Sequential Monte Carlo Method for counting vertex covers

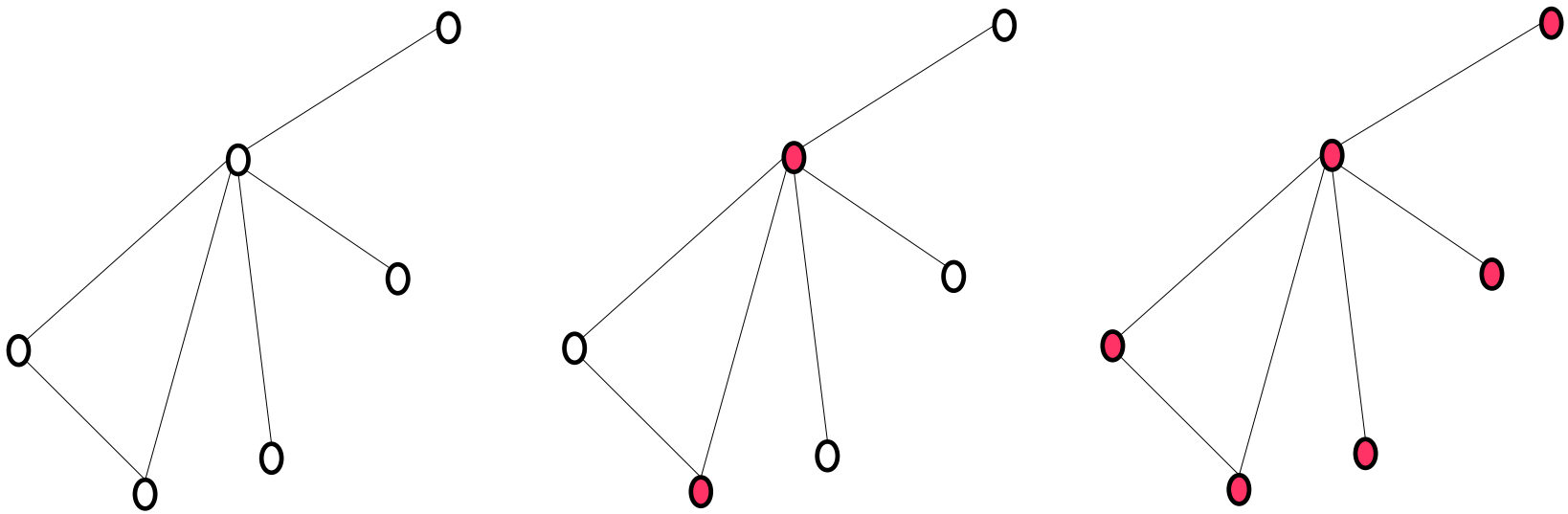
Slava Vaisman
Faculty of Industrial Engineering and Management
Technion, Israel Institute of Technology
Haifa, Israel

May 18, 2013

Table of contents

Vertex Cover

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.



Hardness of vertex cover

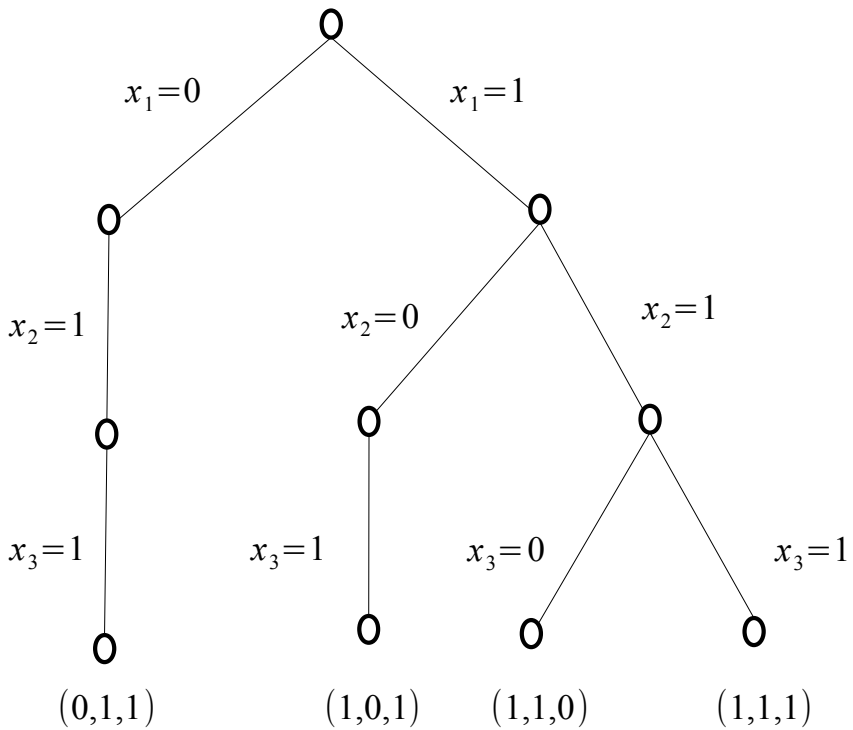
- Finding minimum vertex cover is NP-hard
- Counting all vertex covers - $\sharp P$
- Counting the number of vertex covers remains hard even when restricted to planar bipartite graphs of bounded degree or regular graphs of constant degree

Counting for hard problems

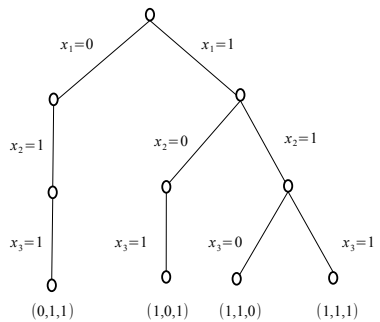
- There is an equivalence between counting and uniform sampling so 2 approaches proved to be useful
- MCMC (Permanent and many more...)
- (Sequential) Importance Sampling (Karp and Lubby - Counting DNF satisfying assignments)

Counting for self reducible problems (1)

$$\Psi = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3),\ x_i \in \{0, 1\}$$

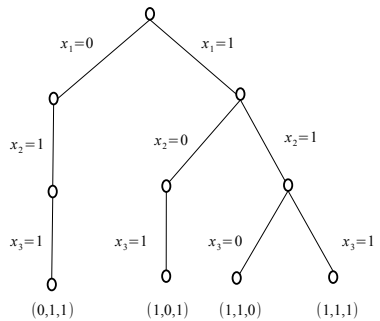


Counting for self reducible problems (2)



- Let us define
  $\mathbb{P}(X_i = 1 | X_1, \cdots X_{i-1}) = \frac{\sharp \text{ leaves in right subtree}}{\text{total } \sharp \text{ of leaves}}$
- If we know the number of leaves - we can sample a solution uniformly from the solution set χ
- For example, sampling $x = (1, 1, 0)$ in our tree will result in
  - $\mathbb{P}(X_1 = 1) = \frac{3}{4}$
  - $\mathbb{P}(X_2 = 1 | X_1 = 1) = \frac{2}{3}$
  - $\mathbb{P}(X_3 = 0 | X_1 = 1, X_2 = 1) = \frac{1}{2}$
  - $\frac{1}{|\chi|} = \frac{3}{4} \cdot \frac{2}{3} \cdot \frac{1}{2} = \frac{1}{4} \Rightarrow |\chi| = 4$
- Conclusion: It will be enough to sample any solution and get an exact cardinality of χ. We can look at this as zero variance importance sampling.

Counting for self reducible problems (3)



- Problem - we do not know the number of leaves. That is exactly what we are trying to find.
- Good news - we do not need to know the number of leaves exactly, even a clue can be very helpful
- Next, we will try to obtain some reasonable approximation to this number. We will use a relaxed domain to achieve this goal.

Vertex Cover Relaxation (1)

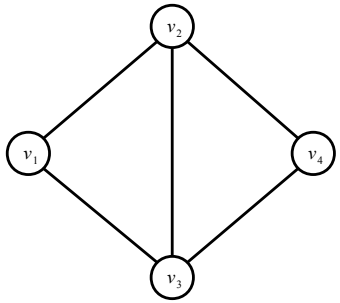
- Given a graph $G = G(V, E)$, $|V| = n, |E| = m$
- Introduce some vertex ordering $\{v_1, v_2, \cdots v_n\}$ and denote by $d_i = \{j|(v_i, v_j) \in E, j > i\}$ a neighbors set of v_i such that each neighbor v_j satisfies $j > i$.

Definition

A probability vector induced by G is given by

$$P = \{p_1, \cdots, p_n\} = \{\frac{|d_1|}{n-1}, \frac{|d_2|}{n-2}, \cdots \frac{|d_{n-1}|}{1}, 0\} \quad (1)$$

Vertex Cover Relaxation (2)



$$P = \{\frac{2}{3}, \frac{2}{2}, \frac{1}{1}, 0\}.$$

It will be more convenient to talk about the complement probability, in other words, the probability that the edge is not present in G'. Formally, this complement probability can be written as

$$Q = \{q_1, \cdots, q_n\} = 1 - P = \{1 - \frac{|d_1|}{n-1}, 1 - \frac{|d_2|}{n-2}, \cdots 1 - \frac{|d_{n-1}|}{1}, 1\}$$

In this case

$$Q = \{\frac{1}{3}, 0, 0, 1\}.$$

Vertex Cover Relaxation (3)

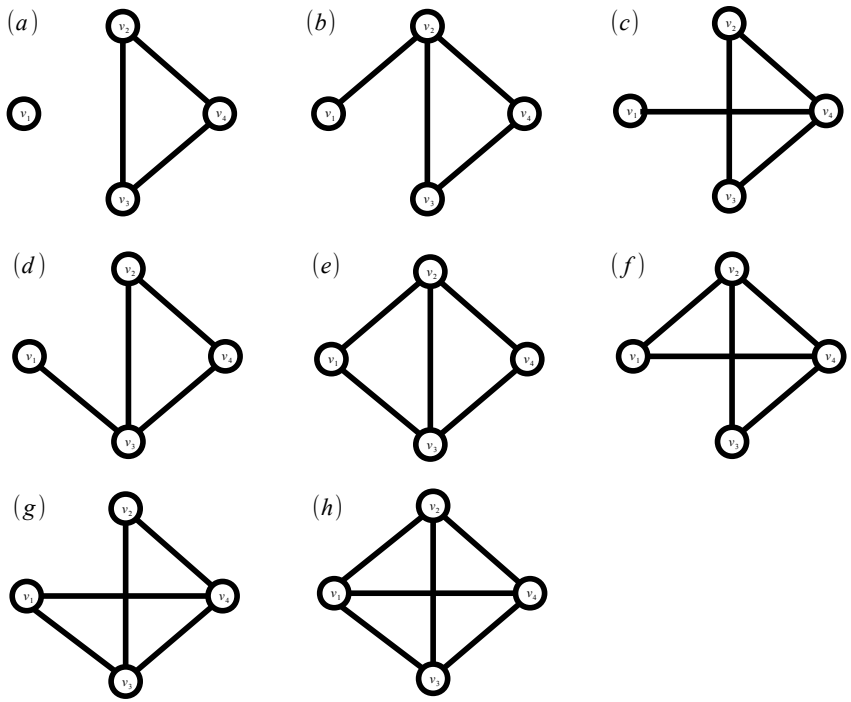
- Consider now a probability space Ω_G of all random graphs $G' = G'(V', E')$ where the set of vertexes remains the same as in G i.e $V' = V$ but each edge (v_i, v_j) $i < j$ is present with probability $p_i = \frac{|d_i|}{n-i}$.
- Having in mind that $\mathbb{P}(G' \in \Omega_G)$ is well defined by probability vector P (or Q) we can write the expected number of vertex covers under Ω_G as

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

- We will use $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ as an approximation to number of leaves in the subtree

Vertex Cover Relaxation example

As an example consider an Ω_G space induced by the previous graph and it's corresponding probability vector $\mathbf{p} = \{\frac{2}{3}, \frac{2}{2}, \frac{1}{1}, 0\}$. All graphs are summarized in the following figure.



It is easy to see that graph (a) is generated with probability $(\frac{1}{3})^3$, $(b), (c), (d)$ with probability $\frac{2}{3}(\frac{1}{3})^2$ $(e), (f), (g)$ with probability $\frac{1}{3}(\frac{2}{3})^2$ and (h) with probability $(\frac{2}{3})^3$. The corresponding number of vertex covers for graphs $(a), (b), \cdots, (h)$ is $8, 7, 7, 7, 6, 6, 6, 5$ so we can compute the expected number of vertex covers that is equal in this case to $\mathbb{E}_{\Omega_G}|\mathscr{X}_{G'}| = (\frac{1}{3})^3 8 + 3\frac{2}{3}(\frac{1}{3})^2 7 + 3\frac{1}{3}(\frac{2}{3})^2 6 + (\frac{2}{3})^3 5 = 6$.

Calculating $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ (1)

Theorem

There exists a deterministic polynomial time Algorithm that calculates $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ *analytically.*

- Let $A(n,k)$ be a probability that exactly $n-k$ vertexes forms a vertex cover in random graph $G' \in \Omega_G$.
- $$\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|] = \sum_{k=0}^{n} \binom{n}{k} A(n,k) \tag{2}$$

Calculating $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ (2)

- Let us look now on some ordered subset $S = \{v_{k,1}, \cdots, v_{k,k} | v_{k,i} \in V\}$ such that $v_{k,1} < v_{k,2}, \cdots < v_{k,k}$ and $|S| = k$ of vertexes that were not chosen to form a vertex cover.
- They are not allowed to have an edge, if they do, this is not a vertex cover
- $$A(n,k) = \frac{\sum_{S \subseteq V, |S|=k} \prod_{i=1}^{k-1} q_{k,i}^{k-i}}{\binom{n}{k}}$$

  where $q_{k,i}$ is the corresponding propagability of $v_{k,i}$ in the induced vector Q.

Calculating $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ (3)

- Suppose $Q = \{q_1, \cdots, q_n\}$
- Define $A(j,k)$ to be a probability that there is a vertex cover of size $j-k$ on the random subgraph $G'_j(V'_j, E'_j)$, $V'_j = \{v_{n-j+1}, \cdots, v_n\}$ and $E'_j = \{(v_l, v_m) | v_l, v_m \in V'_j\}$ for $j \leqslant n$
- Now we add new vertex v_{n-j} and try to calculate $A(j+1,k)$.
  - If we do **use** this vertex in the cover - the number of covers (of size $j+1-k$) in the new subgraph remains $\binom{j}{k} A(j,k)$
  - If we do **not use** this vertex in the cover - the number of covers (of size $j+1-k$) the new subgraph is $q_{n-j}^{k-1} \cdot A(j,k-1)$
- And the following recursive program can be defined:

$$A(j+1,k) = \begin{cases} 0 & k > j+1 \\ \frac{\binom{j}{k} A(j,k) + q_{n-j}^{k-1} \binom{j}{k-1} A(j,k-1)}{\binom{j+1}{k}} & else \end{cases} \quad (3)$$

The starting condition are $A(0,0) = 1$, $A(0,k) = 0 \; \forall k > 0$.

Calculating $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ (4)

The Algorithm for calculating $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$ can be summarized as follows.

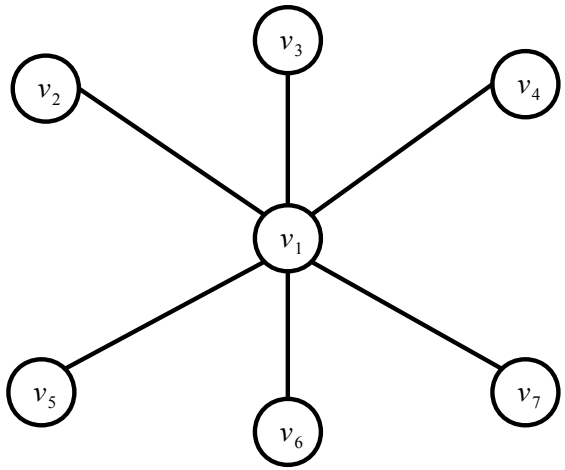
Data: $G = G(V, E)$, $|V| = n$, $|E| = m$
Result: $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$

1 **begin**
2 $\quad Q \longleftarrow$ calculate the probability vector induced by G
3 $\quad \forall k \in \{0, \cdots, n\}$ Calculate $A(n, k)$ using the recursive formula
4 $\quad$ **return** $\mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|] = \sum_{k=0}^{n} \binom{n}{k} A(n, k)$
5 **end**

The overall complexity of the algorithm is $O(n^2)$.

Exact vertex cover computation (1)

- There are some instances for which we can calculate the number of vertex covers exactly
- Consider a *Star Graph*



- If the central vertex participate in the cover, then there is $2^{|V|-1}$ such covers.
- If it is not in the cover, then all the remaining vertices should be there.
- The overall number of vertex covers is $2^{|V|-1}+1$
- If we take the ordered set to be $\{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ the induced probability vector will be $Q = \{0, 1, 1, 1, 1, 1, 1\}$ the dynamic programming will always return $2^{7-1}+1$

Exact vertex cover computation (2)

Lemma

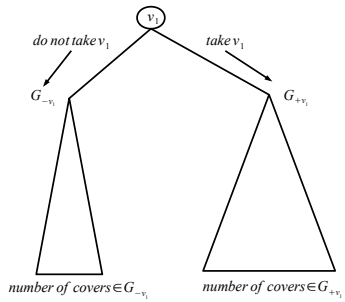
Given that an instance $G = G(V, E)$ *induce a probability vector* $Q = \{q_1, \cdots, q_n\}$ *where each* q_i *satisfies* $q_i \in \{0, 1\}$*, the Dynamic Programming Algorithm provides the exact answer to the number of vertex covers, i.e* $|\mathscr{X}_G| = \mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$.

Proof.

Notice that there is only one graph in Ω_G when G induce a vector Q where $q_i \in \{0, 1\}$, this follows immediately from the construction process of random graph under this particular Ω_G. In other words, we have that $\forall G'(V, E') \in \Omega_G,\ G'(V, E') = G(V, E)$ so $|\mathscr{X}_G| = \mathbb{E}_{\Omega_G}[|\mathscr{X}_{G'}|]$. □

Sequential Procedure

- We will sample some valid vertex cover in a sequential manner
- Set $z \longleftarrow 1$
- Start from v_1 and define 2 subgraphs G_{+v_1} and G_{-v_1}



- Set $p = \frac{\mathbb{E}_{\Omega_{G_{+v_1}}}[|\mathscr{X}_{G'_{+v_1}}|]}{\mathbb{E}_{\Omega_{G_{+v_1}}}[|\mathscr{X}_{G'_{+v_1}}|]+\mathbb{E}_{\Omega_{G_{-v_1}}}[|\mathscr{X}_{G'_{-v_1}}|]}$
- With probability p and $1-p$ select the right or left subtree respectively.
- Multiply Z by p or $1-p$ according to the selection
- Continue to to sample until reaching some leaf
- Deliver $\frac{1}{Z}$

SIS Algorithm for counting

Algorithm 1: SIS Algorithm for counting

Data: $G = G(V, E)$, $|V| = n$, $|E| = m$ and a sample size N

Result: the estimation for number of vertex covers $\widehat{|\mathscr{X}_G|}$

```
1 begin
2 |   Z ⟵ 0
3 |   for i = 1 → N do
4 |   |   Z ⟵ Z + Sequential Procedure
5 |   end
6 |   return Z/N
7 end
```

Probabilistic lower bound

Gomes and Gogate et al.

Theorem

Let $\widehat{Z_1}, \cdots, \widehat{Z_m}$ be the unbiased estimates of Z computed over m independent runs of an importance sampling algorithm. Let $0 < \alpha < 1$ be a constant and let $\beta = \frac{1}{(1-\alpha)^{\frac{1}{m}}}$. Then, the lower bound $Z_{lb} = \frac{1}{\beta} min_{i=1}^{m}(\widehat{Z}_i)$ is a lower bound on Z with probability greater than α.

Model 1

Model 1 A graph with $|V| = 100$ and $|E| = 2,432$. The graph was generated in the following way. We defined the number of vertexes to be 100 and each edge (v_i, v_j) was generated with probability $Ber(p)$ while p is also a random variable such that $p \sim Uni(0, 1)$.

- *cachet* delivers an exact solution of $244,941$ in 0.75 seconds.
- We ran *SampleSearch* for 10 times and it provides an average of $192,251.25$ using 60 seconds time limit.
- The SIS Algorithm $(N = 100)$ delivered 2.440×10^5 in 1.698 seconds. The RE is 1.614×10^{-2}

The following figure provides a typical Histogram of the Importance Weights obtained in a single run of SIS Algorithm.

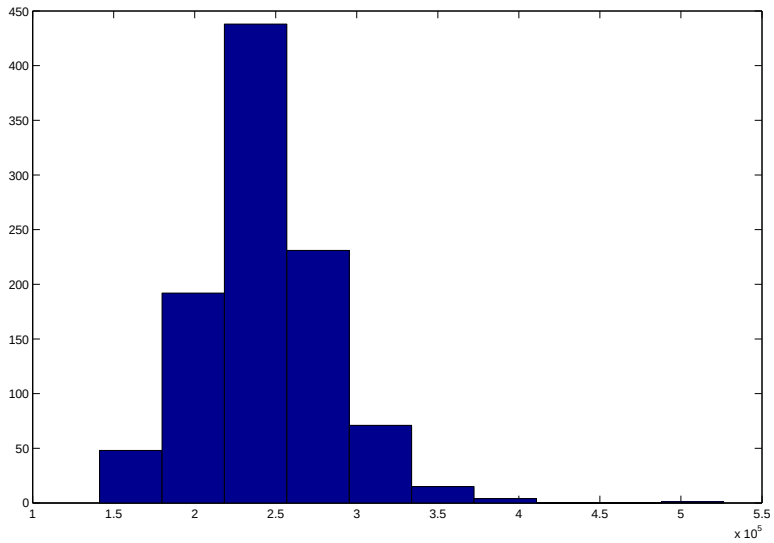


Figure : Histogram of 1,000 Importance Weights for **Model 1**.

Model 2

A graph with $|V| = 300$ and $|E| = 21,094$. The graph was generated in the following way. We defined the number of vertexes to be 300 and each edge (v_i, v_j) was generated with probability $Ber(p_i)$ while p_i is also a random variable such that $p \sim Uni(0, 1)$. The results are self explanatory.

- *cachet* delivers an exact solution of 1.306×10^{14} in about 17 minutes.
- We ran *SampleSearch* for 10 times and it provides an average of 6.001×10^{13} using $1,200$ seconds time limit.
- The SIS Algorithm $(N = 100)$ delivered 1.387×10^{14} in 56.64 seconds. The RE is 4.171×10^{-2}

The following figure provides a typical Histogram of the Importance Weights obtained in a single run of SIS Algorithm.

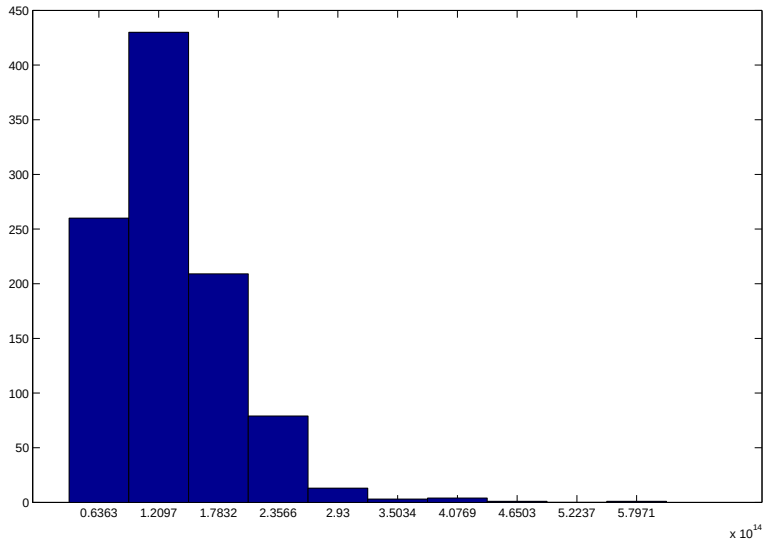


Figure : Histogram of 1,000 Importance Weights for **Model 2**.

Model 3

A graph with $|V| = 1,000$ and $|E| = 64,251$. The graph was generated in the following way. We defined the number of vertexes to be 1,000 and each edge (v_i, v_j) was generated from $Ber(p)$ while each p is generated from truncated Normal distribution with $\mu = 0.1$ and $\sigma = 0.1$ The results are summarized bellow.

- *cachet* was timed out after 2 days and was unable to deliver a solution. The lower bound of $3.439E + 09$ was supplied.
- *SampleSearch* failed to initialize, probably, because the problem is too big.
- The SIS Algorithm $(N = 100)$ delivered 4.261×10^{32} in 648.6 seconds. The RE is 4.813×10^{-2}

The following figure provides a typical Histogram of the Importance Weights obtained in a single run of SIS Algorithm.

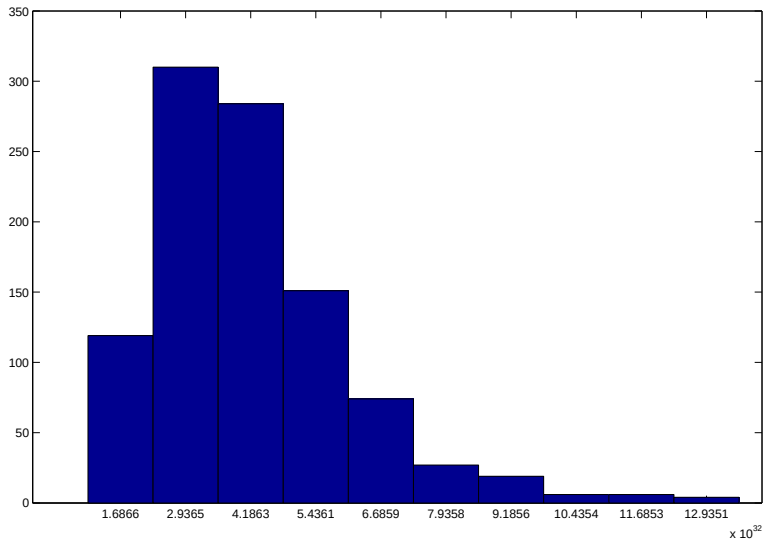


Figure : Histogram of 1,000 Importance Weights for **Model 3**.

Model 4

A graph with $|V| = 1,000$ and $|E| = 249,870$. The graph was generated in the following way. We defined the number of vertexes to be 1,000 and each edge (v_i, v_j) was generated from $Ber(p)$ while each p is generated from truncated Normal distribution with $\mu = 0.5$ and $\sigma = 0.3$ The results are summarized bellow.

- *cachet* was timed out after 2 days and was unable to deliver a solution. The lower bound of $9.601E + 10$ was supplied.
- *SampleSearch* failed to initialize, probably, because the problem is too big.
- The SIS Algorithm $(N = 100)$ delivered 2.773×10^{11} in 1,718 seconds. The RE is 1.579×10^{-2}

The following figure provides a typical Histogram of the Importance Weights obtained in a single run of SIS Algorithm.

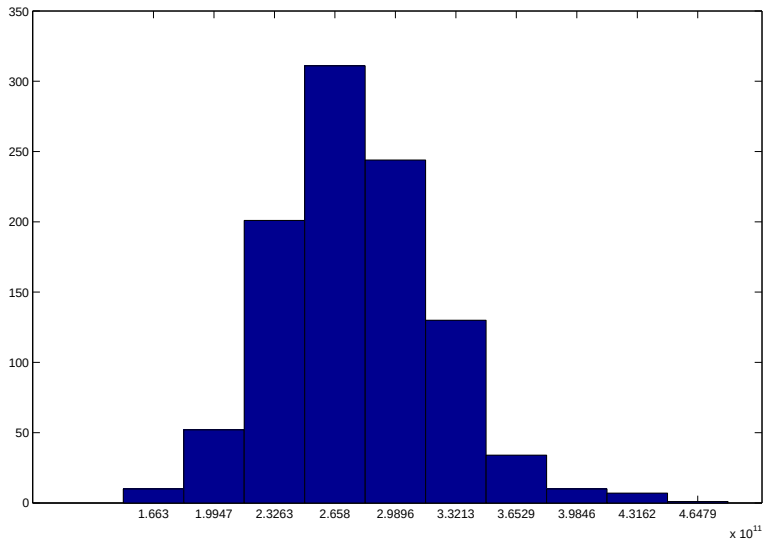
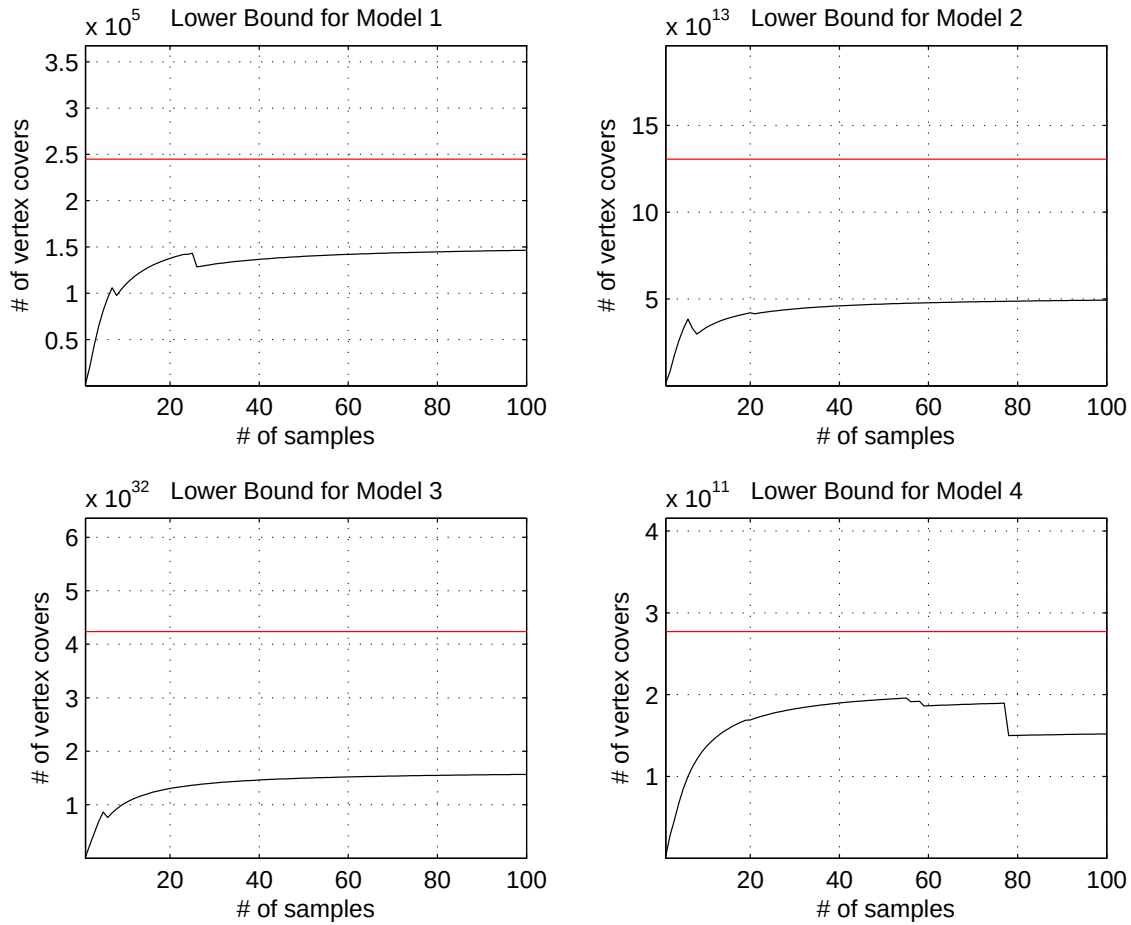


Figure : Histogram of 1,000 Importance Weights for **Model 4**.

Probabilistic lower bounds for the models

For all models we took $\alpha = 0.99$ and $m \in \{1, \cdots, 100\}$. The results are summarized in the following figure.



Non random model - Hypercubes H_4, H_5, H_6 and H_7

- *cachet* delivers 743 254,475 and 1.976×10^{10}, but fails for H_7
- We would like to keeping the RE below 3% so we take the sample sizes to be $50, 250, 1,500$ and $10,000$

Table : Average performance of 10 runs of the *SIS* algorithm for Hypercube graphs.

Instance	$\widehat{\lvert\mathscr{X}_G\rvert}$	$\widehat{RE}$	*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

What next?

Prove the efficiency of this estimator for some class of graphs.

Thank You!
Questions please