Estimating the Number of Vertices in Convex Polytopes

Robert Salomone
School of Mathematics and Physics
The University of Queensland,
Brisbane, Australia
robert.salomone@uqconnect.edu.au

Radislav Vaisman
School of Mathematics and Physics
The University of Queensland,
Brisbane, Australia
r.vaisman@uq.edu.au

Dirk Kroese
School of Mathematics and Physics
The University of Queensland,
Brisbane, Australia
kroese@uq.edu.au

***Abstract*—Estimating the number of vertices of a convex polytope defined by a system of linear inequalities is crucial for bounding the run-time of exact generation methods. It is not easy to achieve a good estimator, since this problem belongs to the #P complexity class. In this paper we present two randomized algorithms for estimating the number of vertices in polytopes. The first is based on the well-known Multilevel Splitting technique. The second, called Stochastic Enumeration, is an improvement of Knuth's backtrack algorithm. Both methods are shown to bring a significant variance reduction, and outperform the current state-of-the-art in test cases.**

***Index Terms*—Multilevel Splitting, Stochastic Enumeration, Rare Events, Vertex Counting, Backtrack Trees**

I. Introduction

Calculating the number of vertices of a bounded convex polyhedron (polytope) is an important computational problem [1], [2], [3], [4], [5], [6]. There exists two basic approaches for handling the vertex generation problem. The first is Motzkin's double description method [6], which exploits a sequential construction of a polytope. For implementations, see the algorithms developed by Seidel [4] and Chazelle [5]. The second approach involves pivoting around the edge skeleton of a polytope. An example of an efficient implementation is the reverse search (RS) algorithm of Avis and Fukuda [1], [3].

It is important to note that pivoting methods, and in particular the RS algorithm, have a runtime that is proportional to the number of vertices of a polytope. Consequently, for instances with a large number of vertices, this algorithm can become computationally impractical. Therefore, prior to using such methods, it is prudent to get an estimate of the number of vertices [2].

Unfortunately, such an estimate is not easy to obtain, since the vertex enumeration problem belongs to the #P complexity class [7]. This complexity class, introduced by Valiant [8], consists of the set of counting problems that are associated with a decision problem in NP (non–deterministic polynomial time). For example, #SAT is the problem of counting the number of feasible solutions to a satisfiability formula (SAT). The #P–complete complexity class is a sub–class of #P consisting of those problems in #P to which any other problem in #P can be reduced via a polynomial reduction. #SAT, for example, is #P–complete. Interestingly, various #P–complete problems are associated with an easy decision problem, i.e., the corresponding decision problem is in P (polynomial time), such as the satisfiability of propositional formulas in disjunctive normal form (DNF), or, existence of a polytope vertex.

For some #P-complete problems there are known efficient approximations. For example, Karp and Lubby [9] introduced a fully polynomial randomized approximation scheme (FPRAS) for counting the solutions of DNF satisfiability formulas. Similar results were obtained for the knapsack and permanent counting problems by Dyer and Jerrum et al. [10], [11]. Unfortunately, there are also many negative results, see Dyer et al. and Vadhan [12], [13].

There are two main approaches to tackle such difficult counting problems. The first one is Markov Chain Monte Carlo (MCMC) and the second is sequential importance sampling (SIS). Both approaches exploit the finding of Jerrum et al. [14] that counting is equivalent to uniform sampling over a suitably restricted set. MCMC methods sample from such restricted regions by constructing an ergodic Markov chain with limiting distribution equal to the desired uniform distribution. A number of MCMC approaches with good empirical performance have been proposed; see, for example, [15], [16], [17], [18]. There are also many examples of successful SIS implementations on various counting problems; see, for example, [9], [19], [20], [21]. More recent advances and background material can be found in [22].

To the best of our knowledge, the problem of estimating the number of vertices of a convex polytope does not have a FPRAS. That is, for this particular problem, an efficient randomized approximation algorithm is not known to exist. Moreover, the sole (to our knowledge) method currently available is that of Avis & Devroye [2], which is based on the backtrack tree size estimator of Knuth [23], implemented within RS. This method, while theoretically unbiased, is flawed in the sense that in practice it has a tendency to vastly underestimate the number of vertices. We address the cause of this, proposing two new estimation methods that outperform the current state of the art.

The first method is Multilevel Splitting. This powerful idea was first used by Kahn and Harris [24] to estimate rare-event probabilities. The main idea is to partition the state space

in such a way that the problem becomes one of estimating conditional probabilities that are not rare. The Generalized Splitting Method (GS) by Botev & Kroese [15], generalizes this to a method able to evaluate wide range of rare-event estimation problems. For a survey of the general methodology and specific examples for counting via splitting, we refer to [21, Chapter 4] and [25], [26].

The second method under consideration is the novel Stochastic Enumeration (SE) algorithm [27] which was originally proposed in [23] for backtrack tree estimation. The SE algorithm belongs to the SIS family of estimators. The main difference between general SIS procedures and SE is that the latter employs polynomial oracles during the execution and runs multiple trajectories (samples) in parallel, instead of repeatedly running single trajectories. The SE algorithm has a budget parameter that limits the number of parallel samples. It was shown in [27] that SE provides powerful variance reduction. For example, it was established that SE is an almost sure FPRAS for estimating the size of certain random trees. See [27] for details.

In this paper we show that both of these methods produce much more reliable results than the method of Avis & Devroye [2]. Despite the superior performance demonstrated by the proposed methods, neither SE nor GS is uniformly "best" as each has its own pros and cons. It will be shown in Section V (numerical results) that SE is generally faster, while GS is more robust to underestimation.

The rest the paper is organized as follows. In Section II we give a brief introduction to the vertex counting problem and show that a good approximation is important. Moreover, we show how the problem can be formulated using a probabilistic setting and give a brief introduction to rare-event estimation. In Sections III and IV we give a general description of the GS and the SE methods respectively and provide an explanation on how both algorithms can be used to deliver reliable estimators for the number of vertices in convex polytopes. Finally, in Section V we present various examples that demonstrates that the same computation effort can lead to much better estimates than currently available.

II. Convex Polytope Vertex Counting Problem

Polyhedral combinatorics is a rich and diverse field that uses techniques from many areas including algebra and topology. We provide a very brief introduction to the theory of convex polytopes by starting with some basic definitions.

Definition 2.1 (Convex polyhedron): A convex polyhedron $\mathcal{P}$ is defined as the solution space of a system of linear inequalities; that is,

$$\mathcal{P} = \left\{ \mathbf{u} \in \mathbb{R}^d \;:\; A\mathbf{u} \leqslant \mathbf{b} \right\},$$

where A is an $n \times d$ matrix, ($n \geqslant d$), and $\mathbf{b}$ is an n-vector. A bounded polyhedron is called a polytope. □

Definition 2.2 (Vertices, bases, and simple polytopes):

1) A point $\overline{\mathbf{u}}$ is called a vertex if there is some $d \times d$ submatrix A' of A such that $\overline{\mathbf{u}}$ is the unique solution of
$$A'\overline{\mathbf{u}} = \mathbf{b}', \quad \overline{\mathbf{u}} \in \mathcal{P},$$
where $\mathbf{b}'$ is the corresponding sub-vector of $\mathbf{b}$.
2) The matrix A' is called a basis for $\overline{\mathbf{u}}$.
3) The vertex $\overline{\mathbf{u}}$ is called degenerate if it can be represented with more than one basis.
4) A polytope with no degenerate vertices is called simple. □

Note that we are interested in approximating the number of vertices, since this quantity is crucial for bounding the running time of the RS algorithm; see [2]. Before one tries to develop an approximation, the natural question of lower and upper bounds is clearly of interest. From [28], we learn the following.

Theorem 2.1 (Upper and lower bounds to the number of vertices): Given a polytope $\mathcal{P}$ (Definition 2.1), the following holds.

1) McMullen upper bound theorem; the number of vertices in polytope $\mathcal{P}$ is at most
$$f(n,d) = \binom{n - \lfloor (d+1)/2 \rfloor}{n-d} + \binom{n - \lfloor (d+2)/2 \rfloor}{n-d}.$$
2) Barnette lower bound theorem; the number of vertices in polytope $\mathcal{P}$ is at least
$$g(n,d) = n(d-1) - (d+1)(d-2).$$

□

Unfortunately, Theorem 2.1 is of little use in determining the running time of the RS method, since the gap between the bounds is generally too large. To see this, consider hypercube in dimension $d = 40$. The exact number of vertices is equal to $2^{40} \approx 1.09 \times 10^{12}$, but the lower and the upper bounds are 1562 and 5.59×10^{15}, respectively.

The above example emphasizes the need for a good approximation algorithm for vertex counting. In this paper we concentrate on solving this problem using randomized algorithms and, in particular, the Monte Carlo technique. We next describe a major problem one can encounter while applying a Monte Carlo algorithm, namely the rare-event setting.

A. Monte Carlo Under a Rare-Event Setting

Consider a set $\mathcal{X}$, a set $\mathcal{X}^* \subseteq \mathcal{X}$ and suppose that the probability
$$\ell = \frac{|\mathcal{X}^*|}{|\mathcal{X}|},$$
is to be estimated. We further suppose that one can sample uniformly at random from $\mathcal{X}$. The Crude Monte Carlo (CMC) procedure for the estimation of ℓ is defined as follows. Sample N independent and uniformly distributed samples, $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from $\mathcal{X}$ and, let Z_i (for $i = 1, \ldots, N$) be (independent) Bernoulli random variable such that
$$Z_i = \mathbb{1}_{\{\mathbf{X}_i \in \mathcal{X}^*\}} = \begin{cases} 1 & \text{if } \mathbf{X}_i \in \mathcal{X}^* \\ 0 & \text{otherwise,} \end{cases}$$

where $\mathbb{1}$ is indicator random variable. The CMC estimator $\widehat{\ell}_{\mathrm{CMC}}$ is given by:

$$\widehat{\ell}_{\mathrm{CMC}} = \frac{1}{N}\sum_{i=1}^{N} Z_i. \tag{1}$$

It is not very hard to see that $\mathbb{E}(Z_i) = \ell$ and $\mathrm{Var}(Z_i) = \sigma^2 = \ell(1-\ell)$ for all $i = 1, \ldots, N$. Consequentially, it holds that the estimator is unbiased; that is,

$$\mathbb{E}\left(\widehat{\ell}_{\mathrm{CMC}}\right) = \mathbb{E}\left(\frac{1}{N}\sum_{i=1}^{N} Z_i\right) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}(Z_i) = \ell. \tag{2}$$

Moreover, the variance of the estimator is given by

$$\begin{aligned}\mathrm{Var}\left(\widehat{\ell}_{\mathrm{CMC}}\right) &= \mathrm{Var}\left(\frac{1}{N}\sum_{i=1}^{N} Z_i\right) = \frac{1}{N^2}\sum_{i=1}^{N}\mathrm{Var}(Z_i) \\ &= \frac{\ell(1-\ell)}{N}.\end{aligned} \tag{3}$$

With (2) and (3), we can measure the accuracy of the CMC estimator. Note that according to central limit theorem, $\widehat{\ell}_{\mathrm{CMC}}$ is approximately $\mathsf{N}\left(\ell, \sigma^2/N\right)$ distributed, where N stands for normal distribution. Let z_γ be the γ quantile of the $\mathsf{N}(0,1)$ distribution; that is, $\Phi(z_\gamma) = \gamma$, where Φ denotes the standard normal cdf. Then,

$$\mathbb{P}\left(\ell \in \widehat{\ell}_{\mathrm{CMC}} \pm z_{1-\gamma/2}\frac{\sigma}{\sqrt{N}}\right) \approx 1-\gamma, \tag{4}$$

holds, and $\widehat{\ell}_{\mathrm{CMC}} \pm z_{1-\gamma/2}\frac{\sigma}{\sqrt{N}}$ defines a confidence interval for the point estimate $\widehat{\ell}_{\mathrm{CMC}}$. The width of this interval is given by

$$w_a = 2z_{1-\gamma/2}\frac{\sigma}{\sqrt{N}},$$

however, when dealing with very small values, say $\ell = 10^{-11}$, an absolute width of this interval, for example, a reasonable 5% ($w_a = 0.05$) is meaningless and, one should instead consider a relative width defined by

$$w_r = \frac{w_a}{\widehat{\ell}_{\mathrm{CMC}}}.$$

Now, divide (4) by $\widehat{\ell}_{\mathrm{CMC}}$ and arrive at

$$\mathbb{P}\left(\ell \in \widehat{\ell}_{\mathrm{CMC}}\left(1 \pm z_{1-\gamma/2}\frac{\sigma}{\widehat{\ell}_{\mathrm{CMC}}\sqrt{N}}\right)\right) \approx 1-\gamma,$$

a much better confidence bound for rare-event estimation problems. This brings us to the standard measure of rare-event estimator efficiency called relative error (RE), [29]. The RE of $\widehat{\ell}_{\mathrm{CMC}}$ is defined by

$$\mathrm{RE}\left(\widehat{\ell}_{\mathrm{CMC}}\right) = \frac{\sigma}{\ell\sqrt{N}} = \frac{\sqrt{\mathrm{Var}\left(\widehat{\ell}_{\mathrm{CMC}}\right)}}{\mathbb{E}\left(\widehat{\ell}_{\mathrm{CMC}}\right)} \underbrace{=}_{(2),(3)} \frac{\sqrt{\frac{\ell(1-\ell)}{N}}}{\ell}.$$

Proceeding with the rare-event setting, for which it holds that $\ell \ll 1$, we see that

$$\mathrm{RE}\left(\widehat{\ell}_{\mathrm{CMC}}\right) = \frac{\sqrt{\frac{\ell(1-\gamma)}{N}}}{\ell} \approx \frac{1}{\sqrt{N\ell}}. \tag{5}$$

The above equation imposes a serious challenge, as illustrated in the following example.

Example 2.1 (Sample size under rare-event setting): Let $\ell = 10^{-11}$, and suppose that we are interested in a modest 10% RE. It will not be very hard to verify from (5), that the required number of experiments N is about 10^{13}. Such N is unmanageable in the sense of computation effort, and one needs to resort to variance minimization techniques. □

B. CMC Algorithm For the Vertex Counting Problem

For the rest of the paper we assume that $\mathcal{P}$ is non-empty and non-degenerate, A has full column rank, and that there are no redundant constraints in the corresponding linear system.

We begin by putting the vertex counting problem into the CMC framework from Section II-A. Recall Definition 2.2, and consider the following random experiment. Given a system of linear inequalities $A\mathbf{u} \leqslant \mathbf{b}$, were A is a $n \times d$ matrix and $\mathbf{b}$ is n-vector, choose a sub-matrix A' of size $d \times d$ and the corresponding d-sub-vector $\mathbf{b}'$, uniformly at random. With these A' and $\mathbf{b}'$, check if A' is a vertex.

More formally, let $\mathcal{X}$ be the set of d-dimensional vectors of indices, taken without replacement from $\{1, \ldots, n\}$, that is, $\mathcal{X}$ is defined by:

$$\{\mathbf{x} = (i_1, \ldots, i_d) \;:\; 1 \leqslant i_1 < i_2 < \cdots < i_d \leqslant n\}.$$

Note that each $\mathbf{x} \in \mathcal{X}$ defines a sub-matrix $A' = A(\mathbf{x})$, the corresponding sub-vector $\mathbf{b}' = \mathbf{b}(\mathbf{x})$, and thus, the linear system:

$$\underbrace{\begin{pmatrix} a_{i_1,1} & a_{i_1,2} & \cdots & a_{i_1,d} \\ a_{i_2,1} & a_{i_2,2} & \cdots & a_{i_2,d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_d,1} & a_{i_d,2} & \cdots & a_{i_d,d} \end{pmatrix}}_{A(\mathbf{x})} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix} = \underbrace{\begin{pmatrix} b_{i_1} \\ b_{i_2} \\ \vdots \\ b_{i_d} \end{pmatrix}}_{\mathbf{b}(\mathbf{x})}. \tag{6}$$

Let $\overline{\mathbf{u}}_{\mathbf{x}}$ be the solution of the linear system (6) and define $\mathsf{U}(\mathcal{X})$ to be the uniform distribution on $\mathcal{X}$.

Remark 2.1 (Random object generation): The generation of $\mathbf{X} \sim \mathsf{U}(\mathcal{X})$ is straightforward. For example, it can be achieved by generating a random permutation of $\{1, \ldots, n\}$ and choosing the first d indices from the latter; see for example, [29, Chapter 2]. □

Let $\mathcal{X}^* \subseteq \mathcal{X}$ be the set of indices that determine feasible bases, that is:

$$\mathcal{X}^* = \{\mathbf{x} \in \mathcal{X} \;:\; A\overline{\mathbf{u}}_{\mathbf{x}} \leqslant \mathbf{b}\}.$$

Next, define the event $\{\mathbf{X} \in \mathcal{X}^*\}$, where $\mathbf{X} \sim \mathsf{U}(\mathcal{X})$. Having in mind that $|\mathcal{X}| = \binom{n}{d}$, since there are $\binom{n}{d}$ possibilities to choose a $d \times d$ sub-matrix from A, the cardinality of $\mathcal{X}^*$ is given by:

$$|\mathcal{X}^*| = |\mathcal{X}|\,\mathbb{P}\left(\mathbf{X} \in \mathcal{X}^*\right) = \binom{n}{d}\mathbb{P}\left(\mathbf{X} \in \mathcal{X}^*\right).$$

The above equation immediately implies the usage of CMC framework from the previous section, by setting the Bernoulli random variable Z to be $Z = \mathbb{1}_{\{\mathbf{X}\in\mathcal{X}^*\}}$ and applying (1) for the estimation of $\mathbb{P}(\mathbf{X} \in \mathcal{X}^*)$.

Unfortunately, this simple idea will generally fail because of the rare events involved. To see this, recall the hypercube ($n = 80$, $d = 40$) example from Section II. In this case the exact value of $\mathbb{P}(\mathbf{X} \in \mathcal{X}^*)$ is equal to $2^{40}/\binom{80}{40} \approx 1.02 \times 10^{-11}$ and we fall into the rare-event trap; see Example 2.1.

To overcome this problem, we describe two randomized algorithms that are capable of handling the rare-event problem discussed above. Both methods are unbiased like the CMC estimator but their variance is significantly much smaller.

III. Generalized Splitting

Note that the main problem with the CMC estimator from the previous section was the high variance. Here we propose to adopt a quite general variance minimization technique for the estimation of rare-event probability $\ell = \mathbb{P}(\mathbf{X} \in \mathcal{X}^*)$ — the GS method, [15].

We already saw that sampling in $\mathcal{X}^*$ is difficult. The main idea of GS is to design a sequential sampling plan, with a view to decompose a "difficult" problem (sampling on $\mathcal{X}^*$), into a number of "easy" ones associated with a sequence of subsets in the sampling space $\mathcal{X}$. In particular, we are interested in estimating the rare-event probability $\ell = |\mathcal{X}^*|/|\mathcal{X}|$, where $\mathcal{X}^* \subseteq \mathcal{X}$ and $|\mathcal{X}|$ is known in advance. We further assume without loss of generality, that generating random variables from the uniform distribution on $\mathcal{X}$ is easy. A very general splitting framework can be summarized as follows, [30], [21].

1) Find a sequence of sets $\mathcal{X} = \mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_T = \mathcal{X}^*$. Assume that the subsets $\mathcal{X}_t$ can be written as level sets of some performance function $S : \mathcal{X} \to \mathbb{R}$ for levels $-\infty = \gamma_0 \leqslant \gamma_1 \cdots \leqslant \cdots \leqslant \gamma_T = \gamma$, that is
$$\mathcal{X}_t = \{\mathbf{x} \in \mathcal{X} \,:\, S(\mathbf{x}) \geqslant \gamma_t\}, \quad t = 0, \ldots, T.$$
Then, the quantity of interest ℓ is given by:
$$\begin{aligned}\ell &= \frac{|\mathcal{X}^*|}{|\mathcal{X}|} = \prod_{t=1}^{T} \frac{|\mathcal{X}_t|}{|\mathcal{X}_{t-1}|} \\ &= \prod_{t=1}^{T} \mathbb{P}(S(\mathbf{X}) \geqslant \gamma_t \mid S(\mathbf{X}) \geqslant \gamma_{t-1}) \\ &= \mathbb{P}(S(\mathbf{X}) \geqslant \gamma_T).\end{aligned}$$
2) For each $t = 1, \ldots, T$, develop an efficient estimator $\widehat{c}_t$ for the conditional probabilities
$$c_t = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma_t \mid S(\mathbf{X}) \geqslant \gamma_{t-1}).$$
To avoid rare-event problems at the intermediate levels (γ_t), we assume that the sets $\mathcal{X}_t$, $t = 1, \ldots, T$, are specifically designed such that the $\{c_t\}$ are not rare-event probabilities.
3) Deliver
$$\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t,$$
as an estimator of ℓ.

With this general framework in hand, we are ready to state the main GS algorithm [30].

Algorithm 3.1 (Generalized Splitting Algorithm for Estimating ℓ):

Input: Given a sequence $\gamma_1, \ldots, \gamma_T$, a performance function $S : \mathcal{X} \to \mathbb{R}$, and a sample size N, execute the following steps.

1) **(Initialization)**. Set $t = 1$, generate N independent samples $\mathcal{W}_0 = \{\mathbf{X}_1, \ldots, \mathbf{X}_N\}$ uniformly from $\mathsf{U}(\mathcal{X})$. Let $\mathcal{W}_1$ be the subset of elements $\mathbf{X}$ in $\mathcal{W}_0$ for which $S(\mathbf{X}) \geqslant \gamma_1$, and let N_1 be the size of $\mathcal{W}_1$. If $N_1 = 0$, go to Step 5.
2) **(Estimation)**. Set:
$$\widehat{c}_t = \frac{N_t}{N}.$$
3) **(Markov chain sampling)**. For each $\mathbf{X}_i$ in $\mathcal{W}_t = \{\mathbf{X}_1, \ldots, \mathbf{X}_{N_t}\}$, sample independently:
$$\mathbf{Y}_{i,j} \sim \kappa_t(\mathbf{y} \mid \mathbf{Y}_{i,j-1}), \quad \mathbf{Y}_{i,0} = \mathbf{X}_i, \quad j = 1, \ldots, S_{ti},$$
where S_{ti} is the splitting factor
$$S_{ti} = \left\lfloor \frac{N}{N_t} \right\rfloor + K_i, \quad K_i \sim \mathsf{Ber}(0.5), \text{ such that,}$$
$$\sum_{j=1}^{N_t} K_j = N \bmod N_t.$$
Here $\kappa_t(\mathbf{y} \mid \mathbf{Y}_{i,j-1})$ is a Markov transition density whose stationary distribution is the uniform distribution on $\mathcal{X}_t$. Reset
$$\mathcal{W}_t = \{\mathbf{Y}_{1,1}, \ldots, \mathbf{Y}_{1,S_{t1}}, \cdots, \mathbf{Y}_{N_t,1}, \ldots, \mathbf{Y}_{N_t,S_{tN_t}}\},$$
where $\mathcal{W}_t$ contains N elements.
4) **(Updating)**. Let $\mathcal{W}_{t+1}$ be the subset of elements $\mathbf{X}$ in $\mathcal{W}_t$ for which $S(\mathbf{X}) \geqslant \gamma_{t+1}$, and let N_{t+1} be the size of $\mathcal{W}_{t+1}$. Increase the level counter: $t = t + 1$.
5) **(Stopping condition)**. If $t = T$ go to Step 6. If $N_t = 0$, set $N_{t+1} = N_{t+1} = \cdots = N_T = 0$ and go to Step 6; otherwise, repeat from Step 2.
6) **(Stopping condition)** Deliver the unbiased estimate of the rare-event probability,
$$\widehat{\ell} = \prod_{t=1}^{T} \widehat{c}_t.$$

□

Remark 3.1 (Sampling from stationary distribution): The crucial step of Algorithm 3.1 is to (approximately) sample from the uniform distribution on each $\mathcal{X}_t$ using MCMC [15], [30] in Step 3. The specific choice of transition density is given in Algorithm 3.2. □

For detailed analysis of the GS method, and in particular for the proof of unbiasedness, see [22], [30]. Next, we continue with GS's implementation details for the convex polytope vertex counting problem.

A. GS Algorithm For Vertex Counting

To deliver an estimator of ℓ and use the GS Algorithm 3.1, we have to resolve the following tasks.

- **Task 1.** Define a sequence of level sets $\mathcal{X} = \mathcal{X}_0 \supset \mathcal{X}_1 \supset \cdots \supset \mathcal{X}_T = \mathcal{X}^*$. Recall that these sets are are defined by the performance function $S : \mathcal{X} \to \mathbb{R}$, and the corresponding performance levels γ_t, $t = 0, \ldots, T$.
- **Task 2.** Develop an MCMC scheme for sampling from uniform distribution on $\mathcal{X}_t$.

The first task is resolved by adopting the CMC algorithm setting from Section II-A. Similar, we define

$$\mathcal{X} = \mathcal{X}_0 = \{\mathbf{x} = (i_1, \ldots, i_d) \ : \ 1 \leqslant i_1 < i_2 < \cdots < i_d \leqslant n\},$$

where $\mathbf{x}$ is a d dimensional vector of indices taken without replacement from $\{1, \ldots, n\}$. Furthermore, we set the nominal distribution to be the uniform distribution on $\mathcal{X}$, and, define

$$\mathcal{X}^* = \{\mathbf{x} \in \mathcal{X} \ : \ A\overline{\mathbf{u}}_{\mathbf{x}} \leqslant \mathbf{b}\},$$

to be the set of feasible bases. With these definitions, let the performance function for each $\mathbf{x} \in \mathcal{X}$ to be the number of satisfied constraints of the linear system $A\mathbf{u} \leqslant \mathbf{b}$, that is:

$$S(\mathbf{x}) = \sum_{i=1}^{n} \mathbb{1}_{\{A(i)\overline{\mathbf{u}}_{\mathbf{x}} \leq b(i)\}}, \tag{7}$$

where $A(i)$ is the ith row of A, $\overline{\mathbf{u}}_{\mathbf{x}}$ is the solution of the linear system (6), and $b(i)$ is the ith component of $\mathbf{b}$.

Noting that (7) defines intermediate sets

$$\mathcal{X}_t = \{\mathbf{x} \in \mathcal{X} \ : \ S(\mathbf{x}) \geqslant \gamma_t\}, \quad t = d+1, \ldots, n, \quad \gamma_t = t,$$

we complete the requirements of the first task.

To complete the second task, all we need to do is to define the MCMC mechanism for approximate sampling from $\mathsf{U}(\mathcal{X}_t)$. This will be accomplished by Gibbs sampling, which is summarized bellow.

*Algorithm 3.2 (**Gibbs Sampler for Vertex Counting**):*
Input: Given a linear system defined by $n \times d$ matrix A, a n-dimensional vector $\mathbf{b}$, and a sample $\mathbf{X} \in \mathcal{X}_t$ which is distributed approximately uniformly on the $\mathcal{X}_t$ set, execute the following steps.

1) **(Initialization)**. Set $j = 1$.
2) **(Sampling)**. Let $\mathbf{X} = (I_1, \ldots, I_d)$. Replace the I_jth component by a (uniform) random index from $\{1, \ldots, n\} \setminus \{I_1, \ldots, I_d\}$. If $S(\mathbf{X}) \geqslant \gamma_t$, go to Step 3, otherwise, repeat the current step.
3) **(Stopping condition)**. If $j < d$, set $j = j + 1$ and go to Step 2. Otherwise, output $\mathbf{X}$ — a sample distributed approximately uniformly on $\mathcal{X}_t$. □

IV. Stochastic Enumeration

In this section we consider a different type of randomized algorithm. The proposed estimator is inspired by the work of Avis & Devroye [2] and is based on an improvement of the well known Knuth's algorithm for tree cost estimation [23].

Note that for a given $\mathcal{P}$, the RS method sequentially constructs a rooted spanning tree T_v, (where v is the tree's root), on the set of all vertices of $\mathcal{P}$ [3]. See for example, the spanning tree for the 3D cube in Figure 1. Specifically, given the matrix A and a vector $\mathbf{b}$, the RS algorithm provides the tree root v and is able to sequentially compute for each node $w \in T_v$ the set od successors of w.

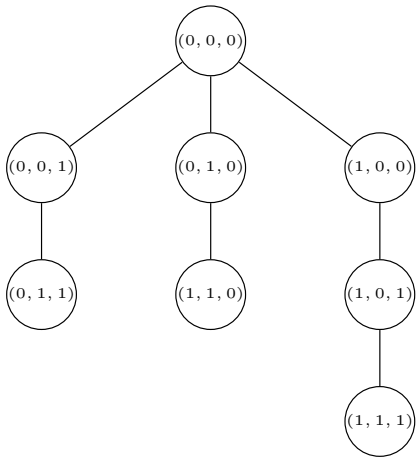

Fig. 1. Spanning tree of the 3D cube.

Note that it is therefore possible to perform a random walk from the root of T_v to one of its leaves without calculating the entire tree. This crucial property of the RS method was exploited in [2]. The authors showed that the combination of RS and Knuth's estimator [23], (which requires only the ability to perform random walks on a tree), yields a straightforward approximation to the number of vertices of a convex polytope.

The main problem with Knuth's algorithm is that it introduces high variance, see [2], [31], [32]. Avis et al. [2] proposed several improvements to the basic Knuth's estimator. Here, we apply a different tree counting technique — the SE method. While SE is a generalization of Knuth's estimator, it is also equipped with powerful variance reduction mechanisms. We show that this algorithm can be very beneficial in the sense of SE's estimator accuracy. Next, we provide a brief overview of the SE method for counting trees.

A. SE Algorithm Description

The SE algorithm is an extension of the one introduced in [21], [23]. Our setting is as follows. Consider a rooted tree $T = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$ (so that $|\mathcal{E}| = |\mathcal{V}| - 1$). We denote the root of the tree by v_0, and for any $v \in \mathcal{V}$ the subtree rooted at v is denoted by T_v. With each node v is associated a non-negative cost $c(v)$. The main quantity of interest is the total cost of the tree,

$$\mathrm{Cost}(T) = \sum_{v \in \mathcal{V}} c(v)$$

or, more generally, the total cost of a subtree T_v — denoted by $\mathrm{Cost}(T_v)$. For each node v we denote the set of successors of v by $S(v)$.

Definition 4.1 (Hyper nodes and forests): Let $\{v_1, \ldots, v_r\} \in \mathcal{V}$ be tree nodes.

- We call a collection of distinct nodes in the same level of the tree $\mathbf{v} = \{v_1, \ldots, v_r\}$ a *hyper node* of cardinality $|\mathbf{v}| = r$.
- Let $\mathbf{v}$ be a hyper node. Generalizing the tree node cost, we define the cost of the hyper node as
$$c(\mathbf{v}) = \sum_{v \in \mathbf{v}} c(v).$$
- Let $\mathbf{v}$ be a hyper node. Define the set of successors of $\mathbf{v}$ as
$$S(\mathbf{v}) = \bigcup_{v \in \mathbf{v}} S(v).$$
- Let $\mathbf{v}$ be a hyper node and let $B \in \mathbb{N}$, $B \geqslant 1$. Define
$$H(\mathbf{v}) = \begin{cases} \{S(\mathbf{v})\} & \text{if} \quad |S(\mathbf{v})| \leqslant B \\ \{\mathbf{w} \mid \mathbf{w} \subseteq S(\mathbf{v}),\ |\mathbf{w}| = B\} & \text{if} \quad |S(\mathbf{v})| > B, \end{cases}$$
to be the set of all possible hyper nodes having cardinality $\max\{B, |S(\mathbf{v})|\}$ that can be formed from the set of $\mathbf{v}$'s successors. Note that if $|S(\mathbf{v})| \leqslant B$, we get a single hyper node with cardinality $|S(\mathbf{v})|$.
- For each hyper node $\mathbf{v}$ let
$$T_{\mathbf{v}} = \bigcup_{v \in \mathbf{v}} T_v$$
be the forest of trees rooted at $\mathbf{v}$. See Figure 2 for an example of hyper node $\mathbf{v} = \{v_1, v_2, v_3, v_4\}$ and its corresponding forest $T_{\mathbf{v}} = \{T_{v_1}, T_{v_2}, T_{v_3}, T_{v_4}\}$.
- For each forest rooted at hyper node $\mathbf{v}$, define its total cost as
$$\mathrm{Cost}\left(T_{\mathbf{v}}\right) = \sum_{v \in \mathbf{v}} \mathrm{Cost}(T_v).$$

□

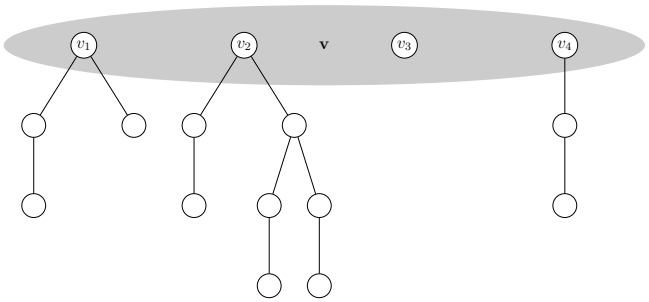

Fig. 2. Hyper node $\mathbf{v}$ that contains regular tree nodes v_1, v_2, v_3 and v_4 with their corresponding subtrees.

With these definitions, we are ready to state the main SE algorithm. The following Algorithm IV-A is a randomized algorithm that outputs a random variable C_{SE}, such that:

$$\mathbb{E}\left(C_{\mathrm{SE}}\right) = \mathrm{Cost}(T).$$

Algorithm 4.1 (Stochastic Enumeration Algorithm): Input: Given a hyper node $\mathbf{v}$ — the forest root, a mechanism that is able to generate a set of successors of any tree node, and a budget $B \geqslant 1$, execute the following steps.

1) **(Initialization)**. Set $k \leftarrow 0$, $D \leftarrow 1$, $\mathbf{X}_0 = \mathbf{v}$ and $C_{\mathrm{SE}} \leftarrow c(\mathbf{X}_0)/|\mathbf{X}_0|$.
2) **(Compute the successors)**. Let $S(\mathbf{X}_k)$ be the set of successors of $\mathbf{X}_k$.
3) **(Terminal position?)**. If $|S(\mathbf{X}_k)| = 0$, the algorithm stops, outputting $|\mathbf{v}|\, C_{\mathrm{SE}}$ as an estimator of $\mathrm{Cost}(T_{\mathbf{v}})$.
4) **(Advance)**. Choose hyper node $\mathbf{X}_{k+1} \in H(\mathbf{X}_k)$ at random, each choice being equally likely. (Thus, each choice occurs with probability $1/|H(\mathbf{X}_k)|$.) Set $D_k = \frac{|S(\mathbf{X}_k)|}{|\mathbf{X}_k|}$ and $D \leftarrow D_k D$, then set $C_{\mathrm{SE}} \leftarrow C_{\mathrm{SE}} + \left(\frac{c(\mathbf{X}_{k+1})}{|\mathbf{X}_{k+1}|}\right) D$. Increase k by 1 and return to Step 2. □

For a detailed analysis of Algorithm IV-A, and in particular for the proof of unbiasedness, see [27].

Remark 4.1 (SE for estimating the number of vertices): The adaptation of Algorithm IV-A to vertex counting estimation problem is straightforward. Define $c(w) = 1$ for any $w \in T_v$. In Step 1, set $\mathbf{v} = \{v\}$, where v is the root of the spanning tree T_v delivered by RS. To calculate $S(\mathbf{X}_k)$ in the second step of the SE algorithm, use RS to get the successors set for each tree node $w \in \mathbf{X}_k$. Note that SE can be used even in degenerate case since it is basically an extension of Avis's algorithm. □

Next, we concentrate on the crucial property of the SE algorithm — the built–in variance reduction mechanism. To do so, we give an intuitive example in Section IV-B and demonstrate numerically in Section V that SE can deliver a much more accurate estimator for the vertex counting problem than the one proposed by Avis & Devroye, [2]. For a detailed discussion about the efficiency analysis of the SE method, see [21] and [27].

B. SE's Variance Reduction Mechanism

Due to the parallel execution of random walks on a tree, the SE algorithm can bring enormous variance reduction [27]. To illustrate this, consider the "hair brush" tree T in Figure 3 and suppose that the cost of all vertices is zero except for v_{n+1}, which has a cost of unity. Our goal is to estimate the cost of this tree, which obviously satisfies:

$$\mathrm{Cost}(T) = 1.$$

It will become clear from the following discussion that the budget parameter B is controlling the variance reduction capability of the SE Algorithm IV-A. We will consider two cases. In particular we examine the behavior of the SE Algorithm IV-A with budgets $B = 1$ and $B = 2$ respectively.

- If we set $B = 1$, the SE Algorithm IV-A essentially adopts the behaviour of Knuth's estimator, [23]. Note that

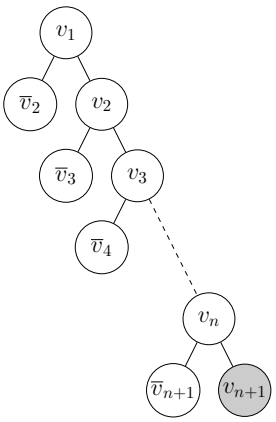

Fig. 3. The hair brush tree.

in this case the algorithm reaches the vertex of interest, v_{n+1}, with probability $1/2^n$ and with $D = 2^n$. In all other cases, the algorithm terminates with some D' and a zero cost node $\bar{v}_i$, $i = 2, \ldots, n+1$. It follows that the expectation and variance of the corresponding SE estimator are

$$\mathbb{E}\left(C_{\mathrm{SE}}\right) = \frac{1}{2^n} \cdot 2^n \cdot 1 + \frac{2^n - 1}{2^n} \cdot D' \cdot 0 = 1,$$

and

$$\mathbb{E}\left(C_{\mathrm{SE}}^2\right) = \frac{1}{2^n} \cdot \left(2^n \cdot 1\right)^2 + \frac{2^n - 1}{2^n} \cdot \left(D' \cdot 0\right)^2 = 2^n \;\Rightarrow$$
$$\Rightarrow\; \mathrm{Var}\left(C_{\mathrm{SE}}\right) = \mathbb{E}\left(C_{\mathrm{SE}}^2\right) - \mathbb{E}\left(C_{\mathrm{SE}}\right)^2 = 2^n - 1.$$

- On the other hand, setting $B = 2$ will force Algorithm IV-A to reach v_{n+1} with probability 1. Note that with this budget and for $i = 2, \ldots, n$, one algorithm trajectory, (random walk from the tree root), is always disappearing at $\bar{v}_i$ vertices from the left, but the second one is always split in two new trajectories at the corresponding v_i nodes from the right. Following the execution steps of the SE Algorithm IV-A one can verify that at the final iteration the cost of the hyper node $\mathbf{X}_{n+1} = \{\bar{v}_{n+1},\ v_{n+1}\}$ is $0 + 1 = 1$, so

$$\left(\frac{c(\mathbf{X}_{n+1})}{|\mathbf{X}_{n+1}|}\right) = \frac{1}{2}.$$

In addition, the final value of D is

$$D = 2 \cdot \underbrace{1 \cdots 1}_{n-1 \text{ times}} = 2.$$

It follows that the expectation and variance of the corresponding SE estimator are

$$\mathbb{E}\left(C_{\mathrm{SE}}\right) = 1 \cdot 2 \cdot \frac{1}{2} = 1,$$

and

$$\mathbb{E}\left(C_{\mathrm{SE}}^2\right) = 1 \cdot \left(2 \cdot \frac{1}{2}\right)^2 = 1 \;\Rightarrow$$
$$\Rightarrow\; \mathrm{Var}\left(C_{\mathrm{SE}}\right) = \mathbb{E}\left(C_{\mathrm{SE}}^2\right) - \mathbb{E}\left(C_{\mathrm{SE}}\right)^2 1 - 1 = 0.$$

By increasing the budget B from 1 to 2 we managed to achieve remarkable variance reduction, from $2^n - 1$ to zero.

The above example is the key for understanding the SE variance reduction mechanism. Note that to deliver some meaningful estimator for the tree cost one needs (at least) to reach the v_{n+1} vertex. In our example, when $B = 1$, this happens with probability $1/2^n$, so we work under the rare-event setting.

For the sake of simplicity, let us concentrate on the simple problem of reaching the vertex v_{n+1}. Suppose that we launch B random walks from the root (v_1). Each walk chooses v_2 or $\bar{v}_2$ with probability $1/2$ respectively; that is, the walks are divided into "good" (those that reach v_2) and "bad" (those that reach $\bar{v}_2$). Now, we split the "good" walks such that there will be B of them and continue the process. Note that the following holds.

- The probability of a single walk ($B = 1$) to reach the v_{n+1} vertex is $1/2^n$.
- A careful choice of B (which can be a polynomial in tree height n), will allow us to reach the vertex of interest — v_{n+1} with reasonably high probability. In particular, note that

$$\mathbb{P}(\text{The process reaches } v_{i+1} \text{ from } v_i) = 1 - 1/2^B,$$

so,

$$\mathbb{P}(\text{The process reaches the } v_{n+1} \text{ vertex}) = (1 - 1/2^B)^n.$$

Now, by choosing $B = \log_2(n)$, we arrive at

$$\mathbb{P}(\text{The process reaches the } v_{n+1} \text{ vertex}) \to \mathrm{e}^{-1}, \quad n \to \infty.$$

In the sense of avoiding rare events, the probability of e^{-1} is much better than $1/2^n$. Moreover, the SE algorithm shares similar behavior. The "bad" walks become extinct and the "good" ones are split to continue to the next level. The only difference is that SE does not randomly generates next level states but uses the full enumeration procedure in Steps 2 and 4, thus introducing an additional variance reduction.

Despite the fact that we presented an artificial example, it is also illustrative enough for our purposes. Generally speaking, by increasing the budget (in a reasonable manner), we cannot expect to obtain a zero variance estimator for hard approximation problems, but we do hope to achieve a significant variance reduction. The benefits are clearly illustrated in the numerical section.

V. Numerical Results

We performed many experiments with all algorithms discussed above. In this section, we introduce various typical example cases in order to demonstrate the efficacy of the proposed methods. While these examples are simple, they illustrate the typical performance of the algorithms. In the first test case (model 1), we demonstrate the performance statistics on a set of relatively small instances, namely 100 random 30×15 polytopes. For the second and third model we choose hypercubes of dimensions thirty and forty, respectively. These hypercube models are interesting since the corresponding

number of vertices is known exactly, but too costly to obtain via full enumeration using **lrs**. All tests were executed on a dual-core 3.2Ghz processor.

To report our findings we use the following notation.

- $\widehat{|\mathcal{X}_{\mathrm{AV}}|}$, $\widehat{|\mathcal{X}_{\mathrm{SE}}|}$ and $\widehat{|\mathcal{X}_{\mathrm{GS}}|}$ are the estimators delivered by the algorithm of Avis (AV), SE and GS, respectively.
- CPU is the execution time in seconds.
- $\widehat{\mathrm{RE}}$ is the numerical RE of an estimator which is calculated via

$$\widehat{\mathrm{RE}} = \frac{\sqrt{\widehat{\mathrm{Var}}\left(\widehat{|\mathcal{X}|}\right)}}{\mathbb{E}\left(\widehat{|\mathcal{X}|}\right)},$$

  where $\widehat{|\mathcal{X}|}$ is an algorithm output, ($\widehat{|\mathcal{X}|}$ can stand for $\widehat{|\mathcal{X}_{\mathrm{AV}}|}$, $\widehat{|\mathcal{X}_{\mathrm{SE}}|}$, or $\widehat{|\mathcal{X}_{\mathrm{GS}}|}$), and $\widehat{\mathrm{Var}}\left(\widehat{|\mathcal{X}|}\right)$ is the estimated variance.
- The relative experimental error (REE) is given by

$$\mathrm{REE} = \frac{\left|\widehat{|\mathcal{X}|} - |\mathcal{X}^*|\right|}{|\mathcal{X}^*|},$$

  where $|\mathcal{X}^*|$ is the exact number of vertices.
- R is the replication parameter that stands for the number of independent repetitions to perform prior to averaging and delivering final result $\widehat{|\mathcal{X}|}$.
- The parameter *maxdepth* is specific for the AV algorithm. This criterion specifies the tree level for which a spanning tree nodes are fully enumerated, that is, no simulation is used. The randomized algorithm is then performed from this particular level. Setting the *maxdepth* to be greater than zero has its pros and cons. The advantage is that it can reduce the estimator variance, but, on the other hand, the computation effort required by the full enumeration procedure can be very high. See [2] for additional details.
- The SE and GS algorithms have the budget parameters denoted by B and N respectively. These parameters control the number of parallel trajectories within a single replication of these algorithms.

Next, we proceed with the models.

A. *Model* 1

For our first model we choose 100 randomly generated 30×15 polytopes. To generate an instance, we choose 30×15 matrix A and 15×1 vector $\mathbf{b}$ components from discrete uniform distributions $\mathsf{U}(1, 100)$ and $\mathsf{U}(1, 1000)$, respectively. For each polytope we obtain an exact number of vertices with RS using the **lrs** package that is publicly available at http://cgm.cs.mcgill.ca/~avis/C/lrslib, [1]. Table I summarizes the average REEs.

Figure 4 presents the natural logarithms of REEs achieved by the AV, SE and GS algorithms for each random instance. All the algorithms were given 100 seconds per polytope. For the AV algorithm we choose *maxdepth* of 3 and $R = 100$, since these parameters delivered the most accurate results, see Table I. The SE and GS methods uses $B = 20, R = 1000$, and $N = 1400, R = 1$, respectively. Figure 4 clearly indicates the superiority of SE and GS as compared to the AV algorithm. In particular, the experiment statistics showed that SE and GS outperformed AV for 94% and 75% of test cases, respectively. Furthermore, SE outperformed GS for 79% of random polytopes.

TABLE I
A SUMMARY OF AVERAGE REES AND $\widehat{\mathrm{RE}}$S OBTAINED FOR THE FIRST MODEL.

Algorithm	*maxdepth*	Average REE	Average $\widehat{\mathrm{RE}}$
AV	0	188.6%	55.38%
AV	1	56.14%	14.49%
AV	2	137.8%	46.20%
AV	**3**	**42.96%**	**11.05%**
AV	4	56.72%	5.84%
SE	—	7.95%	1.65%
GS	—	19.82%	1.84%

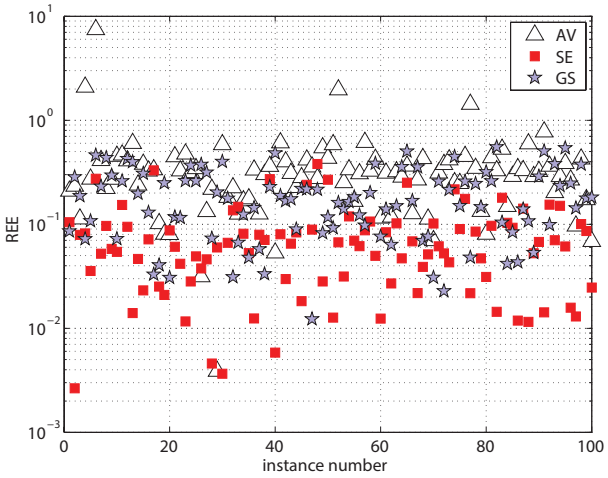

Fig. 4. Comparisson of REEs for 100 random 30×15 polytopes.

B. *Model* 2

For this model we consider the hypercube in dimension 30 which is defined by a 60×30 matrix. The exact number of vertices is $2^{30} \approx 1.074 \times 10^9$. We used the AV algorithm with *maxdepth* parameter equal to 4 and $R = 45$. For SE and GS we set $B = 250, R = 35$, and $N = 100, R = 10$, respectively. Table II summarizes 10 typical runs of the algorithms.

Table III summarizes the $\widehat{\mathrm{RE}}$s and the REEs of the algorithms for the second model.

For this model, the SE and GS introduce a comparable performance (with a slight advantage for SE), and their estimators are quite accurate. However, the algorithm of Avis reports a clear underestimation as shown in Table II. Note that $\widehat{|\mathcal{X}_{\mathrm{AV}}|}$'s $\widehat{\mathrm{RE}}$ is very small (less than 1%), but its REE is about 83%. The reason is that AV underestimates its variance and hence RE.

TABLE II
TYPICAL RESULTS OBTAINED IN 10 INDEPENDENT RUNS OF COUNTING ALGORITHMS FOR THE SECOND MODEL.

Run	$\widehat{\lvert\mathcal{X}_{\mathrm{AV}}\rvert}$	CPU	$\widehat{\lvert\mathcal{X}_{\mathrm{SE}}\rvert}$	CPU	$\widehat{\lvert\mathcal{X}_{\mathrm{GS}}\rvert}$	CPU
1	1.93×10^8	1000	1.50×10^9	953	1.02×10^9	1015
2	1.61×10^8	994	8.78×10^8	955	1.15×10^9	978
3	1.93×10^8	999	9.52×10^8	953	1.24×10^9	1034
4	1.93×10^8	998	1.05×10^9	959	1.05×10^9	1006
5	1.61×10^8	992	1.07×10^9	965	7.64×10^8	1005
6	1.47×10^8	987	1.35×10^9	950	1.83×10^9	1032
7	1.61×10^8	994	1.01×10^9	955	9.38×10^8	999
8	1.93×10^8	998	1.10×10^9	955	9.76×10^8	990
9	1.93×10^8	997	1.27×10^9	959	1.41×10^9	1007
10	1.93×10^8	998	1.14×10^9	953	1.00×10^9	982
Avg	1.79×10^8	995	1.13×10^9	956	1.14×10^9	1005

TABLE III
RELATIVE AND EXPERIMENTAL ERRORS FOR THE SECOND MODEL.

	$\widehat{\lvert\mathcal{X}_{\mathrm{AV}}\rvert}$	$\widehat{\lvert\mathcal{X}_{\mathrm{SE}}\rvert}$	$\widehat{\lvert\mathcal{X}_{\mathrm{GS}}\rvert}$
$\widehat{\mathrm{RE}}$	0.55%	5.60%	8.84%
REE	83.4%	5.43%	5.97%

C. Model 3

For the second hypercube model we consider the hypercube in dimension 40 which is defined by a 80×40 matrix. The exact number of vertices is $2^{40} \approx 1.099 \times 10^{12}$. We used the AV algorithm with *maxdepth* parameter equal to 4 and $R = 40$. For SE and GS we set $B = 750, R = 600$, and $N = 100, R = 10$, respectively. Table IV summarizes 10 typical runs of the algorithms.

TABLE IV
TYPICAL RESULTS OBTAINED IN 10 INDEPENDENT RUNS OF COUNTING ALGORITHMS FOR MODEL 3.

Run	$\widehat{\lvert\mathcal{X}_{\mathrm{AV}}\rvert}$	CPU	$\widehat{\lvert\mathcal{X}_{\mathrm{SE}}\rvert}$	CPU	$\widehat{\lvert\mathcal{X}_{\mathrm{GS}}\rvert}$	CPU
1	2.82×10^{10}	4561	1.04×10^{12}	3992	8.94×10^{11}	4167
2	2.82×10^{10}	4561	7.64×10^{11}	3987	1.29×10^{12}	4079
3	1.07×10^{10}	4547	7.25×10^{11}	3982	7.69×10^{11}	4149
4	2.82×10^{10}	4557	7.89×10^{11}	4002	1.08×10^{12}	4106
5	2.82×10^{10}	4558	7.17×10^{11}	4003	1.21×10^{12}	4248
6	1.07×10^{10}	4544	9.92×10^{11}	3999	1.28×10^{12}	4272
7	1.07×10^{10}	4550	9.61×10^{11}	4002	9.89×10^{11}	4126
8	2.82×10^{10}	4559	7.15×10^{11}	3982	1.20×10^{12}	4272
9	1.07×10^{10}	4548	1.01×10^{12}	4019	1.26×10^{12}	4090
10	2.82×10^{10}	4577	1.33×10^{12}	4004	9.22×10^{11}	4140
Avg	2.12×10^{10}	4556	9.04×10^{11}	3988	1.09×10^{12}	4151

Table V summarizes the average $\widehat{\mathrm{RE}}$ and the REE of the algorithms for the third model.

TABLE V
RELATIVE AND EXPERIMENTAL ERRORS FOR THE THIRD MODEL.

	$\widehat{\lvert\mathcal{X}_{\mathrm{AV}}\rvert}$	$\widehat{\lvert\mathcal{X}_{\mathrm{SE}}\rvert}$	$\widehat{\lvert\mathcal{X}_{\mathrm{GS}}\rvert}$
$\widehat{\mathrm{RE}}$	0.26%	5.73%	5.35%
REE	98.1%	17.8%	0.92%

This model emphasizes the underestimation problem of SIS algorithms, since both $\widehat{\lvert\mathcal{X}_{\mathrm{AV}}\rvert}$ and $\widehat{\lvert\mathcal{X}_{\mathrm{SE}}\rvert}$ provide an underestimation. Table V indicates that the REEs for these algorithms are 98.07% and 17.75% respectively. Consequently, the GS algorithm demonstrates a better performance for this case.

Our numerical results indicates that both SE and GS outperform the existing method of Avis [2]. We recommend the use of the GS algorithm for polytopes with a large number of vertices for a variety of reasons. Primarily this is because the GS estimator does not have the tendency to underestimate that tree-based methods to for large cases, and requires no calibration of parameters outside of an initial pilot run. On the other hand, for Knuth and SE approaches one must ensure that adequate values for *maxdepth* and budget (B) respectively are used so to ensure underestimation is not occurring.

It is important to note that the runtime of GS is dependent on the input size (n and d), whereas SE depends on the tree-size (number of vertices). As such, for polytopes with a relatively small to moderate number of vertices, say less than 10^9, SE often will outperform GS, and in fact deliver estimates much more quickly and with less variance. For this reason, in practice we recommend first running the SE algorithm with high budget parameters, and if the estimate is quite large (greater than 10^9), using the GS estimator instead.

ACKNOWLEDGEMENTS

This work was supported by the Australian Research Council Centre of Excellence for Mathematical & Statistical Frontiers, under grant number CE140100049.

REFERENCES

[1] D. Avis, "A revised implementation of the reverse search vertex enumeration algorithm," in *Polytopes - Combinatorics and Computation*, ser. DMV Seminar, G. Kalai and G. Ziegler, Eds. Birkhäuser Basel, 2000, vol. 29, pp. 177–198.

[2] D. Avis and L. Devroye, "Estimating the number of vertices of a polyhedron," *Information Processing Letters*, vol. 73, no. 34, pp. 137–143, 2000.

[3] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete & Computational Geometry*, vol. 8, no. 1, pp. 295–313, 1992.

[4] B. Chazelle, "An optimal convex hull algorithm and new results on cuttings (extended abstract)," in *FOCS*. IEEE Computer Society, 1991, pp. 29–38.

[5] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*. New York, NY, USA: Springer-Verlag New York, Inc., 1987.

[6] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall, "The double description method," *Contributions to the theory of games. Annals of Mathematics Studies*, vol. 2, no. 28, pp. 51–73, 1953.

[7] N. Linial, "Hard enumeration problems in geometry and combinatorics," *SIAM Journal on Matrix Analysis and Applications*, vol. 7, no. 2, pp. 331–5, 1986.

[8] L. G. Valiant, "The complexity of enumeration and reliability problems," *SIAM Journal on Computing*, vol. 8, no. 3, pp. 410–421, 1979.

[9] 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*, 1983, pp. 56–64.

[10] M. Dyer, "Approximate counting by dynamic programming," in *Proceedings of the 35th ACM Symposium on Theory of Computing*, 2003, pp. 693–699.

[11] 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*, pp. 671–697, 2004.

[12] M. Dyer, A. Frieze, and M. Jerrum, "On counting independent sets in sparse graphs," in *In 40th Annual Symposium on Foundations of Computer Science*, 1999, pp. 210–217.

[13] S. P. Vadhan, "The complexity of counting in sparse, regular, and planar graphs," *SIAM Journal on Computing*, vol. 31, pp. 398–427, 1997.

[14] M. Jerrum, L. G. Valiant, and V. V. Vazirani, "Random generation of combinatorial structures from a uniform distribution," *Theor. Comput. Sci.*, vol. 43, pp. 169–188, 1986.

[15] Z. I. Botev and D. P. Kroese, "Efficient Monte Carlo simulation via the Generalized Splitting method," *Statistics and Computing*, vol. 22, pp. 1–16, 2012.

[16] M. Jerrum and A. Sinclair, "The Markov chain Monte Carlo method: An approach to approximate counting and integration," in *Approximation Algorithms for NP-hard Problems*, D. Hochbaum, Ed. PWS Publishing, 1996, pp. 482–520.

[17] R. Y. Rubinstein, "The Gibbs cloner for combinatorial optimization, counting and sampling," *Methodology and Computing in Applied Probability*, vol. 11, pp. 491–549, 2009.

[18] R. Y. Rubinstein, A. Dolgin, and R. Vaisman, "The splitting method for decision making," *Communications in Statistics - Simulation and Computation*, vol. 41, no. 6, pp. 905–921, 2012.

[19] J. K. Blitzstein and P. Diaconis, "A sequential importance sampling algorithm for generating random graphs with prescribed degrees," *Internet Mathematics*, vol. 6, no. 4, pp. 489–522, 2011.

[20] 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*, vol. 100, pp. 109–120, March 2005.

[21] R. Y. Rubinstein, A. Ridder, and R. Vaisman, *Fast Sequential Monte Carlo Methods for Counting and Optimization*, ser. Wiley Series in Probability and Statistics. John Wiley & Sons, 2013.

[22] J. Blanchet and D. Rudoy, "Rare event simulation and counting problems," in *Rare Event Simulation Using Monte Carlo Methods*. John Wiley & Sons, United Kingdom, 2009.

[23] D. E. Knuth, "Estimating the efficiency of backtrack programs," *Math. Comp.*, vol. 29, pp. 651–656, 1975.

[24] H. Kahn and T. Harris, "Estimation of particle transmission by random sampling," *National Bureau of Standards Applied Mathematics Series*, vol. 12, pp. 27–30, 1951.

[25] M. J. J. Garvels, "The splitting method in rare event simulation," Ph.D. dissertation, University of Twente, Enschede, October 2000.

[26] P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic, "Splitting for rare event simulation: Analysis of simple cases." in *Winter Simulation Conference*, 1996, pp. 302–308.

[27] R. Vaisman and D. Kroese, "Stochastic enumeration method for counting trees," *Methodology and Computing in Applied Probability*, pp. 1–43, 2015.

[28] P. M. Gruber and J. M. Wills, *Handbook of Convex Geometry*, North Holland Publishing, Dordrecht, 1993.

[29] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo Method*, 2nd ed., ser. Wiley Series in Probability and Statistics. John Wiley & Sons, New York, 2008.

[30] D. P. Kroese, T. Taimre, and Z. I. Botev, *Handbook of Monte Carlo methods*. John Wiley & Sons, New York, 2011.

[31] P. C. Chen, "Heuristic sampling: A method for predicting the performance of tree searching programs," *SIAM J. Comput.*, vol. 21, no. 2, pp. 295–315, Apr. 1992.

[32] P. W. Purdom, "Tree size by partial backtracking." *SIAM J. Comput.*, vol. 7, no. 4, pp. 481–491, 1978.