Vaisman, Radislav; Botev, Zdravko; Ridder, Ad

Working Paper

Sequential Monte Carlo for Counting Vertex Covers in General Graphs

Tinbergen Institute Discussion Paper, No. 13-122/III

Provided in Cooperation with:
Tinbergen Institute, Amsterdam and Rotterdam

TI 2013-122/III
Tinbergen Institute Discussion Paper

Sequential Monte Carlo for counting Vertex Covers in General Graphs

Radislav Vaisman[1]
Zdravko I. Botev[2]
Ad Ridder[3]

[1] *Israel Institute of Technology, Haifa, Israel;*
[2] *The University of New South Wales, Sydney, Australia;*
[3] *Faculty of Economics and Business Administration, VU University Amsterdam, and Tinbergen Institute, The Netherlands.*

Tinbergen Institute is the graduate school and research institute in economics of Erasmus University Rotterdam, the University of Amsterdam and VU University Amsterdam.

More TI discussion papers can be downloaded at http://www.tinbergen.nl

Tinbergen Institute has two locations:

Tinbergen Institute Amsterdam
Gustav Mahlerplein 117
1082 MS Amsterdam
The Netherlands
Tel.: +31(0)20 525 1600

Tinbergen Institute Rotterdam
Burg. Oudlaan 50
3062 PA Rotterdam
The Netherlands
Tel.: +31(0)10 408 8900
Fax: +31(0)10 408 9031

Duisenberg school of finance is a collaboration of the Dutch financial sector and universities, with the ambition to support innovative research and offer top quality academic education in core areas of finance.

DSF research papers can be downloaded at: http://www.dsf.nl/

Duisenberg school of finance
Gustav Mahlerplein 117
1082 MS Amsterdam
The Netherlands
Tel.: +31(0)20 525 8579

Sequential Monte Carlo for Counting Vertex Covers in General Graphs

Radislav Vaisman[a], Zdravko I. Botev[b], Ad Ridder[c]

[a] Faculty of Industrial Engineering and Management,
Technion, Israel Institute of Technology, Haifa, Israel
slava@tx.technion.ac.il

[b] The University of New South Wales,
Sydney, NSW 2052, Australia
botev@unsw.edu.au

[c] Faculty of Economics and Business Administration,
Vrije University, Amsterdam, The Netherlands
ad.ridder@vu.nl

August 16, 2013

Abstract

In this paper we describe a Sequential Importance Sampling (SIS) procedure for counting the number of vertex covers in general graphs. The performance of SIS depends heavily on how close the SIS proposal distribution is to a uniform one over a suitably restricted set. The proposed algorithm introduces a probabilistic relaxation technique that uses Dynamic Programming in order to efficiently estimate this uniform distribution. The numerical experiments show that the scheme compares favorably with other existing methods. In particular the method is compared with *cachet* - an exact model counter, and the state of the art *SampleSearch*, which is based on Belief Networks and importance sampling.

Keywords. Vertex Cover, Counting problem, Sequential importance sampling, Dynamic Programming, Relaxation, Random Graphs.

1 Introduction

In graph theory, a vertex cover of a graph is a set of vertices such that each edge of the graph is incident to at least one vertex of the set. The problem of finding a minimum vertex cover is NP-hard. In this article we are interested in approximately counting all vertex covers in a graph using Monte Carlo methods. The area of counting, and the corresponding definition of $\sharp P$ *complete* class introduced by Valiant [15], has received much attention in the computer science community. Efficient algorithms have only been found for some problems. For example, Karp and Lubby [11] introduced a *fully polynomial randomized approximation scheme* (FPRAS) for counting the solutions of *disjunctive normal form* (DNF) satisfiability formula. Similar results were obtained for the knapsack and permanent counting problems, see [4, 9].

Unfortunately, there are negative results [5, 14], showing that counting the number of vertex covers remains hard even when restricted to planar bipartite graphs of bounded degree or regular graphs of constant degree.

There are two Monte Carlo approaches to tackling such difficult counting problems. The first is Markov Chain Monte Carlo (MCMC) and the second is sequential importance sampling. Both approaches exploit the finding of Jerrum et. al. [10] 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 stationary and limiting distribution equal to the desired uniform distribution. A number of MCMC approaches with good empirical performance have been proposed, see [8].

In this article we focus on the SIS approach in much the same sprit as Chen et al. [2]. In addition to [2], there are many examples of successful SIS implementations on various counting problems, see, for example, [1, 11]. The motivation of this article is to find a successful application of SIS to yet another important counting problem — counting the number of vertex covers of a graph.

The rest of the paper is organized as follows: In Section 2 we introduce the probabilistic relaxation to the vertex cover problem and prove that for the relaxed problem the expected number of covers can be calculated analytically in polynomial time. In Section 3 we formulate the proposed SIS algorithm and show that the relaxation introduced earlier leads to the construction of a good proposal distribution. In section 4 we provide numerical support for the accuracy of our method by comparing its performance with existing procedures. Finally, in Section 5 we summarize our findings and discuss possible directions for future research.

2 Vertex Cover Relaxation

In this section we introduce the vertex cover relaxation method. Given an undirected graph $G = G(V, E)$, with vertex set $|V| = n$, and edge set $|E| = m$, define a vertex ordering $v_1, v_2, \cdots, v_n \in V$ and denote by $d_i = \{j|(v_i, v_j) \in E, j > i\}$ the set of neighbors of node v_i such that each neighbor v_j satisfies

$j > i$. Let

$$\boldsymbol{p} = (p_1, \cdots, p_n) = \left(\frac{|d_1|}{n-1}, \frac{|d_2|}{n-2}, \cdots \frac{|d_{n-1}|}{1}, 0 \right)$$

be a vector of probabilities induced by G. Note that $|d_n| = 0$ so we define $p_n = 0$.

Consider now a probability space Ω_G of all graphs $G' = G'(V', E')$ where the set of vertices remains the same as in G, that is, $V' = V$, but each edge (v_i, v_j), $j > i$ is present in E' with probability $p_i = \frac{|d_i|}{n-i}$. Note that

$$\begin{aligned} p_i = 0 \quad &\Rightarrow \quad (v_i, v_j) \notin E' \;\; \forall j > i, \\ p_i = 1 \quad &\Rightarrow \quad (v_i, v_j) \in E' \;\; \forall j > i. \end{aligned}$$

Example 2.1 *Two simple examples are the bridge graph and the star graph. They are depicted in the following figure.*

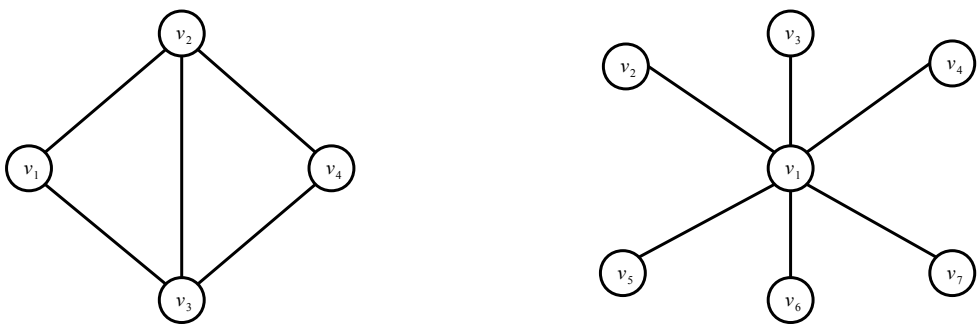

Figure 1: Left panel: bridge graph. Right panel: star graph.

Concerning the bridge graph, one can easily observe that v_1 is connected to v_2 and v_3, v_2 is connected to v_3 and v_4 and, finally, v_3 is connected only to v_4, so that we have $|d_1| = 2$, $|d_2| = 2$, $|d_3| = 1$ respectively. The last vertex v_4 has zero connections under our relaxation, so $|d_4| = 0$ and $\boldsymbol{p} = (\frac{2}{3}, \frac{2}{2}, \frac{1}{1}, 0)$. For the star graph we get $\boldsymbol{p} = (1, 0, 0, 0, 0, 0, 0)$. □

In what follows it will be more convenient to talk about the complementary probability, in other words, the probability that an edge is not present in G'. Formally, this vector of complementary probabilities can be written as

$$\boldsymbol{q} = (q_1, \cdots, q_n) = \mathbf{1} - \boldsymbol{p} = \left(1 - \frac{|d_1|}{n-1}, 1 - \frac{|d_2|}{n-2}, \cdots 1 - \frac{|d_{n-1}|}{1}, 1 \right) . \quad (1)$$

Given a graph $G = G(V, E)$, the calculation of vector $\boldsymbol{q}$ is straightforward.

2.1 Expected Number of Relaxed Vertex Covers

Let $\mathcal{X}_G$ be the set of all vertex covers of the given graph $G = G(V, E)$ and $|\mathcal{X}_G|$ be the its cardinality. Similarly, for any graph $G' \in \Omega_G$ the set of vertex covers is $\mathcal{X}_{G'}$. Denote by $\mathscr{G}$ a random graph in Ω_G, with probability law $\mathbb{P}$ and associated expectation $\mathbb{E}$. Clearly, for any realization $G' \in \Omega_G$

$$\mathbb{P}(\mathscr{G} = G') = \prod_{i=1}^{n-1} p_i^{|d_i'|} q_i^{n-i-|d_i'|},$$

where $0^0 = 1$ and d_i' is defined in the same way as d_i; that is, the set of neigbors of node v_i among the nodes $v_j, j > i$, in the graph G'. In this section we are interested in the expected number of vertex covers of the random graph $\mathscr{G}$. Note that this number is a random variable. If we denote it by $|\mathcal{X}_{\mathscr{G}}|$, then its expectation is given by

$$\mathbb{E}|\mathcal{X}_{\mathscr{G}}| = \sum_{G' \in \Omega_G} \mathbb{P}(\mathscr{G} = G')|\mathcal{X}_{G'}| \ .$$

Example 2.2 *Consider the bridge graph given in Example 2.1 with the vector of probabilities* $\boldsymbol{p} = (\frac{2}{3}, \frac{2}{2}, \frac{1}{1}, 0)$. *The set of 8 possible graphs in the probability space* Ω_G *is summarized below. Note that* $p_2 = 1$, *hence each graph must contain both edges* (v_2, v_3) *and* (v_2, v_4). *Similarly, because* $p_3 = 1$, *edge* (v_3, v_4) *is always present.*

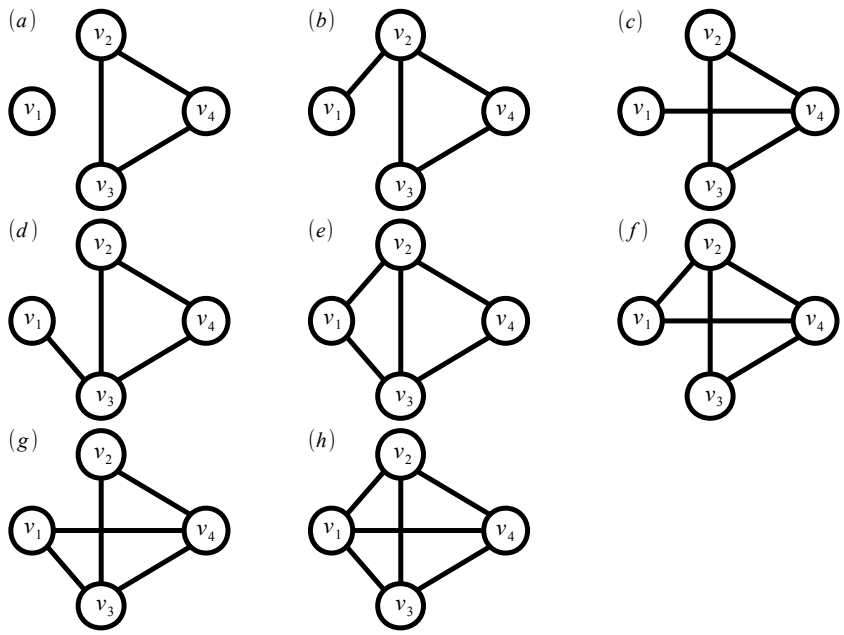

Figure 2: Example graph.

Graph (a) is generated with probability $(\frac{1}{3})^3$; graphs $(b),(c),(d)$ with probability $\frac{2}{3}(\frac{1}{3})^2$; graphs $(e),(f),(g)$ with probability $\frac{1}{3}(\frac{2}{3})^2$, and graph (h) with probability $(\frac{2}{3})^3$. The corresponding number of vertex covers for graphs $(a),(b),\cdots,(h)$ is $8,7,7,7,6,6,6,5$. For instance, consider graph (a), and its subgraph of nodes v_2,v_3,v_4 (with their incident edges). The vertex covers of this subgraph are $\{v_2,v_3\}$, $\{v_2,v_4\}$, $\{v_3,v_4\}$, $\{v_2,v_3,v_4\}$. Because node v_1 has no incident edge, these four sets are also covers for the whole graph. Of course, with node v_1 added, these sets remain vertex covers giving a total of eight covers. Hence, we can compute the expected number of vertex covers

$$\mathbb{E}|\mathcal{X}_{\mathscr{G}}| = \left(\frac{1}{3}\right)^3 8 + 3\frac{2}{3}\left(\frac{1}{3}\right)^2 7 + 3\frac{1}{3}\left(\frac{2}{3}\right)^2 6 + \left(\frac{2}{3}\right)^3 5 = 6\ .$$

□

A crucial property of the proposed relaxation is summarized next.

Proposition 2.1 *There exists a deterministic polynomial time algorithm that calculates $\mathbb{E}|\mathcal{X}_{\mathscr{G}}|$ analytically.*

To prove this proposition we first establish some auxiliary results.

Suppose that a subset of vertices $S \subset V$ has size k. Then we denote the ordered vertices of S by $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$; that is, $i_j < i_{j+1}$ for all $j = 1, 2, \ldots, k-1$.

Lemma 2.1 *Given the vector of complementary probabilities $\boldsymbol{q}$ defined in* (1), *we have*

$$\mathbb{E}|\mathcal{X}_{\mathscr{G}}| = \sum_{k=0}^{n} \sum_{\substack{S\subset V\\ |S|=k}} \prod_{j=1}^{k-1} q_{i_j}^{k-i_j}.$$

Proof. Define, for $k = 0, \ldots, n$, $A(n,k)$ to be the expected number of vertex covers of size $n-k$ in the random graph $\mathscr{G}$. Now observe that, when $C \subset V$ forms a vertex cover of a graph, its complement $S = V \backslash C$ forms an independent set. Thus,

$$\begin{aligned} A(n,k) &= \sum_{\substack{C\subset V\\ |C|=n-k}} \mathbb{P}(C \text{ forms a vertex cover}) \\ &= \sum_{\substack{S\subset V\\ |S|=k}} \mathbb{P}(S \text{ forms an independent set}) \end{aligned}$$

Let $S = \{v_{i_1}, \ldots, v_{i_k}\}$. Then S is an independent set iff none of the edges (v_{i_j}, v_{i_ℓ}) is chosen for all $j = 1, \ldots, k-1$ and all $\ell = j+1, \ldots, k$. This happens exactly with probability $\prod_{j=1}^{k-1} q_{i_j}^{k-i_j}$. The proof is complete by noting that

$$\mathbb{E}|\mathcal{X}_{\mathscr{G}}| = \sum_{k=0}^{n} A(n,k). \tag{2}$$

□

If, for example, the vector of probabilities $\boldsymbol{q}$ satisfies $q_i \equiv q \in (0,1)$ for all i, we obtain

$$A(n,k) = \binom{n}{k} q^{\binom{k}{2}} .$$

Since the last simplification is not valid for general graphs, we next explain how to calculate $A(n,k)$ analytically using a dynamic programming type of recursion.

Consider, for $m = 1, 2, \ldots, n$, the subgraph $G_m = G_m(V_m, E_m)$ of G consisting of the vertices $V_m = \{v_{n-m+1}, \ldots, v_n\}$ and their incident edges $E_m \subset E$. Similarly, we define the random graph $\mathscr{G}_m$ having V_m as vertex set, and edges chosen randomly according to the vector of probabilities $\boldsymbol{p}$ (of the original graph G). Finally, we define $A(m,k)$ to be the expected number of vertex covers of size $m-k$, $k \leq m$, in the random graph $\mathscr{G}_m$ for $m = 1, \ldots, n$. The idea is to compute the $A(m,k)$ numbers via a recursion by considering iteratively vertex covers in $V_1, V_2, \ldots$. Furthermore, the vertex covers of size $m+1-k$ in V_{m+1} can be decomposed into vertex covers of size $m+1-k$ in V_m, and in vertex covers of size $m-k$ in V_m. The precise recursion is formulated in the following.

Lemma 2.2 *For $m = 1, 2, \ldots, n-1$ and $k = 1, \ldots, m+1$, we have the recursion*

$$A(m+1,k) = q_{n-m}^{k-1} A(m,k-1) + A(m,k) , \tag{3}$$

where $A(m,0) = 1$ for $m = 1, \ldots, n$, and $A(1,1) = 1$.

Proof. Again we will use the property that, when $C \subset V_m$ forms a vertex cover of a graph, its complement $S = V_m \setminus C$ forms an independent set. Let S be an 'independent set', and $\#\{U\}$ stand for 'the size of set U'.

$$\begin{aligned} A(m+1,k) &= \mathbb{E}[\#\{S \text{ of size } k \text{ in } \{v_{n-m}, v_{n-m+1}, \ldots, v_n\}\}] \\ &= \mathbb{E}[\#\{S \text{ of size } k \text{ in } \{v_{n-m}, \ldots, v_n\}\} \text{ and } v_{n-m} \in S] + \\ &\quad + \mathbb{E}[\#\{S \text{ of size } k \text{ in } \{v_{n-m}, \ldots, v_n\}\} \text{ and } v_{n-m} \notin S] \end{aligned}$$

The two terms of the decomposition are computed as follows.

- First term: the remaining nodes $S \setminus \{v_{n-m}\}$ form an independent set of size $k-1$ in $\{v_{n-m+1}, \ldots, v_n\}$, and none of these $k-1$ nodes is connected with v_{n-m}. Since $A(m,k-1)$ is the expected number of such independent sets, and since choosing edges between nodes is independent of anything else, the first term yields $q_{n-m}^{k-1} A(m,k-1)$.
- Second term: the remaining nodes $S \setminus \{v_{n-m}\}$ form an independent set of size k in $\{v_{n-m+1}, \ldots, v_n\}$, thus it does not matter whether any of these nodes is connected with v_{n-m}, or not. Hence, the second term yields $A(m,k)$.

□

Hence, the algorithm for calculating $\mathbb{E}|\mathcal{X}_{\mathscr{G}}|$ for a given graph $G = G(V,E)$, $|V| = n$, $|E| = m$, can be summarized as follows.

Algorithm 2.1 ***Calculating Number of Relaxed Covers***
Input: $G = G(V, E)$
Output: $\mathbb{E}|\mathcal{X}_{\mathscr{G}}|$

1: $\boldsymbol{q} \leftarrow$ *calculate the vector of probabilities as in* (1)*;*
2: $\forall k \in \{0, \cdots, n\}$ *calculate* $A(n, k)$ *using recursion* (3)*;*
3: ***return*** $\mathbb{E}|\mathcal{X}_{\mathscr{G}}|$ *as in* (2)*.*

Proof of Proposition 2.1. Step (1) of the algorithm takes $\mathcal{O}(m)$ time; step (2) can be completed in $\mathcal{O}(n^2)$, and step (3) takes only linear time $\mathcal{O}(n)$. Since $|E| = m < n^2$ we conclude that the overall complexity of Algorithm 2.1 is $\mathcal{O}(n^2)$ and Proposition 2.1 follows. □

Algorithm 2.1 lays the groundwork for building a good proposal distribution for SIS. The algorithm can also count vertex covers exactly in some cases. As an example consider the star graph on the right panel of Figure 1, with n nodes. It is not difficult to determine the exact number of vertex covers in this case. If the central vertex participates in the cover, then there are 2^{n-1} covers, because any combination of the remaining $n-1$ vertices yields a valid cover. If the central vertex it is not in the cover, then all the remaining vertices must be part of one cover and we conclude that the exact number of vertex covers in the star graph is $2^{n-1} + 1$. If we take the ordering of nodes such that v_1 is the central vertex, then the induced vector of probabilities will be $\boldsymbol{p} = (1, 0, \ldots, 0)$. Now, running Algorithm 2.1 with a star graph as an input will result in $2^{n-1} + 1$. In general, we have the following result.

Proposition 2.2 *Given that an instance* $G = G(V, E)$ *induces a vector of probabilities* $\boldsymbol{p} = (p_1, \ldots, p_n)$ *where each* $p_i \in \{0, 1\}$*, Algorithm 2.1 provides the exact number of vertex covers, that is,* $\mathbb{E}|\mathcal{X}_{\mathscr{G}}| = |\mathcal{X}_G|$.

Proof. One way to proceed is by induction on the number of vertex covers combined with equation (3). However, it is much simpler to notice that given a vector of probabilities $\boldsymbol{p} = (p_1, \ldots, p_n)$ where each $p_i \in \{0, 1\}$, there is only one graph G with $\boldsymbol{p}$ as its induced vector of probabilities. This observation follows easily from the construction process of the 'random' graph using this particular $\boldsymbol{p}$. For $|\Omega_G| = 1$ we obtain $\mathbb{E}|\mathcal{X}_{\mathscr{G}}| = |\mathcal{X}_G|$. □

It follows from Proposition 2.2 and from Algorithm 2.1 that if there is an ordering in $G = G(V, E)$ such that the induced vector of probabilities $\boldsymbol{q}$ satisfies $q_i \in \{0, 1\}$, then the number of vertex covers is available analytically and can be calculated in $\mathcal{O}(|V|^2)$ time.

3 Sequential Importance Sampling

Consider the probability space $\{0, 1\}^n$ of all binary n-tuples $\boldsymbol{x} = (x_1, \ldots, x_n),\ x_i \in \{0, 1\}$. We denote a random n-tuple by $\boldsymbol{X}$ and a probability mass function (pmf) of $\boldsymbol{X}$ on $\{0, 1\}^n$ by $f(\boldsymbol{x})$. Probabilities and expectations are denoted by $\mathbb{P}_f$ and

$\mathbb{E}_f$. With any n-tuple $\boldsymbol{x} \in \{0,1\}^n$ we associate a subset $V(\boldsymbol{x}) \subset V$ of vertices by letting $v_i \in V(\boldsymbol{x})$ iff $x_i = 1$. Clearly, this subset may or may not be a vertex cover of the given graph $G = G(V, E)$.

In this section we consider estimating the number of vertex covers $|\mathcal{X}_G|$ by importance sampling simulations using a proposal pmf $f(\boldsymbol{x})$ of the random n-tuple. For that purpose, we restrict the class of proposal pmfs by requiring positive probability of all n-tuples for which the associated vertex subset is a vertex cover:

$$\mathscr{F} = \big\{ f : \{0,1\}^n \to [0,1]; \sum_{\boldsymbol{x} \in \{0,1\}^n} f(\boldsymbol{x}) = 1;\, V(\boldsymbol{x}) \in \mathcal{X}_G \;\Rightarrow\; f(\boldsymbol{x}) > 0 \big\}. \quad (4)$$

Using a proposal pmf $f \in \mathscr{F}$, the corresponding single-run importance sampling estimator is

$$Z_f = \frac{\mathbb{I}\{V(\boldsymbol{X}) \in \mathcal{X}_G\}}{f(\boldsymbol{X})}. \quad (5)$$

Clearly, this estimator is unbiased:

$$\mathbb{E}_f[Z_f] = \mathbb{E}_f\left[\frac{\mathbb{I}\{V(\boldsymbol{X}) \in \mathcal{X}_G\}}{f(\boldsymbol{X})}\right] = \sum_{\boldsymbol{x} \in \{0,1\}^n : V(\boldsymbol{x}) \in \mathcal{X}_G} \frac{1}{f(\boldsymbol{x})} \times f(\boldsymbol{x}) = |\mathcal{X}_G| \,.$$

This identity suggests the Monte Carlo estimator

$$\widehat{|\mathcal{X}_G|} = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathbb{I}\{V(\boldsymbol{X}_i) \in \mathcal{X}_G\}}{f(\boldsymbol{X}_i)} \,, \quad (6)$$

where $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_N$ are iid random n-tuples generated by pmf $f(\boldsymbol{x})$, and $V(\boldsymbol{X}_1), \ldots, V(\boldsymbol{X}_N)$ are their associated (random) vertex subsets. A measure of efficiency of an estimator is its *coefficient of variation* (CV) defined as the ratio of the variance to the square of the first moment [2]. Hence, the CV of the single-run estimator is

$$cv^2 = \frac{\mathbb{V}ar[Z_f]}{|\mathcal{X}_G|^2},$$

which is estimated simply by

$$\widehat{cv}^2 = \frac{\frac{1}{N-1}\sum_{i=1}^{N}\left(\frac{\mathbb{I}\{V(\boldsymbol{X}_i)\in\mathcal{X}_G\}}{f(\boldsymbol{X}_i)} - \widehat{|\mathcal{X}_G|}\right)^2}{\widehat{|\mathcal{X}_G|}^2} \,.$$

The CV of the Monte Carlo estimator $\widehat{|\mathcal{X}_G|}$ equals cv^2/N. The square root of the CV is commonly known as the *relative error* (RE). In our numerical tests we will estimate RE by

$$\widehat{\mathrm{RE}} = \widehat{cv}/\sqrt{N}. \quad (7)$$

The importance sampling simulation is implemented more efficiently by a sequential procedure. We now proceed with the details of the sequential importance sampling simulation (SIS) procedure. First, recall that $f(\boldsymbol{x})$ can be decomposed as

$$f(\boldsymbol{x}) = f_1(x_1) f_2(x_2|x_1) \ldots f_n(x_n|x_1, \ldots, x_{n-1}) \,. \quad (8)$$

This decomposition allows us to sample vertex subsets in a sequential manner, ensuring that only valid vertex covers are sampled. In particular, suppose we start adding vertices v_i for $i \in \{1, 2, \cdots\}$ to the vertex cover one by one with probability $f_i(x_i|x_1, \cdots, x_{i-1})$ and consider step i. We start this step with vertices $v_j, j < i$ for which $x_j = 1$ form a vertex cover in the subgraph induced by all vertices $v_1, \ldots, v_{i-1}$.

We either add v_i to the vertex cover, or we do not. While adding v_i to the cover is always feasible, not adding v_i is only feasible if there is no vertex v_j, $j < i$, that is not chosen ($x_j = 0$) and that is a neigbor of v_i ($(v_j, v_i) \in E$). This leads to the following SIS algorithm.

Algorithm 3.1 ***Sequential Sampling of Valid Covers***
Input: $G = G(V, E)$
Output: Importance weight of the generated vertex cover.

```
Z ← 1, x ← (0, ..., 0), so that V(x) = ∅.
for i = 1 → n do                    ▷ Note that at stage i, V(x) ⊆ {v_1, ..., v_{i-1}}
    if v_i must be added to the cover then
        x_i ← 1
    else
        U ~ U(0, 1)
        if U ≤ f_i(1|x_1, ..., x_{i-1}) then
            x_i ← 1, Z ← Z × 1/f_i(1|x_1, ..., x_{i-1})
        else
            Z ← Z × 1/f_i(0|x_1, ..., x_{i-1})
        end if
    end if
end for
Z ← Z × I{V(x) ∈ X_G}
return Repeat the above procedure N times to generate Z_1, ..., Z_N, and
    deliver the average (1/N) Σ_{i=1}^N Z_i.
```

We now explain how we construct a good proposal pmf $f \in \mathscr{F}$ by approximating the zero-variance pmf.

Definition 3.1 *A pmf $f \in \mathscr{F}$ is a zero-variance pmf if the associated importance sampling estimator Z_f has $\mathbb{V}ar[Z_f] = 0$.*

We claim that the uniform distribution on the space of vertex covers is a zero-variance pmf.

Lemma 3.1 *The pmf*

$$f^*(\boldsymbol{x}) = \frac{1}{|\mathcal{X}_G|} \mathbb{I}\{V(\boldsymbol{x}) \in \mathcal{X}_G\} \tag{9}$$

is a zero-variance pmf.

Proof. Clearly $f^* \in \mathscr{F}$, thus f^* is a feasible pmf. The associated importance sampling estimator Z_{f^*} is by (5)

$$Z_{f^*} = |\mathcal{X}_G| \, \mathbb{I}\{V(\boldsymbol{x}) \in \mathcal{X}_G\}$$

Hence, the second moment equals $|\mathcal{X}_G|^2$. □

Because we execute the importance sampling simulation through the implementation of a sequential procedure, we will analyse also the conditional pmfs of the zero-variance distribution. For that purpose, let be given binary $x_1, \ldots, x_{i-1}$ indicating whether nodes $v_j, j < i$, are part of a vertex subset $V(\boldsymbol{x})$. Note that under the zero-variance pmf f^* only vertex covers receive positive probability. Therefore, define subgraphs $G^{[i]}$ and $G^{[-i]}$ for $i = 1, \ldots, n-1$ as follows.

1. $i = 1$.
   - $G^{[1]} = G_1(V_1, E_1)$ where $V_1 = \{v_2, \ldots, v_n\}$, and $E_1 = \{(v_j, v_k) | v_j, v_k \in V_1\} \cap E$;
   - $G^{[-1]} = G_2(V_2, E_2)$ where $V_2 = V_1 \setminus \{v_k | k \geq 2, (v_1, v_k) \in E\}$, and $E_2 = \{(v_j, v_k) | v_j, v_k \in V_2\} \cap E$.
2. $i = 2, \ldots, n-1$.
   - $G^{[i]} = G_1(V_1, E_1)$ where

     $$V_1 = \{v_{i+1}, \ldots, v_n\} \setminus \{v_k | k \geq i+1, \text{and } \exists j \leq i-1 \text{with } x_j = 0, (v_j, v_k) \in E\}, \tag{10}$$

     and $E_1 = \{(v_j, v_k) | v_j, v_k \in V_1\} \cap E$;
   - $G^{[-i]} = G_2(V_2, E_2)$ where

     $$V_2 = V_1 \setminus \{v_k | k \geq i+1, (v_i, v_k) \in E\}, \tag{11}$$

     and $E_2 = \{(v_j, v_k) | v_j, v_k \in V_2\} \cap E$.

Note that these subgraphs depend on the given variables $x_1, \ldots, x_{i-1}$. For convenience we do not denote this explicitly. Note also that a subgraph can be the empty set.

Example 3.1 *Consider the bridge graph in Example 2.1. The $G^{[1]}$ and $G^{[-1]}$ graphs are depicted in the figure below.*

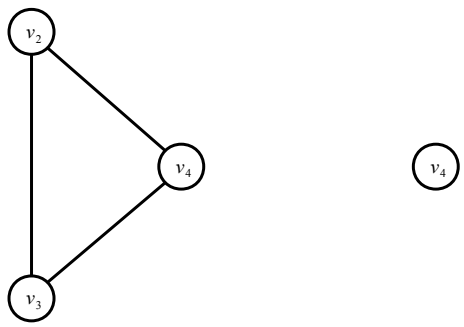

Figure 3: Left panel: $G^{[1]}$ graph. Right panel: $G^{[-1]}$ graph.

Consider $x_1 = 1$ and $i = 2$. Then,

- *$G^{[2]}$ has vertex set $V_1 = \{v_3, v_4\}$ and edge set $E_2 = \{(v_3, v_4)\}$.*
- *$G^{[-2]}$ has vertex set $V_2 = V_1 \setminus \{v_3, v_4\} = \emptyset$.*

Consider $x_1 = 1, x_2 = 1$ and $i = 3$. Then,

- *$G^{[3]}$ has vertex set $V_1 = \{v_4\}$.*
- *$G^{[-3]}$ has vertex set $V_2 = V_1 \setminus \{v_4\} = \emptyset$.*

□

Lemma 3.2 *Let be given binary variables $x_1, \ldots, x_{i-1}$, and denote the number of vertex covers in the associated $G^{[i]}$ and $G^{[-i]}$ graphs by $|\mathcal{X}_{G^{[i]}}|$ and $|\mathcal{X}_{G^{[-i]}}|$, where $|\mathcal{X}_\emptyset| = 1$. Then the zero-variance conditional pmf is as follows.*

(a). Case $i = 1$.

$$f_1^*(1) = \frac{|\mathcal{X}_{G^{[1]}}|}{|\mathcal{X}_{G^{[1]}}| + |\mathcal{X}_{G^{[-1]}}|}$$
$$f_1^*(0) = 1 - f_1^*(1)$$

(b). Case $i = 2, \ldots, n$, and there is a node $v_j, j < i$, such that $x_j = 0$, and $(x_j, x_i) \in E$.

$$f_i^*(1|x_1, \ldots, x_{i-1}) = 1$$
$$f_i^*(0|x_1, \ldots, x_{i-1}) = 0$$

(c). Case $i = 2, \ldots, n-1$, and for all nodes $v_j, j < i$, that have $x_j = 0$, it holds that $(x_j, x_i) \notin E$.

$$f_i^*(1|x_1, \ldots, x_{i-1}) = \frac{|\mathcal{X}_{G^{[i]}}|}{|\mathcal{X}_{G^{[i]}}| + |\mathcal{X}_{G^{[-i]}}|} \tag{12}$$
$$f_i^*(0|x_1, \ldots, x_{i-1}) = 1 - f_i^*(1|x_1, \ldots, x_{i-1})$$

(d). Case $i = n$, and for all nodes $v_j, j < n$, that have $x_j = 0$, it holds that $(x_j, x_n) \notin E$.

$$f_n^*(1|x_1, \ldots, x_{n-1}) = f_n^*(0|x_1, \ldots, x_{n-1}) = \frac{1}{2}$$

Proof. We elaborate case (c). Case (a) follows similarly, while cases (b) and (d) are straightforward. Because the (unconditial) zero-variance pmf f^* is the uniform distribution on the space $\mathcal{X}_G$ of vertex covers, we get for the conditional pmf

$$f_i^*(1|x_1, \ldots, x_{i-1}) = \frac{|\{V(\boldsymbol{y}) \in \mathcal{X}_G : y_1 = x_1, \ldots, y_{i-1} = x_{i-1}, y_i = 1\}|}{|\{V(\boldsymbol{y}) \in \mathcal{X}_G : y_1 = x_1, \ldots, y_{i-1} = x_{i-1}\}|}. \quad (13)$$

The variables $x_1, \ldots, x_{i-1}$ have assigned values 0 or 1 in such a manner that all edges $(v_j, v_k) \cap E, j, k \leq i-1$ are covered. Consider any node $v_k, k \geq i+1$, and suppose that there is a node $v_j, j \leq i-1$ such that $x_j = 0$ and that edge $(v_j, v_k) \in E$. This means that v_j is not part of the vertex cover. Hence, to cover the edge (v_j, v_k), node v_k gets surely assigned $x_k = 1$.

- If we would include v_i in the cover by setting $x_i = 1$, we obtain the subgraph $G^{[i]}$ with vertex set $V_1 \subset \{v_{i+1}, \ldots, v_n\}$ given in (10). Clearly, we can map the vertex covers of $G^{[i]}$ one-to-one on those covers of G that are given by the variables $x_1, \ldots, x_{i-1}$ and $x_i = 1$. That means,

$$|\{V(\boldsymbol{y}) \in \mathcal{X}_G : y_1 = x_1, \ldots, y_{i-1} = x_{i-1}, y_i = 1\}| = |\mathcal{X}_{G^{[i]}}|. \quad (14)$$

- Suppose that we do not include v_i in the cover, by setting $x_i = 0$. To cover an edge $(v_i, v_k) \in E, k \geq i+1$, node v_k gets surely assigned $x_k = 1$. In this way we obtain the subgraph $G^{[-i]}$ with vertex set V_2 given in (11). Now we can map the vertex covers of $G^{[-i]}$ one-to-one on those covers of G that are given by the variables $x_1, \ldots, x_{i-1}$ and $x_i = 0$. That means that

$$|\{V(\boldsymbol{y}) \in \mathcal{X}_G : y_1 = x_1, \ldots, y_{i-1} = x_{i-1}, y_i = 0\}| = |\mathcal{X}_{G^{[-i]}}|. \quad (15)$$

The conclusion that (12) and (13) are equivalent, follows immediately. □

Example 3.2 *Consider the bridge example.*

1. *$i = 1$. The subgraphs $G^{[1]}$ and $G^{[-1]}$ are shown in Figure 3. Clearly we get $|\mathcal{X}_{G^{[1]}}| = 4$ and $|\mathcal{X}_{G^{[-1]}}| = 2$. Thus by (a) in Lemma 3.2,*

$$f_1^*(1) = \frac{4}{6} = \frac{2}{3}; \quad f_1^*(0) = \frac{1}{3}.$$

2. *$i = 2$.*

- *Case* $x_1 = 1$. *See Example 3.1 for* $G^{[2]} = \big(\{v_3, v_4\}, \{(v_3, v_4)\}\big)$ *and* $G^{[-2]} = \emptyset$. *Thus,* $|\mathcal{X}_{G^{[2]}}| = 3$, $|\mathcal{X}_{G^{[-2]}}| = 1$, *and by (c) in Lemma 3.2,*
$$f_2^*(1|1) = \frac{3}{4}; \quad f_2^*(0|1) = \frac{1}{4}.$$
- *Case* $x_1 = 0$. *By (b) in Lemma 3.2*
$$f_2^*(1|0) = 1; \quad f_2^*(0|1) = 0.$$

3. $i = 3$.

- *Case* $x_1 = 1, x_2 = 1$. *See Example 3.1 for* $G^{[3]} = \big(\{v_4\}, \emptyset\big)$ *and* $G^{[-3]} = \emptyset$. *Thus,* $|\mathcal{X}_{G^{[3]}}| = 2$, $|\mathcal{X}_{G^{[-3]}}| = 1$, *and by (c) in Lemma 3.2,*
$$f_3^*(1|1,1) = \frac{2}{3}; \quad f_3^*(0|1,1) = \frac{1}{3}.$$
- *Case* $x_1 = 1, x_2 = 0$. *By (b) in Lemma 3.2*
$$f_3^*(1|1,0) = 1; \quad f_3^*(0|1,0) = 0.$$
- *Case* $x_1 = 0, x_2 = 1$. *By (b) in Lemma 3.2*
$$f_3^*(1|0,1) = 1; \quad f_3^*(0|0,1) = 0.$$

4. $i = 4$.

- *Case* $x_1 = 1, x_2 = 1, x_3 = 1$. *By (d) in Lemma 3.2,*
$$f_4^*(1|1,1,1) = f_4^*(0|1,1,1) = \frac{1}{2}.$$
- *Case* $x_1 = 1, x_2 = 1, x_3 = 0$. *By (b) in Lemma 3.2*
$$f_4^*(1|1,1,0) = 1; \quad f_4^*(0|1,1,0) = 0.$$
- *Case* $x_1 = 1, x_2 = 0, x_3 = 1$. *By (b) in Lemma 3.2*
$$f_4^*(1|1,0,1) = 1; \quad f_4^*(0|1,0,1) = 0.$$
- *Case* $x_1 = 0, x_2 = 1, x_3 = 1$. *By (d) in Lemma 3.2,*
$$f_4^*(1|0,1,1) = f_4^*(0|0,1,1) = \frac{1}{2}.$$

Note that the product (8) *of these conditional pmfs indeed gives the uniform distribution on the space of all vertex covers:*

$$f^*(1,1,1,1) = f_1^*(1)f_2^*(1|1)f_3^*(1|1,1)f_4^*(1|1,1,1) = \frac{2}{3}\,\frac{3}{4}\,\frac{2}{3}\,\frac{1}{2} = \frac{1}{6}$$
$$f^*(1,1,1,0) = f_1^*(1)f_2^*(1|1)f_3^*(1|1,1)f_4^*(0|1,1,1) = \frac{2}{3}\,\frac{3}{4}\,\frac{2}{3}\,\frac{1}{2} = \frac{1}{6}$$
$$f^*(1,1,0,1) = f_1^*(1)f_2^*(1|1)f_3^*(0|1,1)f_4^*(1|1,1,0) = \frac{2}{3}\,\frac{3}{4}\,\frac{1}{3}\,1 = \frac{1}{6}$$
$$f^*(1,0,1,1) = f_1^*(1)f_2^*(0|1)f_3^*(1|1,0)f_4^*(1|1,0,1) = \frac{2}{3}\,\frac{1}{4}\,1\,1 = \frac{1}{6}$$
$$f^*(0,1,1,1) = f_1^*(0)f_2^*(1|0)f_3^*(1|0,1)f_4^*(1|0,1,1) = \frac{1}{3}\,1\,1\,\frac{1}{2} = \frac{1}{6}$$
$$f^*(0,1,1,0) = f_1^*(0)f_2^*(1|0)f_3^*(1|0,1)f_4^*(0|0,1,1) = \frac{1}{3}\,1\,1\,\frac{1}{2} = \frac{1}{6}$$

□

Since we can not calculate the zero-variance conditional pmf $f_i^*(x_i|x_1,\ldots,x_{i-1})$ exactly for the (a) and (c) cases in Lemma 3.2, we approximate it via the proposal

$$f_i(1|x_1,\ldots,x_{i-1}) = \frac{\mathbb{E}|\mathcal{X}_{\mathscr{G}^{[i]}}|}{\mathbb{E}|\mathcal{X}_{\mathscr{G}^{[-i]}}| + \mathbb{E}|\mathcal{X}_{\mathscr{G}^{[i]}}|}\ , \tag{16}$$

Thus, in essence, we approximate $|\mathcal{X}_{G^{[i]}}|$ via $\mathbb{E}|\mathcal{X}_{\mathscr{G}^{[i]}}|$ and $|\mathcal{X}_{G^{[-i]}}|$ via $\mathbb{E}|\mathcal{X}_{\mathscr{G}^{[-i]}}|$, with both expectations readily computed using Algorithm 2.1.

In the next section we provide numerical experiments demonstrating the performance of Algorithm 3.1 with (16) used as a proposal density.

4 Numerical Results

In this section we consider the performance of Algorithm 3.1 on four different graphs. We performed all computation on Core i5 laptop with 4GB RAM. The reported CPU time is measured in seconds. For smaller problems we are able to compute the exact count. In that case we report the numerical relative error of the estimates; in the other cases we report the statistical relative error estimated according to (7).

We compare our algorithm with the following methods.

- *Cachet* is exact model counting software introduced by Sang et al. in [13]. This method uses the well known SAT solver zChaff [12] and combines component caching with traditional clause learning within the setup of model counting.
- *SampleSearch* is a probabilistic model counting technique proposed by Gogate and Dechter in [3, 6]. The method can deliver upper and lower bounds on counting problems and is based on sampling from the search space of a Boolean formula. Similarly to *Cachet*, it also use a DPLL-based SAT solvers during the execution in order to construct the sampling search space.

4.1 Model 1

A graph with $|V| = 100$ and $|E| = 2,432$. The graph was generated in the following way. Each possible edge (v_i, v_j) was present in the graph with probability p, where $p \sim \mathsf{U}(0,1)$. The performance of SIS Algorithm 3.1 is summarized in the following table.

Table 1: Performance of 10 runs of the SIS Algorithm 3.1 on Model 1 with sample size $N = 100$.

Run	$\widehat{\lvert\mathcal{X}_G\rvert}$	error	CPU
1	2.459×10^5	1.604×10^{-2}	1.807
2	2.451×10^5	2.001×10^{-2}	1.610
3	2.512×10^5	1.698×10^{-2}	1.683
4	2.397×10^5	1.380×10^{-2}	1.645
5	2.409×10^5	1.341×10^{-2}	1.723
6	2.414×10^5	1.505×10^{-2}	1.753
7	2.459×10^5	1.599×10^{-2}	1.835
8	2.450×10^5	1.713×10^{-2}	1.708
9	2.398×10^5	1.618×10^{-2}	1.586
10	2.447×10^5	1.679×10^{-2}	1.627
Average	2.440×10^5	1.614×10^{-2}	1.698

For comparison:

- Running *cachet* delivers an exact solution of $244,941$ in 0.75 seconds.
- Running *SampleSearch* 10 times provides an average of $196,277$ in 60 seconds with estimated relative error of about 20%.

For this example *cachet* provides the exact solution in the fastest time. *SampleSearch* provides a good lower bound, but at a high CPU time, and Algorithm 3.1 performs as second best.

4.2 Model 2

A graph with $|V| = 300$ and $|E| = 21,094$, generated in the same manner as Model 1. The SIS performance is given in the following table.

Table 2: Performance of 10 runs of the SIS Algorithm 3.1 for Model 2 with $N = 100$.

Run N_0	$\widehat{\lvert\mathcal{X}_G\rvert}$	error	CPU
1	1.325×10^{14}	4.336×10^{-2}	55.98
2	1.298×10^{14}	3.710×10^{-2}	58.41
3	1.283×10^{14}	3.932×10^{-2}	54.71
4	1.321×10^{14}	3.519×10^{-2}	58.99
5	1.322×10^{14}	4.102×10^{-2}	58.81
6	1.421×10^{14}	5.635×10^{-2}	52.69
7	1.361×10^{14}	4.739×10^{-2}	55.96
8	1.287×10^{14}	4.044×10^{-2}	59.18
9	1.202×10^{14}	4.435×10^{-2}	56.64
10	1.387×10^{14}	4.171×10^{-2}	56.64
Average	1.321×10^{14}	4.262×10^{-2}	56.80

For comparison:

- Running *cachet* delivers an exact solution of 1.306×10^{14} in about 17 minutes.
- Running *SampleSearch* ten times provides an average of 5.791×10^{13} in $1,200$ seconds with estimated relative error of about 55%.

4.3 Model 3

A graph with $|V| = 1,000$ and $|E| = 64,251$, where each edge (v_i, v_j) was present in the graph with probability p and p is generated from a truncated normal distribution on the interval $[0, 1]$ with $\mu = 0.1$ and $\sigma = 0.1$. The results are summarized bellow.

Table 3: Performance of 10 runs of the SIS Algorithm 3.1 on Model 3 with $N = 100$.

Run N_0	$\widehat{\lvert\mathcal{X}_G\rvert}$	$\widehat{\mathrm{RE}}$	CPU
1	4.384×10^{32}	4.450×10^{-2}	661.5
2	4.058×10^{32}	4.151×10^{-2}	703.1
3	4.014×10^{32}	4.814×10^{-2}	691.9
4	4.137×10^{32}	4.215×10^{-2}	721.3
5	4.220×10^{32}	4.437×10^{-2}	691.0
6	4.124×10^{32}	4.555×10^{-2}	688.6
7	4.422×10^{32}	4.788×10^{-2}	667.0
8	4.184×10^{32}	4.563×10^{-2}	682.0
9	4.234×10^{32}	4.103×10^{-2}	698.9
10	4.261×10^{32}	4.813×10^{-2}	648.6
Average	4.204×10^{32}	4.489×10^{-2}	685.4

For comparison:

- *cachet* was unable to deliver a solution within 2 days of CPU time. The lower bound of 3.439×10^9 was only supplied.
- *SampleSearch* failed to initialize possibly due to the large size of the problem.

4.4 Model 4

A graph with $|V| = 1,000$ and $|E| = 249,870$. This time p is generated from a truncated Normal distribution with $\mu = 0.5$ and $\sigma = 0.3$. The results are summarized bellow.

Table 4: 10 runs of the SIS Algorithm 3.1 Model 4 with $N = 100$.

Run N_0	$\widehat{\lvert\mathcal{X}_G\rvert}$	$\widehat{\mathrm{RE}}$	CPU
1	2.749×10^{11}	1.608×10^{-2}	1841
2	2.762×10^{11}	1.531×10^{-2}	1847
3	2.848×10^{11}	1.720×10^{-2}	1658
4	2.737×10^{11}	1.274×10^{-2}	1562
5	2.795×10^{11}	1.521×10^{-2}	1642
6	2.819×10^{11}	1.591×10^{-2}	1853
7	2.764×10^{11}	1.701×10^{-2}	1756
8	2.776×10^{11}	1.768×10^{-2}	1544
9	2.698×10^{11}	1.468×10^{-2}	1767
10	2.778×10^{11}	1.605×10^{-2}	1708
Average	2.773×10^{11}	1.579×10^{-2}	1718

For comparison:

- *cachet* was timed out after 2 days and was unable to deliver a solution. The lower bound of 9.601×10^{10} was supplied.
- *SampleSearch* failed to initialize. We speculate that the reason for this is that the problem is too large.

4.5 Nonrandom models

Finally, consider nonrandom models, where we expect our algorithm's performance to deteriorate, namely the hypercube graphs $H_n,\ n = 4, 5, 6, 7$ with 2^n vertices and $n2^{n-1}$ edges, see [7]. Using *cachet* we was able to determine the exact number of vertex covers for H_4, H_5 and H_6, namely, 743, 254475 and 1.976×10^{10} respectively. The following table summarizes the average values obtained with 10 runs of the SIS algorithm. We set the sample size N to be $50, 250, 1500$ and 10^4 for H_4, H_5, H_6 and H_7, respectively, so that the estimated relative error was below 3%.

Table 5: Performance of 10 runs of the SIS algorithm for the Hypercube graphs.

Instance	$\widehat{\lvert\mathcal{X}_G\rvert}$	$\widehat{\mathrm{RE}}$	CPU
H_4	745.9	2.87×10^{-2}	0.008
H_5	2.550×10^5	2.86×10^{-2}	0.157
H_6	1.983×10^{10}	2.67×10^{-2}	4.841
H_7	7.819×10^{19}	2.89×10^{-2}	199.8

5 Concluding Remarks

In this article we used probabilistic relaxation in combination with SIS to estimate the number of vertex covers of a graph. The probabilistic relaxation method ensures the availability of exact vertex cover computations over the Ω_G space, which are then used to design a good proposal for the sequential importance sampling algorithm. The design of the sequential sampling procedure guarantees that a valid vertex cover is generated.

The proposed algorithm is easy to implement and the numerical results strongly suggest that the practical performance is comparable with and sometimes better than currently existing methods. With a sample size as little as 100, we observed low relative errors on problems with large dimensionality.

Of interest in the future is to theoretically investigate how closely the proposal distribution described in this article approximates the zero-variance measure. In addition, of interest will be the development of similar relaxation techniques to other graph counting problems.

References

[1] Joseph Blitzstein and Persi Diaconis. A sequential importance sampling algorithm for generating random graphs with prescribed degrees. *Internet Mathematics*, 6:487–520, 2010.

[2] Yuguo Chen, Persi Diaconis, Susan P. Holmes, and Jun S. Liu. Sequential Monte Carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100:109–120, March 2005.

[3] Rina Dechter and Vibhav Gogate. A new algorithm for sampling CSP solutions uniformly at random. In *Principles and Practice of Constraint Programming*, May 2006.

[4] Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th ACM Symposium on Theory of Computing*, pages 693–699, 2003.

[5] Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. In *In 40th Annual Symposium on Foundations of Computer Science*, pages 210–217, 1999.

[6] Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. In *Proceedings of the 22nd national conference on Artificial Intelligence - Volume 1*, AAAI'07, pages 198–203. AAAI Press, 2007.

[7] Frank Harary, John P. Hayes, and Horng-Jyh Wu. A survey of the theory of hypercube graphs. *Computers & Mathematics with Applications*, 15(4):277 – 289, 1988.

[8] Mark Jerrum and Alistair Sinclair. The Markov chain Monte Carlo method: An approch to approximate counting and integration. In D. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 482 – 520. PWS Publishing, 1996.

[9] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *Journal of the ACM*, pages 671–697, 2004.

[10] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

[11] Richard M. Karp and Michael 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.

[12] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Annual ACM IEEE Design Automation Conference*, pages 530–535. ACM, 2001.

[13] Tian Sang, Fahiem Bacchus, Paul Beame, Henry Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *Seventh International Conference on Theory and Applications of Satisfiability Testing*, 2004.

[14] Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM Journal on Computing*, 31:398–427, 1997.

[15] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.