```
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0
1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

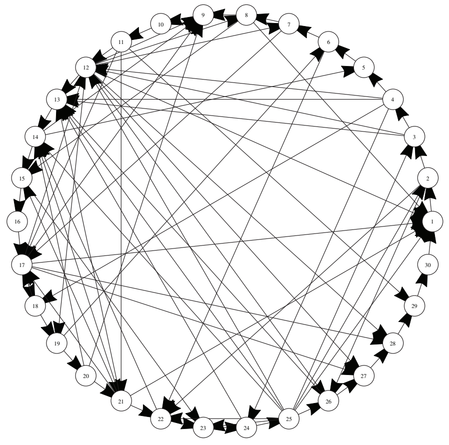

Figure 9.14: How many Hamiltonian cycles does the directed graph on the right have? The adjacency matrix is given on the left.

9.15 The *degree* of a vertex in a graph $G = (V, E)$ is the number of edges starting from that vertex. How many graphs with vertices labeled $1, \ldots, n$ are such that vertex i has a prescribed degree d_i, $i = 1, \ldots, n$? The problem can be viewed as of choosing those edges in the complete graph K_n such that the resulting graph G matches the given sequence $\mathbf{d} = (d_1, \ldots, d_n)$. In K_n all vertices are connected to each other, so there are $m = n(n-1)/2$ edges, labeled $1, \ldots, m$. Let $\mathbf{x} = (x_1, \ldots, x_m)^\top$ be such that $x_j = 1$ when edge j is chosen, and $x_j = 0$ otherwise, $j = 1, \ldots, m$. In order for such an assignment $\mathbf{x} \in \{0,1\}^m$ to match the given degree sequence $(d_1, \ldots, d_n)$, it is necessary that $\sum_{j=1}^m x_j = \frac{1}{2}\sum_{i=1}^n d_i$, since this is the total number of edges. In other words, the configuration space is

$$\mathscr{X} = \left\{ \mathbf{x} \in \{0,1\}^m : \sum_{j=1}^m x_j = \tfrac{1}{2}\sum_{i=1}^n d_i \right\}.$$

For the random graph problem we define the score function $S : \mathscr{X} \to \mathbb{Z}_-$ by

$$S(\mathbf{x}) = -\sum_{i=1}^n |\deg(i(\mathbf{x})) - d_i|,$$

where $\deg(i(\mathbf{x}))$ is the degree of vertex i under the configuration $\mathbf{x}$. Each configuration that satisfies the degree sequence will have a performance function equal to 0.

a) Describe a simple way to sample uniformly from $\mathscr{X}$.

b) Provide a Gibbs move for the splitting algorithm

c) Implement the splitting algorithm, and apply it to the degree sequence $\mathbf{d} = (3, 2, 1, 1, 1, 1, 1)$.

9.16 Apply Algorithm 9.11.2 to the minimization of the Rosenbrock function, defined in (8.64) on Page 301.

Further Reading

The splitting method was first published in [24] as a sequential procedure in which paths are split in order to increase the occurrence of some rare event. A similar *enrichment* method was used in [42] to improve the estimation of polymer dimensions. The fundamental ideas of sequential sampling, importance sampling, and splitting (enrichment, branching, cloning, resampling) have been used in many forms ever since. All of these methods may be viewed as special cases of sequential importance resampling, although such a broad generalization does no justice to the great variety of Monte Carlo methods that combine these ideas [8, 9, 12, 13, 21, 27, 28, 33, 37]. The class of SIR algorithms can itself be generalized to include Markov chain Monte Carlo (MCMC) steps. This larger class of algorithms is probably best described as particle MCMC [1]. It includes generalized splitting [6, 7], cloning, [34, 35] and the resample-move algorithm [18]. Other algorithms similar to splitting include RESTART [40, 41], subset simulation [2, 3], and stopping-time resampling [11].

Discussions on the difference between various splitting algorithms (e.g., fixed effort versus fixed number of splitting factors) and theoretical investigations on the optimal choices of the splitting parameters may be found in [16, 17, 19, 25].

As demonstrated in Section 9.8, the splitting method is well suited to solve difficult counting problems, using the key decomposition (9.3). For good references on #P-complete problems with emphasis on SAT problems, see [29, 30]. The counting class #P was defined by Valiant [39]. Randomized algorithms for approximating the solutions of some well-known counting #P-complete problems and their relation to MCMC are treated in [20, 23, 30, 31, 38, 44]. Bayati and Saberi [4] propose an efficient importance sampling algorithm for generating graphs uniformly at random. Chen et al. [10] discuss the efficient estimation, via sequential importance sampling, of the number of 0-1 tables with fixed row and column sums. Blanchet [5] provides the first importance sampling estimator with bounded relative error for this problem.

REFERENCES

1. C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

2. S.-K. Au and J. L. Beck. Estimation of small failure probabilities in high dimensions by subset simulation. *Probabilistic Engineering Mechanics*, 16(4):263–277, 2001.

3. S.-K. Au, J. Ching, and J. L. Beck. Application of subset simulation methods to reliability benchmark problems. *Structural Safety*, 29(3):183 – 193, 2007.

4. M. Bayati and A. Saberi. Fast generation of random graphs via sequential importance sampling. Manuscript, Stanford University, 2006.

5. J. Blanchet. Importance sampling and efficient counting for binary contingency tables. *Annals of Applied Probability*, 19:949–982, 2009.

6. Z. I. Botev. *The Generalized Splitting Method for Combinatorial Counting and Static Rare-Event Probability Estimation*. PhD thesis, The University of Queensland, 2009.

7. Z. I. Botev and D. P. Kroese. Efficient Monte Carlo simulation via the generalized splitting method. *Statistics and Computing*, 22:1–16, 2012.

8. H.-P. Chan and T.-L. Lai. A sequential Monte Carlo approach to computing tail probabilities in stochastic models. *Annals of Applied Probabability*, 21(6):2315–2342, 12 2011.

9. H.-P. Chan and T.-L. Lai. A general theory of particle filters in hidden Markov models and some applications. *Annals of Statistics*, 41(6):2877–2904, 12 2013.

10. Y. Chen, P. Diaconis, S. P. Holmes, and J. Liu. Sequential Monte Carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100:109–120, 2005.

11. Y. Chen, J. Xie, and J. S. Liu. Stopping-time resampling for sequential Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):199–217, 2005.

12. P. Del Moral, A. Doucet, and A. Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006.

13. A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo methods in practice*. Statistics for Engineering and Information Science. Springer New York, 2001.

14. Q. Duan and D. P. Kroese. Splitting for optimization. *Computers and Operations Research*, 73:119–131, 2016.

15. P. Dupuis, B. Kaynar, R. Y. Rubinstein, A. Ridder, and R. Vaisman. Counting with combined splitting and capture–recapture methods. *Stochastic Models*, 28:478–502, 2012.

16. M. J. J. Garvels. *The Splitting Method in Rare Event Simulation*. PhD thesis, Universiteit Twente, 2000.

17. M. J. J. Garvels and D. P. Kroese. A comparison of restart implementations. In *Simulation Conference Proceedings, 1998. Winter*, volume 1, pages 601–608 vol.1, 1998.

18. W. R. Gilks and C. Berzuini. Following a moving target: Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1):127–146, 2001.

19. P. Glasserman, P. Heidelberger, P. Shahabuddin, and T. Zajic. Multilevel splitting for estimating rare event probabilities. *Operations Research*, 47(4):585–600, 1999.

20. C. P. Gomes and B. Selman. Satisfied with physics. *Science*, pages 784–785, 2002.

21. N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107–113, 1993.

22. J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In *Satisfiability Problem: Theory and Applications*, volume 35 of DIMACS Series in Discrete Mathematics. American Mathematical Society, 1996.

23. M. Jerrum. *Counting, Sampling and Integrating: Algorithms and Complexity*. Birkhauser Verlag, Basel, 2003.

24. H. Kahn and T. E. Harris. Estimation of particle transmission by random sampling. *National Bureau of Standards Applied Mathematics Series*, 12:27–30, 1951.

25. P. L'Ecuyer, V. Demers, and B. Tuffin. Rare events, splitting, and quasi-Monte Carlo. *ACM Trans. Model. Comput. Simul.*, 17(2), 2007.

26. P. L'Ecuyer, V. Demers, and B. Tuffin. Splitting for rare-event simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 17(2):1–44, 2007.

27. P. L'Ecuyer, F. Le Gland, P. Lezaud, and B. Tuffin. Splitting techniques. In G. Rubino and B. Tuffin, editors, *Rare Event Simulation.* John Wiley & Sons, New York, 2009.

28. J. S. Liu. *Monte Carlo strategies in Scientific Computing.* Springer, New York, 2001.

29. M. Mézard and A. Montanari. *Constraint Satisfaction Networks in Physics and Computation.* Oxford University Press, Oxford, 2006.

30. M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, Cambridge, 2005.

31. R. Motwani and R. Raghavan. *Randomized Algorithms.* Cambridge University Press, Cambridge, 1997.

32. S. M. Ross. *A First Course in Probability.* Prentice Hall, Englewood Cliffs, NJ, 7th edition, 2005.

33. D. B. Rubin. The calculation of posterior distributions by data augmentation: Comment: A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest: The SIR algorithm. *Journal of the American Statistical Association*, 82(398):pp. 543–546, 1987.

34. R. Y. Rubinstein. Randomized algorithms with splitting: Why the classic randomized algorithms do not work and how to make them work. *Methodology and Computing in Applied Probability*, 12(1):1–50, 2010.

35. R. Y. Rubinstein. Stochastic enumeration method for counting NP-hard problems. *Methodology and Computing in Applied Probability*, 15(2):249–291, 2013.

36. G.A.F. Seber. *Estimation of Animal Abundance and Related Parameters.* Blackburn Press, Caldwell, N.J., second edition, 2002.

37. Ø. Skare, E. Bølviken, and L. Holden. Improved sampling-importance resampling and reduced bias importance sampling. *Scandinavian Journal of Statistics*, 30(4):719–737, 2003.

38. R. Tempo, G. Calafiore, and F. Dabbene. *Randomized Algorithms for Analysis and Control of Uncertain Systems.* Springer-Verlag, Berlin, 2004.

39. L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

40. M. Villén-Altamirano and J. Villén-Altamirano. RESTART: A method for accelerating rare events simulations. In *Proceedings of the 13th International Teletraffic Congress*, pages 71–76. North-Holland, 1991.

41. M. Villén-Altamirano and J. Villén-Altamirano. On the efficiency of RESTART for multidimensional systems. *ACM Transactions on Modeling and Computer Simulation*, 16(3):251–279, 2006.

42. F. T. Wall and J. J. Erpenbeck. New method for the statistical computation of polymer dimensions. *Journal of Chemical Physics*, 30(3):634–637, 1959.

43. B.-Y. Wang and F. Zhang. On the precise number of (0,1)-matrices in $\mathcal{U}(R,S)$. *Discrete Mathematics*, 187(13):211–220, 1998.

44. D. J. A. Welsh. *Complexity: Knots, Colouring and Counting.* Cambridge University Press, Cambridge, 1993.

CHAPTER 10

STOCHASTIC ENUMERATION METHOD

10.1 INTRODUCTION

Many counting problems can be formulated in terms of estimating the cost of a tree. In this chapter we introduce a new generic Monte Carlo technique, called *stochastic enumeration* (SE), to solve such counting problems effectively. The SE method is a sequential importance sampling technique in which random paths are generated through the tree, in a parallel fashion.

The rest of this chapter is organized as follows: Section 10.2 provides background on tree search and tree counting, and reviews the basic depth-first and breadth-first search algorithms. Knuth's algorithm for estimating the cost of a tree is given in Section 10.3. Section 10.4 describes the SE method and discusses how it can be combined with fast decision-making algorithms, called oracles. In Section 10.5 it is shown how SE can be applied to solve various counting problems, including counting the number of paths in graphs, the number of valid assignments for a satisfiability problem, and the number of perfect matchings in a bipartite graph. For each case several numerical experiments are given, showing the effectiveness of the SE method. Finally, Section 10.6 provides an application of SE to network reliability.

10.2 TREE SEARCH AND TREE COUNTING

Before introducing the stochastic enumeration algorithm, it is prudent to review a few fundamental concepts related to searching and counting trees.

Two of the most useful algorithms for searching trees (or more generally, graphs or networks) are the *depth-first search* (DFS) and *breadth-first search* (BFS) methods. Starting from a root of the tree, the algorithm enumerates all the vertices of the tree in a certain order. In the BFS case all vertices at the first level of the tree are visited first, then all vertices at the second level, and so on. The DFS algorithm, in contrast, tends to descend to low levels of the tree before visiting the upper level vertices. To keep track of which vertices still need to be visited, the DFS algorithm places vertices in a *stack*. This is a data structure where elements are added to the top and are taken out from the top, in a last-in–first-out manner. Similarly, the BFS algorithm places vertices in a *queue*. Here the elements are added to the back of the queue and taken out from the front. Figure 10.1 illustrates these two data structures.

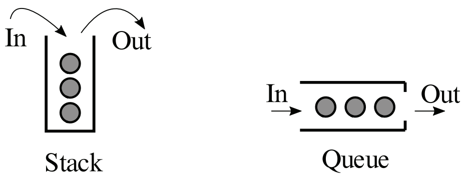

Figure 10.1: Items are added to (pushed onto) the top of the stack and are taken (popped off) from the top in a last-in–first-out manner. In contrast, items enter a queue from the back and leave from the front, in a first-in–first-out manner.

Algorithms 10.2.1 and 10.2.2 describe the DFS and BFS methods, respectively. For simplicity, we represent both the stack and the queue objects as ordered lists (vectors), where new elements are always added to the back of the list. Elements are either taken out from the front of the list (for a queue) or from the back (for a stack).

Algorithm 10.2.1: Depth-First Search

```
   input  : Tree $\mathscr{T}_u$ with root $u$.
   output: List $V$ of all vertices of the tree.
1  $V \leftarrow ()$                       // initialize the list of visited vertices
2  $S \leftarrow (u)$                                    // initialize the stack
3  while $|S| > 0$ do
4      $v \leftarrow S(\text{end})$                     // take the last element of stack
5      $S \leftarrow S(1 : \text{end} - 1)$    // reset the stack by removing last element
6      if $v \notin V$ then
7          $V \leftarrow (V, v)$               // add $v$ to the list of visited vertices
8          for $w \in \mathcal{N}(v)$ do
9              $S \leftarrow (S, w)$            // add all neighbors of $v$ to the stack
10 return $V$
```

Algorithm 10.2.2: Breadth-First Search

```
input  : Tree $\mathscr{T}_u$ with root $u$.
output: List $V$ of all vertices of the tree.
1  $V \leftarrow ()$
2  $Q \leftarrow (u)$
3  while $|Q| > 0$ do
4      $v \leftarrow Q(1)$                // take the first element in the queue
5      $Q \leftarrow Q(2:\text{end})$     // reset the queue by removing first element
6      if $v \notin V$ then
7          $V \leftarrow (V, v)$
8          for $w \in \mathcal{N}(v)$ do
9              $Q \leftarrow (Q, w)$      // add all neighbors of $v$ to the queue
10 return $V$
```

■ **EXAMPLE 10.1**

As an illustration of the workings of the DFS and BFS algorithms, consider the trees in Figure 10.2. Starting from the root A, the DFS algorithm visits the remaining vertices in the order C, H, G, F, B, E, I, D, assuming that the neighbors of a vertex are placed onto the stack from left to right in the figure. For example, the neighbors of A are B and C, and if the B is placed onto the stack before C, then C is visited first. Similarly, the BFS algorithm visits the vertices in the order A, B, C, D, E, F, G, H, I.

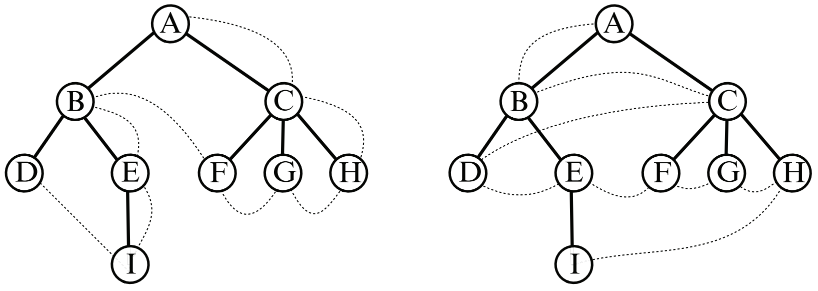

Figure 10.2: Dashed lines show the order in which the vertices are visited for the DFS (left) and BFS (right) algorithms.

We now turn our attention to tree counting. Consider a rooted tree $\mathscr{T} = (\mathscr{V}, \mathscr{E})$ with vertex set $\mathscr{V}$ and edge set $\mathscr{E}$ (so that $|\mathscr{E}| = |\mathscr{V}| - 1$). With each vertex v is associated a nonnegative cost $c(v)$. A quantity of interest is the total cost of the tree,

$$\text{Cost}(\mathscr{T}) = \sum_{v \in \mathscr{V}} c(v) \,. \tag{10.1}$$

If total enumeration of the tree is feasible, then the cost can be determined directly via DFS and BFS.

Many counting problems can be formulated as tree-counting problems. In particular, let $\mathbf{x}$ be an object in some discrete set $\mathscr{X}$ that can be constructed sequentially via intermediate objects $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$. Think of $\mathbf{x}$ as being a vector $(x_1, \ldots, x_n)$ and $\mathbf{x}_t$ as the partial vector $(x_1, \ldots, x_t)$, $t = 1, \ldots, n$. Each $\mathbf{x}$ is thus associated with a path $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_n$ in the tree of all such paths, with the "empty" vector $\mathbf{x}_0 = (\ )$ being the root of the tree. Determining $|\mathscr{X}|$, that is, the number of elements in $\mathscr{X}$, corresponds to counting the cost of the tree where all leaves of the tree have cost 1 and the internal vertices have cost 0. Frequently the situation arises that $|\mathscr{X}|$ is known but the number of elements in a much smaller subset $\mathscr{X}^*$ of $\mathscr{X}$ is not known. This amounts to counting the total costs of the tree in which only the leaves in $\mathscr{X}^*$ have non-zero cost equal to 1.

■ EXAMPLE 10.2 SAT Counting

Consider the 2-SAT CNF formula (see Section 9.9.1)

$$(l_1 \vee l_2) \wedge (l_2 \vee l_3) \wedge (l_2 \vee l_4) ,$$

and let $\mathscr{X}^*$ be the set of truth assignments $\mathbf{x} = (x_1, \ldots, x_4) \in \mathscr{X} = \{0,1\}^4$ for which the formula is satisfied. Direct inspection shows that $\mathscr{X}^* = \{0100, 0101, 0110, 0111, 1011, 1100, 1101, 1110, 1111\}$ (omitting brackets and commas for simplicity). The corresponding binary tree is given in Figure 10.3. Any vertex at the t-level of the tree corresponds to a partial assignment $(x_1, \ldots, x_t)$. Its left-child corresponds to $(x_1, \ldots, x_t, 0)$ and its right-child to $(x_1, \ldots, x_t, 1)$. The black leaves, at tree level $t = 4$, are the valid truth assignments.

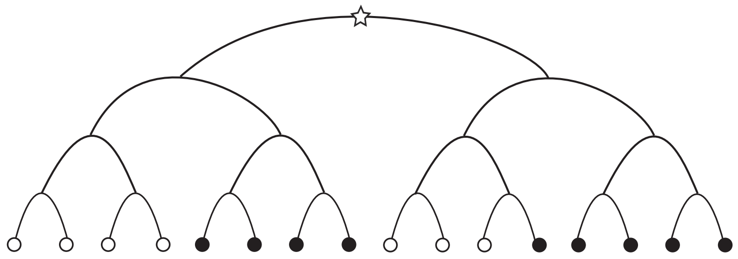

Figure 10.3: Binary counting tree for the 2-SAT counting problem.

■ EXAMPLE 10.3 Counting Paths in Graphs

Figure 10.4 shows the *Petersen graph* of order $(7, 4)$. How many (non-intersecting) paths does the graph have between the two highlighted vertices at the top and bottom-left? A systematic way to determine this is to sequentially construct a tree of paths of various lengths, all starting from the top vertex and ending either at the bottom-left vertex or at a vertex whose neighbors have already been visited. Note that the lengths of these paths vary from 3 to 13.

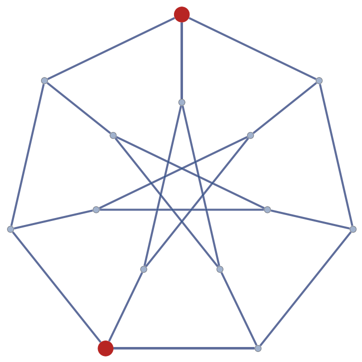

Figure 10.4: The (7,4)-Petersen graph. How many paths are there between the two highlighted vertices?

A similar type of problems is counting self-avoiding walks (see Example 5.17 on Page 162 and Section 9.2 on Page 308). Here the underlying graph is the grid graph on $\mathbb{Z}^2$ and the objective is to count the number of non-intersecting walks of a fixed length n, starting from the origin $(0, 0)$. If a walk is represented as the vector $\mathbf{x} = (x_1, \ldots, x_n)$, where x_i can take four possible values (up, down, left, right), then the corresponding counting tree is a *quadtree* (each parent has exactly four children). The tree leaves correspond to the 4^n walks of length n, and only the non-intersecting ones have non-zero costs 1.

10.3 KNUTH'S ALGORITHM FOR ESTIMATING THE COST OF A TREE

Knuth [10] proposed a randomized algorithm to *estimate* the cost of a tree, when full enumeration of the tree is not feasible. Consider the cost of a tree $\mathscr{T}$ as given in (10.1). For each vertex v we denote the set of children or *successors* of v by $\mathscr{S}(v)$.

Knuth's algorithm constructs a random path $X_0, X_1, \ldots$, through the tree, starting from the root vertex and ending at a leaf, selecting at each step t at random (and uniformly) a successor vertex from the set $\mathscr{S}(X_t)$. In addition, two variables are updated at each step: (1) the product D of the degrees of all vertices encountered and (2) the estimated cost C of the tree. Algorithm 10.3.1 shows the details.

Algorithm 10.3.1: Knuth's Algorithm for Estimating the Cost of a Tree

```
   input  : Tree $\mathscr{T}$ with root $u$.
   output: Estimator $C$ of the total cost of $\mathscr{T}$.
1  $C \leftarrow c(u), D \leftarrow 1, X \leftarrow u$
2  while $|\mathscr{S}(X)| > 0$ do
3      $D \leftarrow |\mathscr{S}(X)| D$
4      $X \leftarrow$ uniformly selected successor in $\mathscr{S}(X)$
5      $C \leftarrow C + c(X) D$
6  return $C$
```

In practice, Algorithm 10.3.1 is repeated via N independent runs, giving estimators $C_1, \ldots, C_N$. In the usual way (see Section 4.2), the combined estimator for these runs is the sample mean

$$\overline{C} = \frac{C_1 + \cdots + C_N}{N}, \tag{10.2}$$

and its accuracy can be assessed via the estimated relative error

$$\widehat{\mathrm{RE}} = \frac{S_C}{\sqrt{N} \times \overline{C}}, \tag{10.3}$$

where S_C is the sample standard deviation of the $\{C_i\}$.

Knuth's algorithm can be viewed as a sequential importance sampling algorithm with a one-step–look-ahead sampling strategy, similar to the algorithm for counting self-avoiding walks in Example 5.17 on Page 162. To see this, extend the tree with 0-cost vertices such that each vertex at depth $t < n$ has at least one successor. Hence, each path returned by the algorithm now has length n, but the total cost of the extended tree is still the same as the original one. Let $\mathscr{V}_t$ be the set of all vertices at the t-th level of the extended tree (hence, $\mathscr{V}_0$ has only the root vertex, and $\mathscr{V}_n$ has all the leaf vertices). For each $t = 0, 1, \ldots, n$, Knuth's algorithm defines a density g_t on the set $\mathscr{X}_t$ of all paths $\mathbf{x}_t = (x_0, x_1, \ldots, x_t)$ from the root to level t. Specifically,

$$g_t(\mathbf{x}_t) = \frac{1}{d(x_0)}\frac{1}{d(x_1)}\cdots\frac{1}{d(x_{t-1})} \overset{\text{def}}{=} \frac{1}{D(\mathbf{x}_t)},$$

where $d(x) = |\mathscr{S}(x)|$ denotes the vertex degree of x. The total cost of the vertices at level t of the tree can thus be written as

$$\sum_{v \in \mathscr{V}_t} c(v) = \sum_{\mathbf{x}_t} \frac{c(x_t)}{g_t(\mathbf{x}_t)} g_t(\mathbf{x}_t) = \mathbb{E}_{g_t}\left[\frac{c(X_t)}{g_t(\mathbf{X}_t)}\right] = \mathbb{E}_{g_t}\left[c(X_t) D(\mathbf{X}_t)\right],$$

so that $c(X_t)D(\mathbf{X}_t)$ is an unbiased estimator of this cost, and the sum $\sum_{t=0}^{n} c(X_t)D(\mathbf{X}_t)$ is an unbiased estimator of the total cost of the tree.

Knuth's algorithm is very effective as long as the costs are evenly distributed over the tree. Consider, for example, a binary tree of depth (maximum level) n, where all the leaves have cost 1 and all the internal vertices have cost 0. In this case a *single* run ($N = 1$) of Algorithm 10.3.1 provides the total cost of the tree, 2^n, regardless of which path $x_0, x_1, \ldots, x_n$ is generated. However, Knuth's algorithm can be very poor when the tree has little symmetry. An extreme example is the "hairbrush" tree in Figure 10.5 of depth n. Suppose that the cost of all vertices is zero except for the right-most vertex at depth n, which has a cost of unity.

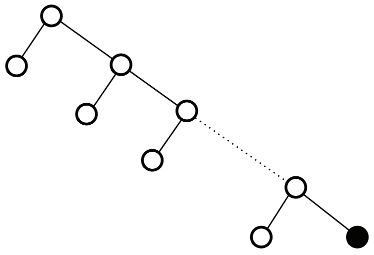
Figure 10.5: Hairbrush tree.

Knuth's algorithm reaches the vertex of interest (colored black in the figure) with probability $1/2^n$ and with $D = C = 2^n$. In all other cases, the algorithm terminates with $C = 0$. It follows that the expectation and variance of Knuth's estimator are, respectively,

$$\mathbb{E}\,[C] = \frac{1}{2^n} \cdot 2^n = 1$$

and

$$\mathrm{Var}\,(C) = \frac{1}{2^n} \cdot (2^n)^2 - 1 = 2^n - 1\ .$$

Hence, although the estimator is unbiased, its variance grows exponentially with n.

10.4 STOCHASTIC ENUMERATION

The *stochastic enumeration* (SE) algorithm generalizes Knuth's algorithm by simulating multiple paths through the tree simultaneously. These paths are also allowed to interact, in the sense that paths avoid each other and therefore explore a larger part of the tree.

Let $B \geqslant 1$ be a fixed parameter, representing the computational budget. We will refer to a set of vertices at the same level of the tree as a *hypervertex*. If $\mathcal{V}$ is a hypervertex, define the collection of hyperchildren of $\mathcal{V}$ by

$$\mathscr{H}(\mathcal{V}) = \begin{cases} \{\mathscr{S}(\mathcal{V})\} & \text{if } |\mathscr{S}(\mathcal{V})| \leqslant B\ , \\ \{\mathcal{W} \subseteq \mathscr{S}(\mathcal{V}) :\ |\mathcal{W}| = B\} & \text{if } |\mathscr{S}(\mathcal{V})| > B\ . \end{cases}$$

Note that if the total number of successors of all vertices in $\mathcal{V}$ is less than or equal to the budget B, then $\mathcal{V}$ has only *one* hyperchild, namely $\mathscr{S}(\mathcal{V})$; otherwise, it has $\binom{|\mathscr{S}(\mathcal{V})|}{B}$ hyperchildren, all containing exactly B vertices. We define the cost $c(\mathcal{V})$ of a hypervertex to be the sum of the costs of the individual vertices, and the successors $\mathscr{S}(\mathcal{V})$ to be the union of the successors of the individual vertices in $\mathcal{V}$.

The stochastic enumeration algorithm is shown in Algorithm 10.4.1 and is very similar in form to Algorithm 10.3.1. The crucial difference is that it constructs a random sequence $\mathcal{X}_0, \mathcal{X}_1, \ldots$, of hypervertices (collections of vertices at the same levels 1, 2, ...) rather than vertices. Note that for $B = 1$ the SE algorithm reduces to Knuth's algorithm.

Algorithm 10.4.1: Stochastic Enumeration Algorithm for Estimating the Cost of a Tree

```
  input  : Tree $\mathscr{T}$ with root $u$ and a budget $B \geqslant 1$.
  output: Estimator of the total cost of $\mathscr{T}$.
1 $D \leftarrow 1, \mathcal{X} = \{u\}, C \leftarrow c(u)$
2 while $|\mathscr{S}(\mathcal{X})| > 0$ do
3     $D \leftarrow \frac{|\mathscr{S}(\mathcal{X})|}{|\mathcal{X}|} D$
4     $\mathcal{X} \leftarrow$ uniformly selected successor (hyperchild) in $\mathscr{H}(\mathcal{X})$
5     $C \leftarrow C + \frac{c(\mathcal{X})}{|\mathcal{X}|} D$
6 return $C$
```

Using similar reasoning as for Knuth's estimator, the SE estimator can be shown to be unbiased; see also [16]. In practice, SE is repeated a number of times to assess the accuracy of the estimate via the corresponding relative error and confidence interval.

Because both Knuth's algorithm and SE use importance sampling, the resulting estimator can misjudge the cost of a tree if the distribution of vertex costs is very skewed, even though the estimator is unbiased, as poignantly demonstrated by the hairbrush tree example in Figure 10.5. By increasing the budget, SE is better able to deal with such trees. Indeed, for the hairbrush tree a budget of size $B = 2$ suffices to produce a zero-variance estimator! However, for some trees Knuth's method may already provide a reliable estimate of the tree cost, so that there is no need to increase B, as the computational effort grows linearly with B. Our advice is to gradually increase the budget, starting with $B = 1$, until the estimates become stable (no longer significantly change).

Example 10.4 illustrates the workings of Algorithm 10.4.1. Note that while Knuth's algorithm operates on vertices at each step, SE operates on collections of vertices, that is, on hypervertices.

EXAMPLE 10.4

Consider the tree $\mathscr{T}$ displayed on the left-hand side of Figure 10.6. Suppose that we wish to count the total number of vertices; so $c(v) = 1$ for all $v \in \mathscr{V}$.

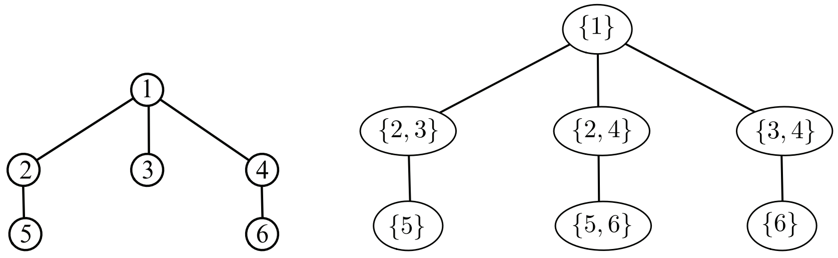

Figure 10.6: Applying SE to the tree on the left with a budget $B = 2$ gives one path through the hypertree on the right.

We step through Algorithm 10.4.1 using a budget $B = 2$. For clarity we index the objects D, $\mathcal{X}$, and C according to the iteration number, starting at iteration 0. The SE algorithm starts at the root and finds that there are 3 successors. Hence, we set $D_0 \leftarrow 1$, $\mathcal{X}_0 \leftarrow \{1\}$, and $C_0 \leftarrow 1$. Next, we determine $\mathscr{S}(\mathcal{X}_0)$ — the set of all successors of $\mathcal{X}_0$, which is $\{2,3,4\}$. Since the budget is 2, the hyperchildren of $\mathcal{X}_0$ are all subsets of size 2 of $\{2,3,4\}$, giving $\mathscr{H}(\mathcal{X}_0) = \{\{2,3\},\{2,4\},\{3,4\}\}$.

D_1 is obtained by multiplying D_0 with $|\mathscr{S}(\mathcal{X}_0)|/|\mathcal{X}_0|$, which may be viewed as the "average degree" of hypervertex $\mathcal{X}_0$. In this case $\mathcal{X}_0$ only contains one vertex, which has degree 3, so $D_1 \leftarrow \frac{3}{1} \cdot 1 = 3$. One of the hyperchildren is chosen at random, say $\mathcal{X}_1 \leftarrow \{2,3\}$. C_1 is set to $C_0 + \frac{c(\mathcal{X}_1)}{|\mathcal{X}_1|} D_1 = 1 + \frac{2}{2} \cdot 3 = 4$.

At the second iteration D_2 is obtained by multiplying D_1 with the average degree of $\{2,3\}$, which is $1/2$, since in the original tree the degree of vertex 2 is 1 and the degree of vertex 3 is 0. So $D_2 \leftarrow \frac{1}{2} \cdot 3 = \frac{3}{2}$. As $\mathcal{X}_1$ only has one hyperchild, $\mathcal{X}_2 \leftarrow \{5\}$, and C_2 becomes to $4 + \frac{1}{1} \cdot \frac{3}{2} = 11/2$. The algorithm then terminates, as there are no more hyperchildren.

Each run of Algorithm 10.4.1 will choose one of 3 possible paths through the hypertree. Below we consider all the paths, the obtained estimator and the corresponding probabilities:

1. $\{1\} \to \{2,3\} \to \{5\}$, $C = 1 + 3 + 3/2 = 11/2$, with probability 1/3.

2. $\{1\} \to \{2,4\} \to \{5,6\}$, $C = 1 + 3 + 3 = 7$, with probability 1/3.

3. $\{1\} \to \{3,4\} \to \{6\}$, $C = 1 + 3 + 3/2 = 11/2$, with probability 1/3.

It follows that

$$\mathbb{E}[C] = \frac{1}{3}\left(\frac{11}{2} + 7 + \frac{11}{2}\right) = 6$$

and

$$\text{Var}(C) = \frac{1}{3}\left[\left(\frac{11}{2}\right)^2 + 7^2 + \left(\frac{11}{2}\right)^2\right] - 6^2 = \frac{1}{2}.$$

The reader may verify (see Problem 10.2) that for a budget $B = 1$ the variance of C is 2. Thus, although running two paths in parallel doubles the computation time (compared with running a single path), the variance is reduced by a factor of 4. For a budget $B = 3$ the SE algorithm will even give the zero-variance estimator, $C = 6$, as the hypertree consists, in this case, of a single path $\{1\} \to \{2,3,4\} \to \{5,6\}$.

10.4.1 Combining SE with Oracles

When running SE one would like to generate only paths in the tree that are useful for estimating the cost of the tree. Suppose that v is a tree vertex that is visited during the execution of the SE algorithm. If it is known that any a path continued via a successor w of v will have cost 0, then it makes sense to delete w from the list of successors of v. The YES–NO decision of keeping w or not can often be calculated quickly via a so-called *oracle*. Example 10.5 illustrates the idea of using SE with an oracle.

■ **EXAMPLE 10.5**

We can also use SE or Knuth's algorithm to count the number of non-intersecting paths, as we do from vertex 1 to 7 in the directed graph of Figure 10.7. Starting from root 1, we construct a tree where at the first level there are 3 potential successors. However, we can quickly establish that paths going through vertex 4 can never reach vertex 7. Hence vertex 4 should be eliminated from the successor list at iteration 1. Similarly 4 should be eliminated at various other possible stages of the algorithm, reducing the original search tree considerably; see Figure 10.8.

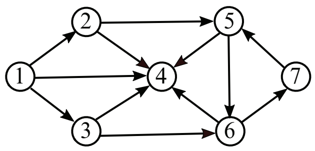

Figure 10.7: A directed graph.

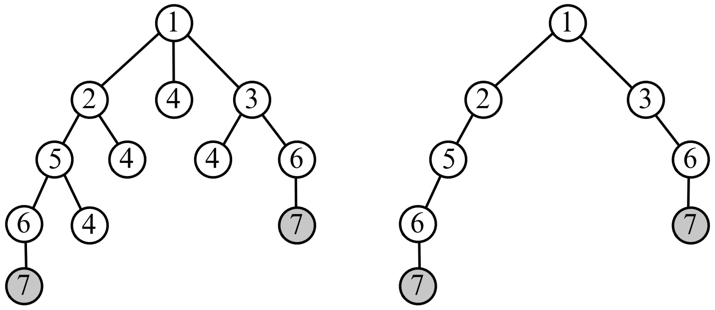

Figure 10.8: Original search tree and the reduced tree obtained by using oracles.

Interestingly, there are many problems in computational combinatorics for which the decision making is easy (polynomial) but the counting is hard. As mentioned in Section 9.8, finding out how many paths there are between two vertices in a graph is #P-complete, while the corresponding YES–NO decision problem (is there such a path?) is easy and can be solved via fast algorithms such as Dijkstra's algorithm [1] or simply via BFS or DFS. Other examples include finding a solution to a 2-SAT problem (easy) versus finding the number of such solutions (hard), and finding perfect matchings in a bipartite graph (easy) versus finding the number of such perfect matching (hard).

Our strategy is thus to incorporate fast polynomial decision-making oracles into SE to solve #P-complete problems.

10.5 APPLICATION OF SE TO COUNTING

In this section we demonstrate three applications of SE involving difficult (#P-complete) counting problems. For each case we provide a worked example, indicate which oracle can be used, and carry out various numerical experiments that show the effectiveness of the method.

10.5.1 Counting the Number of Paths in a Network

Here we use SE to count the number of paths in a network with a fixed source and sink. We start with a toy example.

■ **EXAMPLE 10.6 Extended Bridge**

Our objective is to apply SE to count how many paths are there from s to t in the graph of Figure 10.9. A quick inspection shows there are 7 such paths:

$$\begin{array}{l}(e_1, e_3, e_5, e_7),\ (e_1, e_3, e_6),\ (e_1, e_4, e_5, e_6), (e_1, e_4, e_7) \\ (e_2, e_3, e_4, e_7),\ (e_2, e_5, e_7),\ (e_2, e_6)\ .\end{array} \tag{10.4}$$

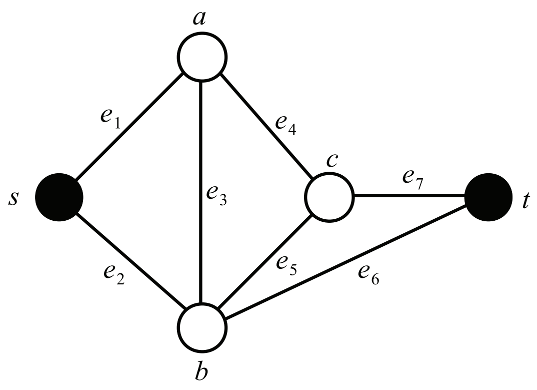

Figure 10.9: Extended bridge.

The associated counting tree is given in Figure 10.10. The SE algorithm only constructs a part of the tree, by running in parallel B paths until a leaf vertex is reached. Each vertex at level t corresponds to a (partial) path $(x_1, x_2, \ldots, x_t)$ through the graph, starting from s. The bottom-left leaf vertex, for example, corresponds to the path (e_1, e_3, e_5, e_7). The leaf vertices have cost 1; all other vertices have cost 0. The root vertex (path of length 0) has been left out in the notation of a path, but could be denoted by the empty vector ( ).

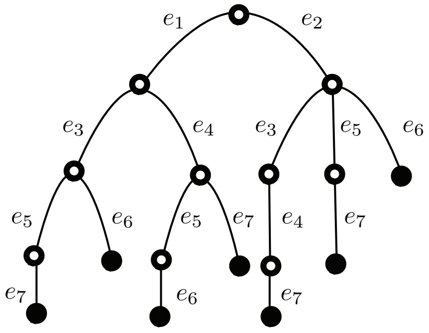

Figure 10.10: Cunting tree corresponding to the extended bridge.

Stepping through a typical run of Algorithm 10.4.1, with a budget $B = 2$, could give the following result:

1. **Initialization**. Initialize D to 1, the cost C to 0, and let $\mathcal{X}$ contain the root of the tree, ( ).

2. **Iteration 1**. The root vertex has potentially 2 children, (e_1) and (e_2) of length 1. We use an oracle to verify that there indeed exists paths of the form $(e_1, \ldots)$ and $(e_2, \ldots)$. Hence, D is updated to $\frac{2}{1} \cdot 1 = 2$. Since the budget $B = 2$, we have $\mathcal{X} = \{(e_1), (e_2)\}$. Neither of the vertices in $\mathcal{X}$ are leaves, so C remains equal to 0.

3. **Iteration 2**. The hypervertex $\mathcal{X}$ has potentially 5 children, $(e_1, e_3), (e_1, e_4)$, (e_2, e_3),(e_2, e_5), and (e_2, e_6). Again, by an oracle all these children can be verified to lead to valid paths. D is therefore updated to $\frac{5}{2} \cdot 2 = 5$. Two children are randomly chosen, without replacement. Suppose that we selected (e_1, e_4) and (e_2, e_5) to form the new hypervertex $\mathcal{X}$. Because the vertices are not leaves, C remains 0.

4. **Iteration 3**. Both paths (e_1, e_4) and (e_2, e_5) have potentially 2 children. However, of these 4 children, only 3 lead to valid paths. In particular, (e_2, e_5, e_4) is not valid, and is therefore discarded by the oracle. Hence, D is updated to $\frac{3}{2} \cdot 5 = 15/2$. The new hypervertex $\mathcal{X}$ is formed by selecting 2 of the 3 children at random, without replacement; say (e_1, e_4, e_5) and (e_2, e_5, e_7) are chosen. Since the last vertex is a leaf, C becomes $0 + \frac{1}{2} \cdot \frac{15}{2} = \frac{15}{4}$.

5. **Iteration 4**. The oracle determines that there is now only one valid child of $\mathcal{X}$, namely (e_1, e_4, e_5, e_6). Hence, D is updated to $\frac{1}{2} \cdot \frac{15}{2} = \frac{15}{4}$, $\mathcal{X} = \{(e_1, e_4, e_5, e_6)\}$, and since the child vertex is a leaf, C becomes $\frac{15}{4} + \frac{1}{1} \cdot \frac{15}{4} = 7.5$. Because there are no further children, the algorithm terminates.

Typical oracles that could be used for path counting are breadth-first search (BFS) or Dijkstra's shortest path algorithm [1]. We will use the former in our numerical experiments.

10.5.1.1 Numerical Results We run SE on two *Erdös–Rényi random graphs* [5]. Such a graph depends on two parameters: the number of vertices, n, and the *density* of the graph, p. There are two types of constructions giving graphs of slightly different properties. The first is to independently choose each of the $\binom{n}{2}$ edges with probability p. The number of chosen edges thus has a $\mathsf{Bin}(n(n-1)/2,\, p)$ distribution. The second method is to fix the number of edges to $m = np$ (assuming this is an integer) and draw uniformly without replacement m edges from the total of $\binom{n}{2}$ possible edges. The graphs used in our numerical results were generated using the latter method.

Model 1. The first graph, with $n = 24$ vertices, is taken from [12]. The adjacency matrix of the graph is

$$A = \begin{pmatrix}
0&0&0&1&0&0&0&0&0&0&0&1&0&0&1&1&1&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&1&1&0&0&0&0&0&0&1&0&0\\
0&0&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&1&0&0&1&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&1&1&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&0&0&0\\
1&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0\\
0&1&1&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&1&0&1&0&1&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
1&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&1&0&0&1&1\\
1&0&0&1&1&0&1&1&1&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0\\
1&0&0&1&1&0&0&0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1&1&0&1&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0
\end{pmatrix}.$$

The objective is to estimate the total number of paths between vertex 1 and 24. In this case the exact solution is 1892724.

Ten independent runs of the SE Algorithm 10.4.1, using a budget of $B = 50$ and $N = 400$ repetitions per run, gave the following results, times 10^6:

$$1.81, 1.93, 1.98, 1.82, 1.90, 1.94, 1.86, 1.87, 1.90, 1.92, 1.89,$$

which gives an estimated relative error per run (i.e., the sample standard deviation divided by the sample mean) of 0.0268. The overall result of the 10 runs gives an estimate of $1.8927 \cdot 10^6$ with an estimated relative error $0.0268/\sqrt{10} \approx 0.0085$. The CPU time was about 4 seconds per run.

Model 2. For our second model, we randomly generated an Erdös–Rényi graph with $n = 200$ vertices and $m = 200$ edges. Ten independent runs of the SE Algorithm 10.4.1, using a budget of $B = 100$ and $N = 500$ repetitions per run, gave the following results, times 10^7:

$$7.08, 7.39, 7.28, 7.38, 7.12, 7.05, 7.22, 7.48, 7.47, 7.40, 7.29,$$

which gives an estimated relative error per run of 0.021.

For $B = 1$ and $N = 50000$, we obtained instead the estimates, times 10^7:

$$7.40, 7.37, 7.38, 7.30, 7.20, 7.40, 7.15, 7.31, 7.27, 7.34, 7.31,$$

which gives an estimated relative error per run of 0.011. This gives confidence that Knuth's algorithm works well on this instance.

10.5.2 Counting SATs

To show how SE works for counting SATs, we illustrate it first for the 2-SAT toy problem in Example 10.2, concerning the CNF formula $(l_1 \vee l_2) \wedge (l_2 \vee l_3) \wedge (l_2 \vee l_4)$.

■ **EXAMPLE 10.7 Example 10.2 (Continued)**

The SE algorithm generates B paths through the binary counting tree of Figure 10.3, while also employing an oracle to see if a partial vector $\mathbf{x}_t$ can lead to a valid assignment. In effect, by using an oracle, the SE generates paths in the reduced tree given in Figure 10.11. Left branching again corresponds to adding a 0, and right branching to adding a 1.

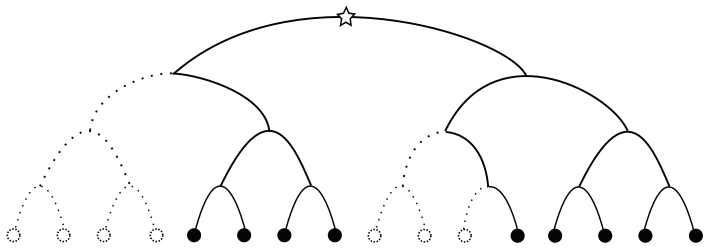

Figure 10.11: Reduced tree for the 2-SAT counting problem

Using a budget $B = 2$, a typical run of the SE algorithm (with an oracle) could be as follows. Note that we leave out the brackets and commas in the notation for the binary vectors, so, for example, 010 is shorthand for (0,1,0).

1. **Initialization**. Initialize D to 1, the cost C to 0, and let $\mathcal{X}$ contain the root of the tree.

2. **Iteration 1**. The root vertex has potentially 2 children: 0 and 1. We apply the oracle twice to verify that there indeed exist assignments of the form $0\cdots$ and $1\cdots$. Hence D is updated to $\frac{2}{1}\cdot 1 = 2$. Since the budget $B = 2$, we have $\mathcal{X} = \{0, 1\}$. Neither of the vertices in $\mathcal{X}$ are leaves, so C remains equal to 0.

3. **Iteration 2**. The hypervertex $\mathcal{X}$ has potentially 4 children: $00, 01, 10, 11$. But the oracle rules out 00, so 3 children remain. D is therefore updated to $\frac{3}{2}\cdot 2 = 3$, and two children are randomly chosen, without replacement; say 01 and 10. Because neither are leaves, C remains 0.

4. **Iteration 3**. We again have 4 potential children, but 100 is ruled out, leaving 010, 011, and 101. Hence, D is updated to $\frac{3}{2}\cdot 3 = 9/2$. Suppose that we select $\mathcal{X} = \{010, 101\}$. Again C remains 0.

5. **Iteration 4**. After applying the oracle, the children of $\mathcal{X}$ are found to be $0100, 0101$, and 1011. All of these are leaf vertices with cost 1. D is updated to $\frac{3}{2}\cdot\frac{9}{2} = \frac{27}{4}$ and C becomes $0 + \frac{2}{2}\cdot\frac{27}{4} = 6.75$. Since there are no further children, the algorithm terminates.

Although decision making for K-SATs is NP-hard for $K \geqslant 3$, there are several very fast heuristics for this purpose. The most popular one is the famous DPLL solver [2, 3]. The basic backtracking algorithm runs by choosing a literal, assigning a truth value to it, simplifying the formula, and then recursively checking if the simplified formula is satisfiable. If this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value. This is known as the splitting rule, as it splits the problem into two simpler sub-problems. The simplification step essentially removes all clauses that become true under the assignment from the formula, and all literals that become false from the remaining clauses.

For the 2-SAT decision problem DPLL has polynomial complexity in the number of clauses. In our experiments we used the freely available `minisat` solver from `http://minisat.se/`, which is based on DPLL.

10.5.2.1 Numerical Results Here, we present numerical results for several SAT models.

We first revisit the 3-SAT (75×325) problem from Section 9.9.1.1, which has 2258 solutions. Running 10 independent replications of SE, with $B = 20$ and $N = 100$, produced the following estimates:

$$2359.780, 2389.660, 2082.430, 2157.850, 2338.100,$$
$$2238.940, 2128.920, 2313.390, 2285.910, 2175.790.$$

This gives an estimated relative error per run of 0.047, and the combined estimate is 2247 with an estimated relative error of 0.015.

Comparison with other methods. We compared the SE method with splitting and also with `SampleSearch` [6, 7]. To make a fair comparison, each method was run for the same amount of time. This also determines the number of replications. For example, if the time limit is 300 seconds, and a run of SE takes 3 seconds, then 100 independent runs can be performed.

We consider three 3-SAT instances:

1. 75×325. This is the instance given above.
2. 75×270. This is the 75×270 instance with the last 55 clauses removed.
3. 300×1080. This is the 75×270 instance "replicated" 4 times. Specifically, its clause matrix is given by the block-diagonal matrix

$$B = \begin{pmatrix} A & O & O & O \\ O & A & O & O \\ O & O & A & O \\ O & O & O & A \end{pmatrix},$$

where A is the clause matrix corresponding to the 75×270 instance.

Regarding the choice of parameters, for SE we used $B = 20$ for the 75×325 instance, $B = 100$ for the 75×270 instance and $B = 300$ for the 300×1080 instance. For splitting, we used $N = 10,000$ and $\varrho = 0.1$ for all instances. `SampleSearch` does not require specific input parameters.

The results of the comparison are given in Table 10.1. For each of the methods, we list the estimate and the estimated relative error.

Table 10.1: Comparison of the efficiencies `SampleSearch` (SaS), stochastic enumeration (SE), and splitting (Split).

Instance	Time	SaS	SaS RE	SE	SE RE	Split	Split RE
75×325	137 sec	2212	2.04E-02	2248.8	9.31E-03	2264.3	6.55E-02
75×270	122 sec	1.32E+06	2.00E-02	1.34E+06	1.49E-02	1.37E+06	3.68E-02
300×1080	1600 sec	1.69E+23	9.49E-01	3.32E+24	3.17E-02	3.27E+24	2.39E-01

For the 300×1080 problem, for example, SE has a relative error that is 30 times smaller than `SampleSearch`, and 7 times smaller than splitting. In terms of computational times, this means a speedup by factors of 900 and 49, respectively.

10.5.3 Counting the Number of Perfect Matchings in a Bipartite Graph

Here, we deal with the application of SE to the estimation of the number of perfect matchings in a bipartite graph. Recall from Section 9.9.3 that in such a graph the vertices can be partitioned into two independent sets such that each edge has one vertex in each of the sets. The adjacency matrix can therefore be decomposed as

$$B = \begin{pmatrix} O_{p\times p} & A \\ A^\top & O_{q\times q} \end{pmatrix}$$

for some $p \times q$ matrix B (called the *biadjacency* matrix) and zero-matrices $O_{p\times p}$ and $O_{q\times q}$. A matching is a subset of the edges with the property that no two edges share the same vertex. For the case where $p = q$, a perfect matching is a matching of all the vertices. The number of perfect matchings in a bipartite graph is identical to the permanent of the biadjacency matrix A; see Section 9.9.3.

A natural implementation of SE involves ordering the vertices from 1 to p and constructing partial vectors of the form $(x_1, x_2, \ldots, x_t)$, where $x_i \neq x_j, i \neq j$. This vector identifies the first t edges (i, x_i), $i = 1, \ldots, t$ of the perfect matching.

■ **EXAMPLE 10.8 Bipartite Graph**

Consider the bipartite graph in Figure 10.12, whose biadjacency is given on the left of the graph.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix},$$

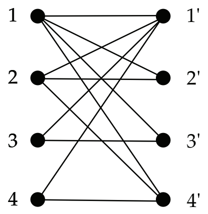

Figure 10.12: Bipartite graph with its biadjacency matrix.

The associated counting tree is given in Figure 10.13.

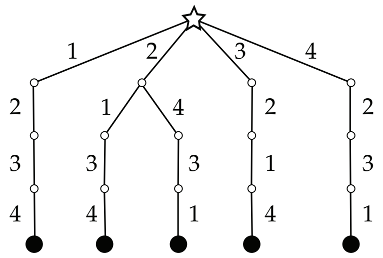

Figure 10.13: Counting tree associated with the bipartite graph in Figure 10.12.

The tree is highly balanced, and hence Knuth's algorithm (SE with a budget of $B = 1$) will perform well. A typical run is as follows:

1. **Initialization**. Initialize D to 1, the cost C to 0, and let X be the root of the tree.

2. **Iteration 1**. The root vertex has potentially 4 children: $(1), (2), (3)$, and (4). We apply the oracle four times to verify that there indeed exist permutations (matchings) for all of these cases. Hence D is updated to $\frac{4}{1} \cdot 1 = 4$. Let X be one of the children, selected at random, say $X = (2)$. C remains equal to 0.

3. **Iteration 2**. X has potentially 3 children (permutation vectors of length 2): $(2, 1), (2, 2)$ and $(2, 4)$. However, the permutation vector $(2, 2)$ is ruled out, because it corresponds to the edges $(1, 2)$ and $(2, 2)$, which violates the requirement that no two edges can have the same end vertices. D is therefore updated to $2 \cdot 4 = 8$, and X becomes a randomly chosen child, say $(2, 1)$. C remains 0.

4. **Iteration 3**. $X = (2, 1)$ only has one child $(2, 1, 3)$, which is not a leaf. So D and C remain unaltered, and X becomes $(2, 1, 3)$.

5. **Iteration 4**. Again, there is only one child: $(2, 1, 3, 4)$, which is a leaf. D remains equal to 8, and C becomes $0 + 8 = 8$. The algorithm then terminates.

We see that the algorithm returns $C = 8$ with probability 1/4 and $C = 4$ with probability 3/4, which has an expectation of 5 — the total number of perfect matchings.

For our oracle we use the well-known *Hungarian method* [11] to check (in polynomial time) if a partial matching can be extended to a perfect matching.

10.5.3.1 Numerical Results We present here numerical results for two models, one small and one large.

Model 1 (Small Model): Consider the biadjacency matrix A given in Section 9.9.3.1 on Page 333. The number of perfect matchings for the corresponding bipartite graph (i.e., the permanent of A) is $|\mathscr{X}^*| = 266$ obtained using full enumeration.

To assess the performance of SE, we repeated Algorithm 10.4.1 ten times with parameter values ($B = 50$ and $N = 10$), giving estimates

$$264.21, 269.23, 270.16, 268.33, 272.10, 259.81, 271.62, 269.47, 264.86, 273.77 .$$

The overall mean is 268.4, with an estimated relative error of 0.005. The estimated relative error per run is $\sqrt{10}$ larger; that is, 0.0158. Each run took about 2 seconds.

Model 2 (Large Model): Here we generate a random 100×100 biadjacency matrix $A = (a_{ij})$ by setting $a_{ij} = 1$ for 200 uniformly chosen index pairs (i, j). The other 9800 entries are set to 0. We perform two SE experiments: one with $B = 100$ and $N = 100$; the other with $B = 1$ and $R = 10000$. For the first case the results were (times 10^7)

$$4.06, 3.95, 3.63, 3.72, 3.93, 3.93, 3.48, 3.70, 3.80, 3.90,$$

with an overall mean of $3.81 \cdot 10^7$ and an estimated relative error per run of 0.063 (overall 0.0147). The average CPU time per run was 340 seconds. The second case (Knuth's estimator) gave the estimates (times 10^7):

$$3.48, 2.98, 4.46, 3.48, 3.43, 3.97, 5.02, 4.27, 3.72, 3.61,$$

with an average CPU time per run of 400 seconds. The overall mean is $3.84 \cdot 10^7$ and the estimated relative error per run is 0.1555 (overall 0.0492).

Clearly, running 100 paths in parallel is beneficial here, as the relative error is less than half of that of Knuth's estimator. Note that after compensating for the increased budget (by taking N 100 times larger) Knuth's estimator actually takes longer to run.

10.6 APPLICATION OF SE TO NETWORK RELIABILITY

Consider a network reliability problem as discussed in Sections 5.4.1 and 9.7 where all edge failure probabilities are equal to some small number q. Under this simplified setting the *Spectra* method gives an efficient approach to calculate the network unreliability $\overline{r}(q)$. We briefly describe the Spectra approach.

Let the network edges (links) be labeled $1, \ldots, n$. Edges can be working (*up*) or failed (*down*). Similarly, the network is either *UP* or *DOWN*. Suppose, that initially all links are down and thus the network is in the *DOWN* state, and let $\boldsymbol{\pi} = (i_1, \ldots, i_n)$ be a permutation of edges. Given the permutation $\boldsymbol{\pi}$, start "repairing" edges (change the edges state from *down* to *up*) moving through the permutation from left to right and check the *UP/DOWN* state of the network after each step. Find the index k of the first edge i_k, $k = 1, \ldots, n$, for which the network switches from *DOWN* to *UP*. This index k is called the *construction anchor* of $\boldsymbol{\pi}$ and is denoted by $a^{\mathrm{C}}(\boldsymbol{\pi})$. The total number of edge permutations that have construction anchor k is denoted by $A^{\mathrm{C}}(k)$. The probability vector

$$\frac{1}{n!}\left(A^{\mathrm{C}}(1), \ldots, A^{\mathrm{C}}(n)\right)$$

is called the *construction spectra*, or simply *C-spectra* of the network. A similar notion is that of *destruction spectra* or *D-spectra*, which is obtained by starting with a network in which all links are up and in which edges are destroyed in the order of the permutation. In particular, given an edge permutation $\boldsymbol{\pi} = (i_1, \ldots, i_n)$, let a^{D} be the first edge i_j for which the system becomes *DOWN* and let $A^{\mathrm{D}}(k)$ be the total number of such *destruction anchors* of level k. Then the D-Spectra is defined as the probability vector

$$\frac{1}{n!}\left(A^{\mathrm{D}}(1), \ldots, A^{\mathrm{D}}(n)\right) .$$

The D-spectra can be obtained from the C-spectra (and vice versa) by reversing the order of the vector: $A^{\mathrm{D}}(k) = A^{\mathrm{C}}(n-k+1), k = 1, \ldots, n$. Namely, by reversing each edge permutation $(i_1, \ldots, i_n)$ to $(i_n, \ldots, i_1)$, we see that the number of permutations for which the network switches from *UP* to *DOWN* after k steps is equal to the number of permutations for which the network switches from *DOWN* to *UP* after $n - k + 1$ steps.

Once the D- or C-spectra is available, one can directly calculate the network unreliability $\overline{r}\,(q)$. Let us define a *failure set* to be a set of edges such that their failure causes the network to be *DOWN*, and denote by $\mathcal{N}_k$ the number of failure sets of size k. It is readily seen that

$$\mathcal{N}_k = \binom{n}{k} \underbrace{\frac{1}{n!} \sum_{j=1}^{k} A^{\mathrm{D}}(j)}_{F_k} . \tag{10.5}$$

This statement has a simple combinatorial explanation: F_k is the fraction of all failure sets of size k among all subsets of size k taken from the set of n components. The vector $(F_1, \ldots, F_N)$ is called the *cumulative D-spectra*.

The probability that in a failure set of size k all of its edges are *down* while all other edges are *up* is $q^k(1-q)^{n-k}$. Because the network is *DOWN* if and only if there is such a failure set, it follows that

$$\overline{r}\,(q) = \sum_{k=1}^{n} \mathcal{N}_k \, q^k (1-q)^{n-k}. \tag{10.6}$$

Thus to calculate the network reliability for any q it suffices to calculate the D-spectra. The latter can be treated as a tree-counting problem, as illustrated in the following example.

■ EXAMPLE 10.9

Consider the simple reliability network with 4 links in Figure 10.14. The system is *UP* if vertices s and t are connected by functioning links.

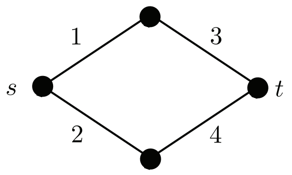

Figure 10.14: Simple network.

To determine the C-spectra, we can construct a permutation tree as in Figure 10.15. Each path in the tree corresponds to a permutation $(i_1, \ldots, i_k)$ up to the corresponding construction anchor k. Let the cost of the leaf that ends this path be equal to $1/(n(n-1)\cdots(n-k)) = (n-k)!/n!$. Then the k-th component of the C-spectra, $A^{\mathrm{C}}(k)/n!$, is equal to the total cost of the leaves in level k of the tree. For example, the 4 leaves at level $k=2$ all have cost $1/12$, and the 16 leaves at level $k=3$ all have cost $1/24$. It is clear that none of the construction anchors are 1 or 4. It follows that the C-spectra is given by $(0, 1/3, 2/3, 0)$, the D-spectra by $(0, 2/3, 1/3, 0)$, and the cumulative D-spectra by $(0, 2/3, 1, 1)$.

Using (10.6), the unreliability is given by

$$\overline{r}(q) = 6 \cdot \frac{2}{3} \cdot q^2(1-q)^2 + 4 \cdot 1 \cdot q^3(1-q)^2 + 1 \cdot 1 \cdot q^4 = 4q^2 - 4q^3 + q^4 = (1-(1-q)^2)^2 .$$

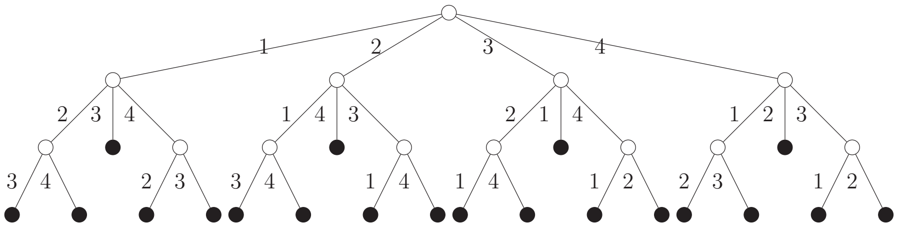

Figure 10.15: Permutation tree of the simple network.

For a general network, estimating the unreliability thus involves the following steps:

1. Construct the permutation tree of the network, where each path ends in a construction anchor point.
2. For each level k set the cost of each leaf to $(n-k)!/n!$, and add all these costs to find the k-th C-spectra.
3. From the C-spectra derive the D-spectra and the cumulative D-spectra.
4. Compute the unreliability via (10.6).

For large n constructing the whole permutation tree is infeasible. Instead, SE can be used to *estimate* the spectra, by estimating the cost of the tree at each level $k = 1, \ldots, n$. In particular, Algorithm 10.4.1 can be directly applied, with the obvious modification that for each level (iteration) the estimated cost at that level, $c(\mathcal{X})/|\mathcal{X}|$ has to be returned.

10.6.1 Numerical Results

In this section we present a numerical comparison between the PMC Algorithm 5.4.2 and the SE Algorithm 10.4.1 for two models.

Recall that the crucial parameter of the SE algorithm is its budget B. Clearly, increasing B means an increase in computation effort (linear in B). In both examples we choose $B = 10$ and repeat the algorithm for $N = 1000$ times. We find numerically that this parameter selection is sufficient for reliable estimation of the spectra. Based on numerous experiments with different models, our general advice for choosing the budget is as follows: Start from small budget, say $B \leqslant 5$ and increase it gradually. As soon as the rare-event probability of interest stops to increase, fix this budget and perform the main experiment. This follows from the fact that the SE tends to underestimate the rare-event probabilities.

Model 1 (Dodecahedron Network): Consider the dodecahedron network in Figure 9.4. Suppose that the network is functioning if vertices 10 and 19 are connected. For this example the estimation of the components of the D-spectra is not a rare-event estimation problem. For such models PMC and SE deliver accurate and comparable estimations for small failure probabilities.

We run the SE algorithm with $N = 1000$ and $B = 10$ and the spectra algorithm with $N = 350000$. The running time for both algorithms is about 4.7 seconds. In this example the minimal D-spectra component has the value $\approx 5 \cdot 10^{-4}$, so both algorithms deliver excellent performances in a reasonable amount of time. Table 10.2 provides typical values obtained during the algorithm's execution, and Table 10.3 summarizes the corresponding probability that the network is *DOWN*.

Table 10.2: PMC and SE spectra estimations for the dodecahedron graph.

Cum. D-spectra	SE	PMC	Cum. D-spectra	SE	PMC
$F(1)$	0	0	$F(16)$	$7.90 \cdot 10^{-1}$	$7.91 \cdot 10^{-1}$
$F(2)$	0	0	$F(17)$	$8.48 \cdot 10^{-1}$	$8.49 \cdot 10^{-1}$
$F(3)$	$5.01 \cdot 10^{-4}$	$4.90 \cdot 10^{-4}$	$F(18)$	$8.91 \cdot 10^{-1}$	$8.93 \cdot 10^{-1}$
$F(4)$	$2.22 \cdot 10^{-3}$	$2.18 \cdot 10^{-3}$	$F(19)$	$9.24 \cdot 10^{-1}$	$9.26 \cdot 10^{-1}$
$F(5)$	$6.30 \cdot 10^{-3}$	$6.33 \cdot 10^{-3}$	$F(20)$	$9.49 \cdot 10^{-1}$	$9.51 \cdot 10^{-1}$
$F(6)$	$1.41 \cdot 10^{-2}$	$1.43 \cdot 10^{-2}$	$F(21)$	$9.67 \cdot 10^{-1}$	$9.68 \cdot 10^{-1}$
$F(7)$	$2.90 \cdot 10^{-2}$	$2.86 \cdot 10^{-2}$	$F(22)$	$9.80 \cdot 10^{-1}$	$9.80 \cdot 10^{-1}$
$F(8)$	$5.53 \cdot 10^{-2}$	$5.45 \cdot 10^{-2}$	$F(23)$	$9.89 \cdot 10^{-1}$	$9.88 \cdot 10^{-1}$
$F(9)$	$9.80 \cdot 10^{-2}$	$9.68 \cdot 10^{-2}$	$F(24)$	$9.94 \cdot 10^{-1}$	$9.94 \cdot 10^{-1}$
$F(10)$	$1.65 \cdot 10^{-1}$	$1.65 \cdot 10^{-1}$	$F(25)$	$9.97 \cdot 10^{-1}$	$9.97 \cdot 10^{-1}$
$F(11)$	$2.62 \cdot 10^{-1}$	$2.60 \cdot 10^{-1}$	$F(26)$	$9.99 \cdot 10^{-1}$	$9.99 \cdot 10^{-1}$
$F(12)$	$3.81 \cdot 10^{-1}$	$3.79 \cdot 10^{-1}$	$F(27)$	1	1
$F(13)$	$5.03 \cdot 10^{-1}$	$5.03 \cdot 10^{-1}$	$F(28)$	1	1
$F(14)$	$6.19 \cdot 10^{-1}$	$6.18 \cdot 10^{-1}$	$F(29)$	1	1
$F(15)$	$7.15 \cdot 10^{-1}$	$7.15 \cdot 10^{-1}$	$F(30)$	1	1

Table 10.3: PMC and SE reliability estimations for the dodecahedron graph.

	SE		PMC	
q	$\widehat{\overline{r}(q)}$	$\widehat{\mathrm{Var}\,(\overline{r}(q))}$	$\widehat{\overline{r}(q)}$	$\widehat{\mathrm{Var}\,(\overline{r}(q))}$
10^{-5}	$2.01 \cdot 10^{-15}$	$2.20 \cdot 10^{-32}$	$1.96 \cdot 10^{-15}$	$1.54 \cdot 10^{-32}$
10^{-4}	$2.00 \cdot 10^{-12}$	$2.20 \cdot 10^{-26}$	$1.96 \cdot 10^{-12}$	$1.53 \cdot 10^{-26}$
10^{-3}	$2.01 \cdot 10^{-9}$	$2.18 \cdot 10^{-20}$	$1.97 \cdot 10^{-9}$	$1.48 \cdot 10^{-20}$
10^{-2}	$2.07 \cdot 10^{-6}$	$2.02 \cdot 10^{-14}$	$2.03 \cdot 10^{-6}$	$1.03 \cdot 10^{-14}$
10^{-1}	$2.86 \cdot 10^{-3}$	$1.49 \cdot 10^{-8}$	$2.84 \cdot 10^{-3}$	$1.90 \cdot 10^{-9}$

Model 2 (Hypercube Graph): For our second model we consider the hypercube graph of order 5, denoted H_5. This is a regular graph with $2^5 = 32$ vertices and $5 \cdot 2^4 = 80$ edges; see also Figure 9.7 on Page 331. Each vertex corresponds to a binary vector $(x_1, \ldots, x_5)$, and two vertices are connected by an edge whenever their Hamming distance (minimum number of substitutions required to change one

Table 10.5: Estimated spectra components.

Algorithm	$F(4)$	$F(5)$	$F(6)$	$F(7)$	$F(8)$	$F(9)$
SE	$8.93 \cdot 10^{-8}$	$4.96 \cdot 10^{-7}$	$1.67 \cdot 10^{-6}$	$4.42 \cdot 10^{-6}$	$9.60 \cdot 10^{-6}$	$2.03 \cdot 10^{-5}$
PMC	0	0	0	0	$1.00 \cdot 10^{-5}$	$2.67 \cdot 10^{-5}$

vector into the other) is 1. Suppose that the network functions when vertices $(0, 0, 0, 0, 0)$ and $(1, 1, 0, 0, 0)$ are connected.

We run the SE Algorithm 10.4.1 with $N = 1000$ and $B = 10$ and the PMC Algorithm 5.4.2 with $N = 300000$. The running time for both SE and PMC is 28 seconds. Table 10.4 provides the estimated network unreliability with both PMC and SE algorithms.

Table 10.4: PMC and SE based reliability estimations for H_5 graph.

	SE		PMC	
q	$\widehat{\overline{r}(q)}$	$\widehat{\mathrm{Var}\,(\overline{r}(q))}$	$\widehat{\overline{r}(q)}$	$\widehat{\mathrm{Var}\,(\overline{r}(q))}$
10^{-5}	$1.94 \cdot 10^{-25}$	$7.26 \cdot 10^{-52}$	$2.32 \cdot 10^{-39}$	$1.15 \cdot 10^{-48}$
10^{-4}	$1.94 \cdot 10^{-20}$	$7.24 \cdot 10^{-42}$	$2.31 \cdot 10^{-30}$	$1.14 \cdot 10^{-38}$
10^{-3}	$1.93 \cdot 10^{-15}$	$7.10 \cdot 10^{-32}$	$2.20 \cdot 10^{-21}$	$1.01 \cdot 10^{-28}$
10^{-2}	$2.05 \cdot 10^{-10}$	$5.94 \cdot 10^{-22}$	$1.38 \cdot 10^{-12}$	$3.45 \cdot 10^{-19}$
10^{-1}	$1.94 \cdot 10^{-5}$	$1.28 \cdot 10^{-12}$	$2.07 \cdot 10^{-5}$	$1.25 \cdot 10^{-11}$

Note the difference in the *DOWN* probabilities for small values of q. The PMC Algorithm 5.4.2 cannot estimate rare-event probabilities; hence the delivered estimate is underestimating the true probability. Table 10.5 summarizes the first components of the obtained spectra with PMC and SE algorithms respectively. We do not report $F(0), \ldots, F(3)$, because they were evaluated as zero by full enumeration. With the same procedure, we obtain the exact value of $F(4)$, which is equal to $8.3195 \cdot 10^{-8}$, close (about 7%) to the reported value of $8.93 \cdot 10^{-8}$.

Next, we run the PMC Algorithm 5.4.2 with sample sizes $10^6, 10^7$, and 10^8, respectively. It is not surprising that $N = 10^6$ and $N = 10^7$ are insufficient to provide a meaningful estimator for the rare-event probability of $8.3195 \cdot 10^{-8}$. Table 10.6 summarizes the results obtained for $N = 10^8$. The running time of PMC is about 9000 seconds and the estimated variance for the rare-event probabilities is larger than the one obtained by SE.

Table 10.6: PMC reliability estimation for $N = 10^8$.

q	$\widehat{\overline{r}(q)}$	$\widehat{\mathrm{Var}\,(\overline{r}(q))}$
10^{-5}	$1.80 \cdot 10^{-25}$	$5.80 \cdot 10^{-51}$
10^{-4}	$1.80 \cdot 10^{-20}$	$5.74 \cdot 10^{-41}$
10^{-3}	$1.81 \cdot 10^{-15}$	$5.18 \cdot 10^{-31}$
10^{-2}	$1.88 \cdot 10^{-13}$	$1.85 \cdot 10^{-21}$
10^{-1}	$2.00 \cdot 10^{-5}$	$2.66 \cdot 10^{-14}$

PROBLEMS

10.1 As a persuasive application of the DFS algorithm, consider the 8-queens problem of Example 6.13. Construct a search tree for all the ways the queens can be arranged such that no queen can capture another. Each vertex $(x_1, \ldots, x_t)$ of the tree represents a valid arrangement of the queens in rows $1, \ldots, t$. At the first level of the tree there are 8 possible positions for the first queen, but to avoid symmetric solutions, you may assume that the first queen can only occupy positions in the first 4 columns, so that the children of the root ( ) vertex are $(1), (2), (3), (4)$. At the second level, the children of, for example, vertex (1) are $(2, 3), \ldots, (2, 8)$. And so on. Implement a DFS algorithm to find all solutions $(x_1, \ldots, x_8)$ of the 8-queens problem. Note that two queens at grid positions (a, b) and (c, d), where $a, b, c, d \in \{1, \ldots, 8\}$, share the same diagonal if $(a - c)/(b - d)$ is equal to 1 or -1.

10.2 In Example 10.4 show that Knuth's algorithm returns an estimator C with $\mathbb{E}[C] = 6$ and $\mathrm{Var}(C) = 2$.

10.3 It is difficult to guess in advance what B to choose, and also if it is better to perform one run of Algorithm 10.4.1 with budget B, or B runs with budget 1. For both trees in Figure 10.16, find out which scenario is better: two runs with $B = 1$ or one run with $B = 2$.

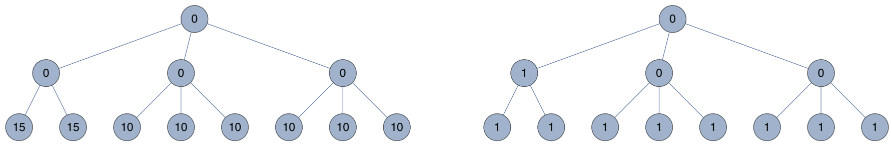

Figure 10.16: Example trees and weights.

10.4 Implement Algorithm 10.4.1 to estimate the number of paths in the $(7, 4)$-Petersen graph of Figure 10.4.

10.5 Consider the 2-SAT CNF formula $C_1 \wedge C_2 \wedge \ldots \wedge C_{n-1}$, where $C_i = l_i \vee \bar{l}_{i+1},\ i = 1, \ldots, n - 1$.

a) Show that the total number of solutions is $|\mathscr{X}^*| = n + 1$ and find these solutions.

b) For each $\mathbf{x} \in \mathscr{X}^*$ give the probability that Knuth's algorithm terminates at $\mathbf{x}$.

c) Implement SE for the cases $n = 21, n = 51, n = 101$, and $n = 151$, using the parameter choices $(B = 1, N = 500), (B = 5, N = 100), (B = 50, N = 10)$, and $(B = 100, N = 1)$. For each case report the estimated relative error and CPU time (for $N = 1$ leave the relative error blank).

10.6 Consider the 2-SAT formula $C_1 \wedge C_2 \wedge \ldots \wedge C_n$ where $C_i = l_i \vee l_{i+1},\ i = 1, \ldots, n$. Implement an SE algorithm to estimate the number of valid assignments for any size m and any budget B. Give the number of valid assignments for $n = 1, \ldots, 10$. What is the pattern?

10.7 Implement an SE algorithm to find the permanent of the 50×50 binary matrix A defined by the following algorithm, which is based on a simple linear congruential generator. Compare the results for two sets of parameter choices: $B = 50$ and $N = 100$ versus $B = 1$ and $N = 5000$.

```
a ← $7^5$,  x ← 12345,  m ← $2^{31} - 1$
for i = 1 to 50 do
    for j = 1 to 50 do
        $x \leftarrow a\,x \bmod m$
        if $x/m < 0.1$ then
            $A(i,j) \leftarrow 1$
        else
            $A(i,j) \leftarrow 0$
```

10.8 The 3×3 grid graph in Figure 10.17 is functioning if the black vertices are connected by functioning links. Each of the 12 links has a failure probability of q and the links fail independently of each other.

a) Compute the construction and destruction spectra via Algorithm 10.4.1, with the modification that the estimated cost at each level of the permutation tree is returned, rather than the total cost.

b) Using the spectra and formula (10.6), draw a graph of the unreliability on a log-log scale from $q = 10^{-30}$ to $q = 1$.

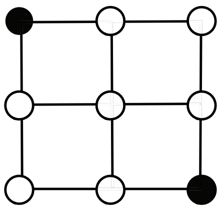
Figure 10.17: Grid graph.

Further Reading

Closely related accounts of stochastic enumeration appeared first in [13], [14], and [15]. It was realized in [16] that the approach is a generalization of Knuth's sequential importance sampling scheme for counting the cost of trees [10], which greatly simplifies the exposition of the algorithm and enhances its applicability.

The use of fast decision-making algorithms (oracles) for solving NP-hard problems is very common in Monte-Carlo methods; see, for example [8], which presents a *fully polynomial randomized approximation scheme* (FPRAS) for network unreliability based on the well-known DNF polynomial counting algorithm (oracle) of [9]. Another example is given in [4], where Kruskal's spanning trees algorithm is used for network reliability estimation. The use of SE for network reliability estimation was introduced in [17]. Wei and Selman proposed the `ApproxCount` algorithm in [18]. It is a *local search* method that uses Markov chain Monte Carlo (MCMC) sampling to approximate the true counting quantity. For SAT counting Gogate and Dechter [6, 7] devised an alternative counting technique called `SampleMinisat`, based on sampling from the search space of a Boolean formula through `SampleSearch`, using importance sampling.

REFERENCES

1. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2009.
2. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
3. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.
4. T. Elperin, I. B. Gertsbakh, and M. Lomonosov. Estimation of network reliability using graph evolution models. *IEEE Transactions on Reliability*, 40(5):572–581, 1991.
5. P. Erdös and A. Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.
6. V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, volume 1 of *AAAI'07*, pages 198–203. AAAI Press, 2007.
7. V. Gogate and R. Dechter. Samplesearch: Importance sampling in presence of determinism. *Artificial Intelligence*, 175(2):694–729, 2011.
8. D. R. Karger. A randomized fully polynomial time approximation scheme for the all terminal network reliability problem. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, STOC '95, pages 11–17, New York, 1995.
9. R. M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.
10. D. E. Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29, 1975.
11. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.
12. B. Roberts and D.P. Kroese. Estimating the number of s-t paths in a graph. *Journal of Graph Algorithms and Applications*, 11(1):195–214, 2007.
13. R. Y. Rubinstein. Stochastic enumeration method for counting NP-hard problems. *Methodology and Computing in Applied Probability*, 15(2):249–291, 2011.

14. R. Y. Rubinstein, A. Ridder, and R. Vaisman. *Fast Sequential Monte Carlo Methods*. John Wiley & Sons, New York, 2013.

15. R. Vaisman. *Stochastic Enumeration Method for Counting, Rare-Events and Optimization*. PhD thesis, Technion, Israel Institute of Technology, Haifa, September 2013.

16. R. Vaisman and D. P. Kroese. Stochastic enumeration method for counting trees. *Methodology and Computing in Applied Probability*, pages 1–43, 2015. doi: 10.1007/s11009-015-9457-4.

17. R. Vaisman, D. P. Kroese, and I. B. Gertsbakh. Improved sampling plans for combinatorial invariants of coherent systems. *IEEE Transactions on Reliability*, 2015. doi: 10.1109/TR.2015.2446471.

18. W. Wei and B. Selman. A new approach to model counting. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing*, SAT'05, pages 324–339, Berlin, 2005. Springer-Verlag.

ABBREVIATIONS AND ACRONYMS

BFS	breadth-first search
cdf	cumulative distribution function
CE	cross-entropy
CMC	crude Monte Carlo
CNF	conjunctive normal form
DEDS	discrete-event dynamical system
DESS	discrete-event statical system
DES	discrete-event simulation
DFS	depth-first search
DNF	disjunctive normal form
ECM	exponential change of measure
FPAUS	fully polynomial almost uniform sampler
FPRAS	fully polynomial randomized approximation scheme
GS	generalized splitting
HMM	hidden Markov model
iid	independent and identically distributed
ITLR	inverse-transform likelihood ratio
KKT	Karush–Kuhn–Tucker

MRG	Multiple-recursive generator
max-cut	maximal cut
MCMC	Markov chain Monte Carlo
MinxEnt	minimum cross-entropy
pdf	probability density function (both discrete and continuous)
PERT	program evaluation and review technique
PMC	permutation Monte Carlo
RSAT	random SAT
SAT	satisfiability problem
SAW	self-avoiding walk
SDE	stochastic differential equation
SE	stochastic enumeration
SF	score function
SIS	sequential importance sampling
SIR	sequential importance resampling
SLR	standard likelihood ratio
TLR	transform likelihood ratio
TSP	traveling salesman problem
VM	variance minimization

LIST OF SYMBOLS

$\gg$	much greater than
$\propto$	proportional to
$\sim$	is distributed according to
$\approx$	approximately
$\top$	transpose
∇	∇f is the gradient of f
∇^2	$\nabla^2 f$ is the Hessian of f
$\mathbb{E}$	expectation
$\mathbb{N}$	set of natural numbers $\{0, 1, \ldots\}$
$\mathbb{P}$	probability measure
$\mathbb{R}$	the real line = one-dimensional Euclidean space
$\mathbb{R}^n$	n-dimensional Euclidean space
$\mathcal{D}$	Kullback–Leibler CE
$\mathcal{H}$	Shannon entropy
$\mathcal{M}$	mutual information
$\mathcal{S}$	score function

Ber	Bernoulli distribution
Beta	beta distribution
Bin	binomial distribution
Exp	exponential distribution
G	geometric distribution
Gamma	gamma distribution
N	normal or Gaussian distribution
Pareto	Pareto distribution
Poi	Poisson distribution
U	uniform distribution
Weib	Weibull distribution
α	smoothing parameter or acceptance probability
γ	level parameter
ζ	cumulant function (log of moment generating function)
ϱ	rarity parameter
$D(\mathbf{v})$	objective function for CE minimization
f	probability density (discrete or continuous)
g	importance sampling density
I_A	indicator function of event A
$\ln$	(natural) logarithm
N	sample size
$\mathcal{O}$	big-O order symbol
S	performance function
$S_{(i)}$	i-th order statistic
$\mathbf{u}$	nominal reference parameter (vector)
$\mathbf{v}$	reference parameter (vector)
$\widehat{\mathbf{v}}$	estimated reference parameter
$\mathbf{v}^*$	CE optimal reference parameter
${}_*\mathbf{v}$	VM optimal reference parameter
$V(\mathbf{v})$	objective function for VM minimization
W	likelihood ratio
$\mathbf{x}, \mathbf{y}$	vectors
$\mathbf{X}, \mathbf{Y}$	random vectors
$\mathscr{X}, \mathscr{Y}$	sets

INDEX