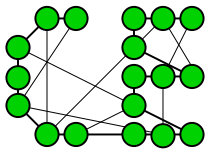


# Stochastic Enumeration Method for Counting NP-hard Problems

*Technion*

Reuven Rubinfeld and Radislav Vaisman

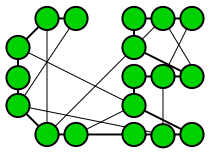
Faculty of Industrial Engineering and Management,  
Technion, Israel



# Contents

1. Sequential Importance Sampling (SIS) method
2. One Step Look-Ahead (OSLA) method for Self-Avoiding Walks (SAW).
3. Extensions of OSLA:  $n$ -Step-Look-Ahead and SE-OSLA Methods.
4. Stochastic Enumeration (SE) Method.
5. Applications of Stochastic Enumeration to
  - Satisfiability,
  - Counting the Number of Paths in a Network
  - Counting the Number of Perfect Matchings
6. Convergence and Numerical Results.

# Sequential Importance Sampling (SIS) Method



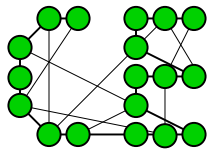
Sequential importance sampling (**SIS**) is importance sampling carried out in a sequential manner. To explain, consider the expected performance

$$\ell = \mathbb{E}_f[S(\mathbf{X})] = \int S(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} , \quad (1)$$

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

Let  $g$  be another probability density such that  $H f$  is *dominated* by  $g$ . That is,  $g(\mathbf{x}) = 0 \Rightarrow S(\mathbf{x}) f(\mathbf{x}) = 0$ . We have

$$\ell = \int S(\mathbf{x}) \frac{f(\mathbf{x})}{g(\mathbf{x})} g(\mathbf{x}) d\mathbf{x} = \mathbb{E}_g \left[ S(\mathbf{X}) \frac{f(\mathbf{X})}{g(\mathbf{X})} \right] . \quad (2)$$



## SIS Method

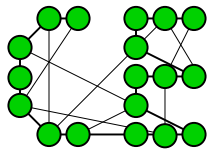
Consequently, if  $\mathbf{X}_1, \dots, \mathbf{X}_N$  is a *random sample* from  $g$ , that is,  $\mathbf{X}_1, \dots, \mathbf{X}_N$  are iid random vectors with density  $g$ , then

$$\hat{\ell} = \frac{1}{N} \sum_{k=1}^N S(\mathbf{X}_k) \frac{f(\mathbf{X}_k)}{g(\mathbf{X}_k)} \quad (3)$$

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

$$W(\mathbf{x}) = \frac{f(\mathbf{x})}{g(\mathbf{x})}, \quad (4)$$

is called the *likelihood ratio*.

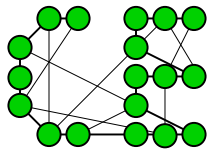


## SIS Method

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

$$g(\mathbf{x}) = g_1(x_1) g_2(x_2 | x_1) \cdots g_n(x_n | x_1, \dots, x_{n-1}). \quad (5)$$

To further simplify the notation, we abbreviate  $(x_1, \dots, x_t)$  to  $\mathbf{x}_{1:t}$  for all  $t$ . In particular,  $\mathbf{x}_{1:n} = \mathbf{x}$ .



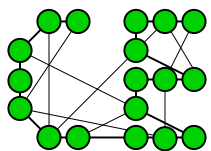
## SIS Method

By the product rule of probability, the target pdf  $f(\mathbf{x})$  can also be written sequentially, that is,

$$f(\mathbf{x}) = f(x_1) f(x_2 | x_1) \cdots f(x_n | \mathbf{x}_{1:n-1}). \quad (6)$$

We can write the likelihood ratio in product form as

$$W(\mathbf{x}) = \frac{f(x_1) f(x_2 | x_1) \cdots f(x_n | \mathbf{x}_{1:n-1})}{g_1(x_1) g_2(x_2 | x_1) \cdots g_n(x_n | \mathbf{x}_{1:n-1})} \quad (7)$$



## SIS Method

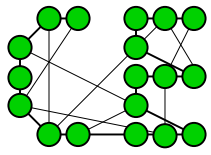
If  $\omega_t(\mathbf{x}_{1:t})$  denotes the likelihood ratio up to time  $t$ , recursively as

$$W_t(\mathbf{x}_{1:t}) = u_t W_{t-1}(\mathbf{x}_{1:t-1}), \quad t = 1, \dots, n, \quad (8)$$

with initial weight  $W_0(\mathbf{x}_{1:0}) = 1$  and *incremental weights*

$u_1 = f(x_1)/g_1(x_1)$  and

$$u_t = \frac{f(x_t | \mathbf{x}_{1:t-1})}{g_t(x_t | \mathbf{x}_{1:t-1})} = \frac{f(\mathbf{x}_{1:t})}{f(\mathbf{x}_{1:t-1}) g_t(x_t | \mathbf{x}_{1:t-1})}, \quad t = 2, \dots, n. \quad (9)$$



# SIS Method

The final estimator is

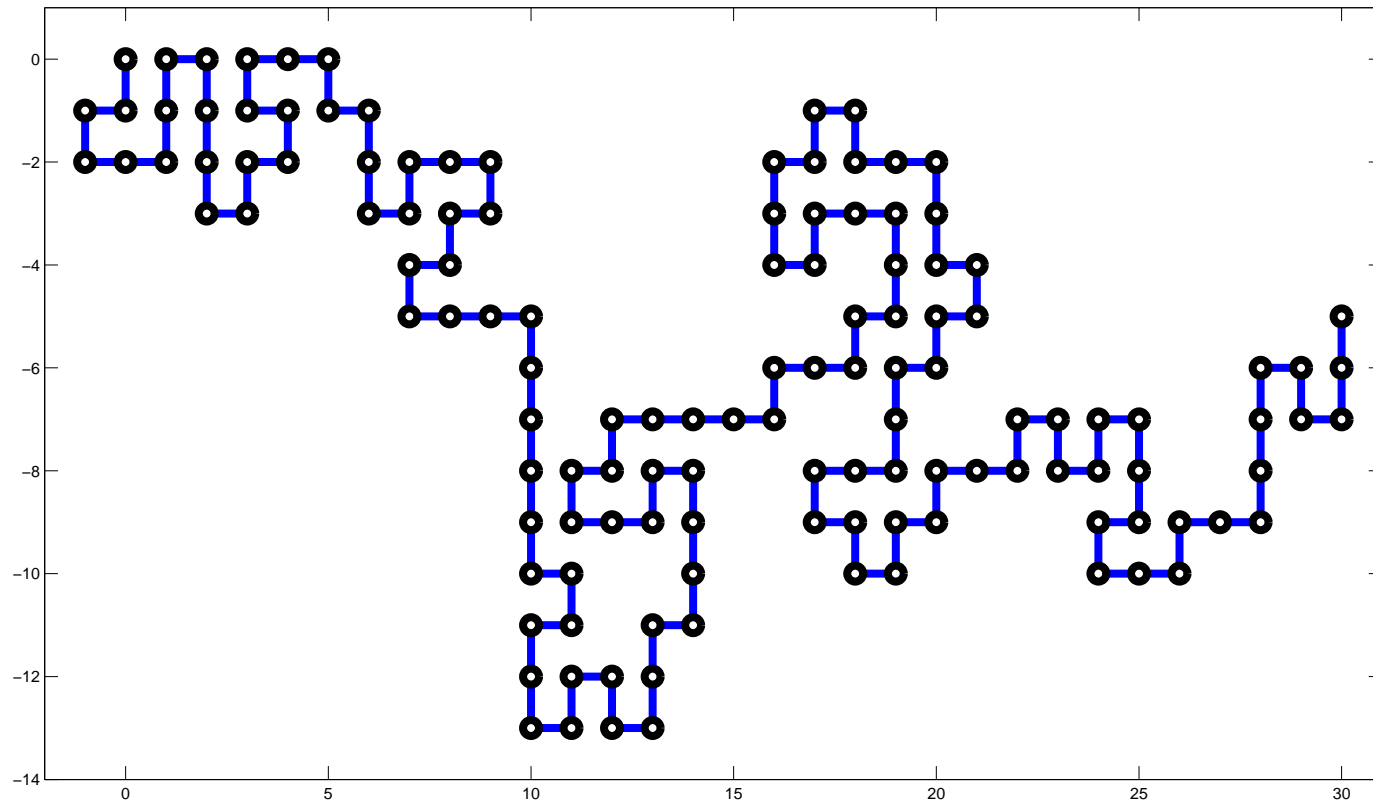
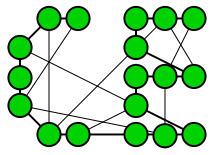
$$\hat{\ell}_w = \frac{\sum_{k=1}^N S(\mathbf{X}_k) w_k}{\sum_{k=1}^N w_k}. \quad (10)$$

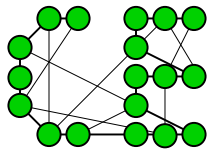
## Algorithm 0.1 (SIS Algorithm)

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

# Self-Avoiding Walk of Length

$$n = 130$$





## One-Step-Look-Ahead (OSLA) Procedure

OSLA is the state of the art procedure due to Rosenbluth and Rosenbluth (1959).

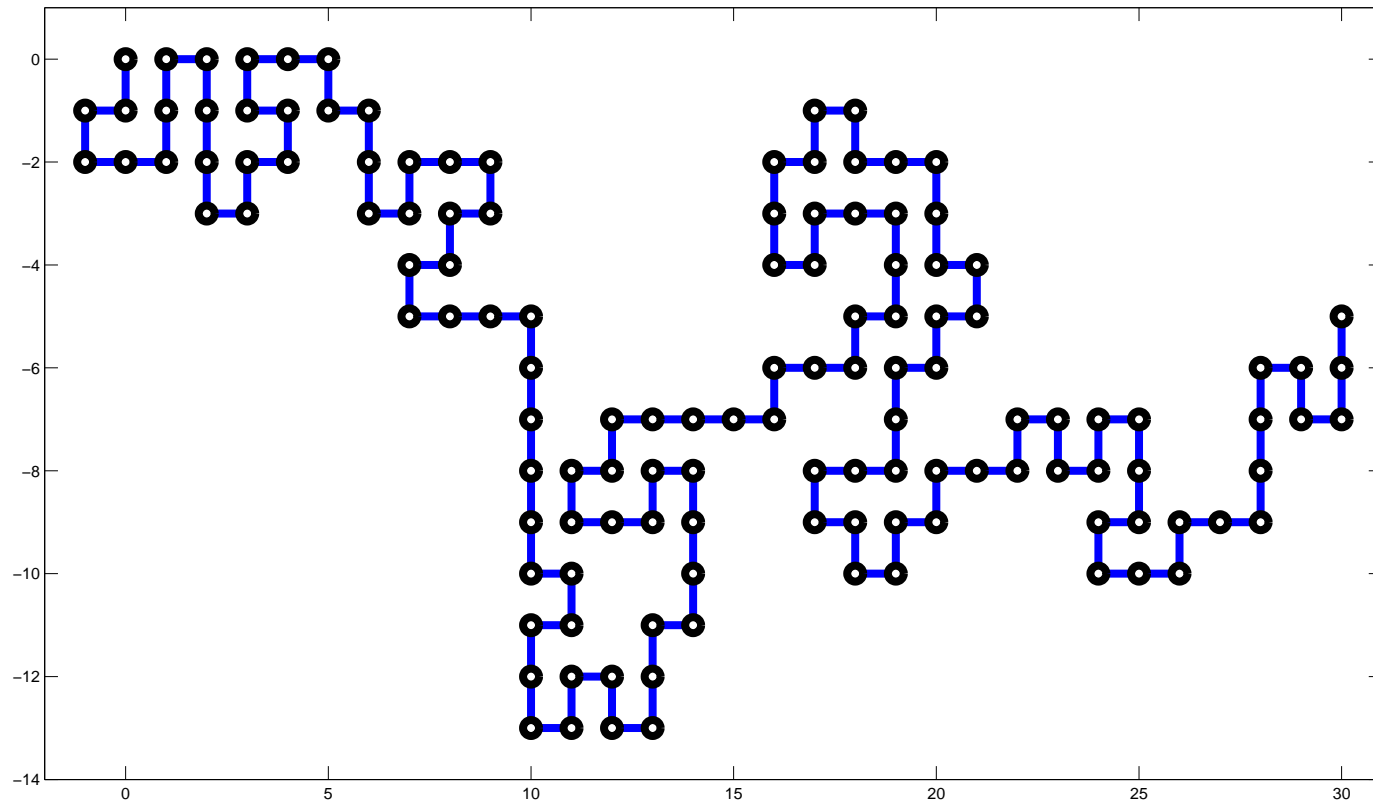
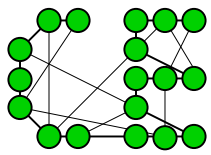
1. Start from  $X_0 = (0, 0)$ . Let  $d_t$  be the number of neighbors of  $X_{t-1}$  that have not yet been visited. If  $d_t > 0$ , choose  $X_t$  with probability  $1/d_t$  from its neighbors. If  $d_t = 0$  stop generating the path.
2. Stop if  $t = n$ . Otherwise increase  $t$  by 1 and go to step 1.

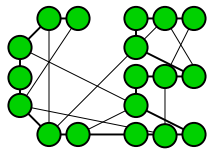
Note that the procedure either generates a SAW  $\mathbf{x}$  of fixed length  $n$  or the path gets value zero. The product rule pdf  $g(\mathbf{x})$  is

$$g(\mathbf{x}) = \frac{1}{d_1} \frac{1}{d_2} \cdots \frac{1}{d_n} = \frac{1}{w(\mathbf{x})}, \quad (w(\mathbf{x}) = d_1 \cdots d_n). \quad (11)$$

# Self-Avoiding Walk of Length

$$n = 130$$

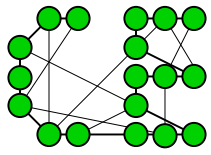




# OSLA Algorithm for SAW

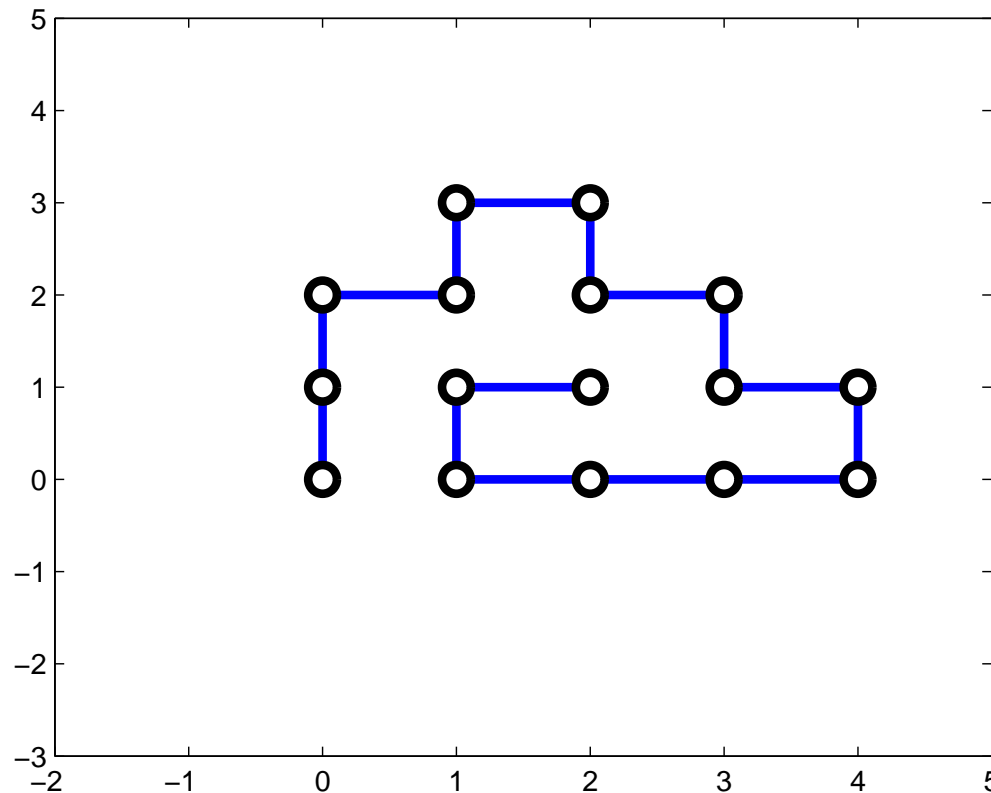
1. Generate independently  $M$  paths  $\mathbf{X}_1, \dots, \mathbf{X}_M$  via the OSLA procedure.
2. For each SAW  $\mathbf{X}_k$  compute the corresponding  $w(\mathbf{X}_k)$  as above. For the other parts (which do not reach the value  $n$ ) set  $w(\mathbf{X}_k) = 0$ .
3. Return

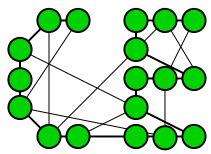
$$|\widehat{\mathcal{X}^*}| = \frac{1}{M} \sum_{i=1}^M w(\mathbf{X}_i) . \quad (12)$$



# OSLA Loses Trajectories

The drawback of OSLA is that it loses most of its trajectories even for moderate  $n$ , say  $n = 100$ . Figure below present a SAW trapped after 15 iterations.





## OSLA Loses Trajectories

To provide more insight on the limitations of OSLA consider:

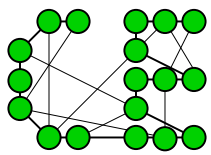
(i) The set  $\mathcal{X}^*$  defined by the following product of clauses.

$$(x_1 + \bar{x}_2)(\bar{x}_1 + \bar{x}_2)(x_1 + x_3)(x_2 + \bar{x}_3),$$

where  $\bar{x} = 1 - x$ . It is not difficult to check that OSLA fails to find the true value  $|\mathcal{X}^*| = 1$  with probability  $7/8$ .

(ii) Consider a 3-SAT model with an adjacency matrix

$\mathbf{A} = (20 \times 80)$  and  $|\mathcal{X}^*| = 15$ . Running the OSLA algorithm we found that it finds all 15 valid assignments (satisfies all 80 clauses) only with probability  $\approx 10^{-4}$ . In addition we found that as the size of the models increases the percentage of valid assignments goes fast to zero.

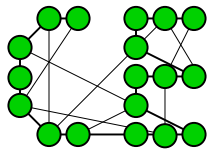


# Extension of OSLA: *n*-step-look-ahead Strategy

We next extend OSLA to  $k$ -step-look ahead and in particular to  $n$ -step-look ahead, called *nSLA*. Here  $n$  denotes the size of the problem, such as the number of variables (literals) in SAT and the number of edges in a network. We assume that all  $n$  variables  $x_1, \dots, x_n$  are binary, that is  $x \in \{0, 1\}$ .

The  $n$ -SLA (based an oracle) is very similar to OSLA.

**Its major advantage versus OSLA: it never loses a trajectory.**



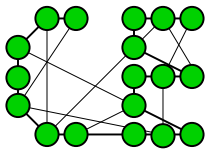
# Extension of OSLA: the $n$ SLA Method

Our main strategy (slogan) is as follows:

**Use fast polynomial decision making oracles to solve #P-sharp problems.**

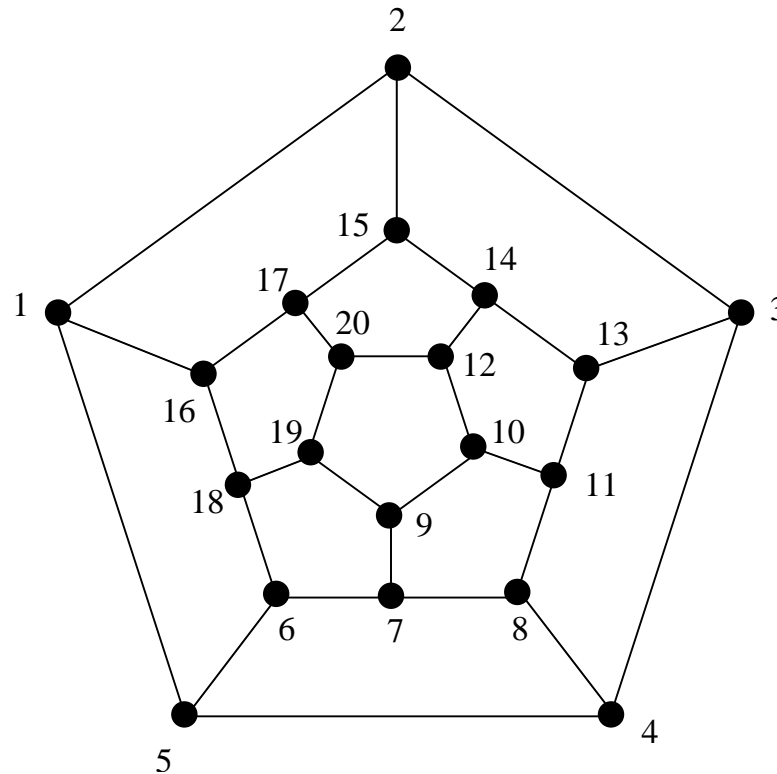
In particular we use

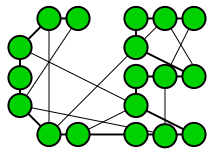
- Dijkstra's algorithm for solving a shortest path problem.
- Hungarian method for solving an assignment problem in polynomial time.
- DPLL decision making algorithm for counting the number of valid assignments in 2-SAT.



## Counting the Number of Paths in a Network

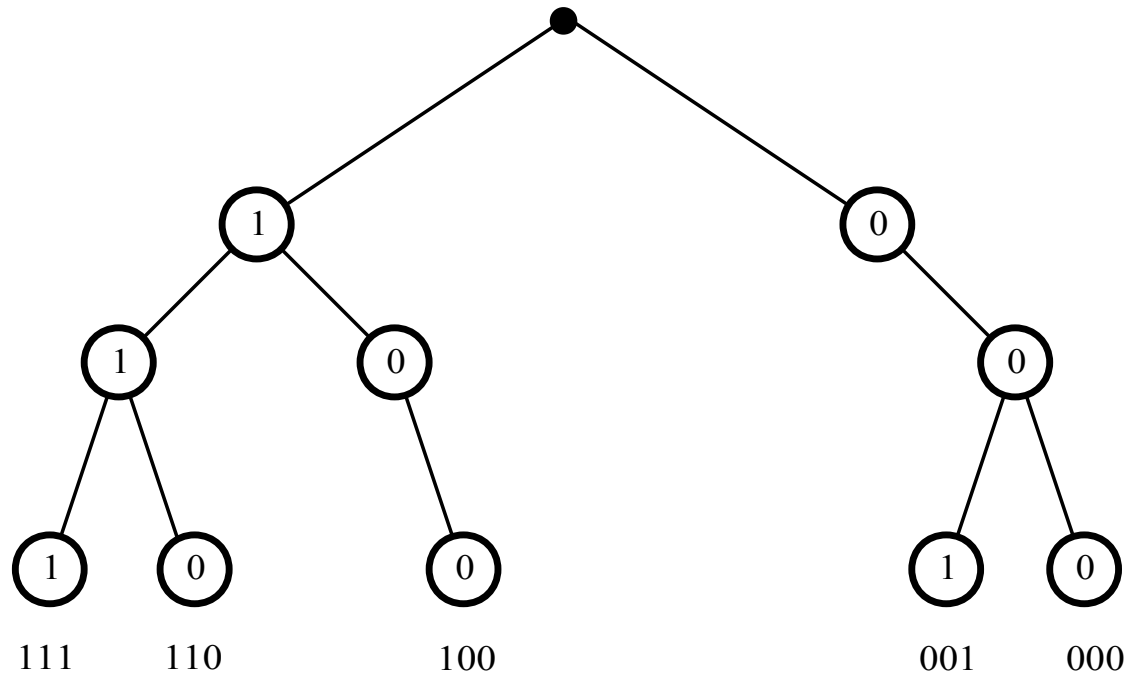
The goal is to count the number of paths  $|\mathcal{X}^*|$  in a dodecahedron graph say from node 1 to node 20.



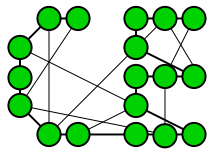


## A Tree

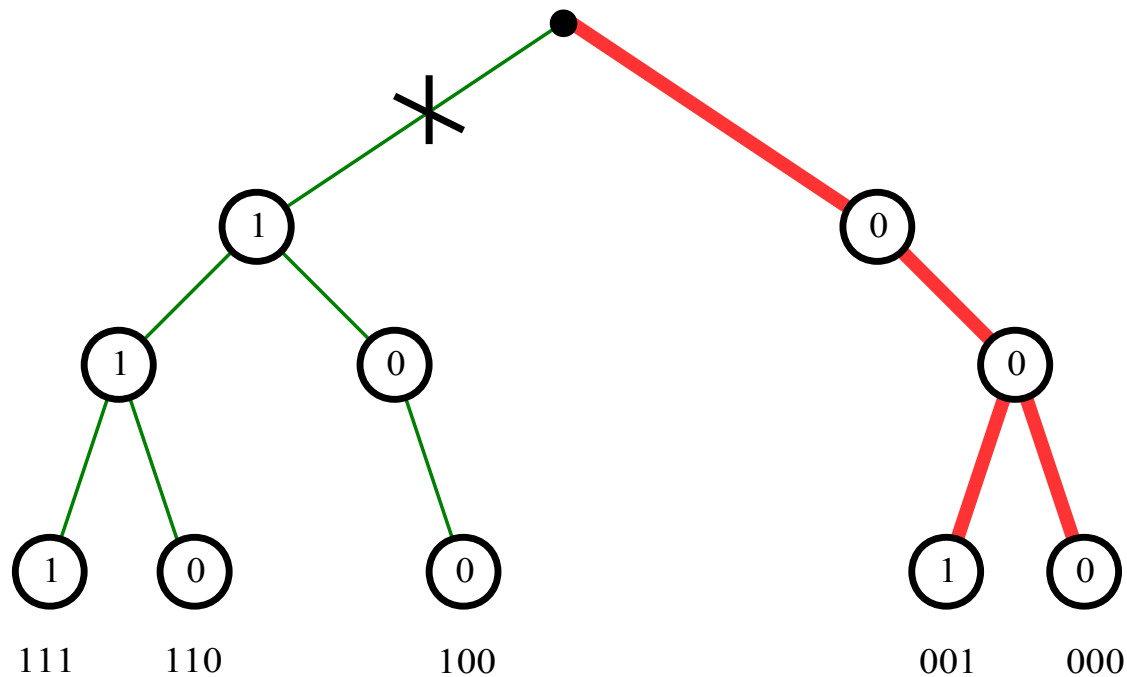
A tree corresponding to the set  $\{000, 001, 100, 110, 111\}$ .

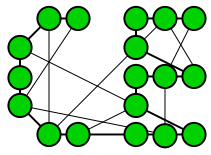


# Extension of OSLA: the $n$ SLA Method



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





## Extension of $n$ SLA: the SE Method

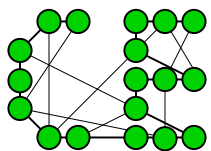
As mentioned the major advantage of  $n$ SLA versus OSLA - it never loses trajectories. Its main drawback is that the generated trajectories are not uniformly distributed. As a result its estimators are heavily biased. To overcome this difficulty we modify  $n$ SLA as:

**Instead of a single trajectory we run in parallel multiple ones.**

This will improve dramatically the non-uniformity issue.

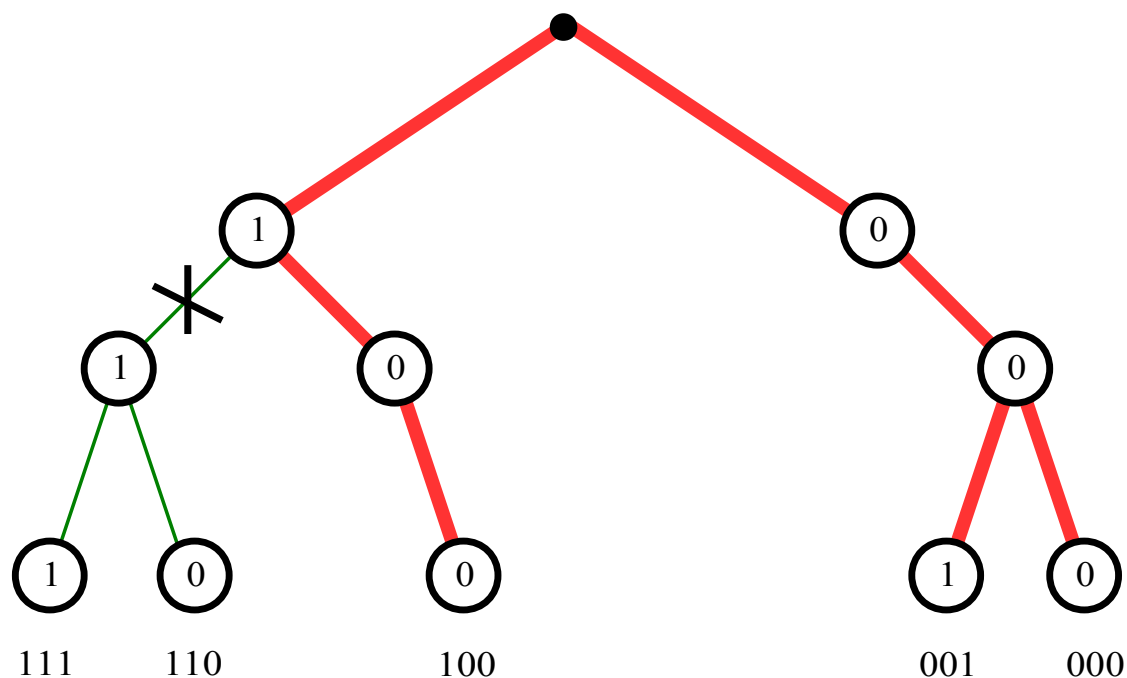
Our strategy is similar to the one proposed by Albert Einstein:

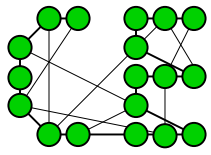
*Everything should be made as simple as possible, but not simpler*



## The SE Method

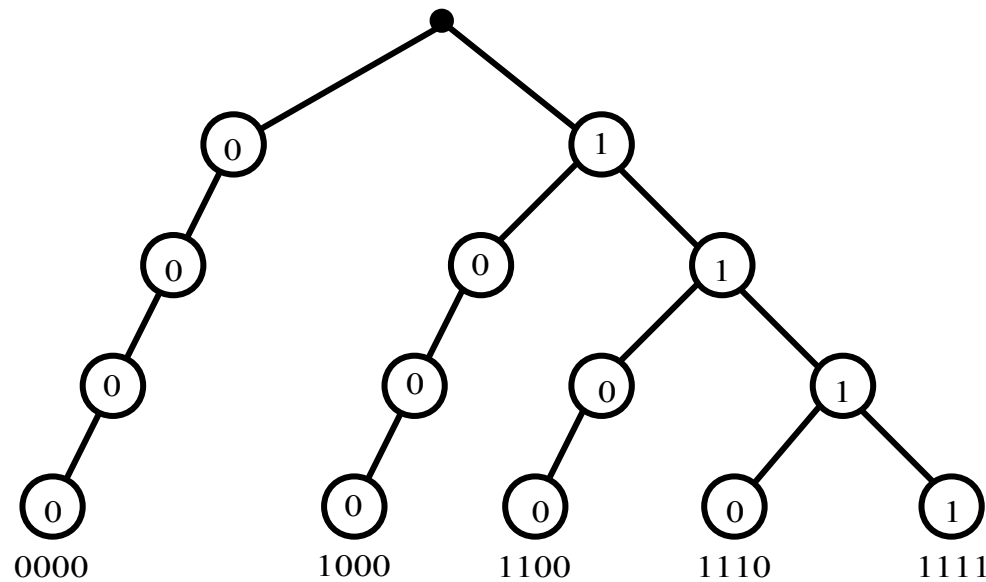
SE in action. The sub-trees  $\{100, 000, 001\}$  (in bold) of the original tree generated with  $N^{(e)} = 2$ .

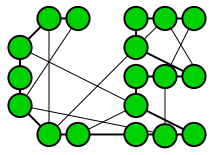




## SE Method

To see how SE improves  $n$ SLA consider a 2-SAT model with clauses  $C_1 \wedge C_2 \wedge \dots \wedge C_n$ , where  $C_i = x_i \vee \bar{x}_{i+1} \geq 1$ . Figure below presents a graph with  $n = 4$  variables and  $|\mathcal{X}^*| = 5$ .

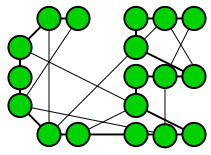




## Extension of $n$ SLA: the SE Method

The table below corresponds to the above figure for  $n = 99$  and  $|\mathcal{X}^*| = 100$ . It shows how bad SE works for  $N^{(e)} = 1$ , (which is  $n$ SLA) and how SE improves for  $N^{(e)} > 1$ . Here  $N^{(e)}$  denotes the number of multiple trajectories and RE-relative error.

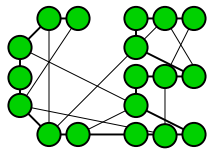
$(N^{(e)}, M)$	$ \widetilde{\mathcal{X}}^* $	$RE$
$(N^{(e)} = 1, M = 500)$	11.110	0.296
$(N^{(e)} = 10, M = 50)$	69.854	0.175
$(N^{(e)} = 50, M = 10)$	100.11	0.032



## Numerical Results for SAT

Performance of SE Algorithm for the 3-SAT  $75 \times 325$  model  
with  $N_t^{(e)} = 20$  and  $M = 100$

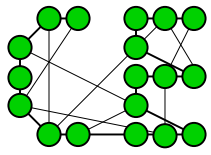
Run $N_0$	Iterations	$ \widetilde{\mathcal{X}}^* $	RE of $ \widetilde{\mathcal{X}}^* $	CPU
1	75	2359.780	0.045	2.74
2	75	2389.660	0.058	2.77
3	75	2082.430	0.078	2.79
4	75	2157.850	0.044	2.85
5	75	2338.100	0.035	2.88
Average	75	2247.077	0.040	2.83



## Numerical Results for SAT

Performance of SE for SAT  $300 \times 1080$  model with  $N_t^{(e)} = 300$ ,  $M = 300$  and  $r = 1$  with exact solution  $|\mathcal{X}^*| = 3.297E + 24$ .

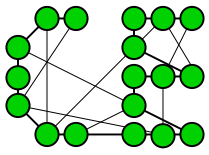
Run $N_0$	Iterations	$ \tilde{\mathcal{X}}^* $	RE of $ \tilde{\mathcal{X}}^* $	CPU
1	300	3.30E+24	2.61E-03	2010.6
2	300	3.46E+24	5.10E-02	2271.8
3	300	3.40E+24	3.22E-02	2036.8
4	300	3.42E+24	4.00E-02	2275.8
5	300	3.39E+24	2.83E-02	2022.4
Average	300	3.36E+24	2.21E-02	2134.1



## Numerical Results for SAT

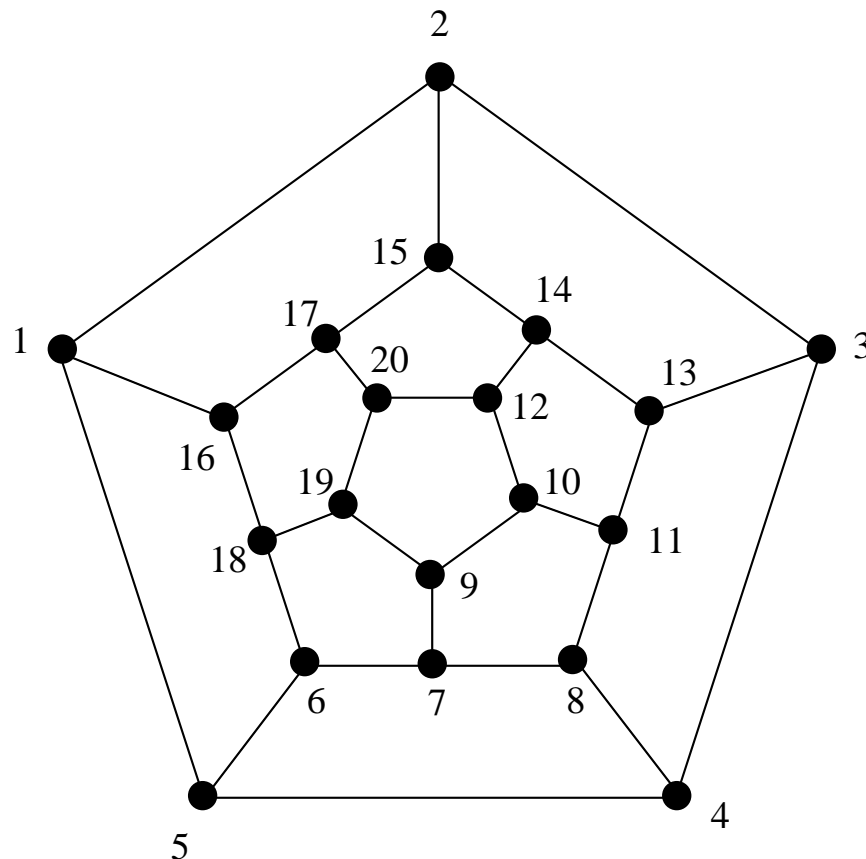
Comparison of the efficiencies of SE and standard splitting. It follows that SE is about 50 times faster than splitting.

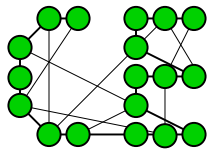
Instance	Time	SE	SE RE	Split	Split RE
20x80	1 sec	15.0158	5.51E-03	14.97	3.96E-02
75x325	137 sec	2248.8	9.31E-03	2264.3	6.55E-02
75x270	122 sec	1.34E+06	1.49E-02	1.37E+06	3.68E-02
300x1080	1600 sec	3.32E+24	3.17E-02	3.27E+24	2.39E-01



## Counting the Number of Paths in a Network

The goal is to count the number of paths  $|\mathcal{X}^*|$  in a dodecahedron graph from node 1 to node 20. Using full enumeration, we obtained  $|\mathcal{X}^*| = 1338$ .



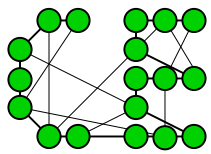


## Counting the Number of Paths in a Network

Performance of the SE Algorithm for the dodecahedron graph with  $N_t^{(e)} = 5$  and  $M = 20$ . Based on 100 runs, we found that  $RE = 0.0121$ .

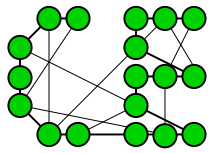
Run $N_0$	Iterations	$ \tilde{\mathcal{X}}^* $	CPU
1	15	1567.3	3.467
2	17	1644.8	3.252
3	15	1220.3	2.956
4	15	1364.4	2.992
5	17	1567.4	3.134

# Counting the Number of Perfect Matchings (Permanent)



Consider the adjacency matrix  $A$  with  $|V| = 20$ ,  $|E| = 78$  and the number of perfect matchings (permanent)  $|\mathcal{X}^*| = 255,112$ , obtained using full enumeration.

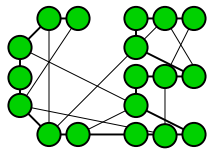
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad (13)$$



# Counting the Number of Perfect Matchings (Permanent)

Performance of the SE Algorithm for the matrix  $A$ . The relative error is near 0.0275.

Run $N_0$	Iterations	$ \tilde{\mathcal{X}}^* $	CPU
1	10	2.59E+05	1.911
2	10	2.48E+05	1.882
3	10	2.67E+05	1.889
4	10	2.44E+05	1.887
5	10	2.53E+05	1.889



**Thank You**

We hope that following Albert Einstein's suggestion we made everything as simple as possible, but not simpler.

*Thank You !!!*