Stochastic Enumeration Method for Counting NP-hard Problems

Technion

Reuven Rubinstein 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

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

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

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

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

SIS Method

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

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

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

$$W(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{g(\boldsymbol{x})}\,, \tag{4}$$

is called the *likelihood ratio*.

SIS Method

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

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

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

SIS Method

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

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

We can write the likelihood ratio in product form as

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

SIS Method

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

$$w_t(\boldsymbol{x}_{1:t}) = u_t\, w_{t-1}(\boldsymbol{x}_{1:t-1}), \quad t = 1, \ldots, n\,, \tag{8}$$

with initial weight $w_0(\boldsymbol{x}_{1:0}) = 1$ and *incremental weights* $u_1 = f(x_1)/g_1(x_1)$ and

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

SIS Method

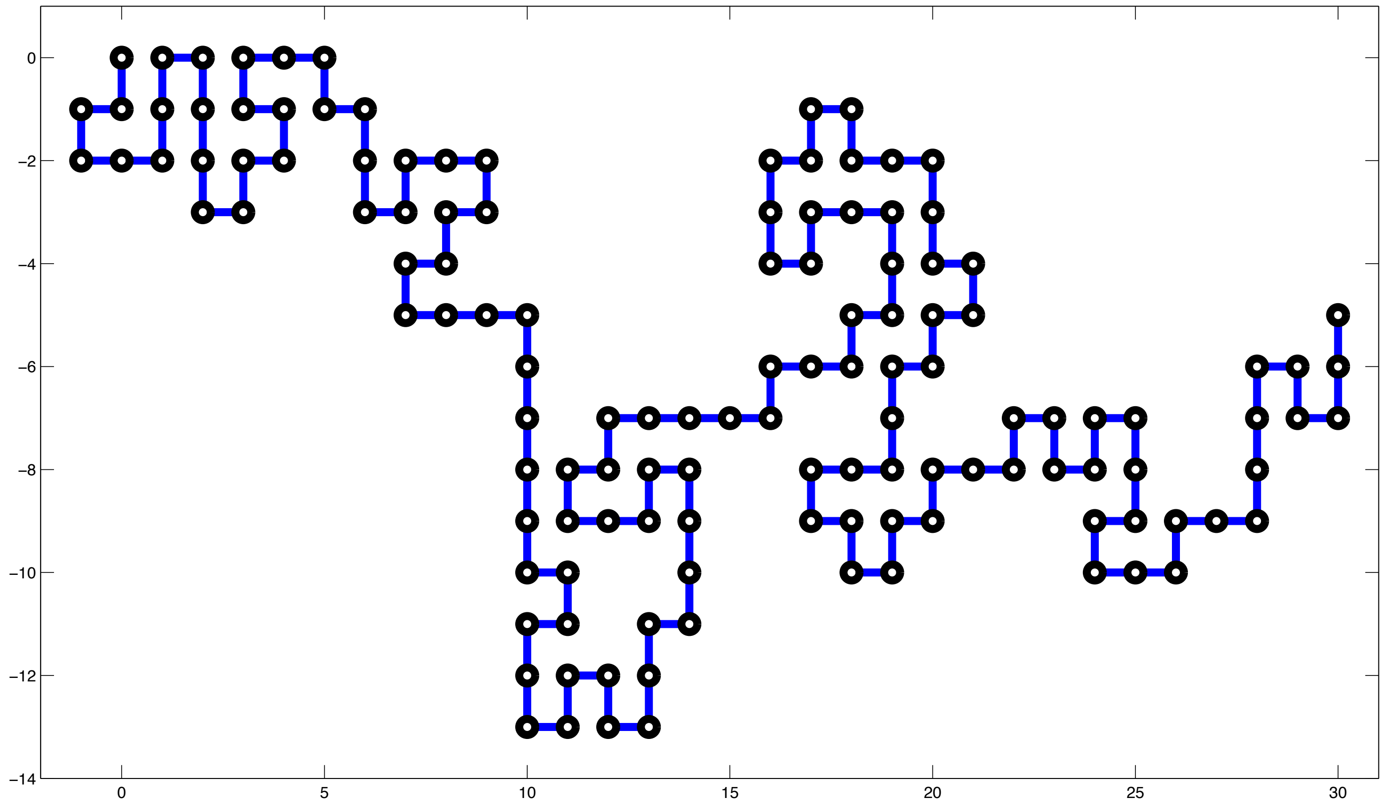
The final estimator is

$$\hat{\ell}_w = \frac{\sum_{k=1}^{N} S(\boldsymbol{X}_k)\, W_k}{\sum_{k=1}^{N} W_k} \; . \qquad (10)$$

Algorithm 0.1 (SIS Algorithm)

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

Self-Avoiding Walk of Length $n = 130$



One-Step-Look-Ahead (OSLA) Procedure

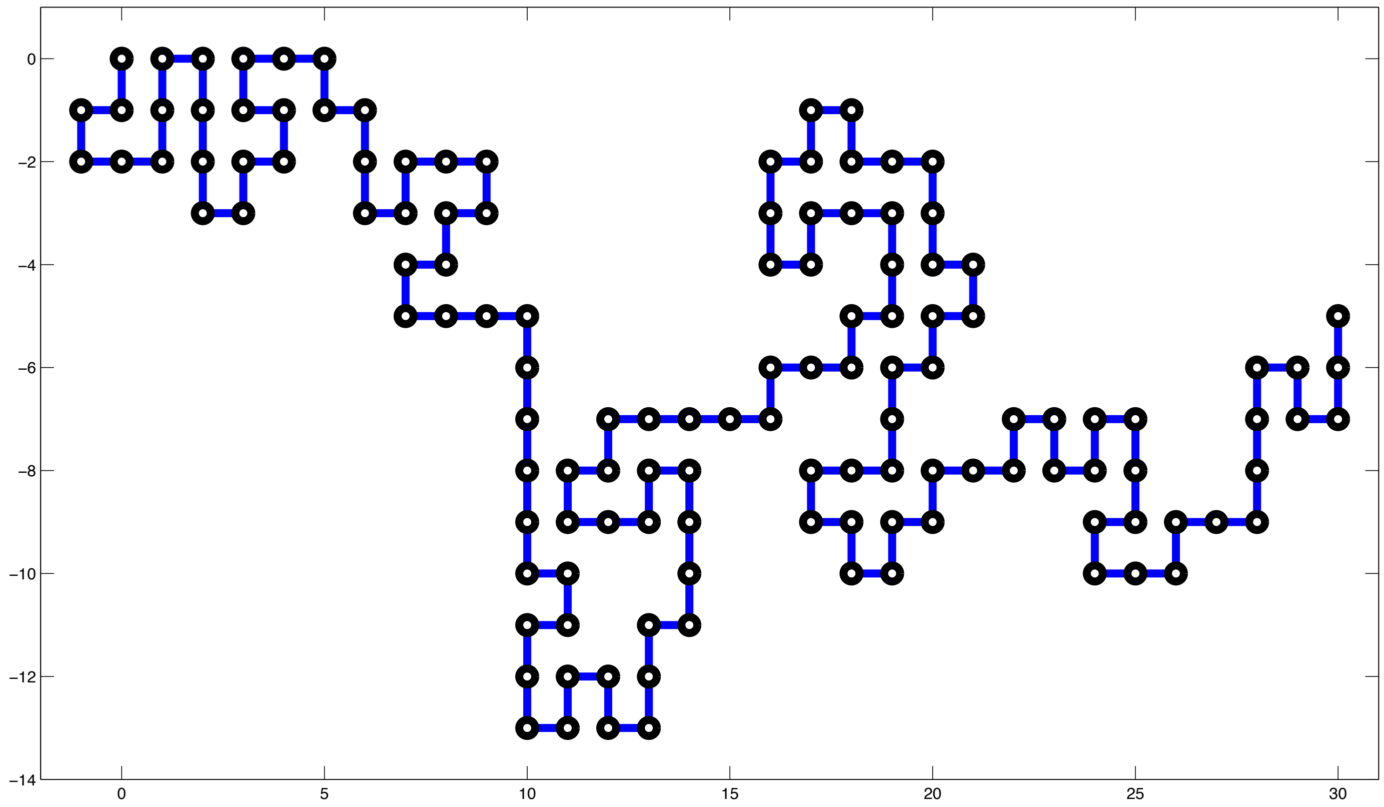
OSLA is the state of the art procedure due to Rosenbluth and Rosenbluth (1959).

1. Start from $X_0 = (0,0)$. Set $t = 1$. 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 2.

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

$$g(\boldsymbol{x}) = \frac{1}{d_1}\frac{1}{d_2}\cdots\frac{1}{d_n} = \frac{1}{|\widehat{\mathscr{X}^*}|}, \ (|\widehat{\mathscr{X}^*}| = d_1 \ldots d_n) \ . \qquad (11)$$

Self-Avoiding Walk of Length $n = 130$



OSLA Algorithm for SAW

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

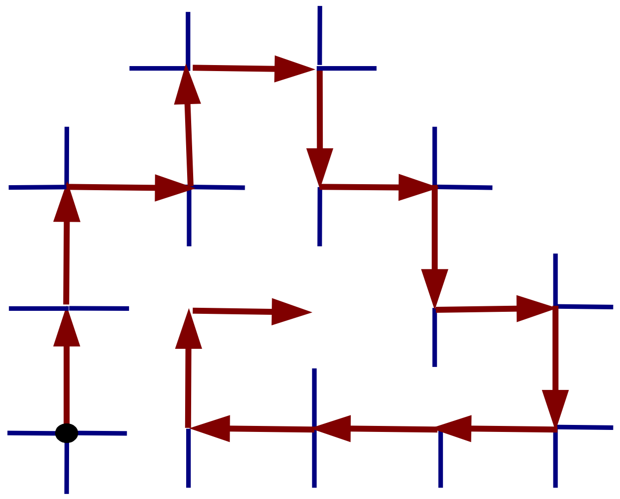
$$\widetilde{|\mathscr{X}^*|} = \frac{1}{M} \sum_{i=k}^{M} \widehat{|\mathscr{X}^*|} \ . \tag{12}$$

OSLA Looses Trajectories

A SAW (with arrows) trapped after 15 iterations. The corresponding ν values (based on short lines without arrows) are

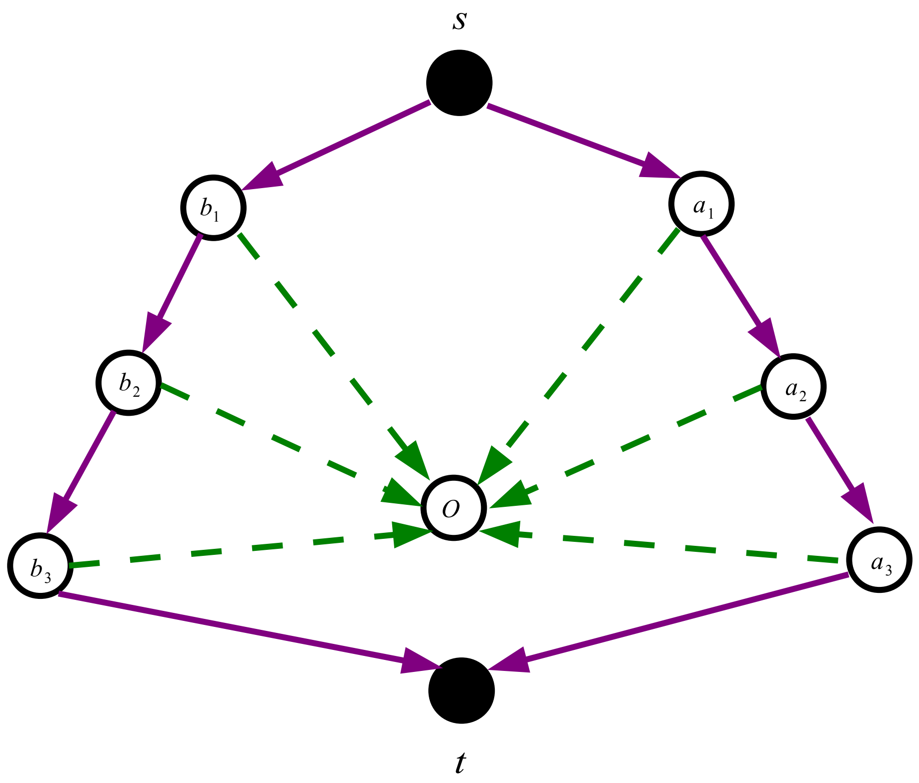
$$\nu_1 = 4,\ \nu_2 = 3,\ \nu_3 = 3,\ \nu_4 = 3,\ \nu_5 = 3,\ \nu_6 = 3,\ \nu_7 = 2,\ \nu_8 = 3,$$

$$\nu_9 = 3,\ \nu_{10} = 3,\ \nu_{11} = 2,\ \nu_{12} = 3,\ \nu_{13} = 2,\ \nu_{14} = 1,\ \nu_{15} = 0.$$



OSLA Looses Trajectories

As for another situation where OSLA can be readily trapped consider a directed graph below with source s and sink t.



OSLA Looses Trajectories

The exception is the OSLA algorithm of Rasmussen for counting the permanent. Rasmussen proofs that if the a_{ij} entries of the permanent matrix $\boldsymbol{A}$ are Bernoulli outcomes each generated randomly with probability $p = 1/2$ then OSLA estimator is FPRAS.

This is quite a remarkable result!

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 $x_1, \ldots, x_n$ 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 looses a trajectory.

Extension of OSLA: the nSLA Method

Our main strategy (slogan) is as follows:

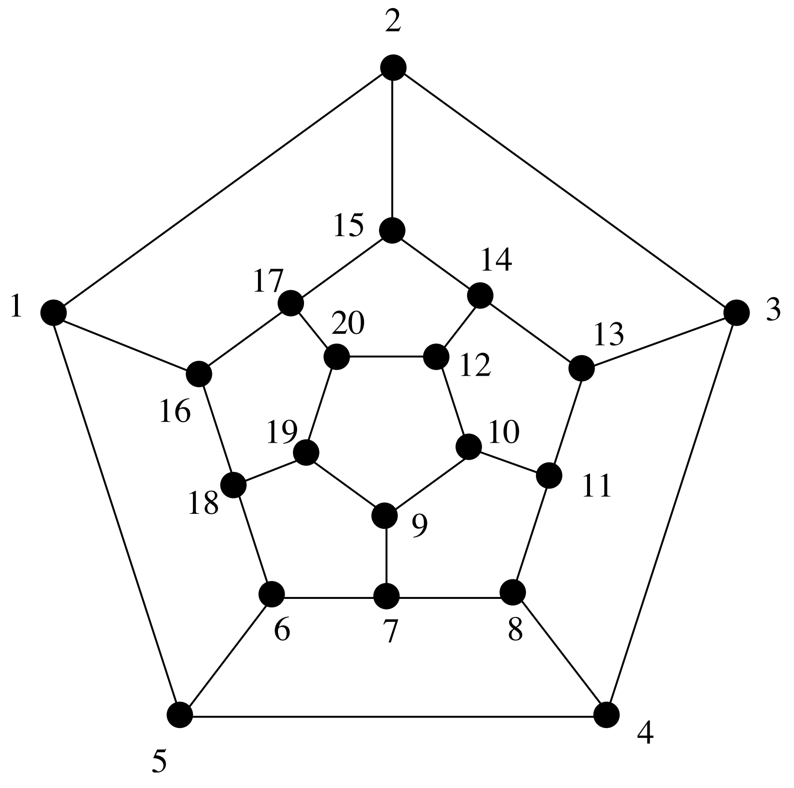
Use fast polynomial decision making oracles to solve #P-sharp problems.

In particular use

- Breadth first search (BFS) or Dijkstra's decision making algorithms for counting the number of paths in a network.
- Hungarian decision making algorithm for counting the number of perfect matchings (permanent) in a bipartite graph.
- DPLL decision making algorithm for counting the number of valid assignments in 2-SAT.

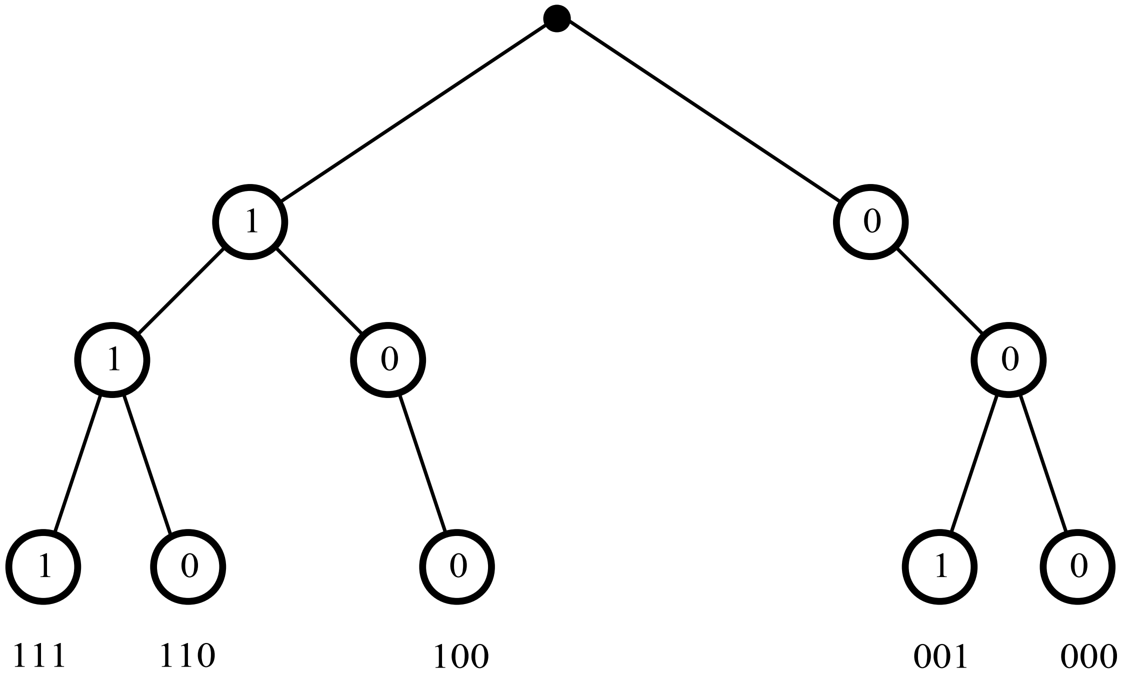
Counting the Number of Paths in a Network

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



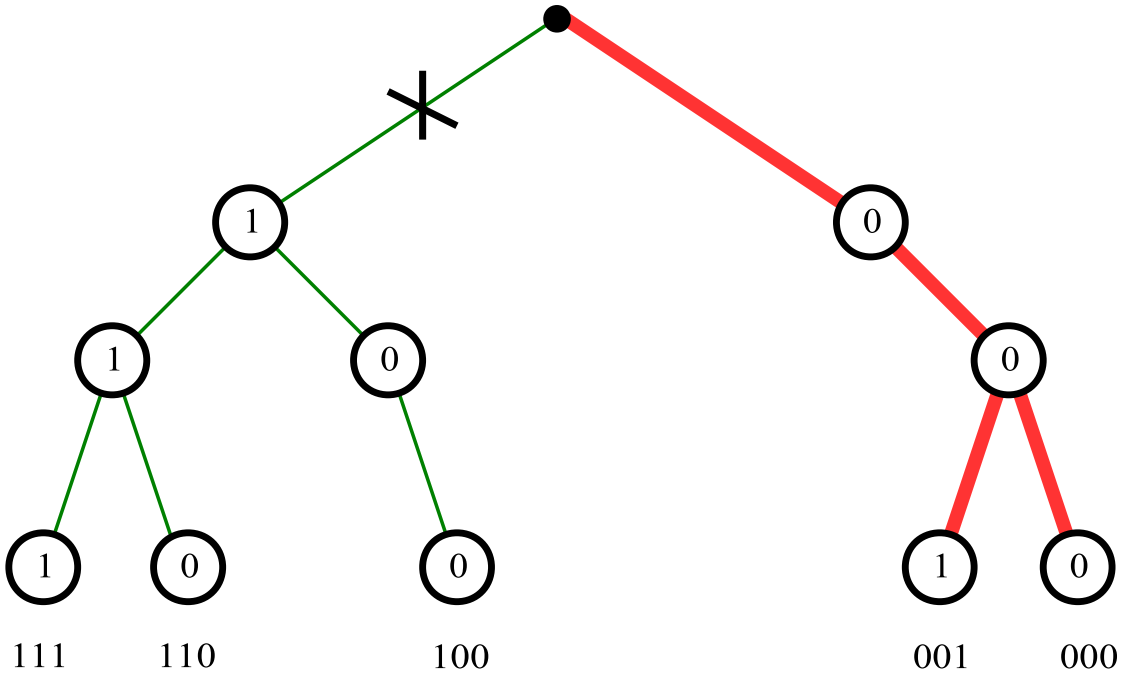
nSLA in Practice

To see how nSLA works consider a tree with the set of paths $\{000, 001, 100, 110, 111\}$.



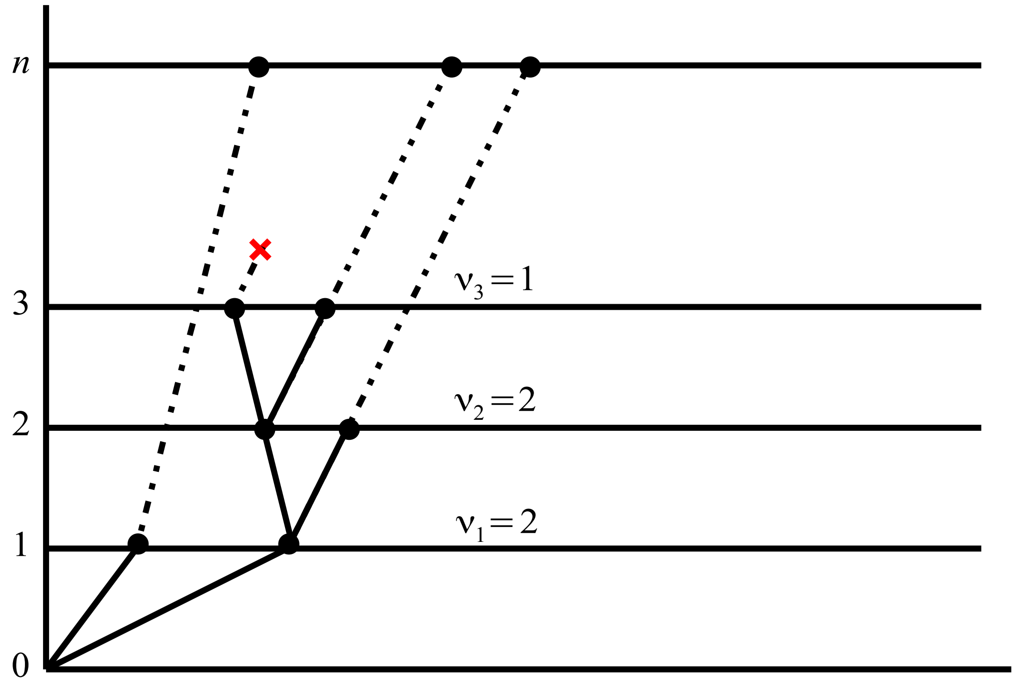
Extension of OSLA: the nSLA Method

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



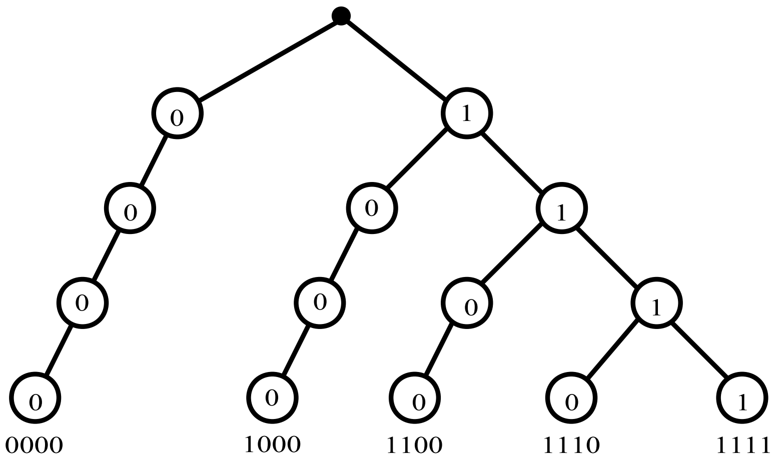
nSLA as Splitting Method

Figure below presents the dynamics of the SE Algorithm for the first 3 iterations in a model with n variables using $N^{(e)} = 1$. The accumulated weights are $\nu_1 = 2, \nu_2 = 2, \nu_3 = 1$.



Drawback of nSLA Method

Although nSLA never looses trajectories its main drawback is that the generated trajectories are **not uniformly distributed**. As results its estimators are heavily biased. To see this consider a graph with $n = 4$ variables and $|\mathscr{X}^*| = 5$. This is a 2-SAT model with clauses $C_1 \wedge C_2 \wedge, \ldots, \wedge C_n$, where $C_i = x_i \vee \bar{x}_{i+1} \geq 1$.



Extension of nSLA - the SE Method

Straightforward calculation yield that for this particular case ($|\mathscr{X}^*| = 5$) variance reduction obtained from using $N^{(e)} = 2$ instead of $N^{(e)} = 1$ is about 150 times.

To overcome the drawback of high variance of nSLA we modify it as:

Instead of a single trajectory we ran in parallel multiple ones.

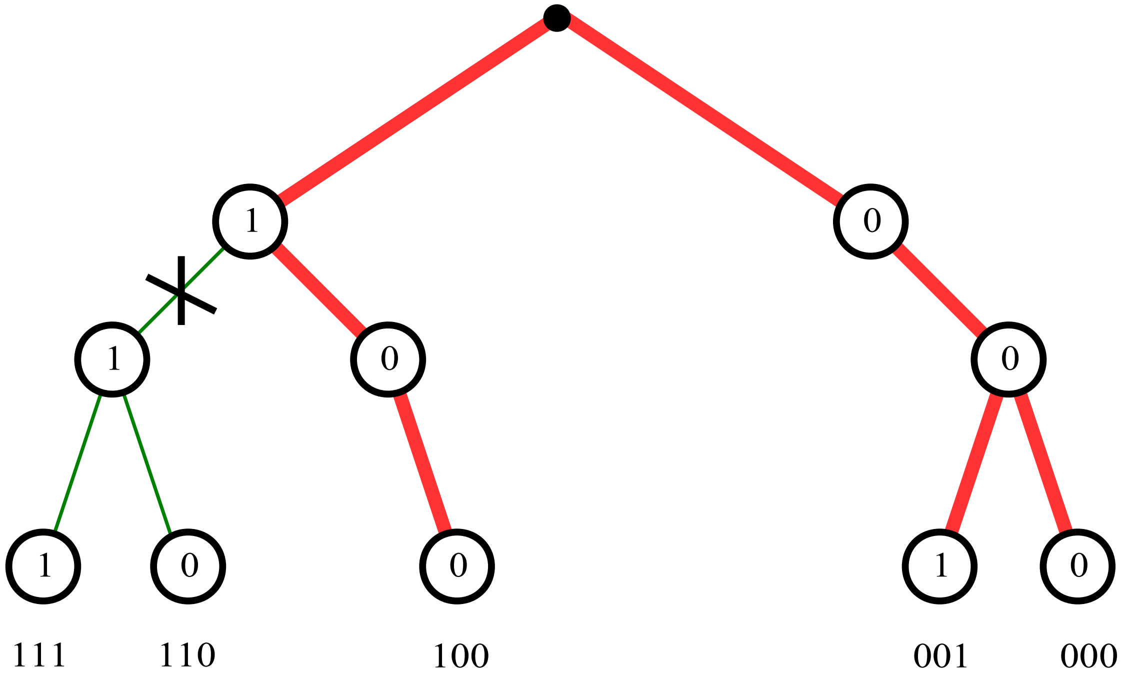
This will improve dramatically the non- uniformity issue.

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

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

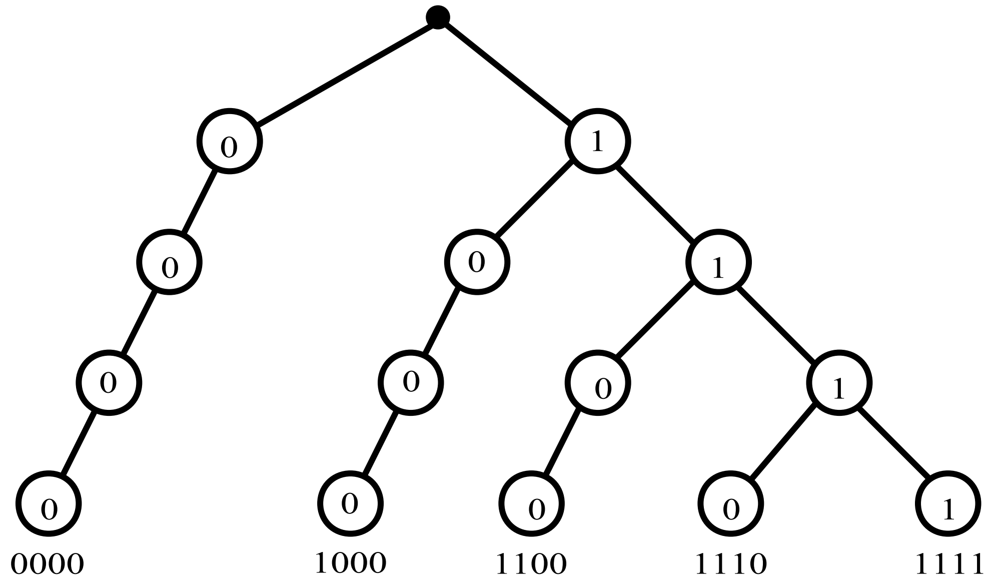
The SE Method

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 nSLA consider again the 2-SAT model with clauses $C_1 \wedge C_2 \wedge, \ldots, \wedge C_n$, where $C_i = x_i \vee \ \bar{x}_{i+1} \geq 1$. Figure below presents a graph with $n = 4$ variables and $|\mathscr{X}^*| = 5$.



Extension of nSLA: the SE Method

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

$(N^{(e)},\ M)$	$\lvert\widetilde{\mathscr{X}^*}\rvert$	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 75×325 model with $N_t^{(e)} = 20$ and $M = 100$

Run N_0	Iterations	$\lvert\widetilde{\mathscr{X}}^*\rvert$	RE of $\lvert\widetilde{\mathscr{X}}^*\rvert$	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 300×1080 model with $N_t^{(e)} = 300$, $M = 300$ and $r = 1$ with exact solution $|\mathscr{X}^*| = 3.297E + 24$.

Run N_0	Iterations	$\lvert\widetilde{\mathscr{X}^*}\rvert$	RE of $\lvert\widetilde{\mathscr{X}^*}\rvert$	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

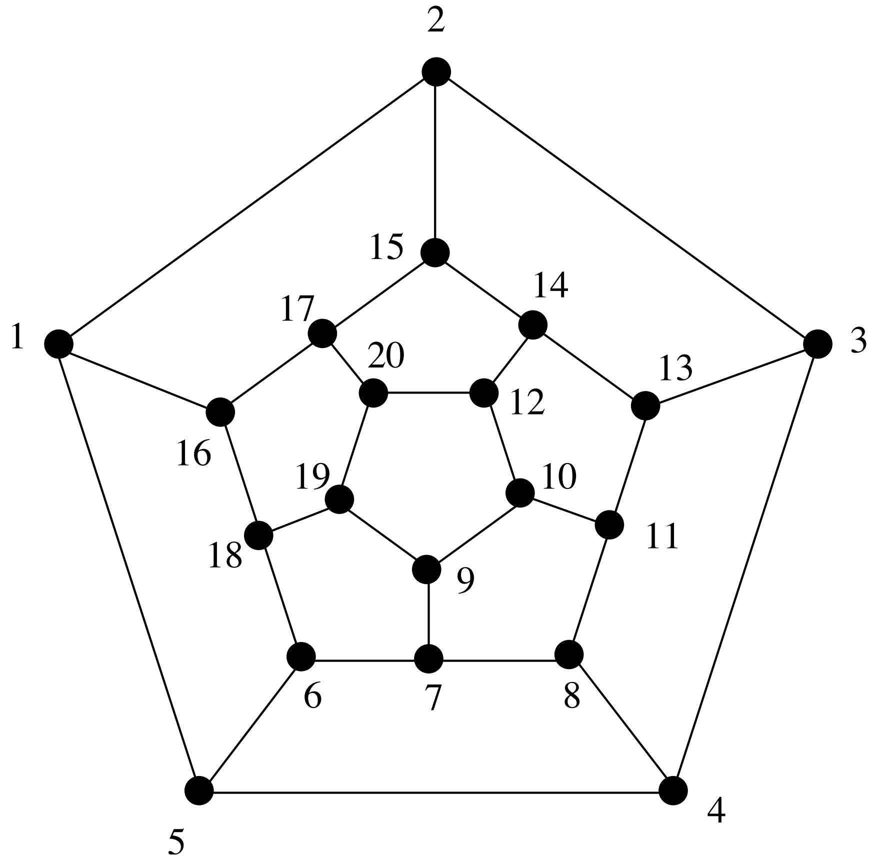
Numerical Results for SAT

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 $|\mathscr{X}^*|$ in a dodecahedron graph from node 1 to node 20. Using full enumeration, we obtained $|\mathscr{X}^*| = 1338$.



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

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

Counting the Number of Perfect Matchings (Permanent)

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

Run N_0	Iterations	$\lvert\widetilde{\mathscr{X}}^*\rvert$	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 !!!