxCount`, is introduced by Wei and Selman [98]. It is a local search method that uses Markov Chain Monte Carlo (MCMC) sampling to compute an approximation of the true model count of a given formula. It is fast and has been shown to provide good estimates for feasible solution counts, but, in contrast with our proposed splitting method, there are no guarantees as to the uniformity of the MCMC samples. Gogate and Dechter [41] recently proposed a second model counting technique called `SampleMinisat`, which is 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 the uniform distribution over all solutions. Experiments with `SampleMinisat` show that it is very fast and typically it provides very good estimates.

The splitting method discussed in this work for counting in deterministic problems is based on its classic counterpart for efficient estimation of rare-event probabilities in stochastic problems. The relation between rare-event simulation methods and approximate counting methods have also been discussed, for instance, by Blanchet and Rudoy [3], Botev and Kroese [7], Rubinstein [83]; and Chapter 9 in Rubinstein and Kroese [86].

As said, we propose to apply the sequential sampling method presented in Botev and Kroese [8] which yields a product estimator for counting the number of solutions $|\mathcal{X}^*|$, where the product is taken over the estimators of the consecutive conditional probabilities, each of which represents an "easy" problem. In addition, we shall consider an alternative version, in which we use the generated samples after the last iteration of the splitting algorithm as a sample for the capture-recapure method. This method gives us an alternative estimate of the counting problem. Furthermore, we shall study an extended version of the capture-recapture method when the problem size is too large for the splitting method to give reliable estimates. The idea is to decrease artificially the problem size and then apply a backwards estimation. Whenever applicable, the estimators associated with our proposed enhancements outperform the splitting estimators in terms of variance normalized by computational effort.

The paper is organized as follows. We first start with describing the splitting method in detail in Section 5.2. Section 5.3 deals with the combination of the classic capture-recapture method with the splitting algorithm. Finally, numerical results and concluding remarks are presented in Sections 5.4 and 5.5, respectively.

5.2 Splitting Algorithms for Counting

The splitting method is one of the main techniques for the efficient estimation of rare-event probabilities in stochastic problems. The method is based on the idea of restarting the simulation in certain states of the system in order to obtain more occurrences of the rare event. Although the method originated as a rare event simulation technique (see Cérou and Guyader [11], L'Ecuyer et al. [58], Garvels [28], Glasserman et al. [40], Lagnoux [55], Melas [64]), it has been modified in Blanchet and Rudoy [3], Botev and Kroese [7], and Rubinstein [83], for counting and combinatorial optimization problems.

Consider a NP decision problem with solution set $\mathcal{X}^*$, i.e., the set containing all solutions to the problem. We are interested in computing the size $|\mathcal{X}^*|$ of the solution set. Suppose that there is a larger set $\mathcal{X} \supset \mathcal{X}^*$ which can be represented by a simple description or formula; specifically, its size $|\mathcal{X}|$ is known and easy to compute. We call $\mathcal{X}$ the state space of the problem. Let $p = |\mathcal{X}^*| \,/\, |\mathcal{X}|$ denote the fraction (or "probability") of the solution set w.r.t. the state space. Since $|\mathcal{X}|$ is known, it suffices to compute p. In most cases p is extremely small, in other words we deal with a rare-event probability. However, assuming we can estimate p by $\widehat{p}$, we obtain automatically

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}|\widehat{p}$$

as an estimator of $|\mathcal{X}^*|$. Note that straightforward simulation based on generation of i.i.d. uniform samples $X_i \in \mathcal{X}$ and delivering the Monte Carlo estimator $\widehat{p}_{\text{MC}} = \frac{1}{N}\sum_{i=1}^{N} I_{\{X_i \in \mathcal{X}^*\}}$ as an unbiased estimator of $|\mathcal{X}^*|/|\mathcal{X}|$ fails when p is a rare-event probability. To be more specific, assume a parametrization of the decision problem. The size of the state space $|\mathcal{X}|$ is parameterized by n, such that $|\mathcal{X}| \to \infty$ as $n \to \infty$. For instance, in SAT n represents the number of variables. Furthermore we assume that the fraction of the solution set $p \to 0$ as $n \to \infty$. The required sample size N to obtain a relative accuracy ε of the 95% confidence interval by the Monte Carlo estimation method is (see Section 1.13 in [86])

$$N \approx \frac{1.96^2}{\varepsilon^2 p},$$

which increases like p^{-1} as $n \to \infty$.

The purpose of the splitting method is to estimate p more efficiently via the following steps:

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

2. Write $|\mathcal{X}^*| = |\mathcal{X}_m|$ as the telescoping product

$$|\mathcal{X}^*| = |\mathcal{X}_0| \prod_{t=1}^{m} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}, \tag{5.1}$$

   thus the target probability becomes a product $p = \prod_{t=1}^{m} c_t$, with ratio factors

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|}. \tag{5.2}$$

3. Develop an efficient estimator $\widehat{c}_t$ for each c_t and estimate $|\mathcal{X}^*|$ by

$$\widehat{\ell} = \widehat{|\mathcal{X}^*|} = |\mathcal{X}_0|\,\widehat{p} = |\mathcal{X}_0| \prod_{t=1}^{m} \widehat{c}_t. \tag{5.3}$$

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

(i). Put the counting problem into the framework (5.1) by making sure that

$$\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_m = \mathcal{X}^*, \tag{5.4}$$

such that each c_t is not a rare-event probability.

(ii). Obtain a low-variance estimator $\widehat{c}_t$ of each ratio c_t.

To deal with both problems, we propose an adaptive version of the splitting method. As a demonstration, consider a specific family of decision problems, namely those whose solution set is finite and given by linear integer constraints. In other words, $\mathcal{X}^* \subset \mathbb{Z}_+^n$ is given by

$$\begin{cases} \sum_{j=1}^{n} a_{ij}x_j = b_i, & i = 1, \ldots, m_1; \\ \sum_{j=1}^{n} a_{ij}x_j \geq b_i, & i = m_1+1, \ldots, m_1+m_2 = m; \\ x_j \in \{0, 1, \ldots, d\}, & \forall j = 1, \ldots, n. \end{cases} \tag{5.5}$$

Our goal is to count the number of feasible solutions (or points) to the set (5.5). Note that we assume that we know, or can compute easily, the bounding finite set $\mathcal{X} = \{0, 1 \ldots, d\}^n$, with points $\boldsymbol{x} = (x_1, \ldots, x_n)$ (in this case $|\mathcal{X}| = (d+1)^n$) as well for other counting problems.

Below we follow Rubinstein [83]. Define the Boolean functions $C_i : \mathcal{X} \to \{0, 1\}$ $(i = 1, \ldots, m)$ by

$$C_i(\boldsymbol{x}) = \begin{cases} I_{\{\sum_{j=1}^{n} a_{ij}x_j = b_i\}}, & i = 1, \ldots, m_1; \\ I_{\{\sum_{j=1}^{n} a_{ij}x_j \geq b_i\}}, & i = m_1+1, \ldots, m_1+m_2. \end{cases} \tag{5.6}$$

Furthermore, define the function $S : \mathcal{X} \to \mathbb{Z}_+$ by counting how many constraints are satisfied by a point $\boldsymbol{x} \in \mathcal{X}$, i.e., $S(\boldsymbol{x}) = \sum_{i=1}^{m} C_i(\boldsymbol{x})$. Now we can formulate the counting problem as a probabilistic problem of evaluating

$$p = \mathbb{E}_f \left[I_{\{S(\boldsymbol{X})=m\}} \right], \tag{5.7}$$

where $\boldsymbol{X}$ is a random point on $\mathcal{X}$, uniformly distributed with probability density function (pdf) $f(\boldsymbol{x})$, denoted by $\boldsymbol{X} \overset{\mathrm{d}}{\sim} f = \mathcal{U}(\mathcal{X})$. Consider an increasing sequence of thresholds $0 = m_0 < m_1 < \cdots < m_{T-1} < m_T = m$, and define the sequence of decreasing sets (5.4) by

$$\mathcal{X}_t = \{\boldsymbol{x} \in \mathcal{X} : S(\boldsymbol{x}) \geq m_t\}.$$

Note that in this way

$$\mathcal{X}_t = \{\boldsymbol{x} \in \mathcal{X}_{t-1} : S(\boldsymbol{x}) \geq m_t\},$$

for $t = 1, 2, \ldots, T$. The latter representation is most useful since it shows that the ratio factor c_t in (5.2) can be considered as a conditional expectation:

$$c_t = \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} = \mathbb{E}_{g_{t-1}}[I_{\{S(\boldsymbol{X}) \geq m_t\}}], \tag{5.8}$$

where $\boldsymbol{X} \overset{\mathrm{d}}{\sim} g_{t-1} = \mathcal{U}(\mathcal{X}_{t-1})$. Note that $g_{t-1}(\boldsymbol{x})$ is also obtained as a conditional pdf by

$$g_{t-1}(\boldsymbol{x}) = f(\boldsymbol{x}|\mathcal{X}_{t-1}) = \begin{cases} \frac{f(\boldsymbol{x})}{f(\mathcal{X}_{t-1})}, & \boldsymbol{x} \in \mathcal{X}_{t-1}; \\ 0, & \boldsymbol{x} \notin \mathcal{X}_{t-1}. \end{cases} \tag{5.9}$$

To draw samples from the uniform pdf $g_{t-1} = \mathcal{U}(\mathcal{X}_{t-1})$ on a complex set given implicitly, one applies typically MCMC methods. For further details we refer to Rubinstein [83].

5.2.1 The Basic Adaptive Splitting Algorithm

We describe here the adaptive splitting algorithm from Botev and Kroese [8]. The thresholds (m_t) are not given in advance, but determined adaptively via a simulation process. Hence, the number T of thresholds becomes a random variable. In fact, the (m_t)-thresholds should satisfy the requirements $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}| \approx \rho_t$, where the parameters $\rho_t \in (0, 1)$ are not too small, say $\rho_t \geq 0.01$, and set in advance. We call these the splitting control parameters. In most applications we choose these all equal, that is $\rho_t \equiv \rho$.

Consider a sample set $[\boldsymbol{X}]_{t-1} = \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N\}$ of N random points in $\mathcal{X}_{t-1}$. That is, all these points are uniformly distributed on $\mathcal{X}_{t-1}$. Let m_t be the $(1 - \rho_{t-1})$-th quantile of the ordered statistics values of the scores $S(\boldsymbol{X}_1), \ldots, S(\boldsymbol{X}_N)$. The elite set $[\boldsymbol{X}]_{t-1}^{(\mathrm{e})} \subset [\boldsymbol{X}]_{t-1}$ consists of those points of the sample set for which $S(\boldsymbol{X}_i) \geq m_t$. Let N_t be the size of the elite set. If all scores $S(\boldsymbol{X}_i)$ are distinct, it follows that the number of elites $N_t = \lceil N\rho_{t-1} \rceil$, where $\lceil \cdot \rceil$ denotes rounding to the largest integer. However, dealing with a discrete space, typically we will find more samples with $S(\boldsymbol{X}_i) \geq m_t$. All these are added to the elite set. Finally we remark that from (5.9) it easily follows that the elite points are distributed uniformly on $\mathcal{X}_t$.

Regarding the elite set based in $\mathcal{X}_{t-1}$ as a subset of $\mathcal{X}_t$, we do two things. First, we screen out (delete) duplicates, so that we end up with a set of size $N_t^{(\mathrm{s})}$ of distinct elites. Secondly, each screened elite is the starting point of an independent Markov chain simulation (MCMC method) in $\mathcal{X}_t$ using a transition probability matrix P_t with $g_t = \mathcal{U}(\mathcal{X}_t)$ as its stationary distribution. Because the starting point is uniformly distributed, all consecutive points on the sample path are uniformly distributed on $\mathcal{X}_t$. Therefore, we may use all these points in the next iteration.

Thus, we simulate $N_t^{(\mathrm{s})}$ independent trajectories, each trajectory for $b_t = \lfloor N/N_t^{(\mathrm{s})} \rfloor$ steps. This produces a total of $N_t^{(\mathrm{s})} b_t \leq N$ uniform points in $\mathcal{X}_t$. To continue with the next iteration again with a sample set of size N, we choose randomly $N - N_t^{(\mathrm{s})} b_t$ of these sample paths and extend them by one point. Denote the new sample set by $[\boldsymbol{X}]_t$, and repeat the same procedure as above. The algorithm iterates until we find $m_t = m$, say at iteration T, at which stage we stop and deliver

$$\widehat{|\mathcal{X}^*|} = |\mathcal{X}| \prod_{t=1}^{T} \widehat{c}_t \tag{5.10}$$

as an estimator of $|\mathcal{X}^*|$, where $\widehat{c}_t = N_t/N$ at iteration t.

In our experiments we applied a Gibbs sampler to implement the MCMC simulation for obtaining uniformly distributed samples. To summarize, we give the algorithm.

Algorithm 5.2.1 (Basic splitting algorithm for counting)**.**

- *Input: the counting problem* (5.5)*; the bounding set* $\mathcal{X}_0$*; sample size* N*; splitting control parameters* $(\rho_t)_t$*.*

- *Output: counting estimator* (5.10)*.*

1. *Set a counter $t = 1$. Generate a sample set $[\boldsymbol{X}]_0$ of N points uniformly distributed in $\mathcal{X}_0$. Compute the threshold m_1, and determine the size N_1 of the elite set. Set $\widehat{c}_1 = N_1/N$ as an estimator of $c_1 = |\mathcal{X}_1|/|\mathcal{X}_0|$.*

2. *Screen out the elite set to obtain $N_t^{(\mathrm{s})}$ distinct points uniformly distributed in $\mathcal{X}_t$.*

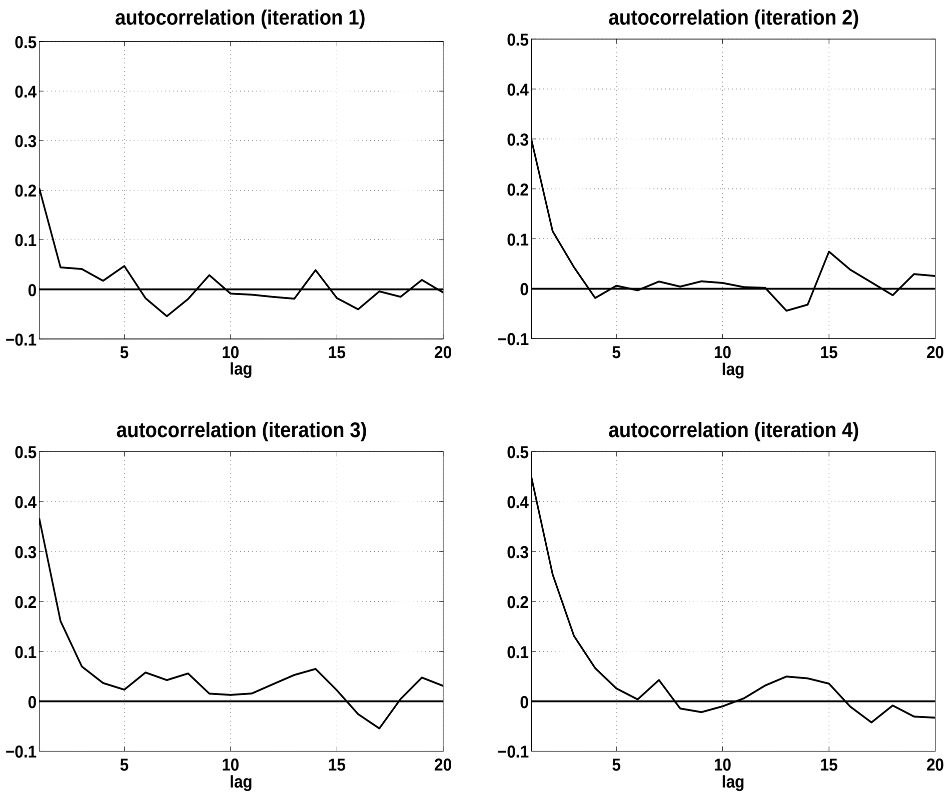
3. *Let $b_t = \lfloor N/N_t^{(\mathrm{s})} \rfloor$. For all $i = 1, 2, \ldots, N_t^{(\mathrm{s})}$, starting at the i-th screened elite point run a Markov chain of length b_t in $\mathcal{X}_t$ with $g_t = \mathcal{U}(\mathcal{X}_t)$ as its stationary distribution. Extend $N - N_t^{(\mathrm{s})} b_t$ randomly chosen sample paths with one point. Denote the new sample set of size N by $[\boldsymbol{X}]_t$.*

4. *Increase the counter $t = t + 1$. Compute the threshold m_t, and determine the size N_t of the elite set. Set $\widehat{c}_t = N_t/N$ as an estimator of $c_t = |\mathcal{X}_t|/|\mathcal{X}_{t-1}|$.*

5. *If $m_t = m$ deliver the estimator* (5.10)*; otherwise repeat from step 2.*

Remark 5.2.1. Note that the goal of our algorithm 5.2.1 is to produce points uniformly distributed on each subregion $\mathcal{X}_t$. At the first iteration (due to the acceptance-rejection technique) the samples are indeed uniformly distributed on the entire space $\mathcal{X}_0$. In the subsequent iterations we generate in parallel a substantial number of independent sequences of uniform points. We also make sure that they have sufficient lengths. By doing so we guarantee that the generated points at each $\mathcal{X}_t$ are distributed approximately uniform, see the discussion in Gelman and Rubin [32]. We found numerically that this is obtained by choosing the sample size about 10-100 times larger then dimension n and the splitting parameters ρ_t not too small, say $10^{-1} \geq \rho_t \geq 10^{-3}$.

The following figures support these rapid mixing properties. We applied the splitting algorithm to a 3-SAT problem from the SATLIB benchmark problems, consisting of $n = 75$ literals and $m = 375$ clauses, see Section 5.4.1 for further details. In each iteration we chose arbitrarily one of the screened elite points as a starting point of the Gibbs sampler of length $N = 1000$. From this sequence of points $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ in the subset $\mathcal{X}_t$ we constructed the time series $y_1, \ldots, y_N$ of their partial sums $y_i = \sum_{j=1}^{n} \boldsymbol{X}_{i,j}$, and computed the autocorrelation function of the times series. The figures show these autocorrelation functions of the first four iterations, up to lag 20.



5.3 Combining Splitting and Capture–Recapture

In this section we discuss how to combine the well known capture-recapture (CAP-RECAP) method with the basic splitting Algorithm 5.2.1. First we present the classical capture-recapture algorithm in the literature.

5.3.1 The Classic Capture–Recapture in the Literature

Originally the capture-recapture method was used to estimate the size, say M, of an unknown population on the basis of two independent samples, each taken without replacement from it. To see how the CAP-RECAP method works, consider an urn model with a total of M identical balls. Denote by N_1 and N_2 the sample sizes taken at the first and second draws, respectively. Assume, in addition, that

- The second draw takes place after all N_1 balls have been returned to the urn.
- Before returning the N_1 balls, each is marked, say we painted them a different color.

Denote by R the number of balls from the first draw that reappear in the second. Then a biased estimate $\widetilde{M}$ of M is

$$\widetilde{M} = \frac{N_1 N_2}{R}. \tag{5.11}$$

This is based on the observation that $N_2/M \approx R/N_1$. Note that the name capture-recapture was borrowed from a problem of estimating the animal population size in a particular area on the basis of two visits. In this case R denotes the number of animals captured on the first visit and recaptured on the second.

A slightly less biased estimator of M is

$$\widehat{M} = \frac{(N_1+1)(N_2+1)}{(R+1)} - 1. \tag{5.12}$$

See Seber [88] for an analysis of its bias. Furthermore, defining the statistic

$$V = \frac{(N_1+1)(N_2+1)(N_1-R)(N_2-R)}{(R+1)^2(R+2)},$$

Seber [88] shows that

$$\mathbb{E}[V] \sim \mathbb{V}\mathrm{ar}[\widehat{M}]\ \left(1+\mu^2 e^{-\mu}\right),$$

where

$$\mu = \mathbb{E}[R] = N_1 N_2/M,$$

so that V is an approximately unbiased estimator of the variance of $\widehat{M}$.

5.3.2 Splitting algorithm combined with Capture–Recapture

Application of the CAP-RECAP to counting problems is trivial. The target is to estimate size $M = |\mathcal{X}^*|$. Consider the basic splitting algorithm 5.2.1 at the last iteration T, when we have found a set of elite points that satisfy all m constraints; i.e., points in the target set $\mathcal{X}^*$. For the capture-recapture method we screen out duplicates which gives us a set $[\boldsymbol{X}]_T^{(\mathrm{s})} \subset \mathcal{X}^*$ of $N_T^{(\mathrm{s})}$ distinct points.

Then (i) we execute step 3 of Algorithm 5.2.1 (the MCMC simulation) with sample size $N_1^{(\mathrm{cap})}$; (ii) we screen out duplicates; and (iii) we record the resulting set of N_1 distinct points. Independently, we execute (i)-(iii) a second time, starting from the points $[\boldsymbol{X}]_T^{(\mathrm{s})}$, now with sample size $N_2^{(\mathrm{recap})}$. After screening out we record a set of N_2 distinct points. Finally, we count the number R of distinct points that occur in both recordings and deliver either estimator (5.11) or (5.12).

To summarize, we give the algorithm.

Algorithm 5.3.1 (Splitting with capture-recapture algorithm for counting)**.**

- *Input: the counting problem* (5.5)*; the bounding set* $\mathcal{X}_0$*; sample sizes* $N, N_1^{(\mathrm{cap})}, N_2^{(\mathrm{recap})}$*; splitting control parameters* $(\rho_t)_t$*.*
- *Output: counting estimator* (5.11) *or* (5.12)*.*

1. *Set a counter* $t = 1$*. Generate a sample set* $[\boldsymbol{X}]_0$ *of* N *points uniformly distributed in* $\mathcal{X}_0$*. Compute the threshold* m_1*, and determine the elite set of points satisfying* m_1 *constraints.*
2. *Screen out the elite set to obtain* $N_t^{(\mathrm{s})}$ *distinct points uniformly distributed in* $\mathcal{X}_t$*.*
3. *Let* $b_t = \lfloor N/N_t^{(\mathrm{s})} \rfloor$*. For all* $i = 1, 2, \ldots, N_t^{(\mathrm{s})}$*, starting at the* i*-th screened elite point run a Markov chain of length* b_t *in* $\mathcal{X}_t$ *with* $g_t = \mathcal{U}(\mathcal{X}_t)$ *as its stationary distribution. Extend* $N - N_t^{(\mathrm{s})} b_t$ *randomly chosen sample paths with one point. Denote the new sample set of size* N *by* $[\boldsymbol{X}]_t$*.*
4. *Increase the counter* $t = t + 1$*. Compute the threshold* m_t*, and determine the elite set of points satisfying* m_t *constraints.*
5. *If* $m_t < m$ *repeat from step 2 otherwise, set* $T = t$ *and go to step 6.*
6. *Screen out the elite set to obtain* $N_T^{(\mathrm{s})}$ *distinct points uniformly distributed in* $\mathcal{X}_T = \mathcal{X}^*$*.*
7. *Let* $b_T(1) = \lfloor N_1^{(\mathrm{cap})}/N_T^{(\mathrm{s})} \rfloor$*. For all* $i = 1, 2, \ldots, N_T^{(\mathrm{s})}$*, starting at the* i*-th screened elite point run a Markov chain of length* $b_T(1)$ *in* $\mathcal{X}_T$ *with* $g_T = \mathcal{U}(\mathcal{X}_T)$ *as its stationary distribution. Extend* $N_1^{(\mathrm{cap})} - N_T^{(\mathrm{s})} b_T(1)$ *randomly chosen sample paths with one point. Screen out duplicates to obtain a set* $[\boldsymbol{X}]_1^{(\mathrm{cap})}$ *of* N_1 *distinct points in* $\mathcal{X}^*$*.*
8. *Repeat step 7 with* $b_T(2) = \lfloor N_2^{(\mathrm{recap})}/N_T^{(\mathrm{s})} \rfloor$*. After screening out, the remaining set is* $[\boldsymbol{X}]_2^{(\mathrm{recap})}$ *of* N_2 *distinct points in* $\mathcal{X}^*$*.*
9. *Compute the number* R *of points in* $[\boldsymbol{X}]_1^{(\mathrm{cap})} \cap [\boldsymbol{X}]_2^{(\mathrm{recap})}$*.*
10. *Deliver estimator* (5.11) *or* (5.12)*.*

In Section 5.4 we report numerical results of simulation experiments executed by the splitting algorithm 5.2.1 and by the capture-recapture algorithm 5.3.1.

As a general observation we found that the performances of the corresponding counting estimators depend on the choice of sample size N in the two algorithms, and on the size of the target set $\mathcal{X}^*$. When we keep the sample N limited to 10000, then for $|\mathcal{X}^*|$ sizes up to 10^6 the CAP-RECAP estimator (5.12) is more accurate than the product estimator (5.10), that is

$$\mathbb{V}\mathrm{ar}[\widehat{|\mathcal{X}^*|}_{\mathrm{cap}}] \leq \mathbb{V}\mathrm{ar}[\widehat{|\mathcal{X}^*|}_{\mathrm{product}}].$$

However, if $10^6 < |\mathcal{X}^*| \leq 10^9$, we propose to apply an extended version of the capture-recapture method, as we will describe in the next section; and for larger target sets ($|\mathcal{X}^*| > 10^9$), we propose to execute just the splitting algorithm because the capture-recapture method performs poorly.

5.3.3 Extended Capture–Recapture Method for SAT

Recall that the regular CAP-RECAP method

1. Is implemented at the last iteration T of the splitting algorithm, that is when some configurations have already reached the desired set $\mathcal{X}^*$.
2. It provides reliable estimators of $|\mathcal{X}^*|$ if it is not too large, say $|\mathcal{X}^*| \leq 10^6$. (We assume sample sizes $N \leq 10000$.)

In rare events counting problems, $|\mathcal{X}^*|$ is indeed $\leq 10^6$, nevertheless we present below an extended CAP-RECAP version, which extends the original CAP-RECAP for 2-3 orders more, that is, it provides reliable counting estimators for $10^6 < |\mathcal{X}^*| \leq 10^9$. The enhancement is based on adding a random number τ of random constraints to the original solution set $\mathcal{X}^*$ that was given in (5.5). For reasons of exposition we consider the SAT problem only, since it is easy to generate a random clause involving n literals.

Algorithm 5.3.2 (Extended CAP-RECAP for SAT).

- *Input: the counting problem* (5.5)*; the bounding set* $\mathcal{X}_0$*; sample sizes* N, $N_1^{(\mathrm{cap})}$, $N_2^{(\mathrm{recap})}$, N_m*, where* $N \leq 10000$*; splitting control parameters* $(\rho_t)_t$*;* c^* *a relatively small number, say* $10^{-2} \leq c^* \leq 10^{-3}$.
- *Output: estimator of* $|\mathcal{X}^*|$.

1. *Execute Algorithm 5.2.1 and compute estimator* $\widehat{|\mathcal{X}^*|}$ *given in* (5.10).
2. *If* $\widehat{|\mathcal{X}^*|} > 10^9$*, stop; else, if* $\widehat{|\mathcal{X}^*|} \leq 10^6$*, execute steps 6-10 of Algorithm 5.3.1; else continue with step 3.*

3. *Recall that Algorithm 5.2.1 stops at iteration T with an elite sample set $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{N_T}\} \subset \mathcal{X}_T = \mathcal{X}^*$. Execute step 2 and step 3 of Algorithm 5.2.1; i.e., screen out duplicates and run the MCMC simulations (target sample size N_m) to obtain a sample set $[\boldsymbol{X}]_T \subset \mathcal{X}_T$ of size N_m. Set auxiliary counter $j = 1$.*

4. *Add one arbitrary (random) auxiliary clause to the model. Let N_{T+j} be the number of points in $[\boldsymbol{X}]_T$ that satisfy all $m + j$ clauses.*

5. *If $N_{T+j}/N_m > c^*$, increase auxiliary counter $j = j + 1$ and repeat from step 4. Else, set $\tau = j$; denote $\mathcal{X}_{T+\tau}$ for the extended model with these new τ auxiliary clauses; and define*

$$\widehat{c_{T+\tau}} = \frac{N_{T+\tau}}{N_m} \leq c^*. \tag{5.13}$$

6. *Execute steps 6-10 of the capture-recapture Algorithm 5.3.1 for the CAP-RECAP estimator $\widehat{|\mathcal{X}_{T+\tau}|}_{\text{cap}}$ of the size of the extended model.*

7. *Deliver estimator*

$$\widehat{|\mathcal{X}^*|}_{\text{ecap}} = \widehat{c_{T+\tau}}^{-1} \cdot \widehat{|\mathcal{X}_{T+\tau}|}_{\text{cap}}. \tag{5.14}$$

We call $\widehat{|\mathcal{X}^*|}_{\text{ecap}}$ the extended CAP-RECAP estimator. It is essential to bear in mind that

- $\widehat{|\mathcal{X}_{T+\tau}|}_{\text{cap}}$ is a CAP-RECAP estimator rather than a splitting (product) one.

- $\widehat{|\mathcal{X}^*|}_{\text{ecap}}$ does not contain the original estimators $\widehat{c}_1, \ldots, \widehat{c}_T$ generated by the splitting method.

- Since we only need here the uniformity of the samples at $\mathcal{X}_T$, we can run the splitting method of Section 5.2.1 all the way with relatively small values of sample size N and splitting control parameter ρ until it reaches the vicinity of $\mathcal{X}^*$ (meaning that the points of the elite set satisfy $m - r$ constraints, where $r = 1$ or $r = 2$; and then switch to larger N and ρ.

- In contrast to the splitting estimator which employs a product of T terms, formula (5.14) employs only a single c factor. Recall that this additional $\widehat{c_{T+\tau}}^{-1}$ factor allows to enlarge the CAP-RECAP estimators of $|\mathcal{X}^*|$ for about two-three additional orders, namely from $|\mathcal{X}^*| \approx 10^6$ to $|\mathcal{X}^*| \approx 10^9$.

5.4 Numerical Results

Below we present numerical results for the splitting algorithm for counting. In particular we consider the following problems:

1. The 3-satisfiability problem (3-SAT)
2. Graphs with prescribed degrees
3. Contingency tables

For the 3-SAT problem we shall also use the CAP-RECAP method when appropriate. We shall show that typically CAP-RECAP outperforms the splitting algorithm. The other two problems typically have too large solution sets to be applied by CAP-RECAP. Clearly, if we would make artificially the associated matrix of the restriction coefficients (see (5.5)) very sparse, the number of solutions could be made $< 10^9$ and CAP-RECAP would be applicable again. However, in this paper we did not follow such untypical situation.

We shall use the following notations.

Notation 5.4.1. For iteration $t = 1, 2, \ldots$

- N_t and $N_t^{(\mathrm{s})}$ denote the actual number of elites and the number after screening, respectively;
- m_t^* and m_{*t} denote the upper and the lower elite levels reached, respectively (the m_{*t} levels are the same as the m_t levels in the description of the algorithm);
- ρ_t is the splitting control parameter (we chose $\rho_t \equiv \rho$);
- $\widehat{c}_t = N_t/N$ is the estimator of the t-th conditional probability;
- product estimator $\widehat{|\mathcal{X}_t^*|} = |\mathcal{X}| \prod_{i=1}^{t} \widehat{c}_i$ after t iterations.

5.4.1 The 3-Satisfiability Problem (3-SAT)

There are m clauses of length 3 taken from n boolean (or binary) variables $x_1, \ldots, x_n$. A literal of the j-th variable is either TRUE ($x_j = 1$) or FALSE ($x_j = 0 \Leftrightarrow \bar{x}_j = 1$, where $\bar{x}_j = \mathrm{NOT}(x_j)$). A clause is a disjunction of literals. We assume that all clauses consist of 3 literals. The 3-SAT problem consists of determining if the variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ can be assigned in a such way as to make all clauses TRUE. More formally, let $\mathcal{X} = \{0,1\}^n$ be the set of all configurations of the n variables, and let $C_i : \mathcal{X} \to \{0,1\}$, be the m clauses. Then define $\phi : \mathcal{X} \to \{0,1\}$ by

$$\phi(\boldsymbol{x}) = \bigwedge_{i=1}^{m} C_i(\boldsymbol{x}).$$

The original 3-SAT problem is to find a configuration of the x_j variables for which $\phi(\boldsymbol{x}) = 1$. In this work we are interested in the total number of such configurations (or feasible solutions). Then as discussed in Section 5.2, $\mathcal{X}^*$ denotes the set of feasible solutions. Trivially, there are $|\mathcal{X}| = 2^n$ configurations.

The 3-SAT problems can also be converted into the family of decision problems (5.5) given in Section 5.2. Define the $m \times n$ matrix $\boldsymbol{A}$ with entries $a_{ij} \in \{-1,0,1\}$ by

$$a_{ij} = \begin{cases} -1 & \text{if } \bar{x}_j \in C_i, \\ 0 & \text{if } x_j \notin C_i \text{ and } \bar{x}_j \notin C_i, \\ 1 & \text{if } x_j \in C_i. \end{cases}$$

Furthermore, let $\boldsymbol{b}$ be the m-(column) vector with entries $b_i = 1 - |\{j : a_{ij} = -1\}|$. Then it is easy to see that for any configuration $\boldsymbol{x} \in \{0,1\}^n$

$$\boldsymbol{x} \in \mathcal{X}^* \Leftrightarrow \phi(\boldsymbol{x}) = 1 \Leftrightarrow \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}.$$

Below we compare the efficiencies of the classic, the CAP-RECAP, and the extended CAP-RECAP algorithms. Efficiency is measured by the reciprocal of the product of the variance and the computational effort (see, e.g., [58]).

As an example we consider the estimation of $|\mathcal{X}^*|$ for the 3-SAT problem with an instance matrix $\boldsymbol{A}$ of dimension (122×515), meaning $n = 122, m = 515$. In particular Table 5.1 presents the the performance of the splitting Algorithm 5.2.1 based on 10 independent runs using $N = 25,000$ and $\rho = 0.1$, while Table 5.2 shows the dynamics of a run of the Algorithm 5.2.1 for the same data.

Table 5.1: Performance of splitting algorithm for the 3-SAT (122×515) model with $N = 25,000$ and $\rho = 0.1$.

Run	nr. of its.	$\widehat{\lvert\mathcal{X}^*\rvert}$	CPU
1	33	1.41E+06	212.32
2	33	1.10E+06	213.21
3	33	1.68E+06	214.05
4	33	1.21E+06	215.5
5	33	1.21E+06	214.15
6	33	1.47E+06	216.05
7	33	1.50E+06	252.25
8	33	1.73E+06	243.26
9	33	1.21E+06	238.63
10	33	1.88E+06	224.36
Average	33	1.44E+06	224.38

The relative error, denoted by RE is $1.815E - 01$. Notice that the relative error of a random variable Z is calculated by the standard formula, namely

$$RE = S/\widehat{\ell},$$

where

$$\widehat{\ell} = \frac{1}{N}\sum_{i=1}^{N} Z_i, \quad S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(Z_i - \widehat{\ell})^2.$$

Table 5.2: Dynamics of a run of the splitting algorithm for the 3-SAT (122×515) model using $N = 25,000$ and $\rho = 0.1$.

t	$\widehat{\lvert\mathcal{X}_t^*\rvert}$	N_t	$N_t^{(\mathrm{s})}$	m_t^*	m_{*t}	$\widehat{c}_t$
1	6.53E+35	3069	3069	480	460	1.23E-01
2	8.78E+34	3364	3364	483	467	1.35E-01
3	1.15E+34	3270	3270	484	472	1.31E-01
4	1.50E+33	3269	3269	489	476	1.31E-01
5	2.49E+32	4151	4151	490	479	1.66E-01
6	3.37E+31	3379	3379	492	482	1.35E-01
7	3.41E+30	2527	2527	494	485	1.01E-01
8	6.19E+29	4538	4538	495	487	1.82E-01
9	9.85E+28	3981	3981	497	489	1.59E-01
10	1.31E+28	3316	3316	498	491	1.33E-01
11	1.46E+27	2797	2797	501	493	1.12E-01
12	4.61E+26	7884	7884	501	494	3.15E-01
13	1.36E+26	7380	7380	501	495	2.95E-01
14	3.89E+25	7150	7150	502	496	2.86E-01
15	1.06E+25	6782	6782	505	497	2.71E-01
16	2.69E+24	6364	6364	503	498	2.55E-01
17	6.42E+23	5969	5969	504	499	2.39E-01
18	1.42E+23	5525	5525	506	500	2.21E-01
19	3.03E+22	5333	5333	505	501	2.13E-01
20	5.87E+21	4850	4850	506	502	1.94E-01
21	1.06E+21	4496	4496	507	503	1.80E-01
22	1.71E+20	4061	4061	507	504	1.62E-01
23	2.50E+19	3647	3647	509	505	1.46E-01
24	3.26E+18	3260	3260	510	506	1.30E-01
25	3.62E+17	2778	2778	510	507	1.11E-01
26	3.68E+16	2539	2539	510	508	1.02E-01
27	3.05E+15	2070	2070	511	509	8.28E-02
28	2.17E+14	1782	1782	512	510	7.13E-02
29	1.21E+13	1398	1398	513	511	5.59E-02
30	5.00E+11	1030	1030	513	512	4.12E-02
31	1.49E+10	743	743	514	513	2.97E-02
32	2.39E+08	402	402	515	514	1.61E-02
33	1.43E+06	150	150	515	515	6.00E-03

We increased the sample size at the last two iterations from $N = 25,000$ to $N = 100,000$ to get a more accurate estimator.

As can be seen from Table 5.1, the estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{product}} > 10^6$, hence for this instance the extended CAP-RECAP Algorithm 5.3.2 can also be used. We shall show that the relative error (RE) of the extended CAP-RECAP estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{ecap}}$ is less than that of $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{product}}$. Before doing so we need to find the extended 3-SAT instance matrix $(122 \times 515 + \tau)$, where τ is the number of auxiliary clauses. Applying the extended CAP-RECAP Algorithm 5.3.2 with $c^* = 0.05$, we found that $\tau = 5$ and thus the extended instance matrix is (122×520). Recall that the cardinality $|\mathcal{X}_{T+\tau}|$ of the extended (122×520) model should be manageable by the regular CAP-RECAP algorithm 5.3.1, that is, we assumed that $|\mathcal{X}_{T+\tau}| < 10^6$. Indeed, Table 5.3 presents the performance of the regular CAP-RECAP algorithm for that extended (122×520) model. Here we used again $\rho = 0.1$. As for the sample size, we took $N = 1,000$ until iteration 28 and then switched to $N = 100,000$. The final CAP-RECAP estimator is obtained by taking two equal samples, each of size $N = 70,000$ at the final subset $\mathcal{X}_{T+\tau}$.

Table 5.3: Performance of the regular CAP-RECAP for the extended (122×520) model with $N = 1,000$ (up to iteration 28), $N = 100,000$ (from iteration 29), $N = 70,000$ (for the two capture-recapture draws), and $\rho = 0.1$.

Run	nr. of its.	$\widehat{\lvert\mathcal{X}^*\rvert}_{\text{cap}}$	CPU
1	34	5.53E+04	159.05
2	35	5.49E+04	174.46
3	35	5.51E+04	178.08
4	34	5.51E+04	166.36
5	34	5.52E+04	159.36
6	33	5.52E+04	152.38
7	33	5.54E+04	137.96
8	34	5.50E+04	157.37
9	35	5.51E+04	179.08
10	34	5.51E+04	163.7
Average	34.1	5.51E+04	162.78

The relative error of $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{cap}}$ over 10 runs is $2.600E-03$.

Next we compare the efficiency of the regular CAP-RECAP algorithm 5.3.1 for the extended (122×520) model (as per Table 5.3) with that of the splitting algorithm 5.2.1 for this model. Table 5.4 presents the performance of the splitting algorithm for $\rho = 0.1$ and $N = 100,000$.

It readily follows that the relative error of the regular CAP-RECAP is about 30 times less than that of splitting. Notice in addition that the CPU time of CAP-RECAP is about 6 times less than that of splitting. This is so since the total sample size of the former is about 6 time less than of the latter. Using that the relative error involves the square root of the variance, the efficiency improvement by CAP-RECAP is about $5,400$.

Table 5.4: Performance of splitting algorithm for the 3-SAT (122×520) model with $N = 100,000$ and $\rho = 0.1$.

Run	nr. of its.	$\widehat{\lvert\mathcal{X}^*\rvert}$	CPU
1	34	6.03E+04	900.28
2	34	7.48E+04	904.23
3	34	4.50E+04	913.31
4	34	5.99E+04	912.27
5	34	6.03E+04	910.44
6	33	4.94E+04	898.91
7	34	5.22E+04	931.88
8	34	5.74E+04	916.8
9	34	5.85E+04	919.63
10	34	5.72E+04	927.7
Average	33.9	5.75E+04	913.54

The relative error of $\widehat{\lvert\mathcal{X}^*\rvert}$ over 10 runs is $1.315E - 01$.

With these results at hand we can proceed with the extended CAP-RECAP and compare its efficiency with splitting (as per Table 5.1) for the instance matrix (122×515). Table 5.5 presents the performance of the extended CAP-RECAP estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{ecap}}$ for the (122×515) model. We set again $\rho = 0.1$. Regarding the sample size we took $N = 1,000$ for the first 31 iterations and then switched to $N = 100,000$ until reaching the level $m = 515$. Recall that the level $m+\tau = 520$ and the corresponding CAP-RECAP estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{cap}}$ was obtained from the set $\mathcal{X}_T = \mathcal{X}_{515}$ by adding $\tau = 5$ more auxiliary clauses. In this case we used for $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{cap}}$ two equal samples each of length $N = 100,000$.

Comparing the results of Table 5.1 with that of Table 5.5 it is readily seen that the extended CAP-RECAP estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{ecap}}$ outperforms the splitting one $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{product}}$ in terms of efficiency. In particular, we have that both RE and CPU times of the former are about 1.6 times less than of the latter. This means that the overall efficiency improvement obtained by $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{ecap}}$ versus $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{product}}$ is about $1.6^2 \cdot 1.6 \approx 4$. Note finally that the total number of samples used in the extended CAP-RECAP estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{ecap}}$ is about $N = 31 \times 1,000 + 5 \times 100,000 = 531,000$, while in its counterpart - the splitting estimator $\widehat{\lvert\mathcal{X}^*\rvert}_{\text{product}}$ is about $N = 33 \times 25,000 = 825,000$.

Table 5.5: Performance of the extended CAP-RECAP estimator $\widehat{|\mathcal{X}^*|}_{\text{ecap}}$ for the (122×515) with $N = 1,000$ (up to iteration 31), $N = 100,000$ (from iteration 31 and the two capture-rcapture draws), $c^* = 0.05$, and $\rho = 0.1$.

Run	nr. its.	$\widehat{\lvert\mathcal{X}^*\rvert}_{\text{ecap}}$	CPU
1	33	1.73E+06	138.99
2	34	1.59E+06	154.64
3	34	1.55E+06	161.78
4	33	1.20E+06	163.53
5	34	1.69E+06	143.84
6	34	1.81E+06	151.1
7	34	1.29E+06	174.08
8	34	1.40E+06	143.27
9	33	1.66E+06	171.07
10	34	1.30E+06	154.71
Average	33.7	1.52E+06	155.70

The relative error of $\widehat{|\mathcal{X}^*|}_{\text{ecap}}$ over 10 runs is $1.315E-01$.

5.4.2 Random graphs with prescribed degrees

Random graphs with given vertex degrees have attained attention as a model for real-world complex networks, including World Wide Web, social networks and biological networks. The problem is basically finding a graph $G = (V, E)$ with n vertices, given the degree sequence $\boldsymbol{d} = (d_1, \ldots, d_n)$ formed of nonnegative integers. Following Blitzstein and Diaconis [6], a finite sequence $(d_1, \ldots, d_n)$ of nonnegative integers is called graphical if there is a labeled simple graph with vertex set $\{1, \ldots, n\}$ in which vertex i has degree d_i. Such a graph is called a realization of the degree sequence $(d_1, \ldots, d_n)$. We are interested in the total number of realizations for a given degree sequence, hence $\mathcal{X}^*$ denotes the set of all graphs $G = (V, E)$ with the degree sequence $(d_1, \ldots, d_n)$.

Similar to (5.5) for SAT we convert the problem into a counting problem. To proceed consider the complete graph K_n of n vertices, in which each vertex is connected with all other vertices. Thus the total number of edges in K_n is $m = n(n-1)/2$, labeled $e_1, \ldots, e_m$. The random graph problem with prescribed degrees is translated to the problem of choosing those edges of K_n such that the resulting graph G matches the given sequence $\boldsymbol{d}$. Set $x_i = 1$ when e_i is chosen, and $x_i = 0$ otherwise, $i = 1, \ldots, m$. In order that such an assignment $\boldsymbol{x} \in \{0,1\}^m$ matches the given degree sequence $(d_1, \ldots, d_n)$, it holds necessarily that $\sum_{j=1}^{m} x_j = \frac{1}{2} \sum_{i=1}^{n} d_i$, since this is the total number of edges. In other words, the configuration space is

$$\mathcal{X} = \left\{ \boldsymbol{x} \in \{0,1\}^m : \sum_{j=1}^{m} x_j = \tfrac{1}{2} \sum_{i=1}^{n} d_i \right\}.$$

Let $\boldsymbol{A}$ be the incidence matrix of K_n with entries

$$a_{ij} = \begin{cases} 0 & \text{if } v_i \notin e_j \\ 1 & \text{if } v_i \in e_j. \end{cases}$$

It is easy to see that whenever a configuration $\boldsymbol{x} \in \{0,1\}^m$ satisfies $\boldsymbol{Ax} = \boldsymbol{d}$, the associated graph has degree sequence $(d_1, \ldots, d_n)$. We conclude that the problem set is represented by

$$\mathcal{X}^* = \{\boldsymbol{x} \in \mathcal{X} : \boldsymbol{Ax} = \boldsymbol{d}\}.$$

We first present a small example as illustration. Let $\boldsymbol{d} = (2,2,2,1,3)$ with $n = 5$, and $m = 10$. After ordering the edges of K_5 lexicographically, the corresponding incidence matrix is given as

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

It is readily seen that the following $\boldsymbol{x} = (0,0,1,1,1,0,1,0,1,0)'$, and $\boldsymbol{x} = (1,0,0,1,1,0,0,0,1,1)'$ present two solutions of this example.

For the random graph problem we define the score function $S : \mathcal{X} \to \mathbb{Z}_-$ by

$$S(\boldsymbol{x}) = -\sum_{i=1}^{n} |\deg(v_i) - d_i|,$$

where $\deg(v_i)$ is the degree of vertex i under the configuration $\boldsymbol{x}$. Each configuration that satisfies the degree sequence will have a performance function equal to 0.

The implementation of the Gibbs sampler for this problem is slightly different than for the 3-SAT problem, since we keep the number of edges in each realization fixed to $\sum d_i/2$. Our first algorithm takes care of this requirement and generates a random $\boldsymbol{x} \in \mathcal{X}$.

Algorithm 5.4.1. *Let $(d_1, \ldots, d_n)$ be the prescribed degrees sequence.*

- *Generate a random permutation of $1, \ldots, m$.*

- *Choose the first $\sum d_i/2$ places in this permutation and deliver a vector $\boldsymbol{x}$ having one's in those places.*

The adaptive thresholds in the basic splitting algorithm are negative, increasing to 0:

$$m_1 \leq m_2 \leq \cdots \leq m_{T-1} \leq m_T = 0.$$

The resulting Gibbs sampler (in Step 3 of the basic splitting algorithm starting with a configuration $\boldsymbol{x} \in \mathcal{X}$ for which $S(\boldsymbol{x}) \geq m_t$) can be written as follows.

Algorithm 5.4.2 (Gibbs Algorithm for random graph sampling)**.** *For each edge $x_i = 1$, while keeping all other edges fixed, do:*

1. *Remove x_i from $\boldsymbol{x}$, i.e. make $x_i = 0$.*
2. *Check all possible placements for the edge resulting a new vector $\bar{\boldsymbol{x}}$ conditioning on the performance function $S(\bar{\boldsymbol{x}}) \geq m_t$*
3. *With uniform probability choose one of the valid realizations.*

We will apply the splitting algorithm to two problems taken from [6].

A small problem

For this small problem we have the degree sequence

$$\boldsymbol{d} = (5, 6, \underbrace{1, \ldots, 1}_{11 \text{ ones}}).$$

The solution can be obtained analytically and already given in [6]:

> "To count the number of labeled graphs with this degree sequence, note that there are $\binom{11}{5} = 462$ such graphs with vertex 1 not joined to vertex 2 by an edge (these graphs look like two separate stars), and there are $\binom{11}{4}\binom{7}{5} = 6930$ such graphs with an edge between vertices 1 and 2 (these look like two joined stars with an isolated edge left over). Thus, the total number of realizations of $\boldsymbol{d}$ is 7392."

As we can see from Table 5.6, the algorithm easily handles the problem. Table 5.7 presents the typical dynamics.

Table 5.6: Performance of the splitting algorithm for a small problem using $N = 50,000$ and $\rho = 0.5$.

Run	nr. of its.	$\widehat{\lvert\mathcal{X}^*\rvert}$	CPU
1	10	7146.2	15.723
2	10	7169.2	15.251
3	10	7468.7	15.664
4	10	7145.9	15.453
5	10	7583	15.555
6	10	7206.4	15.454
7	10	7079.3	15.495
8	10	7545.1	15.347
9	10	7597.2	15.836
10	10	7181.2	15.612
Average	10	7312.2	15.539

The relative error of $\widehat{\lvert\mathcal{X}^*\rvert}$ over 10 runs is $2.710E-02$.

Table 5.7: Typical dynamics of the splitting algorithm for a small problem using $N = 50,000$ and $\rho = 0.5$ (recall Notation 5.4.1 at the beginning of Section 5.4).

t	$\widehat{\lvert\mathcal{X}_t^*\rvert}$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	$\widehat{c}_t$
1	4.55E+12	29227	29227	-4	-30	0.5845
2	2.56E+12	28144	28144	-4	-18	0.5629
3	1.09E+12	21227	21227	-6	-16	0.4245
4	3.38E+11	15565	15565	-4	-14	0.3113
5	7.51E+10	11104	11104	-4	-12	0.2221
6	1.11E+10	7408	7408	-2	-10	0.1482
7	1.03E+09	4628	4628	-2	-8	0.0926
8	5.37E+07	2608	2608	-2	-6	0.0522
9	1.26E+06	1175	1175	0	-4	0.0235
10	7223.9	286	280	0	-2	0.0057

A large problem

A much harder instance (see [6]) is defined by

$$\boldsymbol{d} = (7,8,5,1,1,2,8,10,4,2,4,5,3,6,7,3,2,7,6,1,2,9,6,1,3,4,6,3,3,3,2,4,4).$$

In [6] the number of such graphs is estimated to be about 1.533×10^{57} Table 5.8 presents 10 runs using the splitting algorithm.

Table 5.8: Performance of the splitting algorithm for a large problem using $N = 100,000$ and $\rho = 0.5$.

Run	nr. its.	$\widehat{\|\mathcal{X}^*\|}$	CPU
1	39	1.66E+57	4295
2	39	1.58E+57	4223
3	39	1.58E+57	4116
4	39	1.53E+57	4281
5	39	1.76E+57	4301
6	39	1.75E+57	4094
7	39	1.46E+57	4512
8	39	1.71E+57	4287
9	39	1.39E+57	4158
10	39	1.38E+57	4264
Average	39	1.58E+57	4253

The relative error of $\widehat{|\mathcal{X}^*|}$ over 10 runs is $8.430E - 02$.

5.4.3 Binary Contingency Tables

Given are two vectors of positive integers $\boldsymbol{r} = (r_1, \dots, r_m)$ and $\boldsymbol{c} = (c_1, \dots, c_n)$ such that $r_i \leq n$ for all i, $c_j \leq n$ for all j, and $\sum_{i=1}^m r_i = \sum_{j=1}^n c_j$. A *binary contingency table* with row sums $\boldsymbol{r}$ and column sums $\boldsymbol{c}$ is a $m \times n$ matrix $\boldsymbol{X}$ of zero-one entries x_{ij} satisfying $\sum_{j=1}^n x_{ij} = r_i$ for every row i and $\sum_{i=1}^m x_{ij} = c_j$ for every column j. The problem is to count all contingency tables.

The extension of the proposed Gibbs sampler for counting the contingency tables is straightforward. We define the configuration space $\mathcal{X} = \mathcal{X}^{(\mathrm{r})} \cup \mathcal{X}^{(\mathrm{c})}$ as the space where all column or row sums are satisfied:

$$\mathcal{X}^{(\mathrm{c})} = \left\{ \boldsymbol{X} \in \{0,1\}^{m+n} : \sum_{i=1}^m x_{ij} = c_j \ \forall j \right\},$$
$$\mathcal{X}^{(\mathrm{r})} = \left\{ \boldsymbol{X} \in \{0,1\}^{m+n} : \sum_{j=1}^n x_{ij} = r_i \ \forall i \right\}.$$

Clearly we can sample uniformly at random from $\mathcal{X}$ without any problem. The score function $S : \mathcal{X} \to \mathbb{Z}_-$ is defined by

$$S(\boldsymbol{X}) = \begin{cases} -\sum_{i=1}^m |\sum_{j=1}^n x_{ij} - r_i|, & \text{for } \boldsymbol{X} \in \mathcal{X}^{(\mathrm{c})}, \\ -\sum_{j=1}^n |\sum_{i=1}^m x_{ij} - c_j|, & \text{for } \boldsymbol{X} \in \mathcal{X}^{(\mathrm{r})}, \end{cases}$$

that is, the difference of the row sums $\sum_{j=1}^n x_{ij}$ with the target r_i if the column sums are right, and vice versa.

The Gibbs sampler is very similar to the one in the previous section concerning random graphs with prescribed degrees.

Algorithm 5.4.3 (Gibbs algorithm for random contingency tables sampling)**.** *Given a matrix realization $\boldsymbol{X} \in \mathcal{X}^{(c)}$ with score $S(\boldsymbol{X}) \geq m_t$. For each column j and for each 1-entry in this column ($x_{ij} = 1$) do:*

1. *Remove this 1, i.e. set $x'_{ij} = 0$.*
2. *Check all possible placements for this 1 in the given column j conditioning on the performance function $S(\boldsymbol{X}') \geq m_t$ ($\boldsymbol{X}'$ is the matrix resulting by setting $x'_{ij} = 0$, $x'_{i'j} = 1$ for some $x_{i'j} = 0$, and all other entries remain unchanged).*
3. *Suppose that the set of valid realization is $\mathcal{A} = \{\boldsymbol{X}' | S(\boldsymbol{X}') \geq m_t\}$. (Please note that this set also contains the original realization $\boldsymbol{X}$). Then with probability $\frac{1}{|\mathcal{A}|}$ pick any realization at random and continue with step 1.*

Note that in this way we keep the column sums correct. Similarly, when we started with a matrix configuration with all row sums correct, we execute these steps for each row and swap 1 and 0 per row.

Model 1

The date are $m = 12, n = 12$ with row and column sums

$$\boldsymbol{r} = (2,2,2,2,2,2,2,2,2,2,2,2), \ \boldsymbol{c} = (2,2,2,2,2,2,2,2,2,2,2,2).$$

The true count value is known to be $21,959,547,410,077,200$. Table 5.9 presents 10 runs using the splitting algorithm. Table 5.10 presents a typical dynamics.

Table 5.9: Performance of the splitting algorithm for Model 1 using $N = 50,000$ and $\rho = 0.5$.

Run	nr.its.	$\widehat{\lvert\mathcal{X}^*\rvert}$	CPU
1	7	2.15E+16	4.54
2	7	2.32E+16	4.55
3	7	2.23E+16	4.54
4	7	2.11E+16	4.58
5	7	2.05E+16	4.57
6	7	2.23E+16	4.54
7	7	2.02E+16	4.55
8	7	2.38E+16	4.58
9	7	2.06E+16	4.57
10	7	2.14E+16	4.55
Average	7	2.17E+16	4.56

The relative error of $\widehat{|\mathcal{X}^*|}$ over 10 runs is $5.210E-02$.

Table 5.10: Typical dynamics of the splitting algorithm for Model 1 using $N = 50,000$ and $\rho = 0.5$.

t	$\widehat{\lvert\mathcal{X}_t^*\rvert}$	N_t	$N_t^{(s)}$	m_t^*	m_{*t}	$\widehat{c}_t$
1	4.56E+21	13361	13361	-2	-24	0.6681
2	2.68E+21	11747	11747	-2	-12	0.5874
3	1.10E+21	8234	8234	-2	-10	0.4117
4	2.76E+20	5003	5003	-2	-8	0.2502
5	3.45E+19	2497	2497	0	-6	0.1249
6	1.92E+18	1112	1112	0	-4	0.0556
7	2.08E+16	217	217	0	-2	0.0109

Model 2

Darwin's Finch Data from Yuguo Chen, Persi Diaconis, Susan P. Holmes, and Jun S. Liu: $m = 12, n = 17$ with row and columns sums

$\boldsymbol{r} = (14, 13, 14, 10, 12, 2, 10, 1, 10, 11, 6, 2)$, $\boldsymbol{c} = (3, 3, 10, 9, 9, 7, 8, 9, 7, 8, 2, 9, 3, 6, 8, 2, 2)$.

The true count value is known to be $67,149,106,137,567,600$. Table 5.11 presents 10 runs using the splitting algorithm.

Table 5.11: Performance of the splitting algorithm for Model 2 using $N = 200,000$ and $\rho = 0.5$.

Run	nr. its.	$\widehat{\lvert\mathcal{X}^*\rvert}$	CPU
1	24	6.16E+16	246.83
2	24	6.50E+16	244.42
3	24	7.07E+16	252.71
4	24	7.91E+16	247.36
5	24	6.61E+16	260.99
6	24	6.77E+16	264.07
7	24	6.59E+16	269.86
8	24	6.51E+16	273.51
9	24	7.10E+16	272.49
10	24	5.91E+16	267.23
Average	24	6.71E+16	259.95

The relative error of $\widehat{|\mathcal{X}^*|}$ over 10 runs is $7.850E-02$.

5.5 Concluding Remarks

In this paper we applied the splitting method to several well-known counting problems, like 3-SAT, random graphs with prescribed degrees and binary contingency tables. While implementing the splitting algorithm, we discussed several MCMC algorithms and in particular the Gibbs sampler. We show how to incorporate the classic capture-recapture method in the splitting algorithm in order to obtain a low variance estimator for the desired counting quantity. Furthermore, we presented an extended version of the capture-recapture algorithm, which is suitable for problems with a larger number of feasible solutions. We finally presented numerical results with the splitting and capture-recapture estimators, and showed the superiority of the latter.

Acknowledgement

The authors are grateful to two anonymous referees for their comments and suggestions.

Chapter 6

Permutational Methods for Performance Analysis of Stochastic Flow Networks

Ilya Gertsbakh [a], Reuven Rubinstein[b] [1],
Yoseph Shpungin [c] and Radislav Vaisman [d]

[a] Department of Mathematics,
Ben Gurion University, Beer-Sheva 84105, Israel
elyager@bezeqint.net

[b] Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology, Haifa, Israel
ierrr01@ie.technion.ac.il,

[c] Department of Software Engineering,
Shamoon College of Engineering, Beer-Sheva 84105, Israel
yosefs@sce.ac.il

[d] Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology, Haifa, Israel
slvaisman@gmail.com

[1]Corresponding author (http://iew3.technion.ac.il/Home/ Users/ierrr01.phtml)

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

Abstract

In this paper we show how the permutation Monte Carlo method, originally developed for reliability networks, can be successfully adapted for stochastic flow networks, and in particular for estimation of the probability that the maximal flow in such a network is above some fixed level, called the *threshold.* A stochastic flow network is defined as one, where the edges are subject to random failures. A failed edge is assumed to be erased (broken) and, thus, not able to deliver any flow. We consider two models; one where the edges fail with the same failure probability and another where they fail with different failure probabilities. For each model we construct a different algorithm for estimation of the desired probability; in the former case it is based on the well known notion of the D-spectrum and in the later one - on the permutational Monte Carlo. We discuss the convergence properties of our estimators and present supportive numerical results.

Keywords. Permutation, Performance Analysis, Reliability Networks, Stochastic Flow Networks.

Contents

6.1 Introduction

The purpose of this paper is to investigate the probabilistic properties of the maximal flow in a network with randomly failing edges. Edge failure means that it is erased (broken) and is not able to deliver any flow. Because of the randomness of these failures, the maximum flow from the source to the sink is also a random variable. We call such a network, the *stochastic flow network*.

Before proceeding let us define formally our network. The network is a triple $\mathbf{N} = (\mathbf{V}, \mathbf{E}, \mathbf{C})$, where $\mathbf{V}$ is the set of vertices (nodes), $|\mathbf{V}| = n$, $\mathbf{E}$ is the set of edges, $|\mathbf{E}| = m$, and $\mathbf{C}$ is the set of edge capacities $\mathbf{C} = (c_1; \ldots; c_m)$, where c_i is an item of type $c_i = \{(a, b), w_i\}$, where w_i is the maximal flow capacity from node a to node b along the directed edge (a, b). In case that there are directed edges from a to b and from b to a, these edges get different numbers, say r and s, and $\mathbf{C}$ will contain two items of the above type: $c_r = \{(a, b), w_r\}$ and $c_s = \{(b, a), w_s\}$.

Denote by s and t the source and sink nodes of the network. Denote next by M the maximum flow when all edges are operational. Note that there exist an extensive literature with several fast polynomial time algorithms for finding the maximum flow in networks with perfect edges [75]. Unless stated otherwise we shall use the Goldberg-Rao algorithm [43] with the complexity $\mathcal{O}(min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E| \log(\frac{|V|^2}{|E|}) \log(U))$, where U is the largest edge capacity in the network.

The main goal of this paper is to obtain the probability that the maximal flow in a stochastic flow network is below some fixed level $\Phi = \gamma M$, $(\gamma < 1)$, called the *threshold*. We say that the network is in *DOWN* state if its maximal flow is below Φ, otherwise it is in *UP* state. It is important to note that if the maximal flow drops to zero, the network (s and t nodes) become disconnected. Therefore, the flow model can be viewed as a generalization of the $s - t$ connectivity in the classic reliability model [35], which is based on the *permutational Monte Carlo* (PMC) method.

There exists a vast literature on stochastic flow networks with a number of clever algorithms. For a good survey see [60] and the references therein. It is not our goal to evaluate the PMC method for flow networks versus its alternatives, but rather to show the beauty of this method, discuss its convergence properties and present supportive numerical results.

We consider the following two models:

Model 1. All edges fail independently with the same failure probability q. For this model our goal is to find the probability $\mathbb{P}(DOWN; q)$ that the network is $DOWN$ as a function of q, $0 < q \leq 1$.

Model 2. All edges fail independently, with arbitrary failure probabilities $q_1, \ldots, q_m$. For this model our goals is to find the probability $\mathbb{P}(DOWN; \boldsymbol{q})$ for fixed vector $\boldsymbol{q} = (q_1, \ldots, q_m)$.

The rest of the paper is organized as follows. In Section 6.2 we consider **Model 1** and derive a closed expression for the function $\mathbb{P}(DOWN; q)$. It is based on the D-spectrum, which represents the distribution of the so-called *anchor* for randomly generated permutations and which is approximated via a Monte Carlo procedure. The D-spectrum has been widely used in the literature on stochastic network reliability analysis [26, 25, 38, 33, 34, 35, 36, 37]. Numerically it is identical with the so-called *signature* introduced in [87] and later on independently by [26] under the name *Internal Distribution* (ID). Here we will also consider the main properties of the D-spectrum and its usefulness to stochastic flow networks, as well as present numerical results for a flow network.

Section 6.3 is devoted to **Model 2** and in particular to estimation of $\mathbb{P}(DOWN; \mathbf{q})$ for non equal components of the vector $\boldsymbol{q}$. Here we introduce a specially designed *evolution*, also called *construction or birth* process, first introduced in [26] and then widely used in network reliability analysis [35, 53].

We describe in detail the procedure of obtaining permutations for this process and construct an efficient Monte Carlo estimator for $\mathbb{P}(DOWN; \mathbf{q})$. A numerical example concludes this section.

Section 6.4 extends **Model 2** to the case of random capacities. We show that although the estimator of $\mathbb{P}(DOWN; \mathbf{q})$ is not as accurate as in the case of the fixed capacity vector $\boldsymbol{C}$, the algorithm derived for fixed $\boldsymbol{C}$ is also applicable here. A numerical example supporting our findings is presented as well.

Section 6.5 presents concluding remarks and some directions for further research.

6.2 Max Flow with Equal Failure Probabilities

Here we derive an analytic expression for $\mathbb{P}(DOWN, q)$ while considering the **Model 1**, that is for the one with equal failure probabilities of the edges. Our derivation is based on the notion of D-spectrum [35].

6.2.1 D-spectrum and its Properties

Denote network edges by $e_1, e_2, \ldots, e_m$. Suppose that all edges are initially operational and thus the network is UP. Let $\boldsymbol{\pi} = (e_{i_1}, \ldots, e_{i_m})$ be a permutation of the network edges. Then the D-spectrum algorithm can be described as follows:

Algorithm 6.2.1. (**D-spectrum Algorithm**) Given a network and a set of terminal nodes s and t, execute the following steps.

1. Start turning the edges down (erase them) moving through the permutation from left to right, and check the state (*UP*/*DOWN*) of the network after each step.

2. Find the position of the first edge i_r when the network switches from *UP* to *DOWN*. This can be done, for example, by using the Goldberg-Rao maximum flow polynomial algorithm (oracle) [13]. The serial number r of this edge of $\boldsymbol{\pi}$ is called the *anchor* of $\boldsymbol{\pi}$ and denoted as $r(\boldsymbol{\pi})$.

3. Consider the set of all $m!$ permutations and assign to each of them the probability $1/m!$

4. Define the event $A(i) = \{r(\boldsymbol{\pi}) = i\}$ and denote $f_i = \mathbb{P}(A(i))$. Obviously,

$$f_i = \frac{\#\text{ of permutations with } r(\boldsymbol{\pi}) = i}{m!}. \tag{6.1}$$

   The set of $\{f_i\ , i = 1, \ldots, m\}$ defines a proper discrete density function. It is called the *density D-spectrum*, where "D" stands for "destruction".

5. Define the cumulative D-spectrum or simply *D-spectrum* as

$$F(x) = \sum_{i=1}^{x} f_i,\ \ x = 1, \ldots, m. \tag{6.2}$$

Note that Algorithm 6.2.1 can be speeded up by using a bisection procedure for turning edges down instead of the sequential one-by-one. This is implemented in our main permutational Algorithm 6.2.2 below.

The nice feature of the D-spectrum is that once $F(x)$ is available one can calculate directly the sought failure probability $\mathbb{P}(DOWN; \boldsymbol{q})$ (see (6.4) below). Indeed, denote by $\mathcal{N}(x)$ the number of network failure sets of size x. Note that each such set is a collection of x edges whose failure result in *DOWN* state of the network. So, if the network is *DOWN* when edges $A_x = \{e_{j_1}, \ldots, e_{j_x}\}$ are down (erased), and all other edges are operational, we say that A_x is a failure set of size x. It is readily seen that

$$\mathcal{N}(x) = F(x)\binom{m}{x}. \tag{6.3}$$

This statement has a simple combinatorial explanation: $F(x)$ is a fraction of all failure sets of size x among all subsets of size x taken randomly from the set of m components.

From (6.3) we immediately obtain our main result for **Model 1**.

$$\mathbb{P}(DOWN; q) = \sum_{x=1}^{m} \mathcal{N}(x) q^x (1-q)^{m-x}. \tag{6.4}$$

Indeed, (6.4) follows from the facts that

- Network is $DOWN$ if and only if it is in one of its failure states.
- For fixed q each failure set of size x has the probability $\rho_x = q^x(1-q)^{m-x}$.
- All failure sets of size x have the probability $\mathcal{N}(x)\rho_x$.

Example 6.2.1. Figure 6.1 represents a simple directed graph with $n = 3$ nodes denoted by s, b, t (s and t being the source and the sink), $m = 3$ edges denoted by sb, bt, st and a 3-dimensional flow capacity vector $\boldsymbol{C} = (sb, bt, st) = (1, 2, 2)$.

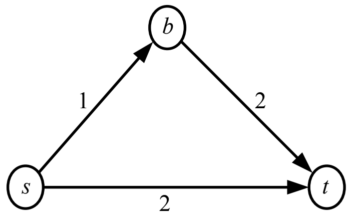


Figure 6.1: A network with $e_1 = (s, b), e_2 = (b, t), e_3 = (s, t)$ and capacity vector $\boldsymbol{C} = (1, 2, 2)$

It is easy to check that the maximal flow is $M = 3$. Assume that $\Phi = 2$, that is the network is $DOWN$ when the max flow drops below level 2. Let us find its D-spectrum. The total number of permutations is 3!=6. If the permutation starts with edge e_3, the anchor is $r = 1$. In permutations $(1, 3, 2)$ and $(2, 3, 1)$ $DOWN$ appears at the second step, $r = 2$. In permutations having e_3 on the third position, the flow becomes 0 at the third step. Thus $f_1 = f_2 = f_3 = 1/3$ and $F(1) = 1/3, F(2) = 2/3, F(3) = 1$. Now by (6.3) we obtain that $\mathcal{N}(1) = 1$, $\mathcal{N}(2) = 2$, and

$\mathcal{N}(3) = 1$. Indeed, in order for the network to be in the *DOWN* state, there is one failure set of size one containing the edge $\{e_3\}$, two failure sets of size two containing the edges $\{e_2, e_3\}$ and $\{e_1, e_3\}$ and one failure set of size 3 containing the edges $\{e_1, e_2, e_3\}$.

Simple calculations of (6.4) yield that $\mathbb{P}(DOWN; q) = q$.

A nice feature of (6.4) is that once $\mathcal{N}(x)$ is available we can calculate analytically the probability $\mathbb{P}(DOWN; q)$ simultaneously for multiple values of q.

Remark 6.2.1. The D-spectrum is a purely combinatorial characteristic of the network. It depends only on its topology, the edge capacity vector $\boldsymbol{C}$ and the threshold value Φ. It does *not* depend on the probabilistic mechanism which governs edge failure.

Note that instead of the destruction process one can use its dual version, the so-called *construction* one. In the latter we start the system at *DOWN* state and turn the edges from down to up one-by-one (or using bisection) until the system becomes *UP*. We shall use the construction process in Section 6.3.

Since network failure probability is a monotone function of its component reliability, we immediately obtain the following

Corollary Let edges fail independently, and edge i fails with probability q_i. Suppose that for all i, $q_i \in [q_{min}, q_{max}]$. Then, obviously,

$$\mathbb{P}(DOWN; q_1, \dots, q_m) \in [\mathbb{P}(DOWN; q_{min}), \mathbb{P}(DOWN; q_{max})] \qquad (6.5)$$

This corollary may be useful for the case where exact information about edge failure probabilities is not available and the only statement we can be made is that the edges fail independently and that their failure probabilities lie within some known interval.

6.2.2 Estimation of D-spectrum and $\mathbb{P}(DOWN; q)$

For $m \leq 10$, the total number of permutations $m!$ is not too large, and the probabilities f_i, $i = 1, \dots, m$ and $\mathbb{P}(DOWN; q)$ might be computed by full enumeration. For $m > 10$ we need to resort to Monte Carlo simulation. In our numerical examples below we shall show that with a sample size of $N = 10^6$ of permutations one can estimate $\mathbb{P}(DOWN : q)$ with relative error not exceeding 2% for flow networks with the number of edges $m = 200 - 300$. Note that the most time consuming part of the simulation process is to check *after edge destruction* whether or not the system switches from *UP* to *DOWN*. As mentioned this can be done by a maximum flow algorithm (oracle), and in particular by the Goldberg-Rao algorithm, which finds the location of permutation anchor in $\mathcal{O}(min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E| \log(\frac{|V|^2}{|E|}) \log(U))$ operations.

The Monte Carlo estimators of $F(x)$ and $\mathbb{P}(DOWN;q)$ is straightforward. Basically we apply Algorithm 6.2.1 N times. During each replication we find the anchor $r(\boldsymbol{\pi})$ of $\boldsymbol{\pi}$. In analogy to (6.1) we estimate the density $f(x),\ \ x=1,\dots,m$ as follows

$$\widehat{f}(x) = \frac{\#\text{ of permutations with } r(\boldsymbol{\pi}) = x}{N}. \tag{6.6}$$

Note that (6.6) differs from (6.1) that $m!$ is replaced by N. Note also that $\widehat{f}(x)$ represents a histogram of $f(x)$ in (6.1).

The corresponding estimators of $F(x)$ and $\mathbb{P}(DOWN;q)$ (see (6.2) and (6.4)) are

$$\widehat{F}(x) = \sum_{i=1}^{x} \widehat{f}_i,\ \ x = 1,\dots,m \tag{6.7}$$

and

$$\widehat{\mathbb{P}}(DOWN;q) = \sum_{x=1}^{m} \widehat{F}(x) \binom{m}{x} q^x (1-q)^{m-x}, \tag{6.8}$$

respectively.

Below we present our main algorithm for estimating $F(x)$ and $\mathbb{P}(DOWN;q)$.

Algorithm 6.2.2. (Main D-spectrum Algorithm for Estimating $F(x)$ and $\mathbb{P}(DOWN;q)$)

Given a network and a set of terminal nodes s and t, execute the following steps.

1. Simulate a random permutation $\boldsymbol{\pi} = (\pi_1,\dots,\pi_m)$ of the edges $1,\dots,m$.

2. Set $low = 1$ and $high = m$

3. Set $b = low + \lceil \frac{high - low}{2} \rceil$

4. Consider $\pi_1,\dots,\pi_b$ and $\pi_1,\dots,\pi_{b+1}$ and use Goldberg-Rao algorithm to check if the network changed its state from *UP* to *DOWN* at index $b+1$ and b respectively.
   If so, denote by $r = r(\boldsymbol{\pi})$ the final number of the *anchor* of $\boldsymbol{\pi}$, corresponding to the non-operational network , output $r(\boldsymbol{\pi}) = b$ as the anchor at which the network is in *DOWN* state and go to step 5.
   If the network state at b is still *UP* set $high = b$ else, if for both b and $b+1$ the network is in the *DOWN* state, set $low = b$ and repeat step 3.

5. Output $r(\boldsymbol{\pi}) = b$ as the anchor at which the network is in *DOWN* state.

6. Repeat steps 1-5 N times and deliver $\widehat{F}(x)$ and $\widehat{\mathbb{P}}(DOWN;q)$ as per (6.7) and (6.8), respectively.

6.2.3 Numerical Results

Below we present simulation results for the D-spectrum and $\mathbb{P}(DOWN; q)$ for the following two models (i) *dodecahedron graph* and (ii) *Erdos - Renyi* graph.

(i) *Dodecahedron graph* with $|V| = 20$, $|E| = 54$ is depicted in Figure 6.2. We set $s = 1$ ant $t = 10$.

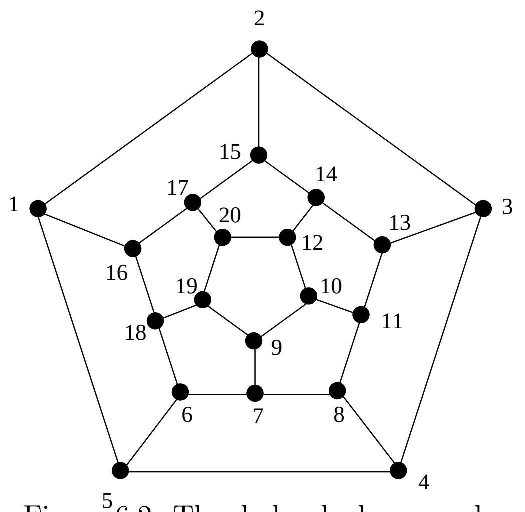


Figure 6.2: The dodecahedron graph.

Table 6.1 presents the values of the edge capacities $c_1, \dots, c_{54}$. They where generated (for each edge independently) using a discrete uniform pdf $\mathcal{U}(5, 10)$.

Table 6.1: Edge capacities for the dodecahedron graph

$e = (a, b)$	$c(e)$	$e = (a, b)$	$c(a, b)$	$e(a, b)$	$c(a, b)$	$e = (a, b)$	$c(a, b)$
(1,2)	5	(18,6)	8	(12,10)	7	(19,18)	5
(1,16)	6	(18,19)	10	(9,10)	7	(7,6)	8
(1,5)	5	(6,7)	8	(3,2)	5	(8,4)	9
(2,3)	9	(4,8)	5	(15,2)	7	(14,13)	7
(2,15)	9	(13,14)	7	(17,16)	8	(11,13)	7
(16,17)	6	(13,11)	6	(18,16)	7	(12,14)	6
(16,18)	7	(14,12)	6	(6,5)	8	(19,20)	5
(5,6)	6	(20,19)	5	(4,5)	8	(12,20)	9
(5,4)	8	(20,12)	10	(4,3)	9	(9,19)	9
(3,4)	10	(19,9)	9	(13,3)	5	(8,7)	6
(3,13)	6	(7,8)	7	(17,15)	7	(9,7)	6
(15,17)	10	(7,9)	9	(14,15)	8	(11,8)	7
(15,14)	7	(8,11)	9	(20,17)	6		
(17,20)	6	(11,10)	8	(6,18)	8		

Using the Goldberg-Rao algorithm we found that the maximum flow with the perfect edges equals $M = 16$. For our simulation studies we set the threshold level $\Phi = 14$. In all our experiments below we took $N = 50,000$ samples.

Table 6.2 presents the D-spectrum estimator $\widehat{F}(x)$ as function of x for the dodecahedron graph with $\Phi = 14$ based on $N = 5 \cdot 10^4$ replications.

Table 6.2: D-spectrum estimator $\widehat{F}(x)$ for the dodecahedron graph with $\Phi = 14$

x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$
1	0.0547	15	0.8899	29	0.99999	43	1
2	0.1148	16	0.9199	30	0.99999	44	1
3	0.1780	17	0.9435	31	0.99999	45	1
4	0.2475	28	0.9606	32	1	46	1
5	0.3212	19	0.9732	33	1	47	1
6	0.3993	20	0.9822	34	1	48	1
7	0.4804	21	0.9882	35	1	49	1
8	0.5592	22	0.9924	36	1	50	1
9	0.6359	23	0.9952	37	1	51	1
10	0.7072	24	0.9969	38	1	52	1
11	0.7701	25	0.9982	39	1	53	1
12	0.8263	26	0.9990	40	1	54	1
13	0.8721	27	0.9995	41	1		
14	0.9086	28	0.9997	42	1		

It follows from Table 6.2 that the network fails with probability equal at least 0.7, 0.9 and 0.99, when the number x of failed edges exceeds 10, 14, and 22, respectively. It fails with probability one when x exceeds 31.

Recall that once $\widehat{F}(x)$ is available one can calculate analytically the estimator $\widehat{\mathbb{P}}(DOWN; q)$ of the true probability $\mathbb{P}(DOWN; q)$ for any q by applying (6.8).

Table 6.3 presents $\widehat{\mathbb{P}}(DOWN; q)$ for different values of q. It also present the corresponding relative error RE based on $K = 10$ independent runs of the entire algorithm. The RE was calculated as

$$RE = \frac{S}{\widetilde{\ell}}, \tag{6.9}$$

where

$$\widehat{\ell} = \widehat{\mathbb{P}}(DOWN; q),\ S^2 = \frac{1}{K-1}\sum_{i=1}^{K}(\widehat{\ell}_i - \widetilde{\ell})^2 \text{ and } \widetilde{\ell} = \frac{1}{\mathrm{K}}\sum_{\mathrm{i}=1}^{\mathrm{K}}\widehat{\ell}_{\mathrm{i}}.$$

Table 6.3: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the dodecahedron graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; q)$	2.99E-06	2.99E-05	2.99E-04	3.00E-03	3.05E-02	3.57E-01	5.54E-01
RE	1.84E-02	2.08E-02	1.84E-02	1.79E-02	1.41E-02	4.50E-03	2.96E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN;q)$ equals to 6.6 seconds.
It follows from Table 6.3 that

- In order to guarantee network reliability $1-\widehat{\mathbb{P}}(DOWN;q)$ =0.9, the probability q of edges failure must not exceed 0.1.
- A sample $N = 5 \cdot 10^4$ guaranties relative error $\leq 1.9\%$ for all q scenarios.

(ii) *Erdos - Renyi* graph, named for Paul Erdos and Alfred Renyi. Our graph was generated according to what is called the $G(n,p)$ Erdos - Renyi random graph [27].

In the $G(n,p)$ model, (n is fixed) 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 the same probability

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

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. Note that the expected number of edges in $G(n,p)$ is $p\binom{n}{2}$, and each vertex has expected degree $p(n-1)$. Clearly as p increases from 0 to 1, the model becomes more dense in the sense that is it is more likely that it will include graphs with more edges than less edges.

We considered an instance of Erdos-Renyi $G(n,p)$ random graph with $p=0.1$ and $|V|=35$ vertices. While generating it we obtained $|E|=109$ connected edges. We set $s=1$ and $t=35$.

Similar to the dodecahedron graph, each capacity $c_1,\ldots,c_{109}$ was generated independently using a discrete uniform $\mathcal{U}(5,10)$ pdf. Using the Goldberg-Rao algorithm we found that the maximum flow with the perfect edges equals $M=30$. For our simulation studies we set the threshold level $\Phi=27$. Again, as before, we set $N=50,000$.

Tables 6.4 and 6.5 present data similar to these of Tables 6.2 and 6.3 for the above Erdos-Renyi graph with $|V|=35$ vertices and $|E|=109$. Note again that each value of $\widehat{\mathbb{P}}(DOWN;q)$ and the corresponding values RE were calculated based on 10 independent runs. The results of the tables are self explanatory.

Table 6.4: The estimator $\widehat{F}(x)$ of the D-spectrum for the Erdos-Renyi graph with $\Phi = 27$

x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$	x	$\widehat{F}(x)$
1	0.0354	31	0.8931	61	0.99994	91	1
2	0.0722	32	0.9074	62	0.99996	92	1
3	0.1078	33	0.9200	63	0.99996	93	1
4	0.1436	34	0.9312	64	0.99997	94	1
5	0.1786	35	0.9414	65	0.99997	95	1
6	0.2122	36	0.9504	66	0.99998	96	1
7	0.2457	37	0.9584	67	0.99998	97	1
8	0.2798	38	0.9655	68	1	98	1
9	0.3128	39	0.9716	69	1	99	1
10	0.3469	40	0.9769	70	1	100	1
11	0.3785	41	0.9814	71	1	101	1
12	0.4100	42	0.9851	72	1	102	1
13	0.4420	43	0.9884	73	1	103	1
14	0.4735	44	0.9908	74	1	104	1
15	0.5059	45	0.9928	75	1	105	1
16	0.5362	46	0.9944	76	1	106	1
17	0.5669	47	0.9955	77	1	107	1
18	0.5955	48	0.9967	78	1	108	1
19	0.6239	49	0.9974	79	1	109	1
20	0.6526	50	0.9982	80	1		
21	0.6812	51	0.9987	81	1		
22	0.7076	52	0.9990	82	1		
23	0.7319	53	0.9992	83	1		
24	0.7563	54	0.9995	84	1		
25	0.7797	55	0.9996	85	1		
26	0.8015	56	0.9997	86	1		
27	0.8223	57	0.9998	87	1		
28	0.8422	58	0.9999	88	1		
29	0.8608	59	0.9999	89	1		
30	0.8777	60	0.9999	90	1		

Table 6.5: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the Erdos-Renyi graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	4.05E-06	4.05E-05	4.05E-04	4.04E-03	3.98E-02	3.74E-01	5.45E-01
RE	1.68E-02	1.68E-02	1.67E-02	1.57E-02	9.54E-03	3.36E-03	2.32E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 22.4 seconds.

All numerical experiments were performed on Intel Core $i5$ 650 3.20 GHz CPU having GB RAM.

We also estimated $\mathbb{P}(DOWN; q)$ for the Erdos-Renyi models with several hundreds edges. For example, setting $|V| = 55$ and $p = 0.1$ we generated a random graph with the number of connected edges $|E| = 313$. We found that

- $M = 28$ and we set $\Phi = 25$.

- In order to provide relative error RE $\leq$ 0.02 we need to take the sample size $N = 6 \cdot 10^5$

The CPU time was about 400 seconds.

6.3 Max Flow with Non-Equal Edge Failure Probabilities

The simplest way to estimate $\mathbb{P}(DOWN, \boldsymbol{q})$ for different failure probability vector $\boldsymbol{q} = (q_1, \dots, q_m)$ is to simulate the state vector $\boldsymbol{X} = (X_1, \dots, X_m)$ of edges $i_1, \dots, i_m$ failure from Ber$(\boldsymbol{q})$ distribution with independent components and then calculate for each realization of $\boldsymbol{X} = (X_1, \dots, X_m)$ the maximum flow in the network and the corresponding network state. This naive Monte Carlo procedure is extremely time consuming, however.

To overcome this difficulty we shall adopt here the *evolution* process of Elperin, Gertsbakh and Lomonosov [26] originally developed for reliability network estimation.

6.3.1 Transformation of the Static Flow Model into a Dynamic

The main idea of the evolution process is to replace a failing edge i having a Bernoulli Ber(q_i) distribution by one with an exponentially distributed *birth time* τ_i with birth rate $\lambda_i = -\ln(q_i)$. More specifically, at time $t = 0$ edge i starts its birth process which terminates at some random moment τ_i. Once this edge is "born" it stays forever in up state. The probability that this event happens before time $t = 1$ is

$$\mathbb{P}(\tau_i \leq 1) = 1 - \exp(-\lambda_i \cdot 1) = 1 - \exp(\ln q_i) = 1 - q_i = p_i. \tag{6.10}$$

Thus, if we take a snapshot of the state of all edges at time instant $t = 1$ we can see the static picture in the sense that edge i will be up with probability $p_i = 1 - q_i$ and down with probability q_i.

With this in mind, we can represent the edge birth process as one evolving in time in a form of a sequence of random sequential births

$\{Y_j(k)\}$, $j = 1, 2, \dots, m$, where $Y_j(\cdot)$ is the instant of the $j - th$ birth, and $k,\ k = 1, \dots, m$ is the number of the born edge. Then the whole birth history can be represented by the following sequence

$$0 < Y_1(k_1) < Y_2(k_2) < \dots < Y_j(k_j) < \dots < Y_m(k_m). \tag{6.11}$$

Note that here, in contrast to **Model 1**

- $\boldsymbol{\pi} = (k_1, k_2, \dots, k_m)$ represents an *ordered* sample reflecting the order of edge birth sequence and not an arbitrary random permutation.

- It describes a *construction* process instead of a destruction one, namely starting with the network in *DOWN* state and ending in *UP* state.

Since all birth times are exponential, it is easy to generate a single birth "history", also called the sample path or trajectory.

6.3.2 Permutational Algorithm for Estimating $\mathbb{P}(DOWN; \mathbf{q})$

Before presenting the algorithm for estimating $\mathbb{P}(DOWN; \mathbf{q})$ we make the following observations.

1. The time to the first birth is a random variable ξ_1 distributed exponentially with parameter $\Lambda_1 = \sum_i^m \lambda_i$. It follows that
   - The first birth will occur at edge k_1 with probability λ_{k_1}/Λ_1.
   - At the instant ξ_1 all edges except the born one are not yet born, but because of the memoryless property they behave as if they all are born at that instant ξ_1. This means that the time interval ξ_2 between the first and second births, given that edge k_1 is already born, is distributed exponentially with parameter $\Lambda_2 = \Lambda_1 - \lambda_{k_1}$.
2. The second birth will occur at edge k_2 with probability λ_{k_2}/Λ_1, and so on. Clearly that given the edges $k_1, k_2, \ldots, k_s$ are already born, the time $\xi_{k_{s+1}}$ to the next birth will be exponentially distributed with parameter Λ_{s+1}, which equals the sum of λ-s of all non born edges, and with probability λ_j/Λ_{s+1} the next birth will occur at edge j.

Based on the above we can design a simple permutational Monte Carlo algorithm for estimating $\mathbb{P}(DOWN; \mathbf{q})$.

Algorithm 6.3.1. (**Permutational Algorithm for Estimating** $\mathbb{P}(DOWN; \mathbf{q})$
Given a network and a set of terminal nodes s and t, execute the following steps.

1. Generate the birth times of the edges $\tau_1, \ldots, \tau_m$, with $\tau_i \sim \exp(\lambda_i)$, and rate $\lambda_i = -\ln(q_i)$.
2. Arrange these birth times in increasing order obtaining the order statistics $\tau_{(1)}, \ldots, \tau_{(m)}$, then ignore the actual birth times retaining only the order in which the edges are born. This yields a permutation $\pi = (\pi(1), \ldots, \pi(m))$ of the edge numbers under which their birth times occur.
3. Similar to Algorithm 6.2.2, use a combination of the Goldberg-Rao algorithm (oracle) and the bisection procedure to allocate the edge r_α whose birth brings the network *UP*. Call the sequence $(\pi_1, \ldots, \pi_\alpha) = \omega$.

4. Calculate analytically $\mathbb{P}(\xi_1 + \xi_2 + ... + \xi_\alpha \leq 1|\omega)$. Note that $\xi_1 + \xi_2 + ... + \xi_\alpha$ has a hypo-exponential distribution. Note also that the event $\sum_{i=1}^{\alpha} \xi_i \leq 1$ is equivalent to the network being *UP*, by the definition of α.

5. Repeat N times Steps 1- 4, and estimate $\mathbb{P}(DOWN; \mathbf{q})$ as

$$\widehat{\mathbb{P}}(DOWN; \mathbf{q}) = 1 - \frac{1}{N} \sum_{i=1}^{N} \mathbb{P}(\sum_{j=1}^{\alpha} \xi_j \leq 1|\omega_i). \tag{6.12}$$

Note that

- The complexity of Algorithm 6.3.1 is governed by the Goldberg-Rao oracle and is therefore

$$\mathcal{O}(N \log(m)[min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E| \log(\frac{|V|^2}{|E|}) \log(U)]) \tag{6.13}$$

  This follows from the fact that a single run has complexity $\mathcal{O}(\log(m)[min(|V|^{\frac{2}{3}}, \sqrt{|E|})|E| \log(\frac{|V^2}{|E|}) \log(U)])$ and we perform N such runs.

- As for networks reliability the relative error of the estimator $\widehat{\mathbb{P}}(DOWN; \mathbf{q})$ is uniformly bounded with respect to the λ_i values, (see [25, 33]).

- Besides the permutation Algorithm 6.3.1 one could apply some other alternatives, such as the turnip [26], cross-entropy [52] and splitting [8, 9, 80].

Example 6.3.1. Example 6.2.1 continued

Consider again the network of Example 1 and assume that edge e_i fails with probability q_i. Let $\Phi = 2$, that is assume that network is *UP* if the maximum flow is either 2 or 3. Figure 6.3 shows the tree of all five possible trajectories of the evolution process. It starts moving from the root to the *UP* state shown by double circles.

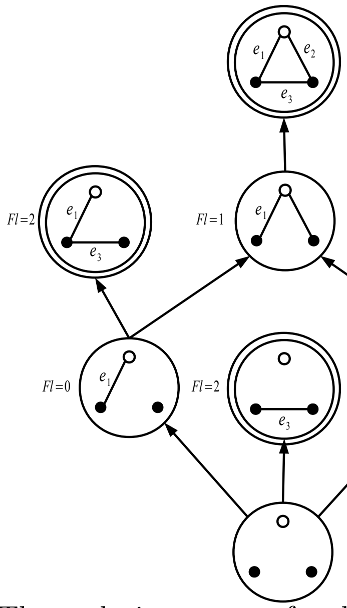


Figure 6.3: The evolution process for the network of Example 6.2.1

Let us consider one of them, namely the one corresponding to $\omega = \{1, 3\}$, meaning that the first birth occurs at edge e_1 and the second at edge e_3. This trajectory will occur with probability

$$\frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} \cdot \frac{\lambda_3}{\lambda_2 + \lambda_3}.$$

The movement along this trajectory from the root to the *UP* state lasts $\xi = \xi_1 + \xi_2$ units of time, where $\xi_1 \sim \exp(\lambda_1 + \lambda_2 + \lambda_3)$ and $\xi_2 \sim \exp(\lambda_2 + \lambda_3)$. Computing $\mathbb{P}(\xi \leq 1|\omega)$ is a simple task.

6.3.3 Numerical Results

Here we apply the permutation Algorithm 6.3.1 to estimate $\mathbb{P}(DOWN; \boldsymbol{q})$ for the same two models in Section 6.2.

Tables 6.6 and 6.7 present data similar to Tables 6.3 and 6.5 while assuming that all components of the vector $\boldsymbol{q}$ are equal. This was done purposely with view to see how the performance of the permutation Algorithm 6.3.1 compares with that of the D-Spectrum Algorithm 6.2.2.

It follows from comparison of Tables 6.6, 6.7 with Tables 6.3, 6.5 that both produce almost identical $\widehat{\mathbb{P}}(DOWN; q)$ values and they also close in terms of RE and CPU times. The main difference is that with Algorithm 6.2.2 we was calculated $\widehat{\mathbb{P}}(DOWN; q)$ simultaneously for different values of q using a *single* simulation, while with Algorithm 6.3.1 we required to calculate $\widehat{\mathbb{P}}(DOWN; q)$ separately for each q value, that is using *multiple* simulations.

Table 6.6: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the dodecahedron graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; q)$	2.99E-06	3.02E-05	3.00E-04	3.00E-03	3.06E-02	3.57E-01	5.54E-01
RE	2.08E-02	1.94E-02	1.53E-02	2.11E-02	1.09E-02	3.13E-03	2.81E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 8.6 seconds.

Table 6.7: $\widehat{\mathbb{P}}(DOWN; q)$ and RE for different values of q for the Edrdos-Renyi graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; q)$	4.01E-06	4.00E-05	4.03E-04	3.99E-03	3.97E-02	3.73E-01	5.44E-01
RE	2.84E-02	1.64E-02	2.36E-02	2.74E-02	1.07E-02	3.84E-03	3.77E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 34 seconds.

Tables 6.8 and 6.9 present data similar to Tables 6.6 and 6.7, but for different values of the components of the vector $\boldsymbol{q}$. More specifically, define α_1 and α_2 to be the minimal and the maximal failure probability, respectively. Define next $\delta = (\alpha_2 - \alpha_1)/|E|$. Then, we set $q_i = \alpha_1 + i\delta$ for $i \in \{0, \dots |E| - 1\}$. In all tables below we set $\alpha_2 = q$ and $\alpha_1 = \frac{\alpha_2}{10} = \frac{q}{10}$.

Table 6.8: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	3.42E-07	3.50E-06	3.50E-05	3.55E-04	3.97E-03	8.04E-02	1.52E-01
RE	1.76E-02	2.01E-02	2.16E-02	2.21E-02	2.39E-02	1.32E-02	5.23E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ was 9 seconds.

Table 6.9: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Edrdos-Renyi graph

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	2.16E-06	2.14E-05	2.16E-04	2.13E-03	2.14E-02	2.10E-01	3.11E-01
RE	2.54E-02	1.29E-02	2.95E-02	2.49E-02	1.71E-02	5.55E-03	5.47E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ was 34.8 seconds. The results are self explanatory.

We also applied Algorithm 6.3.1 for Erdos-Renyi models with the size of several hundreds edges. As for Algorithm 6.2.2 we

- Considered the model with $p = 0.1$, $|V| = 55$ and $|E| = 313$.
- In order to keep the RE ≤ 0.05 we required to increase the sample size from $N = 5 \cdot 10^4$ to $N = 10^5$.

The CPU time was about 9 minutes for each q value.

6.4 Extension to Random Capacities

Algorithm 6.3.1 can be readily modified for the case where not only the edges are subject of failure, but the capacity vector is random as well, that is when the capacity c_k of each edge $k, k = 1, ..., m$ is a random variable independent of edge failure Ber $(\boldsymbol{q})$ distribution. We assume that the capacity random vector is unbiased with respect to the true vector $\mathbf{C}$ and has independent components with some known distribution $\mathcal{F} = (\mathcal{F}_1, \dots, \mathcal{F}_m)$. Denote the joint distribution of $\mathcal{F}_k$ and $\text{Ber}(1 - q_k)$ by $\mathcal{R}_k$, that is $\mathcal{R}_k = \mathcal{F}_k \cdot \text{Ber}(1-q_k)$, $k = 1, \dots, m$ and call it the modified capacity distribution. In this case, only Step 1 of Algorithm 3.1 should be modified as

Generate the birth times $\tau_1, \dots, \tau_m$ of the edges $\tau_i \sim \exp(\lambda_i)$, and put $\lambda_i = -\ln(q_i)$. Generate a capacity random vector $\boldsymbol{Z} = (Z_1, \dots, Z_m)$ according to the modified capacity distribution $\mathcal{R} = (\mathcal{R}_1, \dots, \mathcal{R}_m)$, while the rest of Algorithm 6.3.1 remains the same. Note that as soon the edge birth times and the edge capacities were generated at step 1 all the remaining steps do not change and the Goldberg-Rao algorithm is applied on those capacities.

Clearly the variability of such a noise estimator $\mathbb{P}(DOWN; \boldsymbol{q})$ increases with the variability of $\boldsymbol{Z}$. Our numerical results below support this.

6.4.1 Numerical Results

We model each component Z_k of the random capacity vector $\boldsymbol{Z} = (Z_1, \dots, Z_m)$ by setting $Z_k = \zeta_k c_k (1 + \varepsilon\eta)$, where $\zeta_k \sim \text{Ber}(1 - q_k)$ and η is a random variable with $\mathbb{E}\eta = 0$ and $\mathbb{V}\text{ar}\eta = \sigma^2$. Note that for $\varepsilon = 0$ (deterministic capacities) we have $Z_k = c_k\zeta_k$ and for perfect edges, Z_k reduces to $Z_k = c_k$. We assume for concreteness that $\eta \sim \mathcal{U}(-0.5, 0.5)$.

Before proceeding with numerical results lets us discuss the choice of the threshold level Φ for random capacities. Consider, for example the dodecahedron graph. Recall that with deterministic capacities the maximal flow was $M = 16$ and we selected $\Phi = 14$ for our simulation studies. Observe also that in the case of random capacities (even with perfect edges) the maximal flow is a random variable, which will fluctuate from replication to replication. Denote by $\boldsymbol{M} = (\mathcal{M}_1, \dots, \mathcal{M}_N)$ the maximal flow random vector corresponding to a simulation of length N. Clearly, that the variability of the components of $\boldsymbol{M} = (\mathcal{M}_1, \dots, \mathcal{M}_N)$ increases in ε and it is quite easy to chose an ε (perhaps large enough) such that many the components of $\boldsymbol{M} = (\mathcal{M}_1, \ldots, \mathcal{M}_N)$ will be below some fixed threshold Φ and in particular below our earlier one, $\Phi = 14$. It follows from above that setting, say $\Phi = 14$ for large ε is meaningless.

To resolve this difficulty we propose the following adaptive algorithm based on what is called *elite* sampling [86] for finding a meaningful Φ.

Algorithm 6.4.1. (Adaptive Algorithm for Fining Φ in a Flow Network with Random Capacities)
Given a network and a set of terminal nodes s and t, execute the following steps.

1. Generate a sample of random capacity vectors $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N$. Calculate the corresponding maximum flow values $\mathcal{M}_1, \ldots, \mathcal{M}_N$ using the Goldberg-Rao algorithm.

2. Order them from the smallest to the largest. Denote the ordered values by $\mathcal{M}_{(1)}, \ldots, \mathcal{M}_{(N)}$. Write $\mathcal{M}_{(1)}, \ldots, \mathcal{M}_{(N)}$ as

$$\mathcal{M}_{(1)}, \ldots, \mathcal{M}_{(e)}; \mathcal{M}_{(e+1)}, \ldots, \mathcal{M}_{(a)}; \mathcal{M}_{(a+1)}, \ldots, \mathcal{M}_{(N)}. \qquad (6.14)$$

   Here

   - $\mathcal{M}_{(a)} \leq \mathcal{M}_a \leq \mathcal{M}_{(a+1)}$, $\mathcal{M}_{(a)} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{M}_i$ is the sample average of $\mathcal{M}_1, \ldots, \mathcal{M}_N$.
   - $\mathcal{M}_{(e)}$ corresponds to the sample $(1-\rho)$-quantile of the sequence (6.14), namely $\mathcal{M}_{(e)} = \mathcal{M}_{N^e}$ and $N^e = \lceil (1-\rho)N \rceil$. In other words $\mathcal{M}_{(e)}$ corresponding to $\rho\%$ of the largest sample value in the sequence (6.14) starting from the left.

3. Set $\Phi = \mathcal{M}_{(e)}$ and call it the *adaptive* threshold. Note that the "distance" $|\mathcal{M}_a(\varepsilon) - \mathcal{M}_{(e)}(\varepsilon)|$ between $\mathcal{M}_a$ and $\mathcal{M}_{(e)}$ increases in ε.

If not stated otherwise we assume that the sample quantile $\rho = 0.05$.

Tables 6.10 and 6.11 present $\mathcal{M}_{(e)}$ and $|\mathcal{M}_a - \mathcal{M}_{(e)}|$ as function of ε for the dodecahedron and Erdos- Renyi graphs with $\rho = 0.05$.

Table 6.10: $\mathcal{M}_{(e)}$ and $|\mathcal{M}_a - \mathcal{M}_{(e)}|$ as function of ε for the dodecahedron graph with $\rho = 0.05$

ε	0.5	1	2	3	4	5	6
$\mathcal{M}_{(e)}$	15.516	15.033	14.063	13.099	12.119	11.145	10.168
$\lvert\mathcal{M}_a - \mathcal{M}_{(e)}\rvert$	0.484	0.967	1.939	2.901	3.866	4.794	5.682

Table 6.11: $\mathcal{M}_{(e)}$ and $|\mathcal{M}_a - \mathcal{M}_{(e)}|$ as function of ε for the Erdos-Renyi graph with $\rho = 0.05$

ε	0.5	1	2	3	4	5	6
$\mathcal{M}_{(e)}$	29.440	28.880	27.762	26.645	25.518	24.390	23.231
$\lvert\mathcal{M}_a - \mathcal{M}_{(e)}\rvert$	0.560	1.120	2.238	3.356	4.466	5.560	6.648

It follows from Tables 6.10 and 6.11 that

1. For $\varepsilon \leq 2$ one can use our earlier (deterministic) threshold values $\Phi = 14$ and $\Phi = 27$ for the dodecahedron and Erdos-Renyi graphs, respectively and still be within the 5% of elites.
2. For $\varepsilon > 2$ one should use the appropriate threshold values given in Tables 6.10 and 6.11. For example, for $\varepsilon = 3$ the thresholds are $\Phi = 13.099$ and $\Phi = 26,645$ for the dodecahedron and the Erdos-Renyi graphs, respectively and for $\varepsilon = 6$ they are $\Phi = 10.146$ and $\Phi = 23.231$, respectively.

We proceed below with the following 3 scenarios of ε: (i) $\varepsilon = 0.5$, (ii) $\varepsilon = 3$ and (iii) $\varepsilon = 6$. Note that (i) corresponds to the above case 1 ($\varepsilon \leq 2$), while (ii) and (iii) corresponds to the case 2 ($\varepsilon > 2$).

In all three experiments we set the sample size $N = 5 \cdot 10^4$.

Experiment (i), $\varepsilon = 0.5$. It follows from Tables 6.10 and 6.11 that in this case we can still use the original thresholds $\Phi = 14$ and $\Phi = 27$ chosen for the deterministic capacities for the dodecahedron and Erdos-Renyi graphs, respectively. Based on that Tables 6.12 and 6.13 present data similar to Tables 6.8 and 6.9 for $\varepsilon = 0.5$, with $\Phi = 14$ and $\Phi = 27$, respectively.

Table 6.12: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph with $\varepsilon = 0.5$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	9.76E-07	9.77E-06	9.98E-05	9.98E-04	1.02E-02	1.27E-01	2.11E-01
RE	2.48E-02	2.41E-02	2.51E-02	2.16E-02	2.02E-02	8.29E-03	5.53E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 6.53 seconds.

Table 6.13: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Erdos-Renyi graph with $\varepsilon = 0.5$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	2.45E-06	2.42E-05	2.45E-04	2.46E-03	2.44E-02	2.35E-01	3.43E-01
RE	1.77E-02	2.31E-02	2.34E-02	2.38E-02	2.64E-02	4.52E-03	3.84E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 26.1 seconds It is readily seen that with $N = 5 \cdot 10^4$ samples we obtained RE $\leq 2.5\%$ in both cases.

Experiment (ii), $\varepsilon = 3$. The corresponding thresholds (see Tables 6.10 and 6.11) are $\Phi = 13.099$ and $\Phi = 26,645$ for the dodecahedron and the Erdos-Renyi graphs, respectively.

Based on that Tables 6.14 and 6.15 present data similar to Tables 6.12 and 6.13 for $\varepsilon = 3$ for the dodecahedron and the Erdos-Renyi graphs, respectively.

Table 6.14: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph with $\varepsilon = 3$ and $\Phi = 13.099$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	8.61E-07	8.58E-06	8.55E-05	8.64E-04	8.58E-03	7.87E-02	1.12E-01
RE	2.33E-02	2.59E-02	3.64E-02	2.12E-02	2.08E-02	1.68E-02	8.21E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 2.64 seconds.

Table 6.15: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Erdos-Renyi graph with $\varepsilon = 3$ and $\Phi = 26,645$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	1.02E-06	1.02E-05	1.01E-04	1.01E-03	1.01E-02	9.84E-02	1.43E-01
RE	2.33E-02	3.22E-02	4.09E-02	4.30E-02	1.59E-02	8.16E-03	9.98E-03

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 26.1 seconds It is readily seen that with $N = 5 \cdot 10^4$ samples we obtained the RE $\leq 2.5\%$ in both cases.

Experiment (iii), $\varepsilon = 6$. The corresponding thresholds (see Tables 6.10 and 6.11) are $\Phi = 10.168$ and $\Phi = 23.231$ for the dodecahedron and the Erdos-Renyi graphs, respectively. Based on that Tables 6.16 and 6.17 present data similar to Tables 6.14 and 6.15 for $\varepsilon = 6$ for the dodecahedron and the Erdos-Renyi graphs, respectively.

Table 6.16: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the dodecahedron graph with $\varepsilon = 6$ and $\Phi = 10.168$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	6.44E-07	6.46E-06	6.51E-05	6.37E-04	6.31E-03	5.98E-02	8.55E-02
RE	3.58E-02	3.33E-02	2.15E-02	3.23E-02	2.31E-02	9.60E-03	1.38E-02

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 1.77 seconds.

Table 6.17: $\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$ and RE for different values of $\boldsymbol{q}$ for the Erdos-Renyi graph with $\varepsilon = 6$ and $\Phi = 23.231$

q	10^{-6}	10^{-5}	10^{-4}	10^{-3}	10^{-2}	0.1	0.15
$\widehat{\mathbb{P}}(DOWN; \boldsymbol{q})$	8.82E-07	9.01E-06	9.01E-05	9.10E-04	9.02E-03	8.89E-02	1.31E-01
RE	4.00E-02	2.68E-02	3.13E-02	2.42E-02	2.65E-02	1.46E-02	1.07E-02

The CPU time for each $\widehat{\mathbb{P}}(DOWN; q)$ equals to 9.2 seconds. It is readily seen that with $N = 5 \cdot 10^4$ samples we obtained the RE $\leq 4.0\%$ in both cases.

We also estimated $\mathbb{P}(DOWN; \boldsymbol{q})$ for the Erdos-Renyi models with several hundreds edges with different ε values. In particular we considered our previous model with $p = 0.1$, $|V| = 55$ and $|E| = 313$. We found that

- For $\varepsilon = 3$ one should set $\Phi = 24.634$ in order to insure $\rho = 0.05$ (5% elites).
- In order for the relative error RE ≤ 0.05 (with $\Phi = 24.634$), we need to increase the sample size from $N = 5 \cdot 10^4$ to $N = 5 \cdot 10^5$.

The CPU time for this scenario was about 25 minutes.

6.5 Concluding Remarks and Further Research

We show how the permutation Monte Carlo method originally developed for reliability networks [26] can be successfully adapted for stochastic flow networks, and in particular for estimation of the probability that the maximal flow in such a network is above some fixed level, called the *threshold*. A stochastic flow network is defined as one, where the edges are subject to random failures. A failed edge is assumed to be erased (broken) and, thus, not able to deliver any flow. We consider two models; one where the edges fail with the same failure probability and another where they fail with different failure probabilities. For each model we construct a different algorithm for estimation of the desired probability; in the former case it is based on the well known notion of the D-spectrum and in the later one - on the permutational Monte Carlo. We discuss the convergence properties of our estimators and present supportive numerical results.

One of our directions of further research will be in obtaining efficient Monte Carlo estimators based on network edge *important measures* (see [38, 35]) and their applications to stochastic flow networks with a particular emphasis on the optimal network design under some budget constraints.

Another direction will be an extension of our permutational technique to a wider class of functionals associated with network edges. The crucial property for the applicability of that methodology for computing the probability that the maximal $s - t$ flow M exceeds the threshold level Φ is the monotonicity of the maximal $s-t$ in the process of edge destruction (construction). This means that the maximal $s-t$ flow in the destruction process,can only *decrease*, or equivalently, the "birth" of new edges can lead only to a *increase* of the maximal $s-t$ flow and similarly for the construction one. But this "edge monotonicity" property holds true not only for the network flow. Consider, for example, the total weight $W_{\max}$ of the maximal spanning tree of the network. $W_{\max}$ *decreases* if the edges are sequentially eliminated. Similar monotone behavior exhibit the minimal distance between any pair of fixed nodes in the network, the total number of $s-t$ paths in it, and many other of its important combinatorial characteristics.

ACKNOWLEDGMENT
We are thoroughly grateful to anonymous Reviewer for his/her valuable constructive remarks and suggestions.

Chapter 7

Stochastic Enumeration Method for Counting, Rare-Events and Optimization

Radislav Vaisman

Faculty of Industrial Engineering and Management,

Technion, Israel Institute of Technology, Haifa , Israel

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.

7.1 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 [50] and Metropolis et al [65]. Since then hundreds of insightful papers have been written on that topic. We refer the reader to [7, 8], [18]-[11], [68]-[40], [55, 56] and also to the papers by Glasserman et al. [40], Cerou et al. [11] and Melas [64], 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 [68] 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 [86]) that its straightforward application may typically yield very poor approximations of the quantity of interest. For example, it is shown by Gogate and Dechter [41], [42] 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 [82] 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 [35] 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 [92].

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 [67]. Jerrum, Valiant and Vazirani [49] 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 [35]: *use fast polynomial decision making oracles to solve #P-complete problems.* In particular one can easily incorporate into SE

- Breadth first search (BFS) or Dijkstra's shortest path algorithm [17] for counting the number of path in the networks.

- Hungarian decision making assignment problem method [54] for counting the number of perfect matchings.

- Davis et al [19, 20] decision making algorithm for counting the number of valid assignments in 2-SAT.

- 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.

We shall implement here the first three polynomial decision making oracles.

As mentioned SE represents a natural extension of the classic *one-step-look-ahead* OSLA. We shall show that OSLA-type algorithms (see Algorithm 7.2.1) have the following two drawbacks:

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

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

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 [61] that most of its trajectories of length $n \geq 100$ in OSLA Algorithm 7.2.1 become not self-avoiding. A similar pattern was observed in other counting problems including satisfiability.

The exception is the OSLA algorithm of Rasmussen [73] for counting the permanent. Rasmussen [73] proofs that if the a_{ij} entries of the permanent matrix $\boldsymbol{A}$ 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* $g(\boldsymbol{x})$, SE runs in parallel multiple trajectories (see Algorithm 7.5.1 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 7.5 below.

The rest of the paper is organized as follows. Section 7.2 presents background on the OSLA method. Section 7.3 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 7.4 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 7.5 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 7.7 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 7.8 discus how to choose the main parameter, the number of elites samples in the SE algorithm. In Section 7.9 we present numerical results for SE-OSLA and SE. Here we show that SE outperforms the well known splitting and SampleSearch 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 7.10 presents some conclusions. Finally, the Appendix is devoted to some auxiliary material.

7.2 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 nods and the number of SAW's of length n. We wish to estimate $|\mathcal{X}^*|$ by employing a SIS pdf $g(\boldsymbol{x})$ defined in Section 7.11.1 of Appendix . To do so we use the following *one-step-look-ahead* (OSLA) procedure due to Rosenbluth and Rosenbluth [77], which was originally introduced for SAW's:

Procedure 1: One-Step-Look-Ahead

1. **Initial step** Start from the origin point $\boldsymbol{y}_0$. For example, in a two-dimensional SAW, $\boldsymbol{y}_0 = (0, 0)$ and an a network with source s and think t, $\boldsymbol{y}_0 = s$. Set $t = 1$.
2. **Main step** Let ν_t be the number of neighbors of $\boldsymbol{y}_{t-1}$ that have not yet been visited. If $\nu_t > 0$, choose $\boldsymbol{y}_t$ 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

$$g(\boldsymbol{x}) = \frac{1}{\nu_1}\frac{1}{\nu_2}\cdots\frac{1}{\nu_n} = \frac{1}{\widehat{|\mathcal{X}^*|}(\boldsymbol{x})} \tag{7.1}$$

as a SIS pdf. Here

$$\widehat{|\mathcal{X}^*|} = \nu_1 \dots \nu_n \; . \tag{7.2}$$

Note that the procedure either generates a path $\boldsymbol{x}$ of fixed length n or the path gets value zero (in case of $\nu_t = 0$, for which $g(\boldsymbol{x}) = \infty$).

The OSLA counting algorithm now follows:

Algorithm 7.2.1 (OSLA Algorithm).

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

2. For each path $\boldsymbol{X}_k$ compute the corresponding $\widehat{|\mathcal{X}_k^*|}$ as in (7.2). For the other parts (which do not reach the value n) set $\widehat{|\mathcal{X}_k^*|} = 0$.

3. Return

$$\bar{|\mathcal{X}^*|} = \frac{1}{M}\sum_{i=k}^{M} \widehat{|\mathcal{X}_k^*|} \; . \tag{7.3}$$

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 $\boldsymbol{y}_{t-1}$ 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 7.1 depicts a SAW (with arrows) trapped after 15 iterations. One can easily find from it the corresponding values $\nu_t, \; t = 1, \dots, 15$ (using the corresponding short lines without arrows) of each of the 15 points. They are

$$\nu_1 = 4, \; \nu_2 = 3, \; \nu_3 = 3, \; \nu_4 = 3, \; \nu_5 = 3, \; \nu_6 = 3, \; \nu_7 = 2, \; \nu_8 = 3,$$

$$\nu_9 = 3, \; \nu_{10} = 3, \; \nu_{11} = 2, \; \nu_{12} = 3, \; \nu_{13} = 2, \; \nu_{14} = 1, \; \nu_{15} = 0.$$

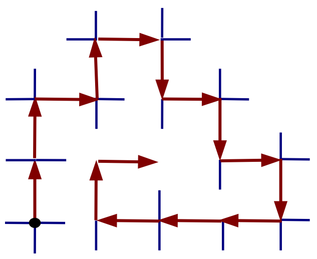


Figure 7.1: SAW trapped after 15 iterations and its corresponding values ν_t, $t = 1, \dots, 15$

As for another situation where OSLA can be readily trapped consider a directed graph in Figure 7.2 with source s and sink t.

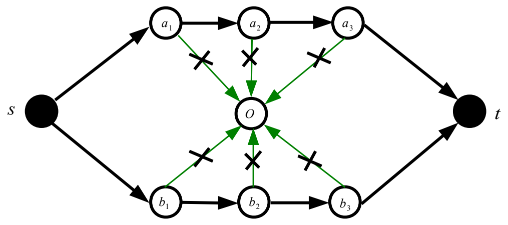


Figure 7.2: Directed graph

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 7.2 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 most of the random walks (with probability $1 - (1/2)^n$) will be trapped at node o.

7.3 Extension of OSLA: nSLA Method

A natural extension of the one-step-look-ahead (OSLA) procedure is the k-step-look-ahead, where $2 \le k \le n$ 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 7.1. 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 $\boldsymbol{y}_{13}$) 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* (nSLA) 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 nSLA algorithm is identical to the OSLA Algorithm 7.2.1. 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 $\boldsymbol{y}_{t-1}$ that have not yet been visited. Choose $\boldsymbol{y}_t$ 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 nSLA works in practice consider a simple example following the main steps of the OSLA Algorithm 7.2.1.

Example 7.3.1. Let $\boldsymbol{x} = (x_1, x_2, x_3)$ 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 nSLA 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 7.3 presents a tree corresponding to the set $\{000, 001, 100, 110, 111\}$.

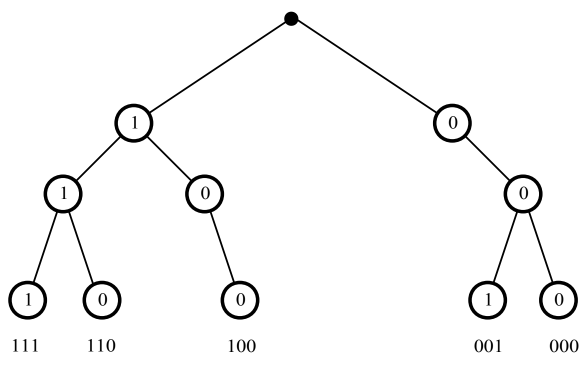


Figure 7.3: Tree corresponding to the set $\{000, 001, 100, 110, 111\}$.

According to nSLA **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 wether 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 7.4).

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 7.4 presents the sub-trees $\{000, 001\}$ (in bold) generated by nSLA using the oracle.

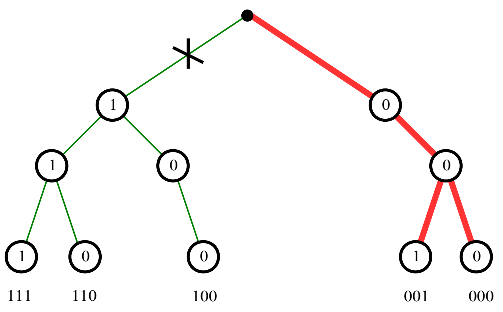


Figure 7.4: The sub-trees $\{000, 001\}$ (in bold) generated by nSLA using the oracle

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

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

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

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

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

To see the non-uniformity of $g(\boldsymbol{x})$ 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 7.5 presents the corresponding graph with 4 literals and $|\mathcal{X}^*| = 5$.

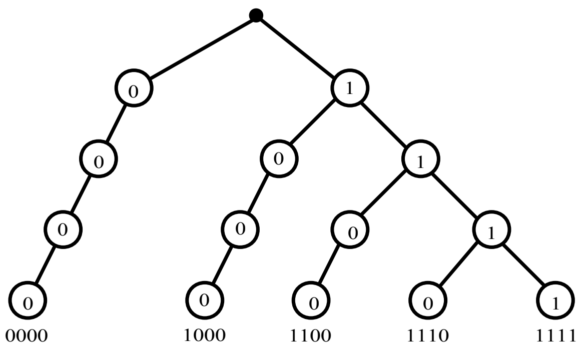


Figure 7.5: A graph with 4 clauses and $|\mathcal{X}^*| = 5$.

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

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

which is highly non-uniform.

To improve the non-uniformity of $g(\boldsymbol{x})$ we will develop in Section 7.8 a modified version of nSLA, called *stochastic enumeration* (SE) algorithm. In contrast to nSLA 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 7.2.1 for SAW's.

7.4 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.

7.4.1 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 [15], [62]. For a nice recent survey see [94].

Although empirically the pivot algorithm [15] 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 $\boldsymbol{x} = (x_1, x_2, \ldots, x_{n-1}, x_n)$, 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 7.6.

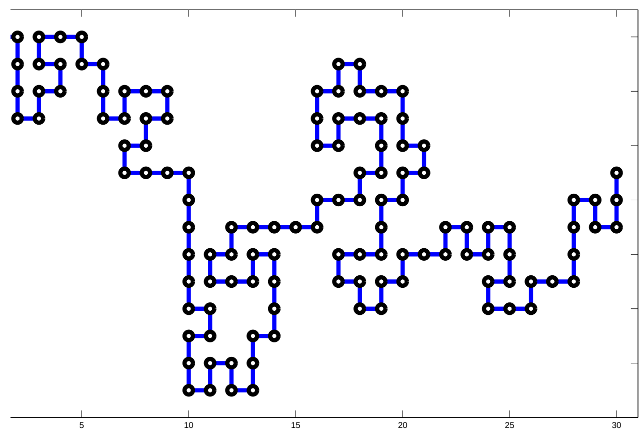


Figure 7.6: A SAW of length $n = 121$.

Algorithm 7.4.1 (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
    $$\nu_1 = \frac{N_1}{N_1^{(e)}} \tag{7.4}$$
    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
    $$\nu_t = \frac{N_t}{N_t^{(e)}} \tag{7.5}$$
    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, $t = 1, \ldots, \frac{n-n_0}{r}$ and calculate
$$\widehat{|\mathcal{X}^*|} = N_1^{(e)} \prod_{t=1}^{\frac{n-n_0}{r}} \nu_t. \tag{7.6}$$

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 (7.2). Note that since the number of levels is fixed, $\widehat{|\mathcal{X}^*|}$ presents an unbiased estimator of $|\mathcal{X}^*|$ (see [7]).

4. **Final Estimator** Run SE-OSLA for M independent replications and deliver
$$\widetilde{|\mathcal{X}^*|} = \frac{1}{M}\sum_{k=1}^{M}\widehat{|\mathcal{X}_k^*|} \tag{7.7}$$
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 (7.3).

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 $\widehat{|\mathcal{X}^*|}$ is
$$S^2(\widetilde{|\mathcal{X}^*|}) = \frac{1}{M-1}\sum_{k=1}^{M}(\widehat{|\mathcal{X}_k^*|} - \widetilde{|\mathcal{X}^*|})^2 \tag{7.8}$$
and the relative error is
$$RE(\widetilde{|\mathcal{X}^*|}) = \frac{\widetilde{|\mathcal{X}^*|}}{S(\widetilde{|\mathcal{X}^*|})}. \tag{7.9}$$

The first two iterations of Algorithm 7.4.1 for $n_0 = 1$, $r = 1$ and $N_t^{(e)} = N^{(e)} = 4$ are given below.

Example 7.4.1. *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 7.7).

  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 7.8).

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

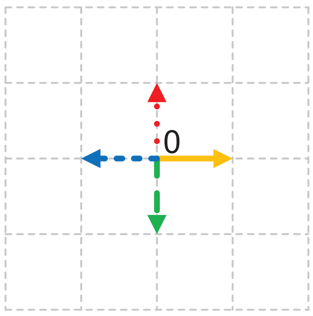


Figure 7.7: The First Four Elites $N_1^{(e)} = N^{(e)} = 4$

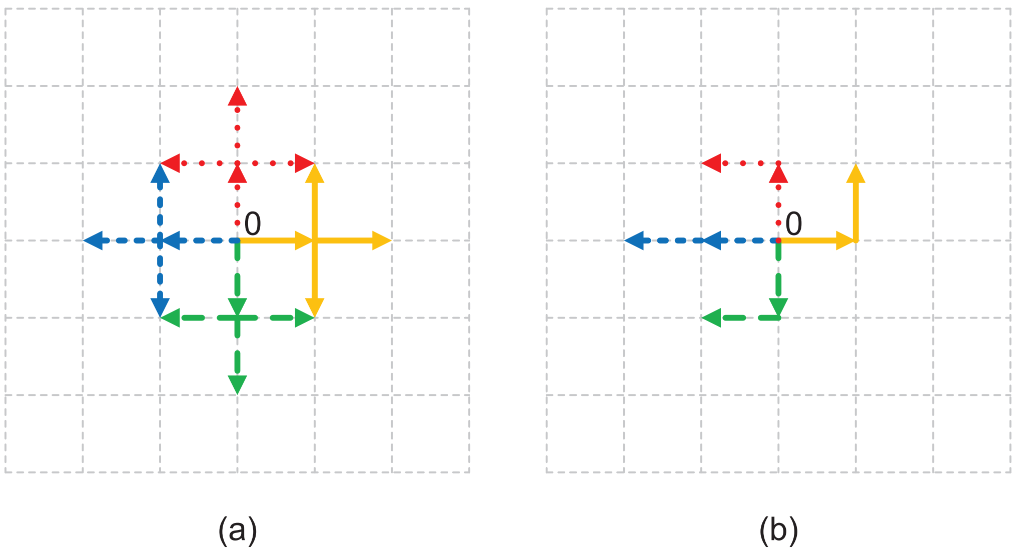


Figure 7.8: First Iteration of Algorithm 7.4.1

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 7.8).
- **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 7.9.
- **Calculation of the second weight factor**. We have $\nu_2 = \frac{N_2}{N_2^{(e)}} = \frac{12}{4} = 3$.

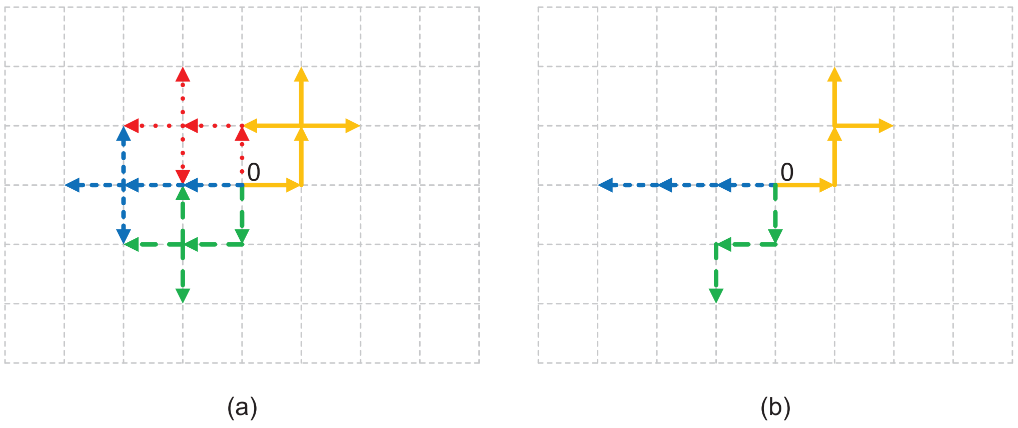


Figure 7.9: Second Iteration of Algorithm 7.4.1

We call Algorithm 7.4.1 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 7.4.1. It follows from Algorithm 7.4.1 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 (7.6) 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 7.4.2. We next support empirically Remark 7.4.1 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 7.1 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 $N_1^{(e)} = 1, \ N_2^{(e)} = 2, \ N_3^{(e)} = 5, \ N_4^{(e)} = 15, \ N_4^{(e)} = 25, \ N_4^{(e)} = 35$ are arranged in Table 7.1 in the increasing order.

It readily follows from the results of Table 7.1 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 7.1: The length of the SAW's R_{ki}, $k = 1, \ldots, 20$; $i = 1, \ldots, 6$ 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 $(N_t^{(e)}, N_t),\ t = 1, \ldots, R$ for one of the runs with $N_1^{(e)} = 2$ with the outcome $R = 30$.

$$\begin{array}{l}(2,6),(2,6),(2,6),(2,6),(2,6),(2,6),(2,5),(2,5),(2,5),(2,4),\\ (2,4),(2,6),(2,5),(2,6),(2,6),(2,6),(2,5),(2,4),(2,6),(2,6),\\ (2,6),(2,5),(2,4),(2,4),(2,3),(2,4),(2,3),(2,2),(2,2),(2,0).\end{array} \tag{7.10}$$

It is crucial to note that for general counting problems, like SAT's, Remark 7.4.1 does not apply in the sense that even for a relatively small model size the SE-OSLA Algorithm 7.4.1 generates many zeros (trajectories of zero length). For this reason it has limited applications.

To see this consider, for example, the set $\mathcal{X}^*$ defined by the following product of clauses

$$(x_1 + \bar{x}_2)(\bar{x}_1 + \bar{x}_2)(x_1 + x_3)(x_2 + \bar{x}_3). \tag{7.11}$$

It is not difficult to check that with $N^{(e)} = 1$ SE-OSLA (which is fact OSLA), fails to find the true value $|\mathcal{X}^*| = 1$ with probability 7/8, that is it loses in average the unique trajectory with probability 0.88.

As for another example consider a 3-SAT model with an adjacency matrix $\boldsymbol{A} = (20 \times 80)$ and $|\mathcal{X}^*| = 15$. Running the SE-OSLA Algorithm 7.4.1 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.

7.5 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 7.4.1 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 7.4.1.

We present SE for the models where the components of the vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ are assumed to be binary variables, such as SAT's, and where fast decision making oracles are available (see [19, 20] for SAT's). Its modification to arbitrary discrete variables is straightforward.

7.5.1 SE Algorithm

Consider SE-OSLA Algorithm 7.4.1 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 $(x_1, \ldots, x_t)$ 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 $(x_{t+2}, \ldots, x_n)$.
- Sets next $x_{t+1} = 1$ and then again makes a similar (YES-NO) decision for the same set $(x_{t+2}, \ldots, x_n)$.

Algorithm 7.5.1 (SE Algorithm)**.**

1. **Iteration 1**
   - **Full Enumeration** (Similar to Algorithm 7.4.1). Let n_0 be the number corresponding to the first n_0 variables $x_1, \ldots, x_{n_0}$. 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 7.4.1).
2. **Iteration** t, $(t \geq 2)$

- **Full Enumeration** (Same as in Algorithm 7.4.1, 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 7.4.1).
- **Calculation of the t-th Weight Factor**. (Same as in Algorithm 7.4.1). Recall that in contrast to SE-OSLA, where $\nu_t \geq 0$, here $\nu_t \geq 1$.

3. **Stopping Rule** Same as in Algorithm 7.4.1.

4. **Final Estimator** Same as in Algorithm 7.4.1.

It follows that if $N_t^{(e)} \geq |\mathcal{X}^*|$, $\forall t$, the SE Algorithm 7.5.1 will be exact.

Remark 7.5.1. The SE method can be viewed as a multi-lever splitting with a specially chosen *importance function* [28]. Note that in the traditional splitting [7, 8], [82] 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* [66]. 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 7.10 presents dynamics of the SE Algorithm 7.5.1 for the first 3 iterations in 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 an oracle), and 1 invalid path (corresponding to the line depicted as ·— x. It follows from Figure 7.10 that the accumulated weights are $\nu_1 = 2, \nu_2 = 2, \nu_3 = 1$.

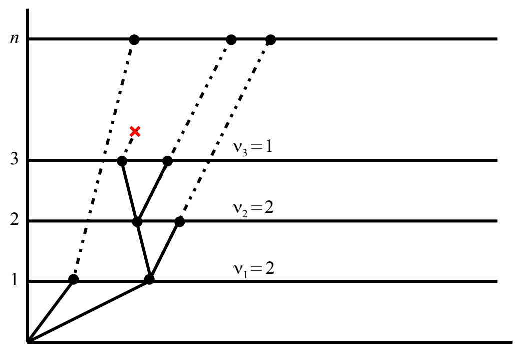


Figure 7.10: Dynamics of the SE Algorithm 7.5.1 for the first 3 iterations

Note again that the SE estimator $\widehat{|\mathcal{X}^*|}$ is unbiased for the same reason as its SE-OSLA counterpart (7.6) is so; namely both can be viewed as multiple splitting methods with fixed (non-adaptive) levels [8].

Below we show how SE works for several toy examples

Example 7.5.1. Consider the SAT model $C_1 \wedge C_2 \wedge, \ldots, \wedge C_n$, where $C_i = x_i \vee x_{i+1} \geq 1$, $i = 1, \ldots, n$. 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 7.5.2. 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 7.5.1.
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 (7.6) decreases and for $N^{(e)} \geq \widehat{|\mathcal{X}^*|}$ we have $\mathbb{V}\text{ar}\widehat{|\mathcal{X}^*|} = 0$.

Example 7.5.3. Example 7.3.1 continued Let again $\boldsymbol{x} = (x_1, x_2, x_3)$ 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 7.3.1, 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 $\widehat{|\mathcal{X}^*|} = 5$. The variance of $\widehat{|\mathcal{X}^*|}$ is

$$\mathbb{V}\text{ar}\widehat{|\mathcal{X}^*|} = 1/3\{2(9/2 - 5)^2 + 2(6 - 5)^2\} = 1/2.$$

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

Figure 7.11 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 $N^{(e)} = 2$.

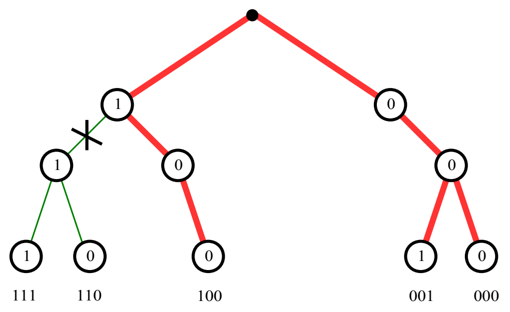


Figure 7.11: The sub-trees $\{100, 000, 001\}$ (in bold) corresponding to $N^{(e)} = 2$.

Figures 7.12, 7.13 and 7.14 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.

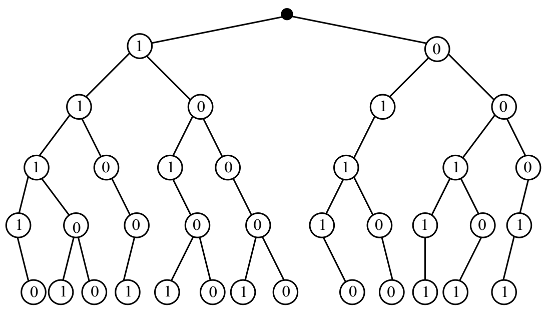


Figure 7.12: A tree with 5 variables.

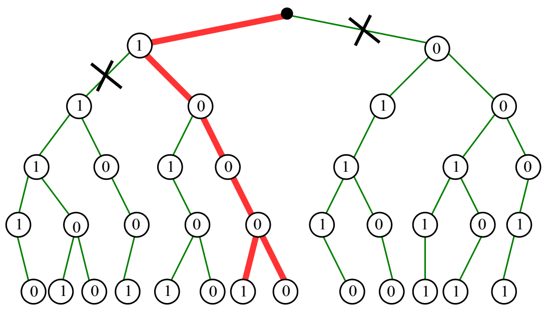


Figure 7.13: Sub-trees (in bold) corresponding to $N^{(e)} = 1$.

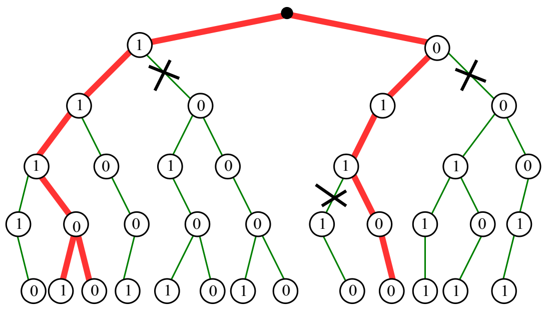


Figure 7.14: Sub-trees (in bold) corresponding to $N^{(e)} = 2$.

As mentioned earlier the major advantage of SE Algorithm 7.5.1 versus its nSLA counterpart is that the uniformity of $g(\boldsymbol{x})$ in the former increases in $N^{(e)}$. In other words $g(\boldsymbol{x})$ becomes "closer" to the ideal pdf $g^*(\boldsymbol{x})$. 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 7.5 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 7.15 presents such a graph with $|\mathcal{X}^*| = 5$ paths and $s - t$ being the source and sink, respectively.

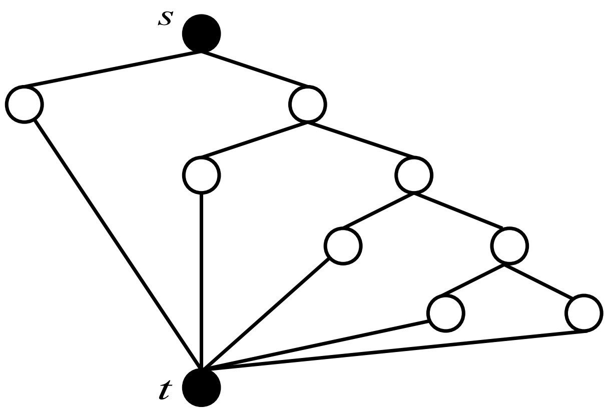


Figure 7.15: A graph with $|\mathcal{X}^*| = 5$ paths.

(i) Table 7.2 presents the efficiency of the SE Algorithm 7.5.1 for for the 2-SAT model with $|\mathcal{X}^*| = 100$ for different values of $N^{(e)}$ and M. The comparison was done for

$$\begin{aligned}&(N^{(e)} = 1,\ M = 500),\ (N^{(e)} = 5,\ M = 100),\ (N^{(e)} = 10,\ M = 50),\\ &(N^{(e)} = 25,\ M = 20),\ (N^{(e)} = 50,\ M = 10)\ (N^{(e)} = 75,\ M = 5),\\ &(N^{(e)} = 100,\ M = 1).\end{aligned} \tag{7.12}$$

Table 7.2: The efficiencies of the SE Algorithm 7.5.1 for the 2-SAT model with $|\mathcal{X}^*| = 100$

$(N^{(e)},\ M)$	$\widetilde{\lvert\mathcal{X}^*\rvert}$	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 where obtained for the graph in Figure 7.15 with $|\mathcal{X}^*| = 100$ paths.

As expected for small $N_t^{(e)}$ $(1 \le N_t^{(e)} \le 10)$ the relative error RE of the estimator $\widetilde{|\mathcal{X}^*|}$ 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

$$(00\cdots00),\ (10\cdots00),\ (11\cdots00),\ldots,\ (11\cdots10),\ (11\cdots11)$$

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

The variance of the SE method can be reduced by using the so-called *empirical length-distribution* method due to [76], which is described below.

Remark 7.5.2. 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 $\boldsymbol{x}$ by $|\boldsymbol{x}|$. 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 7.5.1 to find an estimate of the length-distribution $\boldsymbol{r} = (r_1, \ldots, r_{n-1})$, where

$$r_k = \frac{\text{number of paths of length k}}{\text{number of paths of length } > \text{k}}. \tag{7.13}$$

We can visualize r_k in the following way: suppose a path $\boldsymbol{x}$ 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 $\boldsymbol{r}$ as

$$\widehat{r_k} = \frac{\sum_{j=1}^{N_0} I\{|\boldsymbol{X}_i| = k\} I\{\boldsymbol{X}_i \in \mathcal{X}^*\}/f(\boldsymbol{X}_i)}{\sum_{j=1}^{N_0} I\{|\boldsymbol{X}_i| \geq k\} I\{\boldsymbol{X}_i \in \mathcal{X}^*\}/f(\boldsymbol{X}_i)}, \tag{7.14}$$

where $\boldsymbol{X} \sim f(\boldsymbol{x})$. We then use $\widehat{\boldsymbol{r}} = (\widehat{r_1}, \ldots, \widehat{r_{n-1}})$ to generate paths similarly to the SE Algorithm 7.5.1 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 $\boldsymbol{x} \in \mathcal{X}^*$, 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 $\widehat{\boldsymbol{r}}$ in (7.14) 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.

7.6 Backward Method

Here we describe a new counting method, called the *backward method*, which appears to be very efficient, provided the number of solutions is not too large, say $|\mathcal{X}^*| \leq 10^9$. We demonstrate it for SAT's. Its name derives from the fact that we model the SAT problem as a tree and move backwards from the branches toward its root. It is assumed that we have access to a SAT solver of the oracle type that can decide whether or not there exists a valid assignment to a fixed n-dimensional vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ of some given instance matrix $\boldsymbol{A} = n \times m$.

The underlying idea is straightforward. As usual, denote by $\mathcal{X}_0 = \{0, \ldots, 2^n - 1\}$ the entire space and let $\mathcal{X}^* \subseteq \mathcal{X}_0$ be the set of valid solutions. In terms of our binary tree graph, this means that we have 2^n branches. Our procedure starts moving to the right from the leftmost point $\mathbf{0} = (0, \ldots 0)$ searching for a valid solution using a SAT solver.

Procedure 1: SAT-SOLVE ($\boldsymbol{x}$) where $\boldsymbol{x} = (x_1, \ldots, x_k)$
Input: Some partial assignment (possibly empty) of variables $\boldsymbol{x} = x_{i_1}, \ldots, x_{i_k}$ $i_j \in \{1, \ldots, n\}$, $x_{i_j} \in \{0, 1\}$ and $k \leq n$
Output: The solution found by the solver. If no solution exists, output $\emptyset$.

The following procedure determines the closest rightmost solution to an arbitrary initial vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ in $\mathcal{X}_0 = \{0, \ldots, 2^n - 1\}$.

Procedure 2: FIND-NEXT-RIGHT-SOLUTION($\boldsymbol{x}_i$)
Input: Given an initial vector $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,n})$ (not necessarily a valid solution)
Output: the first valid solution $\boldsymbol{x}_{i+1}$ on the right side of $\boldsymbol{x}_i$ if exists, else return $\emptyset$.

For given $\boldsymbol{x}_0 = (x_{0,1}, \ldots, x_{0,n})$ proceed as follows

```
1. $solutionFound \longleftarrow \emptyset$
2. $index \longleftarrow n - 1$
3. $tmp \longleftarrow (x_{0,1}, \ldots, x_{0,n})$
4. While $solutionFound = \emptyset$ OR $index = 0$ DO:
   (a) Go up in the tree to node $tmp \longleftarrow (tmp_{0,1}, \ldots, tmp_{0,index})$
   (b) $solutionFound \longleftarrow \mathbf{SAT-SOLVE}(tmp_{0,1}, \ldots, tmp_{0,index}, 1)$
   (c) IF($solutionFound = \emptyset$) THAN $index \longleftarrow index - 1$
   (d) IF($solutionFound \neq \emptyset$) THAN
       i. IF $solutionFound \leq \boldsymbol{x}_0$ THAN $solutionFound = \emptyset$
5. $tmp \longleftarrow (tmp_{0,1}, \ldots, tmp_{0,index})$
6. $solutionFound \longleftarrow \emptyset$
7. While length of binary string $tmp < n$ DO:
   (a) $tmp_0 = (tmp_{0,1}, \ldots, tmp_{0,index}, 0)$
   (b) $tmp_1 = (tmp_{0,1}, \ldots, tmp_{0,index}, 1)$
   (c) $solutionFound \longleftarrow \mathbf{SAT-SOLVE}(tmp_0)$
   (d) IF($solutionFound = \emptyset$) THAN $tmp \longleftarrow tmp_1$
   (e) ELSE $tmp \longleftarrow tmp_0$
8. $\boldsymbol{x}_1 \longleftarrow tmp$
9. IF($\boldsymbol{x}_1 \leq \boldsymbol{x}_0$) THAN $\boldsymbol{x}_1 = \emptyset$
```

```
10. Output $\boldsymbol{x}_1$
```

Procedure 3: Backward Algorithm

Output: The number of solutions X.

```
1. $X \longleftarrow 0$
2. $\boldsymbol{x} \longleftarrow \{0, \dots, 0\}$
3. $\boldsymbol{x} \longleftarrow$ FIND-NEXT-RIGHT-SOLUTION($\boldsymbol{x}$)
4. While $\boldsymbol{x} \neq \emptyset$
   (a) $X \longleftarrow X + 1$
   (b) $\boldsymbol{x} \longleftarrow$ FIND-NEXT-RIGHT-SOLUTION($\boldsymbol{x}$)
5. Output $X$
```

It is important to note that the running time of **Procedure 2** is bounded by $O(2log(2^n) \cdot (\mathbf{SAT-SOLVE}))$. Assuming that **Procedure 1** is also fast enough, we can see that the total running time is polynomial in n. If $|\mathcal{X}^*|$ is also polynomial in n, then so is the full enumeration using the backward method.

Figure 7.16 demonstrates **Procedure 2** in action. In particular, the backward algorithm starts from 000 and has to climb all the way up to node 0. Next, the algorithm "tries" to go "as far left as possible" from 0 in order to find the solution closest to 000. Accordingly the SAT solver identifies the first valid solution (to the right of 000 and to the left of 0), which corresponds to 010. It also follows that in order to find the next solution (to the right of 010) corresponding to 100, the SAT solver has to climb all the way from 010 up to the node *Root.*

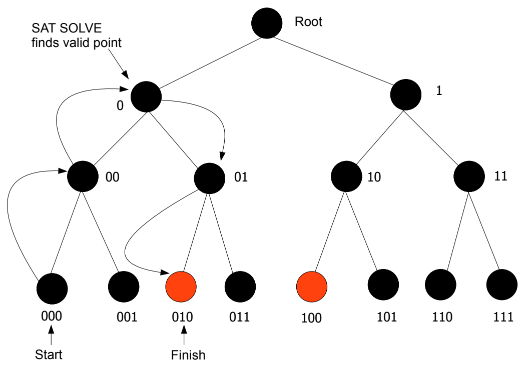


Figure 7.16: Procedure 2

7.7 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.

7.7.1 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 7.7.1. Bridge Network Consider the undirected graph in Figure 7.17.

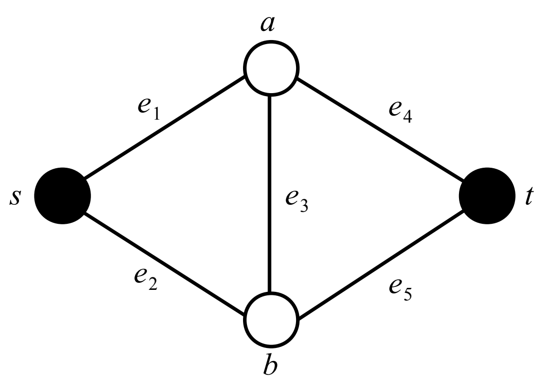


Figure 7.17: Bridge network: number of path from A to B.

Suppose we wish to count the 4 trajectories

$$(e_1, e_4),\ (e_1, e_3, e_5),\ (e_2, e_3, e_4),\ (e_2, e_5) \tag{7.15}$$

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 (7.15) we would have a directed graph with the following 3 trajectories

$$(e_1, e_4),\ (e_1, e_3, e_5),\ \ (e_2, e_5), \tag{7.16}$$

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 7.7.2. Extended Bridge

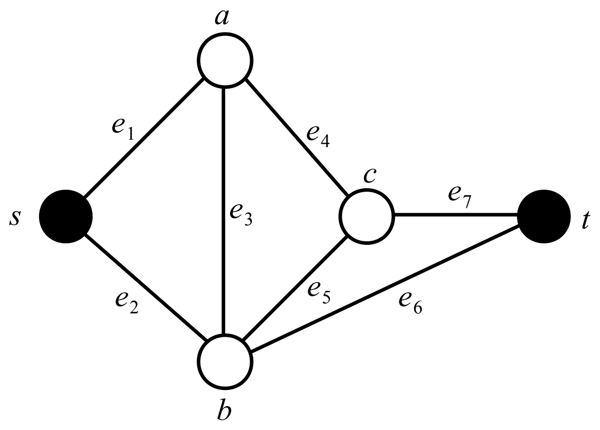


Figure 7.18: Extended bridge

We have the following 7 trajectories

$$\begin{aligned}&(e_1, e_4, e_7),\ (e_1, e_3, e_6),\ (e_1, e_3, e_5, e_7),\ (e_1, e_4, e_5, e_6),\\ &(e_2, e_6)\ (e_2, e_5, e_7),\ (e_2, e_3, e_4, e_7).\end{aligned} \tag{7.17}$$

1. **Iteration 1** This iteration coincides with **Iteration 1** of Example 7.7.1. 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 7.19 presents the sub-tree (in bold) corresponding to the path (e_2, e_3, e_4, e_7) for the extended bridge in Figure 7.18.

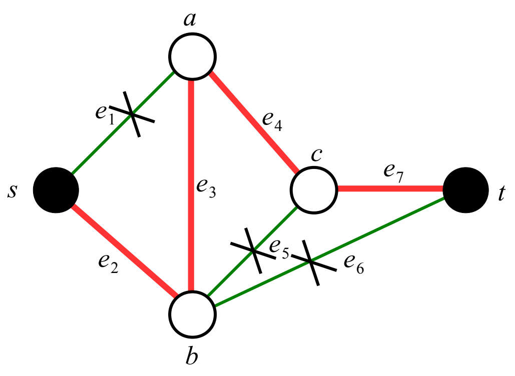


Figure 7.19: Sub-tree (in bold) corresponding to the path (e_2, e_3, e_4, e_7).

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 7.18 and assume that $N^{(e)} = 2$. In this case at iteration 1 of SE Algorithm 7.5.1 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 7.5.1 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.

SE for Estimation of Probabilities in a Network

Algorithm 7.5.1 can be readily modified for the estimation of different probabilities in a network, such as the probability that the length $S(\boldsymbol{X})$ of a randomly chosen path $\boldsymbol{X}$ is greater than a fixed number γ, i.e.

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

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

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

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

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

Algorithm 7.7.1 (SE for Estimation Probabilities)**.**

1. **Iteration 1**
    - **Full Enumeration** (Same as in Algorithm 7.5.1).
    - **Calculation of the First Weight Factor** (Redundant). Instead, store the lengths of the corresponding edges $\eta_{1,1}, \ldots, \eta_{1,N^{(e)}}$.
2. **Iteration** t, $(t \geq 2)$
    - **Full Enumeration** (Same as in Algorithm 7.5.1.
    - **Stochastic Enumeration** (Same as in Algorithm 7.5.1).
    - **Calculation of the** t**-th Weight Factor**. (Redundant). Instead, store the lengths of the corresponding edges $\eta_{t,1}, \ldots, \eta_{t,N^{(e)}}$.
3. **Stopping Rule** Proceed with iteration $t,\ t = 1, \ldots, \frac{n-n_0}{r}$ and calculate

$$I_j = I_{\{\frac{S(\boldsymbol{x}_j)-n_0}{r} \geq \gamma\}},\ j = 1, \ldots, N^{(e)}, \tag{7.18}$$

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

4. **Final Estimator** Run Algorithm 7.7.1 for M independent replications and deliver

$$\widetilde{\ell} = \frac{1}{MN^{(e)}} \sum_{k=1}^{M} \sum_{j=1}^{N^{(e)}} I_{\{\frac{S(\boldsymbol{X}_{jk})-n_0}{r} \geq \gamma\}} \tag{7.19}$$

   as an unbiased estimator of ℓ.

We performed various numerical studies with Algorithm 7.7.1 and found that it performs properly, provided that γ is chosen such that ℓ is not a rare event probability; otherwise one needs to use the importance sample method. For example, for a random Erdos-Renyi graph with 15 nodes and 50 edges we obtained via full enumeration that number of valid paths is 643,085 and the probability $\ell = \mathbb{P}\{S(\boldsymbol{X}) \geq \gamma\}$ that the length $S(\boldsymbol{X})$ of a randomly chosen path $\boldsymbol{X}$ is greater than $\gamma = 10$ is $\ell = 0.8748$. From our numerical results with $N^{(e)} = 100$ and $M = 100$ based on 10 runs we obtained with Algorithm 7.7.1 an estimator $\widehat{\ell} = 0.8703$ with the relative error about 1%.

7.7.2 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 [69] that the latter number coincides with the permanent of the corresponding $0-1$ matrix $\boldsymbol{A}$. 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 $\boldsymbol{A}$, that is, $|Q_n| = \mathrm{per}(\boldsymbol{A})$, defined as

$$\mathrm{per}(\boldsymbol{A}) = |\mathcal{X}^*| = \sum_{\boldsymbol{x}\in\mathcal{X}} \prod_{i=1}^{n} a_{ix_i}, \tag{7.20}$$

  $\mathcal{X}$ is the set of all permutations $\boldsymbol{x} = (x_1, \dots, x_n)$ 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 7.7.1. 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 7.7.3. Consider the following adjacency matrix

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{7.21}$$

The associated bipartite graph is given in Figure 7.20.

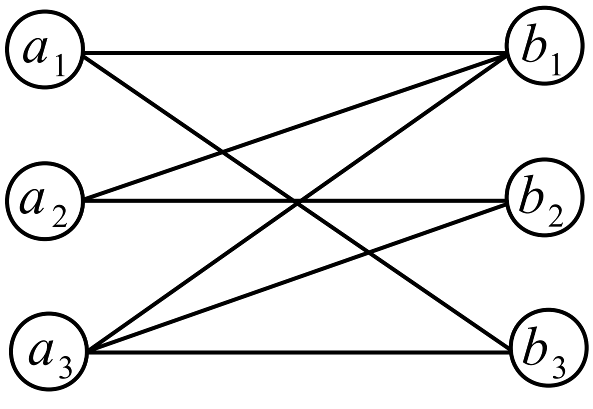


Figure 7.20: The bipartite graph.

It has the following three perfect matchings

$$\begin{aligned} M_1 &= [(a_1, b_1),\ (a_2, b_2),\ (a_3, b_3)], \\ M_2 &= [(a_1, b_3),\ (a_3, b_2),\ (a_2, b_1)], \\ M_3 &= [(a_1, b_3),\ (a_3, b_1),\ (a_2, b_2)]. \end{aligned} \tag{7.22}$$

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 (a_1, b_1) and (a_1, b_3). 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 (a_1, b_1) is active and (a_1, b_3) 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 (a_1, b_1) and (a_1, b_3) we generate two different perfect matchings (see (7.22)), 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 (a_1, b_1) is active. This automatically implies that all three neighboring edges (a_1, b_3), (a_3, b_1), (a_2, b_1) must be passive. Using the perfect matching oracle we will arrive at the next active edge, which is (a_2, b_2). Since the degree of node 3 is two and since (a_2, b_1) must be passive we have that $N_2 = 1, \nu_2 = 1$.

  (c) **Iteration 3** Since (a_2, b_2) is active (a_3, b_2) must be passive. The degree of node 5 is three, but since (a_3, b_2) and (a_3, b_1) are passive (a_3, b_3) 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 (a_1, b_3) is active we automatically set the neighboring edges (a_1, b_1), (a_3, b_3) as passive. Using an oracle we shall arrive at node 5 which has degree three. Since (a_1, b_1) is passive it is easily seen that each edge, (a_3, b_1) and (a_3, b_2), will become active with probability 1/2. This means that with (a_3, b_1) and (a_3, b_2) we are in a similar situation (in the sense of active-passive edges) to that of (a_1, b_1), and (a_1, b_3). We thus have for each case $N_2 = 2, \nu_2 = 2$.

(c) **Iteration 3** It is readily seen that both cases (a_3, b_1), and (a_3, b_2) lead to $N_3 = 1, \nu_3 = 1$. The resulting perfect matchings (see (7.22)) and the estimator of $|\mathcal{X}^*|$ are

$$\begin{aligned} M_3 &= [(a_1, b_3),\ (a_3, b_1),\ (a_2, b_2)], \\ M_2 &= [(a_1, b_3),\ (a_3, b_2),\ (a_2, b_1)] \end{aligned} \tag{7.23}$$

and

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

respectively.

Since each initial edge (a_1, b_1) and (a_1, b_3) 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 $g^*(\boldsymbol{x}) = 1/3$. Its corresponding conditional probabilities (for $N^{(e)} = 1$, starting at node 1) are given below:

$$g^* = \begin{pmatrix} 0 & 1/3 & 0 & 0 & 0 & \mathbf{2/3} \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & (1,\mathbf{1}) & 0 \\ 0 & \mathbf{1/2} & 0 & \mathbf{1/2} & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{7.24}$$

   The plain and bold numbers in (7.24) correspond to the trajectories starting at edges (a_1, b_1) and (a_1, b_3), respectively. Also, the notation $(1, \mathbf{1})$ means that the two perfect matching trajectories (one starting at (a_1, b_1) and one at (a_1, b_3), each with probability 1) pass through node (a_2, b_2).

7.7.3 Counting SAT's

The most common SAT problem comprises the following two components:

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

The binary vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ 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 $\overline{x}_i$, for each $i = 1, \ldots, n$.

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

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

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) [19], 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 [98], 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 [41], [42] 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.

7.8 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 chose 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 7.2). 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 chose $N^{(e)} = K$ and obtain maximum variance reduction (see Table 7.2). 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 necessary 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 7.18 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 $e_1 e_3$ and $e_2 e_3$, 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 runes with several different values of M and selected the best one.

7.9 Numerical Results

Here we present numerical results with the SE-OSLA and SE algorithms. In particular, we use SE-OSLA Algorithm 7.4.1 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 7.5.1, 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 7.5.1 we set M as suggested to follow Section 7.8. 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 runes with several values of M and select the best one.

We use 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. $\nu_t = N_t^{(e)}/N_t$.

7.9.1 Counting SAW's

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

Tables 7.3 and 7.4 present the performance of the SE-OSLA Algorithm 7.4.1 for SAW for $n = 500$ and $n = 1,000$ respectively, with $r = 2$, $n_0 = 4$ and $M = 20$ (see (7.7)). This corresponds to the initial values $N_0^{(e)} = 100$ and $N_1 = 780$, (see also iteration $t = 0$ in Table 7.5).

Table 7.3: Performance of SE-OSLA Algorithm 7.4.1 for SAW with $n = 500$

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

Based on those runs, we found that $RE = 0.0685$.

Table 7.4: Performance of of the SE-OSLA Algorithm 7.4.1 for SAW for $n = 1,000$

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

Based on those runs, we found that $RE = 0.0901$.

Table 7.5 presents dynamics of one of the runs of the SE-OSLA Algorithm 7.4.1 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 7.5: Dynamics of a run of the SE-OSLA Algorithm 7.4.1 for $n = 500$

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

7.9.2 Counting the Number of Trajectories in a Network

Model 1: from Roberts and Kroese [76] with $n = 24$ nodes

Table 7.6 presents the performance of the SE Algorithm 7.5.1 for the **Model 1** taken from Roberts and Kroese [76] with the following adjacency (24×24) matrix.

$$\begin{pmatrix}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&1&1&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&1&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&1&0&0&1&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0\\
1&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
0&1&1&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&1&0&1&0&1&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
1&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&1&1\\
1&0&0&1&1&0&1&1&1&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
1&0&0&1&1&0&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&1&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0
\end{pmatrix} \tag{7.25}$$

We set $N_t^{(e)} = 50$ and $M = 400$ to get a comparable running time.

Table 7.6: Performance of SE Algorithm 7.4.1 for the **Model 1** graph with $N_t^{(e)} = 50$ and $M = 400$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	18.87	1.81E+06	4.218
2	18.83	1.93E+06	4.187
3	18.83	1.98E+06	4.237
4	18.83	1.82E+06	4.232
5	18.79	1.90E+06	4.225
6	18.83	1.94E+06	4.231
7	18.86	1.86E+06	4.207
8	18.77	1.87E+06	4.172
9	18.79	1.90E+06	4.289
10	18.81	1.92E+06	4.287
Average	18.82	1.89E+06	4.229

Based on those runs, we found that $RE = 0.0264$. Comparing the results of Table 7.6 with these in [76] it follows that the former is about 1.5 times faster than the latter.

Model 2: Large Model ($n = 200$ vertices and 199 edges)

Table 7.7 presents the performance of the SE Algorithm 7.5.1 for the **Model 3** with $N_t^{(e)} = 100$ and $M = 500$.

Table 7.7: Performance of SE Algorithm 7.4.1 for the **Model 2** with $N_t^{(e)} = 100$ and $M = 500$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	58.93	1.53E+07	157.37
2	58.86	1.62E+07	153.92
3	58.96	1.69E+07	153.90
4	59.09	1.65E+07	154.50
5	58.73	1.57E+07	153.38
6	58.96	1.57E+07	153.71
7	59.13	1.67E+07	153.91
8	58.43	1.51E+07	153.14
9	59.08	1.62E+07	153.87
10	58.90	1.59E+07	154.81
Average	58.91	1.60E+07	154.25

We found for this model that $RE = 0.03606$.

We also counted the number of paths for Erdos-Renyi random graphs with $p = \ln n/n$ (see Section 7.11.3) We found that SE performs reliable (RE ≤ 0.05) for $n \leq 200$, provided the CPU time is limited by 5-15 minutes.

Table presents the performance of SE Algorithm 7.5.1 for the Erdos-Renyi random graph with $n = 200$ using $N_t^{(e)} = 1$ and $M = 30,000$.

Table 7.8: Performance of SE Algorithm 7.5.1 for the Erdos-Renyi random graph ($n = 200$) with $N_t^{(e)} = 1$ and $M = 30,000$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	80.08	1.43E+55	471.0
2	80.43	1.39E+55	499.0
3	80.51	1.41E+55	525.0
4	80.76	1.38E+55	507.9
5	80.43	1.45E+55	505.5
Average	80.44	1.41E+55	501.7

We found that the RE is 2.04E-02.

Remark 7.9.1. The SE method has some limitation, in particular when dealing with non-sparse instances. In this case one has to use in SE the full enumeration step ("consult" the oracle) very many times. As results the CPU time of SE might increase dramatically. Consider for example, the extreme case, a complete graph K_n. The exact number of s-t paths between any 2 vertices is given by:
$K(n) = \sum_{k=0}^{n-2} \frac{(n-2)!}{k!}$

Table 7.9 presents the performance of the SE Algorithm 7.5.1 for K_{25} with $N_t^{(e)} = 50$ and $M = 100$. The exact solution is $7.0273E + 022$

Table 7.9: Performance of SE Algorithm 7.4.1 for K_{25} with $N_t^{(e)} = 50$ and $M = 100$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	23.00	7.07E+22	28.71
2	23.00	6.95E+22	28.99
3	23.00	7.10E+22	28.76
4	23.00	7.24E+22	28.33
5	23.00	7.03E+22	27.99
6	23.00	6.75E+22	28.32
7	23.00	7.18E+22	29.25
8	23.00	7.15E+22	29.51
9	23.00	6.85E+22	28.20
10	23.00	7.34E+22	28.12
Average	23.00	7.07E+22	28.62

For this model we found that $RE = 0.0248$, so the results are still good. However, as n increases, the CPU time grows rapidly. For example, for K_{100} we found that while using $N_t^{(e)} = 100$ and $M = 100$ the CPU time is about 5.1 hours.

7.9.3 Counting the Number of Perfect Matchings (Permanent) in a Graph

As before we present here numerical results for two models, one small and one large.

Model 1: (Small Model)

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

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Table 7.10 present performance of SE Algorithm 7.5.1 for ($N_t^{(e)} = 50$ and $M = 10$). We found that the relative error is 0.0268.

Table 7.10: Performance of the SE Algorithm 7.5.1 for the **Model** 1 with $N_t^{(e)} =$ 50 and $M = 10$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}^*}\rvert$	CPU
1	24	264.21	2.056
2	24	269.23	2.038
3	24	270.16	2.041
4	24	268.33	2.055
5	24	272.10	2.064
6	24	259.81	2.034
7	24	271.62	2.035
8	24	269.47	2.050
9	24	264.86	2.059
10	24	273.77	2.048
Average	24	268.36	2.048

Applying splitting algorithm for the same model using $N = 15,000$ and $\rho = 0.1$, we found that the relative error is 0.2711. It follows that SE is about 100 times faster than splitting.

Model 2 with 100×100 **(Large Model)**

Table 7.11 presents the performance of the SE Algorithm 7.5.1 for **Model 2** matrix $N_t^{(e)} = 100$ and $M = 100$.

Table 7.11: Performance of SE Algorithm 7.5.1 for the for **Model 2** with $N_t^{(e)} = 100$ and $M = 100$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	93	1.58E+05	472.4
2	93	1.77E+05	482.8
3	93	1.77E+05	472.4
4	93	1.65E+05	482.0
5	93	1.58E+05	475.7
6	93	1.78E+05	468.9
7	93	1.76E+05	469.3
8	93	1.73E+05	480.9
9	93	1.74E+05	473.7
10	93	1.78E+05	472.0
Average	93	1.71E+05	475.0

The relative error is 0.0434.

Model 3: Erdos-Renyi graph with $n = 100$ edges and $p = 0.07$

Table 7.12 presents the performance of the SE Algorithm 7.5.1 for the Erdos-Renyi graph with $n = 100$ edges and $p = 0.07$. We set $N_t^{(e)} = 1$ and $M = 20,000$.

Table 7.12: Performance of SE Algorithm 7.5.1 for the Erdos-Renyi graph using $N_t^{(e)} = 1$ and $M = 20,000$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}}^*\rvert$	CPU
1	50	3.60E+07	3.29E+02
2	50	3.40E+07	3.50E+02
3	50	3.54E+07	3.30E+02
4	50	3.61E+07	3.40E+02
5	50	3.60E+07	3.33E+02
6	50	3.79E+07	3.27E+02
7	50	3.58E+07	3.33E+02
8	50	3.42E+07	3.35E+02
9	50	3.88E+07	3.25E+02
10	50	3.55E+07	3.19E+02
Average	50	3.60E+07	332.09179

We found that the relative error is 0.0387.

We also applied the SE algorithm for counting general matchings in a bipartite graph. The quality of the results was similar to that for paths counting in a network.

7.9.4 Counting SAT's

Here we present numerical results for several SAT models. We set $r = 1$ in the SE algorithm for all models. Recall that for 2-SAT we can use a polynomial decision making oracle [19], while for K-SAT ($K > 2$)- an heuristic based on the DPLL solver [20, 19]. Note that the SE Algorithm 7.5.1 remains exactly the same when a polynomial decision making oracle is replaced by an heuristic one, like the DPLL solver. Running both SAT models we found that the CPU time for 2-SAT is about 1.3 times faster then for K-SAT ($K > 2$).

Table 7.13 presents the performance of the SE Algorithm 7.5.1 for the 3-SAT 75×325 model with exactly 2258 solutions. We set $N_t^{(e)} = 20$ and $M = 100$.

Table 7.13: Performance of SE Algorithm 7.5.1 for the 3-SAT 75×325 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}^*}\rvert$	CPU
1	75	2359.780	2.74
2	75	2389.660	2.77
3	75	2082.430	2.79
4	75	2157.850	2.85
5	75	2338.100	2.88
6	75	2238.940	2.75
7	75	2128.920	2.82
8	75	2313.390	3.04
9	75	2285.910	2.81
10	75	2175.790	2.85
Average	75	2247.077	2.83

Based on those runs, we found that $RE = 0.0448$.

Table 7.14 presents performance of the SE Algorithm 7.5.1 for the 3-SAT 75×270 model. We set $N_t^{(e)} = 100$ and $M = 100$. 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 second (compare with average 16.8 second time of SE in Table 7.14).

Table 7.14: Performance of SE Algorithm 7.5.1 for the 3-SAT 75×270 model.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}^*}\rvert$	CPU
1	75	1.42E+06	16.88
2	75	1.37E+06	16.91
3	75	1.31E+06	17.24
4	75	1.35E+06	17.38
5	75	1.31E+06	16.75
6	75	1.32E+06	16.81
7	75	1.32E+06	16.51
8	75	1.25E+06	16.27
9	75	1.45E+06	16.3
10	75	1.33E+06	16.95
Average	75	1.35E+06	16.80

Based on those runs, we found that $RE = 0.0409$.

Table 7.15 presents the performance of the SE Algorithm 7.5.1 for a 3-SAT 300×1080 model. We set $N_t^{(e)} = 300$ and $M = 300$.

Table 7.15: Performance of SE Algorithm 7.5.1 for SAT 300×1080 model with $N_t^{(e)} = 300$, $M = 300$ and $r = 1$.

Run N_0	Iterations	$\lvert\widetilde{\mathcal{X}^*}\rvert$	CPU
1	300	3.30E+24	2010.6
2	300	3.46E+24	2271.8
3	300	3.40E+24	2036.8
4	300	3.42E+24	2275.8
5	300	3.39E+24	2022.4
6	300	3.35E+24	2267.8
7	300	3.34E+24	2019.6
8	300	3.34E+24	2255.4
9	300	3.32E+24	2031.7
10	300	3.33E+24	2149.6
Average	300	3.36E+24	2134.1

Based on those runs, we found that $RE = 0.0266$.

Note that in this case the estimator of $|\mathcal{X}^*|$ is very large and, thus full enumeration is imposable. 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 7.9.2 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 7.9.2. *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 it 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 $|\mathcal{X}_1^*|, ..., |\mathcal{X}_k^*|$. 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 it's 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 consist 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 7.16 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 7.16: The relative errors as function of r.

Instance	$r = 1$	$r = 2$	$r = 3$
20x80, $N_t^{(e)}$=3, $M = 100$	2.284E-02	1.945E-02	2.146E-02
75x325, $N_t^{(e)} = 20$, $M = 100$	5.057E-02	4.587E-02	5.614E-02
75x270, $N_t^{(e)} = 100$, $M = 100$	4.449E-02	4.745E-02	4.056E-02

Comparison of SE with Splitting and SampleSearch

Here we compare SE with splitting and SampleSearch for several 3-SAT instances. Before doing so we need the following remarks.

Remark 7.9.3. SE versus splitting Note that

1. In the splitting method [82] the rare event ℓ is presented a as

$$\ell = \mathbb{E}\left[I_{\left\{\sum_{j=1}^m C_j(\boldsymbol{X}) = m\right\}} \right], \tag{7.26}$$

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

$$\{(m_0, g^*(\boldsymbol{x}, m_0)),\ (m_1, g^*(\boldsymbol{x}, m_1)),\ (m_2, g^*(\boldsymbol{x}, m_2)), \ldots, (m_T, g^*(\boldsymbol{x}, m_T))\}, \tag{7.27}$$

where $g^*(\boldsymbol{x}, m_t),\ t = 0, 1, \ldots, T$ is uniformly distributed on the set $\mathcal{X}_t$ and such that $\mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_T = \mathcal{X}^*$. It is crucial to note, however, that

2. In contrast to splitting which samples from a sequence of n-dimensional pdf's $g^*(\boldsymbol{x}, m_t)$ (see (7.27)) sampling in the SE Algorithm 7.5.1 is minimal; it resort to sampling only n times. In particular SE draws $N_t^{(e)}$ balls (without replacing) from an urn containing $N_t \geq N_t^{(e)},\ t = 1, \ldots, n$ 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 [82].

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 7.9.4. SE versus SampleSearch

The main difference between `"SampleSearch"` [41, 42] 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 [98] `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 7.17 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 7.17: 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 [82] 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 $x_1, \ldots, x_n$, while the other two sample (random vectors $\boldsymbol{X}$ from the IS pdf $g(\boldsymbol{x})$) in the entire n-dimensional space.

We also ran the models of Table 7.17 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.

7.10 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 problems 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 7.4.1 as compared with the well-known algorithms and in particular over the splitting [82], and SampleSearch [41] methods.

As for further research, we are planning to

- Find the set of optimal parameters $\{N_t^{(e)},\ r,\ M\}$ which, for fixed n, minimizes the variance of the estimator $\widetilde{|\mathcal{X}|^*} = \widetilde{|\mathcal{X}|^*}(N_t^{(e)},\ r,\ M)$ in (7.7) 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 [73] FPRAS result for counting permanent with OSLA to counting some other graphs quantities using nSLA and SE.

7.11 Appendices

7.11.1 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

$$\ell = \mathbb{E}_f[S(\boldsymbol{X})] = \int S(\boldsymbol{x})\, f(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \ , \tag{7.28}$$

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

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

$$\ell = \int S(\boldsymbol{x})\, \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})}\, g(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} = \mathbb{E}_g\left[S(\boldsymbol{X})\, \frac{f(\boldsymbol{X})}{g(\boldsymbol{X})}\right] , \tag{7.29}$$

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

$$\widehat{\ell} = \frac{1}{N} \sum_{k=1}^{N} S(\boldsymbol{X}_k)\, \frac{f(\boldsymbol{X}_k)}{g(\boldsymbol{X}_k)} \tag{7.30}$$

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

$$W(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})} \ , \tag{7.31}$$

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

$$g(\boldsymbol{x}) = g_1(x_1)\, g_2(x_2 \,|\, x_1) \cdots g_n(x_n \,|\, x_1, \ldots, x_{n-1}) \, , \tag{7.32}$$

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 $\boldsymbol{X}$ from $g(\boldsymbol{x})$. Repeating this independently N times, each time sampling from $g(\boldsymbol{x})$, one obtains a random sample $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ from $g(\boldsymbol{x})$ and estimates ℓ according to (7.30). To further simplify the notation, we abbreviate $(x_1, \ldots, x_t)$ to $\boldsymbol{x}_{1:t}$ for all t. In particular, $\boldsymbol{x}_{1:n} = \boldsymbol{x}$. Typically, t can be viewed as a (discrete) time parameter and $\boldsymbol{x}_{1:t}$ as a path or trajectory. By the product rule of probability, the target pdf $f(\boldsymbol{x})$ can also be written sequentially, that is,

$$f(\boldsymbol{x}) = f(x_1)\, f(x_2 \,|\, x_1) \cdots f(x_n \,|\, \boldsymbol{x}_{1:n-1}). \tag{7.33}$$

From (7.32) and (7.33) it follows that we can write the likelihood ratio in product form as

$$W(\boldsymbol{x}) = \frac{f(x_1)\, f(x_2 \,|\, x_1) \cdots f(x_n \,|\, \boldsymbol{x}_{1:n-1})}{g_1(x_1)\, g_2(x_2 \,|\, x_1) \cdots g_n(x_n \,|\, \boldsymbol{x}_{1:n-1})} \tag{7.34}$$

or recursively as

$$w_t(\boldsymbol{x}_{1:t}) = u_t\, w_{t-1}(\boldsymbol{x}_{1:t-1}), \quad t = 1, \ldots, n\ , \tag{7.35}$$

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

$$u_t = \frac{f(x_t \,|\, \boldsymbol{x}_{1:t-1})}{g_t(x_t \,|\, \boldsymbol{x}_{1:t-1})} = \frac{f(\boldsymbol{x}_{1:t})}{f(\boldsymbol{x}_{1:t-1})\, g_t(x_t \,|\, \boldsymbol{x}_{1:t-1})}\ , \quad t = 2, \ldots, n \tag{7.36}$$

are *incremental weights*.

Remark 7.11.1. 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 $W(\boldsymbol{x})$ is known up to a constant as well, say $W(\boldsymbol{x}) = C\, V(\boldsymbol{x})$, where $1/C = \mathbb{E}_g[V(\boldsymbol{X})]$ 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*:

$$\widehat{\ell} = \frac{\sum_{k=1}^{N} S(\boldsymbol{X}_k)\, W_k}{\sum_{k=1}^{N} W_k}\ . \tag{7.37}$$

rather than the likelihood ratio estimator (7.31). Here the $\{W_k\}$, with $W_k = W(\boldsymbol{X}_k)$, are interpreted as *weights* of the random sample $\{\boldsymbol{X}_k\}$, and the sequence $\{(\boldsymbol{X}_k, W_k)\}$ is called a *weighted (random) sample* from $g(\boldsymbol{x})$.

Summarizing, the SIS method can be written as follows.

Algorithm 7.11.1 (SIS Algorithm)**.**

1. For each finite $t = 1, \ldots, n$, sample X_t from $g_t(x_t \,|\, \boldsymbol{x}_{1:t-1})$.

2. Compute $w_t = u_t\, w_{t-1}$, where $w_0 = 1$ and u_t is given in (7.36).

3. Repeat N times and estimate ℓ via $\widehat{\ell}$ in (7.30) or $\widehat{\ell}$ in (7.37).

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

7.11.2 DPLL Algorithm from Wikipedia

Davis-Putnam-Logemann-Loveland (DPLL) algorithm [19] 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. 1 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.

7.11.3 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:

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

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 $p(1-p)(n-1)^2$.

It readily follows from the above that even for $N^{(e)}=1$ we have that

$$\mathbb{V}\text{ar}\{\widehat{|\mathcal{X}^*|}\} \leq p(1-p)\binom{n}{2}^2 < n^4. \tag{7.38}$$

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 $O(\log n)$.

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 $O(\log n)$ 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 $p > \frac{(1+\varepsilon)\ln n}{n}$.

Chapter 8

Conclusion and Further Research

In this work we introduced three general techniques for approximating the solutions of general NP-hard, NP-complete, and #P-complete problems. The main problems handled in this thesis are: combinatorial optimization, uniform sampling on complex regions, counting the number of satisfiability assignments in a cnf formulas, approximating the permanent, counting the number of contingency tables and graphs with prescribed degrees and a rare-event estimation for the network flow reliability. In particular, we showed the following.

1. The *splitting* algorithm is a general and very powerful technique capable to handle many hard problems in sense of optimization and counting. As compared to the classical randomized algorithms, the proposed splitting algorithms require very little warm-up time when running the Gibbs sampler from iteration to iteration, since the underlying Markov chains are already in steady-state from the beginning. The only purpose remaining for them is fine tuning in order for the associated Markov processes to be kept in the steady-state while moving from iteration to iteration. The fine tuning is performed by introducing the splitting and burn-in parameters η and b, respectively. As result we obtain substantial variance reduction and thus, dramatic speedup as compared to the randomized algorithms.

   We introduced an entirely new, flexible and robust version of the *splitting* algorithm - the *smoothed splitting method* (SSM) that operates in the continuous space while solving discrete optimization and counting problems.

   We showed that *splitting* can successfully generate almost uniform samples in hard regions. Finally, a classical capture-recapture method can lead to a substantial variance reduction for many problem instances.

2. We showed how the classical *Evolution* methods, initially developed for network reliability problems can be used to successfully solve network flow problems.

3. We introduced entirely new sequential importance sampling algorithm called the *Stochastic Enumeration* method for counting $\#P$-complete problems like SAT, permanent and self avoiding walks.

4. We presented efficient numerical results with all the algorithms described above for a variety of real-world benchmark problem instances.

Further Research

As for further research, we suggest the following issues:

1. Establish solid mathematical grounding based on the splitting and SSM methods [28] , on use of the Feynman-Kac formulae and on interacting particle systems (Del Moral, 2004). We believe that the prove for (polynomial) speed of convergence for quite general counting problems could be established by using arguments similar to these we used in Appendix under simplifying assumptions.

2. Find the total sample size and the size of the elites at each iteration of the splitting algorithm , while estimating the rare-event probability
$$\ell = \mathbb{E}_f \left[I_{\{\sum_{i=1}^m C_i(\boldsymbol{X}) \geq m\}} \right],$$
which involves a sum of *dependent* Ber(1/2) random variables $C_i(\boldsymbol{X}),\ i = 1, \ldots, m$. Recall that the goal of estimator of ℓ is to insure approximate uniformity of the sample at each sub-region $\mathcal{X}_t$ via formula $|\mathcal{X}^*| = \ell|\mathcal{X}|$. Note that both, the CE and the exponential change of measure have limited applications in this case.

3. Further investigate the convergence properties of some alternatives to the Gibbs sampler used in this thesis, like the hit-and-run and Metropolis-Hastings combined with splitting.

4. In this work we applied the splitting method to some new problems like counting graphs with prescribed degrees and contingency tables. Clearly additional research is needed in sense of employing our method to a broad variety of optimization and counting problems. A special interest can be an adaptation to the continuous case. The splitting framework can provide both optimization and counting the number of multiple extrema in a multi-extremal function.

5. The splitting method can have an impact in various queueing models with heavy tails. As an example, one can consider estimating the probability of buffer overflow in the *GI/G/1* queue.

6. While working with the methods presented in chapter 6, we noticed that it can be used for various problems having monotonic properties. For example, it can be used for counting graph vertex and edge covers or even a more general monotone cnf instances.

7. We showed numerically that the Stochastic Enumeration method is a powerful tool but more rigorous mathematical background should be established. One possible direction of research would be to supply the SE with estimation oracle that will suggest the correct probabilities during the algorithm walk down the tree. The SE can benefit much from it's use. See for example the sequential importance sampling procedure proposed in [74]

Appendix A

Additional topics

A.1 Efficiency of Estimators

In this work we frequently use

$$\widehat{\ell} = \frac{1}{N} \sum_{i=1}^{N} Z_i \, , \tag{A.1}$$

which presents an *unbiased* estimator of the unknown quantity $\ell = \mathbb{E}[Z]$, where $Z_1, \ldots, Z_N$ are independent replications of some random variable Z.

By the central limit theorem, $\widehat{\ell}$ has approximately a $\mathsf{N}(\ell, N^{-1}\mathbb{V}\mathrm{ar}[Z])$ distribution for large N. We shall estimate $\mathbb{V}\mathrm{ar}[Z]$ via the *sample variance*

$$S^2 = \frac{1}{N-1} \sum_{i=1}^{N} (Z_i - \widehat{\ell})^2 \, .$$

By the law of large numbers, S^2 converges with probability 1 to $\mathbb{V}\mathrm{ar}[Z]$. Consequently, for $\mathbb{V}\mathrm{ar}[Z] < \infty$ and large N, the approximate $(1-\alpha)$ confidence interval for ℓ is given by

$$\left(\widehat{\ell} - z_{1-\alpha/2} \frac{S}{\sqrt{N}}, \; \widehat{\ell} + z_{1-\alpha/2} \frac{S}{\sqrt{N}} \right) \, ,$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$ quantile of the standard normal distribution. For example, for $\alpha = 0.05$ we have $z_{1-\alpha/2} = z_{0.975} = 1.96$. The quantity

$$\frac{S/\sqrt{N}}{\widehat{\ell}}$$

is often used in the simulation literature as an accuracy measure for the estimator $\widehat{\ell}$. For large N it converges to the *relative error* of $\widehat{\ell}$, defined as

$$\mathrm{RE} = \frac{\sqrt{\mathbb{V}\mathrm{ar}[\widehat{\ell}]}}{\mathbb{E}[\widehat{\ell}]} = \frac{\sqrt{\mathbb{V}\mathrm{ar}[Z]/N}}{\ell} \, . \tag{A.2}$$

The square of the relative error

$$\mathrm{RE}^2 = \frac{\mathbb{V}\mathrm{ar}[\widehat{\ell}]}{\ell^2} \tag{A.3}$$

is called the *squared coefficient of variation.*

Example A.1.1 (Estimation of Rare-Event Probabilities)**.** Consider estimation of the tail probability $\ell = \mathbb{P}(X \geq \gamma)$ of some random variable X for a *large* number γ. If ℓ is very small, then the event $\{X \geq \gamma\}$ is called a *rare event* and the probability $\mathbb{P}(X \geq \gamma)$ is called a *rare-event probability.*

We may attempt to estimate ℓ via (A.1) as

$$\widehat{\ell} = \frac{1}{N} \sum_{i=1}^{N} I_{\{X_i \geq \gamma\}} \ , \tag{A.4}$$

which involves drawing a random sample $X_1, \ldots, X_N$ from the pdf of X and defining the indicators $Z_i = I_{\{X_i \geq \gamma\}}$, $i = 1, \ldots, N$. The estimator ℓ thus defined is called the *crude Monte Carlo* (CMC) estimator. For small ℓ the relative error of the CMC estimator is given by

$$\mathrm{RE} = \frac{\sqrt{\mathbb{V}\mathrm{ar}[\widehat{\ell}]}}{\mathbb{E}[\widehat{\ell}]} = \sqrt{\frac{1-\ell}{N\,\ell}} \approx \sqrt{\frac{1}{N\,\ell}} \ . \tag{A.5}$$

As a numerical example, suppose that $\ell = 10^{-6}$. In order to estimate ℓ accurately with relative error (say) RE $= 0.01$, we need to choose a sample size

$$N \approx \frac{1}{\mathrm{RE}^2 \ell} = 10^{10} \ .$$

This shows that estimating small probabilities via CMC estimators is computationally meaningless. □

A.1.1 Complexity

The theoretical framework in which one typically examines rare-event probability estimation is based on *complexity theory*, (see, for example, [2]).

In particular, the estimators are classified either as *polynomial-time* or as *exponential-time.* It is shown in [2] that for an arbitrary estimator, $\widehat{\ell}$ of ℓ, to be polynomial-time as a function of some γ, it suffices that its squared coefficient of variation, RE^2, or its relative error, RE, is bounded in γ by some polynomial function, $p(\gamma)$. For such polynomial-time estimators, the required sample size to achieve a fixed relative error does not grow too fast as the event becomes rarer.

Consider the estimator (A.4) and assume that ℓ becomes very small as $\gamma \to \infty$. Note that

$$\mathbb{E}[Z^2] \geq (\mathbb{E}[Z])^2 = \ell^2 \;.$$

Hence, the best one can hope for with such an estimator is that its second moment of Z^2 decreases proportionally to ℓ^2 as $\gamma \to \infty$. We say that the rare-event estimator (A.4) has *bounded relative error* if for all γ

$$\mathbb{E}[Z^2] \leq c\,\ell^2 \tag{A.6}$$

for some fixed $c \geq 1$. Because bounded relative error is not always easy to achieve, the following weaker criterion is often used. We say that the estimator (A.4) is *logarithmically efficient* (sometimes called *asymptotically optimal*) if

$$\lim_{\gamma\to\infty} \frac{\log \mathbb{E}[Z^2]}{\log \ell^2} = 1 \;. \tag{A.7}$$

Example A.1.2 (The CMC Estimator is not Logarithmically Efficient)**.** Consider the CMC estimator (A.4). We have

$$\mathbb{E}[Z^2] = \mathbb{E}[Z] = \ell \;,$$

so that

$$\lim_{\gamma\to\infty} \frac{\log \mathbb{E}[Z^2]}{\log \ell^2(\gamma)} = \frac{\log \ell}{\log \ell^2} = \frac{1}{2} \;.$$

Hence, the CMC estimator is not logarithmically efficient, and therefore alternative estimators must be found to estimate small ℓ. □

A.1.2 Complexity of Randomized Algorithms

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

$$\mathbb{P}(|Z - z| \leq \varepsilon z) \geq 1 - \delta, \tag{A.8}$$

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

One of the main tools in proving (A.8) for various randomized algorithms is the so-called *Chernoff bound*, which states that for any random variable Y and any number a

$$\mathbb{P}(Y \leq a) \leq \min_{\theta>0} \mathrm{e}^{\theta a}\, \mathbb{E}[\mathrm{e}^{-\theta Y}] \;. \tag{A.9}$$

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

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

The bound (A.9) now follows by taking the smallest such θ.

An important application is the following.

Theorem A.1.1. *Let $X_1, \ldots, X_n$ be iid* $\mathsf{Ber(p)}$ *random variables, then their sample mean provides an (ε, δ)-approximation for p, that is,*

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

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

For the proof see, for example Rubinstein and Kroese [86].

Definition A.1.1 (FPRAS)**.** A randomized algorithm is said to provide a *fully polynomial-time randomized approximation scheme (FPRAS)* if for any input vector $\boldsymbol{x}$ and any parameters $\varepsilon > 0$ and $0 < \delta < 1$ the algorithm outputs an (ε, δ)-approximation to the desired quantity $z(\boldsymbol{x})$ in time that is, polynomial in $\varepsilon^{-1}, \ln \delta^{-1}$ and the size n of the input vector $\boldsymbol{x}$.

Note that the sample mean in Theorem A.1.1 provides a FPRAS for estimating p. Note also that the input vector $\boldsymbol{x}$ consists of the Bernoulli variables $X_1, \ldots, X_n$.

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

Definition A.1.2 (ε-Uniform Sample)**.** Let Z be a random output of a sampling algorithm for a finite sample space $\mathcal{X}$. We say that the sampling algorithm generates an *ε-uniform sample* from $\mathcal{X}$ if, for any $\mathcal{Y} \subset \mathcal{X}$

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

Definition A.1.3 (Variation Distance)**.** The variation distance between two distributions F_1 and F_2 on a countable space $\mathcal{X}$ is defined as

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

It is well-known, [67] that the definition of variation distance coincides with that of an ε- uniform sample in the sense that a sampling algorithm returns an ε- uniform sample on $\mathcal{X}$ if and only if the variation distance between its output distribution F and the uniform distribution $\mathcal{U}$ satisfies

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

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

Definition A.1.4 (FPAUS)**.** A sampling algorithm is called a *fully polynomial almost uniform sampler (FPAUS)* if, given an input vector $\boldsymbol{x}$ and a parameter $\varepsilon > 0$, the algorithm generates an ε-uniform sample from $\mathcal{X}(\boldsymbol{x})$ and runs in a time that is, polynomial in $\ln \varepsilon^{-1}$ and the size of the input vector $\boldsymbol{x}$.

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

A.2 Complexity of Splitting Method under Simplifying Assumptions

Denote by
ℓ- the rare-event probability.
m - number of levels. $m_t,\ t = 0, 1, \dots, T$ - intermediate levels ($m_0 = 0, m_T = m$).
c_t - probability of hitting m_t, starting from m_{t-1}.
N_t - number of successful hits of level m_t (elite sample).
$N^{(t)}$ - total sample from level m_t.

Our main goal is to present a calculation for $\mathbb{V}\mathrm{ar}\widehat{\ell}$ and to show how much variance can be obtained with splitting versus the *crude Monte Carlo* (CMC). To do so we shall cite some basic results from [28].

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

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

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

1. The indicators $I_i^{(t)},\ i = 1, \dots, N^{(t)};\ t = 1, \dots, T$ are generated using the splitting algorithm.

2. For fixed m_t the indicators $I_i^{(t)}$ are iid and $\mathbb{E}I_i^{(t)} = c_t$. In addition, we assume that for all combinations (i, j) and for $t \neq k$, they are also independent level-wise, that is, $\mathbb{E}I_i^{(t)} I_j^{(k)} = c_t c_k$. Clearly, this is a simplified assumption, since in practice we generate only approximately uniform samples at each sub-space $\mathcal{X}_t$. As a result, each estimator of c_t might be slightly biased. Recall that $c_t = \mathbb{P}(\mathcal{X}_t | \mathcal{X}_{t-1})$, that is, it represents the conditional probability of the process (particle) reaching the sub-space (event) $\mathcal{X}_t$ from $\mathcal{X}_{t-1}$, or in other words of it reaching the level m_t from level m_{t-1}. Moreover, the indicators $I_i^{(t)}$ and $I_j^{(k)}$ might be slightly correlated as well, in particular when k is

close to t, say $k = t \pm 1$. We use the phrases "slightly biased" and "slightly correlated" being aware that our elites samples remain in "near" *steady-state (warm-up)* position in each sub-space $\mathcal{X}_t$. Recall that we do so by introducing the splitting and the burn-in parameters, b and η, respectively.

With this on hand, we have

$$\mathbb{V}\mathrm{ar}\widehat{\ell} = \mathbb{E}\widehat{\ell}^2 - \ell^2 = \mathbb{E}\prod_{t=1}^{T}\widehat{c}_t^2 - \ell^2. \tag{A.10}$$

Taking into account that

$$\mathbb{V}\mathrm{ar}\widehat{c}_t = \frac{1}{N^{(t)}}c_t(1-c_t), \tag{A.11}$$

we obtain

$$\mathbb{V}\mathrm{ar}\widehat{\ell} = \prod_{t=1}^{T}\left\{\frac{c_t(1-c_t)}{N^{(t)}} + c_t^2\right\} - \ell^2 = \ell^2\left(\prod_{t=1}^{T}\left\{\frac{1-c_t}{c_tN^{(t)}} + 1\right\} - 1\right). \tag{A.12}$$

As a simple example consider estimation of ℓ in the *Sum of Bernoulli* problem, that is estimation

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

where the X_i's are iid, each $X_i \sim$ Ber(1/2). Assume that $T = m$ and $N_t = N,\ t = 1,\ldots,m$. It is readily seen that in this case $c_t = 1/2$ and thus (A.12) reduces to

$$\mathbb{V}\mathrm{ar}\widehat{\ell} = \ell^2\left(1+\frac{1}{N}\right)^m - \ell^2. \tag{A.13}$$

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

$$\mathbb{V}\mathrm{ar}\widehat{\ell} \approx \ell^2(e-1). \tag{A.14}$$

We proceed next with (A.12). To find the optimal parameters T, and $(N_1,\ldots,N_T)$ in (A.12) we solve the following minimization problem

$$\min \mathbb{V}\mathrm{ar}\widehat{\ell} = \min \ell^2\left(\prod_{t=1}^{T}\left\{\frac{(1-c_t)}{c_tN^{(t)}} + 1\right\} - 1\right) \tag{A.15}$$

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

$$\sum_{t=1}^{T} N^{(t)} = M. \tag{A.16}$$

It is not difficult to show that for fixed T, the solution of (A.15)-(A.16) (see Garvels, 2000) is $c_t = c = \ell^{\frac{1}{T}}$ and $N^{(t)} = \frac{M}{T}$, $\forall\, k = 1, \ldots, T$. Mote that we assumed that $\sum_{t=1}^{T} N^{(t)}$ is large and in solving (A.15)-(A.16) we used a continuous approximation for the true discrete program (A.15)-(A.16). It follows that for fixed T, the minimal variance (under $c_t = c = \ell^{\frac{1}{T}}$ and $N^{(t)} = \frac{M}{T}$, $\forall\, k = 1, \ldots, T$) equals

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

It remains to solve

$$\min_T \mathbb{V}\mathrm{ar} \widehat{\ell} = \min_T \left\{ \frac{\ell^2 T^2 (1 - \ell^{\frac{1}{T}})}{\ell^{\frac{1}{T}} M} \right\}. \tag{A.17}$$

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

$$T_r = -\frac{\log \ell}{2}, \tag{A.18}$$

$$c_r = e^{-2}, \tag{A.19}$$

$$\mathbb{V}\mathrm{ar}_r \widehat{\ell} = \frac{(e \ell \log \ell)^2}{4M}, \tag{A.20}$$

$$\kappa_r^2 = \frac{M^{-1} \mathbb{V}\mathrm{ar}_r \widehat{\ell}}{\ell^2} \approx \frac{(e \log \ell)^2}{4}, \tag{A.21}$$

and

$$\varepsilon_r = \frac{M^{-1} \mathbb{V}\mathrm{ar}_r \widehat{\ell}}{\ell (1 - \ell)} \approx \frac{\ell (e \log \ell)^2}{4}, \tag{A.22}$$

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

Even though the assumptions are indeed simplifying, they provide good insight into the polynomial complexity of splitting Algorithm and in particular into the relative error κ_r^2. Note that for general counting problem, our numerical data are in agreement with these results and in particular with κ_r^2 in (A.21). Note finally, that for the Bernoulli model with $\mathbb{V}\mathrm{ar} \widehat{\ell} \approx \ell^2 (e - 1)$ (see (A.14)) we obtain (under the above simplifying assumptions) that $\kappa_r^2 = e - 1$ and thus, bounded relative error.

Bibliography

[1] S. Asmussen and P. W. Glynn. *Stochastic Simulation: Algorithms and Analysis*. Stochastic Modelling and Applied Probability, 57. Springer Science,Business Media, LLC, 2007.

[2] S. Asmussen and R. Y. Rubinstein. Steady state rare event simulation in queueing model and its complexity properties. In J. Dshalalow, editor, *Advances in Queueing: Theory, Methods and Open Problems*, volume I, pages 429–462. CRC Press, 1995.

[3] J. Blanchet and D. Rudoy. *Rare-Event Simulation and Counting Problems*. Wiley, 2009.

[4] J. H. Blanchet. Efficient importance sampling for binary contingency tables. *Annals of Applied Probability*, 19:949–982, 2009.

[5] J. Blitzstein and P. Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. Technical report, 2006.

[6] J. Blitzstein and P. Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. Technical report, 2006.

[7] Z. I. Botev and D. P. Kroese. An Efficient Algorithm for Rare-event Probability Estimation, Combinatorial Optimization, and Counting. *Methodology and Computing in Applied Probability*, 10(4):471–505, December 2008.

[8] Z. I. Botev and D. P. Kroese. Efficient monte carlo simulation via the generalized splitting method. *Statistics and Computing*, 22:1–16, 2012.

[9] Z. I. Botev, P. L'Ecuyer, G. Rubino, R. Simard, and B. Tuffin. Static network reliability estimation via generalized splitting. *INFORMS Journal on Computing*, 2012.

[10] A. Cayley. A theorem on trees. *Quart. J. Math*, 23:376–378, 1889.

[11] F. Cérou and A. Guyader. Adaptive multilevel splitting for rare event analysis. Research Report RR-5710, INRIA, 2005.

[12] F. Cérou, P. D. Moral, F. L. Gland, and P. Lezaud. *Genetic Genealogical Models in Rare Event Analysis*. Publication interne - IRISA. IRISA, 2006.

[13] F. Cérou, P. Del Moral, T. Furon, and A. Guyader. Sequential monte carlo for rare event estimation. *Statistics and Computing*, 22(3):795–808, 2012.

[14] Y. Chen, P. Diaconis, S. P. Holmes, and J. S. Liu. Sequential monte carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100:109–120, March 2005.

[15] N. Clisby. Efficient implementation of the pivot algorithm for self-avoiding walks. Technical Report arXiv:1005.1444, May 2010. Comments: 35 pages, 24 figures, 4 tables. Accepted for publication in the Journal of Statistical Physics.

[16] J. L. Cook and J. E. Ramirez-Marquez. Two-terminal reliability analyses for a mobile ad hoc wireless network. *Rel. Eng. & Sys. Safety*, 92(6):821–829, 2007.

[17] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[18] F. Crou, P. Del, Moral T. Furon, and A. Guyader. Rare event simulation for static distribution. Technical report, 2009.

[19] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962.

[20] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[21] T. Dean and P. Dupuis. Splitting for rare event simulation: a large deviations approach to design and analysis. *Stochastic Processes and their Applications*, 119(2):562–587, 2009.

[22] T. Dean and P. Dupuis. The design and analysis of a generalized RESTART/DPR algorithm for rare event simulation. *Annals of Operations Research*, 189(1):63–102, 2011.

[23] M. Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 693–699, 2003.

[24] M. Dyer, A. Frieze, and M. Jerrum. On counting independent sets in sparse graphs. In *In 40th Annual Symposium on Foundations of Computer Science*, pages 210–217, 1999.

[25] T. Elperin, I. Gertsbakh, and M. Lomonosov. An evolution model for Monte Carlo estimation of equilibrium network renewal parameters. *Probability in the Engineering and Informational Sciences*, 6:457–469, 1992.

[26] T. Elperin, I. B. Gertsbakh, and M. Lomonosov. Estimation of network reliability using graph evolution models. *IEEE Transactions on Reliability*, 40(5):572–581, 1991.

[27] P. Erdos and A. Rnyi. On random graphs. i. *Publicationes Mathematicae*, 6:290–297, 1959.

[28] M. J. J. Garvels. *The splitting method in rare event simulation.* PhD thesis, University of Twente, Enschede, October 2000.

[29] M. J. J. Garvels and D. P. Kroese. A comparison of RESTART implementations. In *Proceedings of the 30th conference on Winter simulation*, WSC '98, pages 601–608, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[30] M. J. J. Garvels, D. P. Kroese, and J. C. W. van Ommeren. On the importance function RESTART simulation. *European Transactions on Telecommunications*, 13(4), 2002.

[31] M. J. J. Garvels and R.Y. Rubinstein. A combined restart - cross entropy method for rare event estimation with applications to atm networks. Technical report, 2000.

[32] A. Gelman and D. B. Rubin. Inference from Iterative Simulation Using Multiple Sequences. *Statistical Science*, 7(4):457–472, 1992.

[33] I. Gertsbakh and Y. Shpungin. Combinatorial approaches to monte carlo estimation of network lifetime distribution. *Applied Stochastic Models in Business and Industry*, 20:49–57, 2004.

[34] I. Gertsbakh and Y. Shpungin. Network reliability importance measures: combinatorics and monte carlo based computations. *WSEAS Trans Comp*, 4:21–23, 2007.

[35] I. Gertsbakh and Y. Shpungin. *Models of network reliability: analysis, combinatorics, and Monte Carlo.* CRC Press, New York, 2009.

[36] I. Gertsbakh and Y. Shpungin. *Network reliability design: combinatorial and Monte Carlo approach.* Proceedings of the International Conference on Modeling and Simulation, pages 88-94, Canada, 2010.

[37] I. Gertsbakh and Y. Shpungin. *Network Reliability and Resilience* -. Springer, Berlin, Heidelberg, 1st edition. edition, 2011.

[38] I. Gertsbakh and Y. Shpungin. *Spectral Approach to Reliability Evaluation of Flow Networks.* Proceedings of the European Modeling and Simulation Symposium, Vienna, pages 68-73, 2012.

[39] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. A large deviations perspective on the efficiency of multilevel splitting. *IEEE Transactions on Automatic Control*, 43(12):1666–1679, 1998.

[40] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. Multilevel splitting for estimating rare event probabilities. *Operations Research*, pages 585–600, 1999.

[41] V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *AAAI*, pages 198–203. AAAI Press, 2007.

[42] V. Gogate and R. Dechter. Samplesearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2):694–729, 2011.

[43] A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[44] C. P. Gomes, A. Sabharwal, and B. Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, 2009.

[45] D. Günneç and F. S. Salman. Assessing the reliability and the expected performance of a network under disaster risk. *OR Spectr.*, 33(3):499–523, July 2011.

[46] M. Jerrum and A. Sinclair. The markov chain monte carlo method: An approach to approximate counting and integration. pages 482–520. PWS Publishing, 1996.

[47] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *Journal of the ACM*, pages 671–697, 2004.

[48] M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[49] M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[50] H. Kahn and T. E. Harris. Estimation of particle transmission by random sampling, monte carlo method. *National Bureau of Standards Applied Mathematics Series*, 12:27–30, 1951.

[51] R. M. Karp and M. Luby. Monte-carlo algorithms for enumeration and reliability problems. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, SFCS '83, pages 56–64, Washington, DC, USA, 1983. IEEE Computer Society.

[52] D. P. Kroese, K. P. Hui, and S. Nariai. Network reliability optimization via the cross-entropy method. *IEEE Transactions on Reliability*, 56(2):275–287, 2007.

[53] D. P. Kroese, T. Taimre, and Z. I. Botev. *Handbook of Monte Carlo methods*. John Wiley & Sons, New York, 2011.

[54] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[55] A. Lagnoux-Renaudie. A two-step branching splitting model under cost constraint for rare event analysis. *Journal of Applied Probability*, 46(2):429–452, 2009.

[56] P. LEcuyer, J. Blanchet, B. Tuffin, and P. Glynn. Asymptotic robustness of estimators in rare-event simulation. *ACM Transactions on Modeling and Computer Simulation - TOMACS*, 20:1–41, 2010.

[57] P. L'Ecuyer, V. Demeres, and B. Tuffin. Splitting for rare-event simulation. In *Proceedings of the 38th conference on Winter simulation*, WSC '06, pages 137–148. Winter Simulation Conference, 2006.

[58] P. L'Ecuyer, V. Demers, and B. Tuffin. Rare events, splitting, and quasi-monte carlo. *ACM Trans. Model. Comput. Simul.*, 17(2), April 2007.

[59] L. Lin and M. Gen. A Self-controlled Genetic Algorithm for Reliable Communication Network Design. pages 640–647.

[60] Y. Lin. Reliability of a stochastic-flow network with unreliable branches & nodes, under budget constraints. *IEEE Transactions on Reliability*, 53(3):381–387, 2004.

[61] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, corrected edition, January 2008.

[62] N. Madras and A. Sokal. The pivot algorithm - a highly efficient monte-carlo method for the self-avoiding walk. *Journal of Statistical Physics*, 50(1-2):109–186, 1988.

[63] M. Marseguerra, E. Zio, L. Podofillini, and D. W. Coit. Optimal design of reliable network systems in presence of uncertainty. *IEEE Transactions on Reliability*, 54(2):243–253, 2005.

[64] V. B. Melas. On the efficiency of the splitting and roulette approach for sensitivity analysis. In *Proceedings of the 29th conference on Winter simulation*, WSC '97, pages 269–274, Washington, DC, USA, 1997. IEEE Computer Society.

[65] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[66] P. Metzner, C. Schutte, and E. Vanden-Eijnden. Illustration of transition path theory on a collection of simple examples. *The Journal of Chemical Physics*, 125(8):84–110, 2006.

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

[68] P. D. Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications.* Springer series in statistics: Probability and its applications. Springer, 2004.

[69] R. Motwani and P. Raghavan. Randomized algorithms. In *The Computer Science and Engineering Handbook*, pages 141–161. 1997.

[70] L. Murray, H. Cancela, and G. Rubino. A splitting algorithm for network reliability estimation. *IIE Transactions*, 45(2):177–189, 2013. 2 2.

[71] C. H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[72] B. T. Polyak and E. N. Gryazina. Randomized methods based on new monte carlo schemes for control and optimization. *Annals OR*, 189(1):343–356, 2011.

[73] L. E. Rasmussen. Approximating the permanent: A simple approach. *Random Struct. Algorithms*, 5(2):349–362, 1994.

[74] L. E. Rasmussen. Approximately counting cliques. *Random Struct. Algorithms*, 11(4):395–411, 1997.

[75] A. K. Ravindra, M. L. Thomas, and O. B. James. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[76] B. Roberts and D. P. Kroese. Estimating the number of s-t paths in a graph. *Journal of Graph Algortihms and Applications*, 11(1):195–214, 2007.

[77] M. N. Rosenbluth and A. W. Rosenbluth. Monte Carlo calculation of the average extension of molecular chains. *Journal of Chemical Physics*, 23:356–359, 1955.

[78] S. M. Ross. *Simulation.* Statistical Modeling and Decision Science Series. Academic Press, 2006.

[79] S. M. Ross. *Introduction to Probability Models, Eighth Edition.* Academic Press, 9 edition, January 2007.

[80] R. Rubinstein, R. Vaisman, and Z. Botev. Hanging edges for fast reliability estimation. Technical report, Technion, Haifa, Israel, 2012.

[81] R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, pages 127–190, 1999.

[82] R. Y. Rubinstein. The Gibbs Cloner for Combinatorial Optimization, Counting and Sampling. *Methodology and Computing in Applied Probability*, pages 491–549, 2009.

[83] 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*, 12:1–50, 2010.

[84] R. Y. Rubinstein. Stochastic enumeration method for counting np-hard problems. *Methodology and Computing in Applied Probability*, pages 1–42, 2012.

[85] R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning (Information Science and Statistics).* Springer, 1 edition, July 2004.

[86] R. Y. Rubinstein and D. P. Kroese. *Simulation and the Monte Carlo Method, Second Edition.* John Wiley and Sons, New York, 2007.

[87] F. J. Samaniego. On closure of the ifr under formation of coherent systems. *IEEE Trans Reliab*, 34:69–72, 1985.

[88] G. A. F. Seber. The effect of trap response on tag recapture estimates. *Biometrics*, pages 13–22, 1970.

[89] D. Siegmund. Importance sampling in the Monte Carlo study of sequential tests. *Annals of Statististics*, 4:673–684, 1976.

[90] R. L. Smith. *Efficient Monte Carlo procedures for generating points uniformly distributed over bounded regions.* 1982.

[91] S. P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31:398–427, 1997.

[92] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, January 1979.

[93] A. W. van der Vaart. *Asymptotic Statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2000.

[94] J. van Rensburg. Monte carlo methods for the self-avoiding walk. *J. Phys. A: Math. Theor.*, 50(42):1–97, 2009.

[95] M. Villén-Altimirano and J. Villén-Altimirano. RESTART: A method for accelerating rare event simulations. In *Proceedings of the 13th International Teletraffic Congress, Performance and Control in ATM*, pages 71–76, June 1991.

[96] M. Villén-Altimirano and J. Villén-Altimirano. RESTART: A straightforward method for fast simulation of rare events. In *Proceedings of the 26th conference on Winter simulation*, WSC '94, pages 282–289, San Diego, CA, USA, 1994. Society for Computer Simulation International.

[97] M. Villén-Altimirano and J. Villén-Altimirano. About the efficiency of RESTART. In *Proceedings of the 1999 RESIM Workshop*, pages 99–128, University of Twente, the Netherlands, 1999.

[98] W. Wei and B. Selman. A new approach to model counting. In *In 8th SAT, volume 3569 of LNCS*, pages 324–339, 2005.

- בדיקת תנאי עצירה: אם תנאי העצירה שנבחר מראש מתקיים עבור לשלב הבא, אחרת תחזור לשלב 2.
- אמידה: אמידה עבור בעיות ספירה.

בעבודה זו, המורכבת ממאמרים, אנו מרחיבים את אלגוריתם הפיצול במספר נקודות.

- אנו מראים שאלגוריתם הפיצול יכול לשמש כאלגוריתם לפתרון בעיות אופטימיזציה בדידות
- אנו מציגים עדויות נומריות ליכולת האלגוריתם להגריל דגימות אחידות בתוך אובייקטים קומבינטורים מסובכים
- אנו מציגים אלגוריתם פיצול חדש עם החלקה אשר יכול לשפר את הביצועים של האלגוריתם המקורי
- אנו מרחיבים את האלגוריתם המקורי ע"י שיטת capture-recapture הידועה בסטטיסטיקה.

בנוסף, אנו מציגים שני אלגוריתמים נוספים שמשמשים גם כן לפתרון בעיות ספירה אך בדרך שונה מזו שהשתמשנו בה באלגוריתם הפיצול. אלגוריתם ה Spectra ואלגוריתם SE – אנומרציה סטובסטית המוצגים בפרקים 6 ו 7 בהתאמה.

במהלך העבודה, אנו מציגים תוצאות נומריות עבור בעיות שונות של ספירה ואופטימיזציה כגון בעיית ספיקות (SAT), אשר תופסת מקום מרכזי בעבודה, בעיית ספירת גרפים בעלי דרגות נתונות ובעיית הBinary Contingency Tables. חשוב לציין, שאף אם קל למצוא פתרון לחלק מבעיות החלטה, קשה למצוא פתרון לבעיית הספירה המתאימה לה.

בעיית ספיקות הינה אחד מהנושאים המרכזיים באופטימיזציה קומבינאטורית. כל בעיה שהיא NP-complete כמו Max-Cut, צביעת גרף ו-TSP ניתנת להצגה בזמן פולינומיאלי כבעיית ספיקות. באופן כללי, בעיית SAT היא הבעיה הבאה: בהינתן נוסחה בתחשיב הפסוקים שמכילה רק קשרים מסוג "גם", "או" ו-"לא", האם קיימת השמה של ערכי אמת למשתנים כך שהנוסחה תקבל ערך אמת? (פה אנו עוסקים בבעיית ספירה הקשורה לבעיית ספיקה שבה אנו מעוניינים לדעת מספר הפתרונות) ניתן להראות שגם אם נגביל את הבעיה לנוסחאות הניתנות בצורה הנורמאלית הקוניונקטיבית (CNF), הבעיה תישאר-NP קשה. לכן, בשל נוחות העבודה עם נוסחאות בצורת CNF, מקובל לנסח את הבעיה רק עבור נוסחאות הנתונות בצורה זו. מקובל להתייחס בשם k-SAT לבעיית ספיקות מנוסחות בצורת CNF אשר כל הפסוקיות המופיעות בהן מאורך k. ניתן לראות כי הבעיות 1,2-SAT ניתנות לפתרון בזמן פולינומי. (כלומר, שייכות ל-P), בעוד שעבור k גדול משתיים הבעיה המתקבלת היא NP קשה. לבעיית ה SAT - חשיבות בתחומים רבים של מדעי המחשב, ובהם אלגוריתמיקה ,בינה מלאכותית ועיצוב חומרה.

בעיית ספירת גרפים עם דרגות נתונות. בעיה זו משכה את תשומת לב החוקרים מכיוון שהיא יכולה למדל בעיות אמתיות כגון בעיות ב World Wide Web , בעיות ברשתות חברתיות וביולוגיות. בעיקרון נתון גרף עם מספר צמתים ידוע ולכל צומת מוגדרת הדרגה שלו. אנו רוצים לספור כמה גרפים כאלה קיימים.

בעיית ה Binary Contingency Tables. בהינתן שני וקטורים $r = \{r_1, \ldots, r_m\}$ ו $c = \{c_1, \ldots, c_n\}$ אנו מגדירים מטריצה של אפסים ואחדים בגודל $m \times n$ כך שסכום האיברים בשורה ה- i יהיה שווה ל r_i וסכום האיברים בעמודה ה-j יהיה שווה ל c_j לכל $i \in \{1, \ldots, m\}$ ו $j \in \{1, \ldots, n\}$. אנו מעוניינים לספור את כמות המטריצות שמקיימות את התכונה הזו.

תקציר

אחד התחומים המאתגרים ביותר במדעי המחשב הוא טיפול בפתרון בעיות NP. בעיות NP הן בעיות שלא קיימת עבורן מכונת טיורינג דטרמיניסטית הפותרת את הבעיה בזמן פולינומיאלי כפונקציה של גודל הבעיה או במילים אחרות, עבור בעיות אלה לא ידוע אם קיים אלגוריתם יעיל הפותר אותם. בעבודה זו אנו מטפלים בבעיות האלה וגם בבעיות הספירה #P הנחשבות אפילו קשות יותר מבעיית ה NP.

למרות שרוב הבעיות שנטפל בהן מוגדרות כבעיות דטרמיניסטיות, אנו נתקוף אותן בעזרת כלים הסתברותיים. על מנת להבין את דרך הפעולה, נתבונן למשל בבעיית אופטימיזציה בדידה כלשהי המוגדרת על וקטור בינארי באורך n. נשים לב שאנו יכולים להגריל אותו למשל מהתפלגות אחידה מעל המרחב $\{0,1\}^n$. כמו כן, ע"י הגרלה חוזרת והצבה לפונקציית המטרה אנו בסוף נגיע לפתרון האופטימאלי של אותה בעיית אופטימיזציה. הבעיה היחידה בגישה זו היא שההגרלה של הפתרון כזה היא מאורע נדיר, כלומר אם למשל קיים מינימום גלובאלי יחיד אז ההסתברות שהגענו אליו בזמן הדגימה מהתפלגות אחידה הוא $\frac{1}{2^n}$. באופן כללי קיימת שקילות מסויימת בין בעיות קשות לבעיית אמידה של מאורעות נדירים במרחב הסתברותי.

ניתן גם להסתכל על בעיות ספירה (#P) כבעיית עמידה. לדוגמא, נתבונן בבעיית אמינות רשת ונניח שיש לנו גרף לא מכוון עם צמתים חשובים שחייבים להיות מקושרים בינם לבין עצמם כל הזמן. כמו כן נניח שהצמתים אמינים אך הקשתות יכולות ליפול. אחד השאלות שניתן לשאול היא מה ההסתברות שהרשת אמינה או במילים אחרות, מה ההסתברות שכל הצמתים החשובים מקושרים. ברור ש בהינתן שכל קשת נופלת בהסתברות מסוימת, ונניח בלי הגבלת הכלליות שהסתברות זו שונה מ 0 או 1 אז קיימים $2^{|E|}$ (כאשר E – קבוצת הקשתות בגרף) ריאליזציות של הגרף ורק בחלקם הצמתים החשובים מקושרים. ניתן כעת לספור את כל הריאליזציות שמגדירים רשת אמינה ולחלק את המספר שהתקבל במספר של כל הריאליזציות האפשריות. התוצאה תיתן לנו את ההסתברות של אמינות הרשת הנתונה. נניח כעת שכל הקשתות מאוד לא אמינות. אם כך, באופן אינטואיטיבי, אמינות הרשת תהיה מאוד נמוכה. ואכן זה מה שקורה באמת ואנו נכנסים שוב לנישה של מאורעות נדירים.

באופן כללי, אפשר להגיד שבעיית אמידה ששקולה לבעיות אופטימיזציה וספירה, הופכת להיות קשה כאשר אנו עוסקים במאורעות נדירים. באופן לא פורמאלי, זה נובע מהתפוצצות השונות של אותו אמד. קיימות מספר שיטות להקטנת השונות ו באמצע שנות התשעים החלו להופיע אלגוריתמים לאמידת התפלגות (או הקטנת שונות). אפשר להגיד שכל האלגוריתמים האלה הם אלגוריתמים איטרטיביים המשתמשים בתיאור הסתברותי של כל אוכלוסיית המבנים או של העלית שלה ליצירת מבנים חדשים, וכך מכליאים בו זמנית מבנים רבים. בכל מקרה, רובם מסתמכים על שתי השיטות הבאות:

- דגימה חשובה (Importance Sampling)
- Markov Chain Monte Carlo - MCMC

דגימה חשובה היא שיטה פרמטרית ו MCMC היא שיטה לא פרמטרית בהתאמה. בעבודה זו אנו נעסוק בעיקר בשיטה השנייה (MCMC) ובפרט בשיטת הפיצול המצליחה המבוססת אליה. ההבדל העיקרי בין שיטת ה Splitting לMCMC הוא שבשיטה זו יש מנגנון מיוחד הנקרא מנגנון "clonning" (שיבוט, פיצול), אשר עושה את האלגוריתם מהיר ומדויק מאוד. נקודה חשובה נוספת היא בכך שאלגוריתם הפיצול מבצע דגימה בכל המרחב ולא רק במרחב קבוע מראש.

הרעיון העיקרי של שיטת MCMC הוא לתכנן סדרת הדגימות כאשר בעיה "קשה" של חישוב עוצמה של הקבוצה מפורקת לבעיות אמידה "קלות" עבור מספר קבוצות הקשורות לקבוצה המקורית. תכנית הפעולה בכל השיטות שנחקרו או פותחו במסגרת העבודה הזאת היא בעצם חישוב אבולוציוני המורכב מהשלבים הבאים:

- אתחול: ההתפלגות ההתחלתית – התפלגות אחידה על פני המרחב הכולל, מספר האיטרציה – 1.
- דגימה: תחולל מבנים המפולגים לפי פונקציית ההתפלגות של האיטרציה הנוכחית, תקבע את רמת האלגוריתם עבור האיטרציה הנוכחית ועל סמך הרמה הזאת תמצא את האוכלוסייה אשר תשפיע על פונקציית ההתפלגות באיטרציה הבאה.
- עדכון פונקצית התפלגות: עדכון פונקציית ההתפלגות.

המחקר נעשה בהנחיית פרופ׳ ראובן רובינשטיין ופרופ׳ אופר שטרייכמן בפקולטה להנדסת תעשייה וניהול.
ברצוני להביע את תודתי העמוקה לפרופ׳ רובינשטיין על הנחייתו המסורה והמקצועית ועל תמיכתו הכספית.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

אני מודה למשפחתי על התמיכה בזמן הלימודים.

עבודה זו מוקדשת לפרופ׳ ראובן רובינשטיין, המורה הטוב ביותר שהיה לי.

שיטות אנומרציה סטוכסטית לבעיות מאורעות נדירים, ספירה ואופטימיזציה

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
דוקטור לפילוסופיה

רדיסלב וייסמן

הוגש לסנט הטכניון – מכון טכנולוגי לישראל

תשרי תשע״ג   חיפה   אוקטובר 2013