Table 8.10: Progression of the CE algorithm for the Hammersley TSP.

t	S_t^b	$\widehat{\gamma}_t$
1	11.0996	13.2284
2	10.0336	11.8518
3	9.2346	10.7385
4	8.27044	9.89423
5	7.93992	9.18102
6	7.54475	8.70609
7	7.32622	8.27284
8	6.63646	7.94316
9	6.63646	7.71491
10	6.61916	7.48252
11	6.43016	7.25513
12	6.20255	7.07624
13	6.14147	6.95727
14	6.12181	6.76876
15	6.02328	6.58972
16	5.95643	6.43456
17	5.89489	6.31772
18	5.83683	6.22153
19	5.78224	6.18498
20	5.78224	6.1044
21	5.78224	6.0983
22	5.78224	6.06036
23	5.78224	6.00794
24	5.78224	5.91265
25	5.78224	5.86394
26	5.78224	5.86394
27	5.78224	5.83645
28	5.78224	5.83645
29	5.78224	5.83645

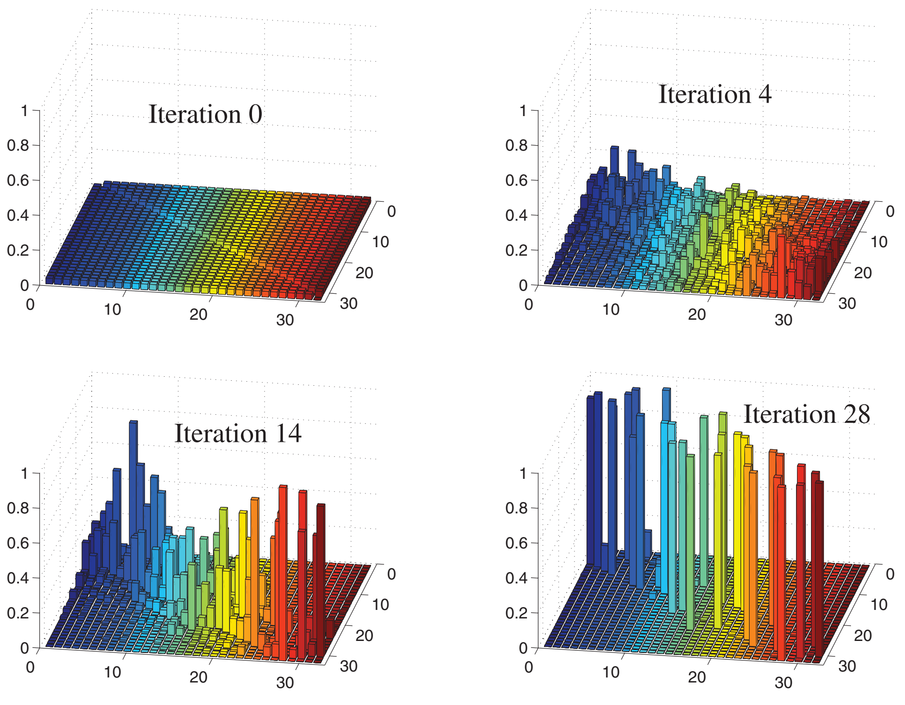

Figure 8.8: Evolution of $\mathbf{P}_t$ in the CE algorithm for the Hammersley TSP.

The optimal tour length for the Hammersley problem is $\gamma^* = 5.78224$ (rounded), which coincides with $\widehat{\gamma}_{29}$ found in Table 8.10. A corresponding solution (optimal tour) is (1, 5, 9, 17, 13, 11, 15, 18, 22, 26, 23, 19, 21, 25, 29, 27, 31, 30, 32, 28, 24, 20, 16, 8, 12, 14, 10, 6, 4, 2, 7, 3), depicted in Figure 8.9. There are several other optimal tours (see Problem 8.14), but all exhibit a straight line through the points (10,10)/32, (14,14)/32, (17,17)/32, and (21,21)/32.

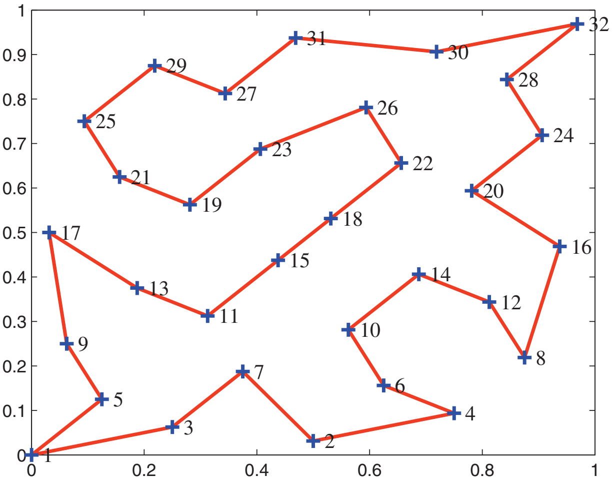

Figure 8.9: An optimal tour through the Hammersley points.

8.6.1 Incomplete Graphs

The easiest way to deal with TSPs on incomplete graphs is, as already remarked in Example 6.12, to make the graph complete by acquiring extra links with infinite cost. However, if many entries in the cost matrix are infinite, most of the generated tours in Algorithm 8.6.1 will initially be invalid (yield a length of ∞). A better way of choosing $\mathbf{P}_0 = (p_{0,ij})$ is then to assign smaller initial probabilities to pairs of nodes for which no direct link exists. In particular, let d_i be the *degree* of node i, that is, the number of finite entries in the i-th row of the matrix C. We can then proceed as follows:

1. If $c_{ij} = \infty$, set $p_{0,ij}$ to $\frac{1-\delta}{d_i}$, where δ is a small number, say $\delta = 0.1$. Set the remaining elements to ε, except for $p_{0,ii} = 0$. Since the rows of $\mathbf{P}_0$ sum up to 1, we have $\varepsilon = \frac{\delta}{n-d_i-1}$.

2. Keep the$p_{0,ij} = \varepsilon = \frac{\delta}{n-d_i-1}$ above for *all iterations* of the CE Algorithm 8.3.1.

Since δ is the sum of all $p_{t,ij}$ corresponding to the ∞ elements in the i-th row of C, and since all such $p_{t,ij}$ are equal to each other (ε), we can generate a transition from each state i using only a $(d_i + 1)$-point distribution rather than the n-point distribution formed by the i-th row of $\widehat{\mathbf{P}}_t$. Indeed, if we *relabel* the elements of this row such that the first d_i entries correspond to existing links, while the next $n - d_i - 1$ correspond to nonexisting links, then we obtain the following faster procedure for generating transitions:

Algorithm 8.6.3: A Fast Procedure for Generating Transitions

```
1 Generate a random variable $U \sim \mathsf{U}(0,1)$.
2 if $U \leqslant 1 - \delta$ then
3     Generate the next transition from the discrete $d_i$-point pdf with
        probabilities $p_{t,ij}/(1-\delta)$, $j = 1, \ldots, d_i$.
4 else
5     Generate the next transition by drawing a discrete random variable $Z$
        uniformly distributed over the points $d_i + 1, \ldots, n-1$ (recall that these
        points correspond to the $\infty$ elements in the $i$-th row of $C$).
```

It is important to note that the small elements of $\mathbf{P}_0$ corresponding to infinities in matrix C should be kept the same from iteration to iteration rather than being updated. By doing so, one obtains considerable speedup in trajectory generation.

8.6.2 Node Placement

We now present an alternative algorithm for trajectory generation due to Margolin [33] called the node *placement algorithm.* In contrast to Algorithm 8.6.1, which generates *transitions* from node to node (based on the transition matrix $P = (p_{ij})$), in Algorithm 8.6.4 below, a similar matrix

$$\mathbf{P} = \begin{pmatrix} p_{(1,1)} & p_{(1,2)} & \cdots & p_{(1,n)} \\ p_{(2,1)} & p_{(2,2)} & \cdots & p_{(2,n)} \\ \vdots & \vdots & \vdots & \vdots \\ p_{(n,1)} & p_{(n,2)} & \cdots & p_{(n,n)} \end{pmatrix} \tag{8.42}$$

generates *node placements.* Specifically, $p_{(i,j)}$ corresponds to the probability of node i being visited at the j-th place in a tour of n cities. In other words, $p_{(i,j)}$ can be viewed as the probability that city (node) i is "arranged" to be visited at the j-th place in a tour of n cities. More formally, a *node placement vector* is a vector $\mathbf{y} = (y_1, \ldots, y_n)$ such that y_i denotes the place of node i in the tour $\mathbf{x} = (x_1, \ldots, x_n)$. The precise meaning is given by the correspondence

$$y_i = j \quad \Longleftrightarrow \quad x_j = i \,, \tag{8.43}$$

for all $i, j \in \{1, \ldots, n\}$. For example, the node placement vector $\mathbf{y} = (3, 4, 2, 6, 5, 1)$ in a six-node network defines uniquely the tour $\mathbf{x} = (6, 3, 1, 2, 5, 4)$. The performance of each node placement $\mathbf{y}$ can be defined as $\overline{S}(\mathbf{y}) = S(\mathbf{x})$, where $\mathbf{x}$ is the unique path corresponding to $\mathbf{y}$.

Algorithm 8.6.4: Trajectory Generation Using Node Placements

```
1 Define $\mathbf{P}^{(1)} \leftarrow \mathbf{P}$. Let $k \leftarrow 1$.
2 for $t = 1$ to $n$ do
3     Generate $Y_t$ from the distribution formed by the $t$-th row of $\mathbf{P}^{(t)}$.
4     Obtain the matrix $\mathbf{P}^{(t+1)}$ from $\mathbf{P}^{(t)}$ by first setting the $Y_t$-th column of
        $\mathbf{P}^{(t)}$ to 0 and then normalizing the rows to sum up to 1.
5 Determine the tour by (8.43) and evaluate the length of the tour by (8.34).
```

It is readily seen that the updating formula for $p_{(i,j)}$ is now

$$\widehat{p}_{(i,j)} = \frac{\sum_{k=1}^{N} I_{\{\overline{S}(\mathbf{Y}_k) \leqslant \gamma\}} \, I_{\{Y_{ki}=j\}}}{\sum_{k=1}^{N} I_{\{\overline{S}(\mathbf{Y}_k) \leqslant \gamma\}}} \, . \tag{8.44}$$

Our simulation results with the TSP and other problems do not indicate clear superiority of either Algorithm 8.6.1 or Algorithm 8.6.4 in terms of the efficiency (speed and accuracy) of the main CE Algorithm 8.3.1.

8.6.3 Case Studies

To illustrate the accuracy and robustness of the CE algorithm, we applied the algorithm to a number of benchmark problems from the TSP library

`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/`

In all cases the *same* set of CE parameters were chosen: $\varrho = 0.03$, $\alpha = 0.7$, $N = 5\,n^2$; and we use the stopping rule (8.21) with the parameter $d = 3$.

Table 8.11 presents the performance of Algorithm 8.3.1 for a selection of *symmetric* TSPs from this library. To study the variability in the solutions, each problem was repeated 10 times. In the table, *min*, *mean*, and *max* denote the smallest (i.e., best), average, and largest of the 10 estimates for the optimal value. The true optimal value is denoted by γ^*.

The average CPU time in seconds and the average number of iterations are given in the last two columns. The size of the problem (number of nodes) is indicated in its name. For example, `st70` has $n = 70$ nodes. Similar case studies for the *asymmetric case* can be found in Table 2.5 of [45].

At this end, note that CE is ideally suitable for parallel computation, since parallel computing speeds up the process by almost a factor of r, where r is the number of parallel processors.

One might wonder why the CE Algorithm 8.2.1, with such simple updating rules and quite arbitrary parameters α and ϱ, performs so nicely for combinatorial optimization problems. A possible explanation is that the objective function S for combinatorial optimization problems is typically close to being additive; see, for example, the objective function S for the TSP problem in (8.34). For other optimization problems (e.g., optimizing complex multiextremal continuous functions), one needs to make a more careful and more conservative choice of the parameters α and ϱ.

Table 8.11: Case studies for the TSP.

File	γ^*	Min	Mean	Max	CPU	$\bar{T}$
burma14	3323	3323	3325.6	3336	0.14	12.4
ulysses16	6859	6859	6864	6870	0.21	14.1
ulysses22	7013	7013	7028.9	7069	1.18	22.1
bayg29	1610	1610	1628.6	1648	4.00	28.2
bays29	2020	2020	2030.9	2045	3.83	27.1
dantzig42	699	706	717.7	736	19.25	38.4
eil51	426	428	433.9	437	65.0	63.35
berlin52	7542	7618	7794	8169	64.55	59.9
st70	675	716	744.1	765	267.5	83.7
eil76	538	540	543.5	547	467.3	109.0
pr76	108159	109882	112791	117017	375.3	88.9

8.7 CONTINUOUS OPTIMIZATION

We will briefly discuss how the CE method can be applied to solve continuous optimization problems. Let $S(\mathbf{x})$ be a real-valued function on $\mathbb{R}^n$. To maximize the function via CE, we need to specify a family of parameterized distributions to generate samples in $\mathbb{R}^n$. This family must include, at least in the limiting case, the degenerate distribution that puts all its probability mass on an optimal solution. A simple choice is to use a multivariate normal distribution, parameterized by a mean vector $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)$ and a covariance matrix Σ. When the covariance matrix is chosen to be diagonal — that is, the components of $\mathbf{X}$ are independent — the CE updating formulas become particularly easy. In particular, denoting $\{\mu_i\}$ and $\{\sigma_i\}$ the means and standard deviations of the components, the updating formulas are (see Problem 8.18)

$$\widehat{\mu}_{t,i} = \frac{\sum_{\mathbf{X}_k \in \mathscr{E}_t} X_{ki}}{|\mathscr{E}_t|} \qquad i = 1, \ldots, n\,, \tag{8.45}$$

and

$$\widehat{\sigma}_{t,i} = \sqrt{\frac{\sum_{\mathbf{X}_k \in \mathscr{E}_t} (X_{ki} - \widehat{\mu}_{t,i})^2}{|\mathscr{E}_t|}}, \qquad i = 1, \ldots, n\,, \tag{8.46}$$

where X_{ki} is the i-th component of $\mathbf{X}_k$ and $\mathbf{X}_1, \ldots, \mathbf{X}_n$ is a random sample from $\mathsf{N}(\widehat{\boldsymbol{\mu}}_{t-1}, \widehat{\Sigma}_{t-1})$. In other words, the means and standard deviations are simply updated via the corresponding maximum likelihood estimators based on the elite samples $\mathscr{E}_t = \{\mathbf{X}_k : S(\mathbf{X}_k) \geqslant \widehat{\gamma}_t\}$.

■ **EXAMPLE 8.12 The Peaks Function**

Matlab's `peaks` function,

$$S(\mathbf{x}) = 3\,(1-x_1)^2\,\exp(-(x_1^2)-(x_2+1)^2) - 10\,(x_1/5 - x_1^3 - x_2^5)\,\exp(-x_1^2-x_2^2) \\ - 1/3\,\exp(-(x_1+1)^2 - x_2^2)\;,$$

has various local maxima. In Section A.5 of the Appendix, a simple Matlab implementation of CE Algorithm 8.3.1 is given for finding the global maximum of this function, which is approximately $\gamma^* = 8.10621359$ and is attained at $\mathbf{x}^* = (-0.0093151, 1.581363)$. The choice of the initial value for $\boldsymbol{\mu}$ is not important, but the initial standard deviations should be chosen large enough to ensure initially a close to uniform sampling of the region of interest. The CE algorithm is stopped when all standard deviations of the sampling distribution are less than some small ε.

Figure 8.10 gives the evolution of the worst and best of the elite samples, that is, $\widehat{\gamma}_t$ and S_t^*, for each iteration t. We see that the values quickly converge to the optimal value γ^*.

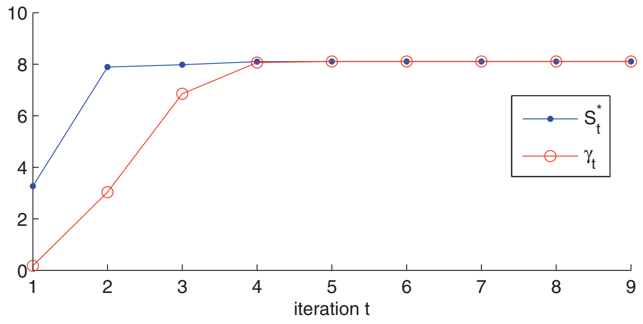

Figure 8.10: Evolution of the CE algorithm for the `peaks` function.

Remark 8.7.1 (Injection) When using the CE method to solve practical optimization problems with many constraints and many local optima, it is sometimes necessary to prevent the sampling distribution from shrinking too quickly. A simple but effective approach is the following *injection* method [6]. Let S_t^* denote the best performance found at the t-th iteration, and (in the normal case) let σ_t^* denote the largest standard deviation at the t-th iteration. If σ_t^* is sufficiently small and $|S_t^* - S_{t-1}^*|$ is also small, then add some small value to each standard deviation, for example, a constant δ or the value $c\,|S_t^* - S_{t-1}^*|$, for some fixed δ and c. When using CE with injection, a possible stopping criterion is to stop after a fixed number of injections.

8.8 NOISY OPTIMIZATION

One of the distinguishing features of the CE Algorithm 8.3.1 is that it can easily handle noisy optimization problems, that is, when the objective function $S(\mathbf{x})$ is corrupted with noise. We denote such a noisy function by $\widehat{S}(\mathbf{x})$. We assume that for each $\mathbf{x}$ we can readily obtain an outcome of $\widehat{S}(\mathbf{x})$, for example, via generation of some additional random vector $\mathbf{Y}$, whose distribution may depend on $\mathbf{x}$.

A classical example of noisy optimization is simulation-based optimization [46]. A typical instance is the *buffer allocation problem*, where the objective is to allocate n buffer spaces among the $m-1$ "niches" (storage areas) between m machines in a serial production line so as to optimize some performance measure, such as the steady-state throughput. This performance measure is typically not available analytically and thus must be estimated via simulation. A detailed description of the buffer allocation problem, and of how CE can be used to solve this problem, is given in [45].

Another example is the noisy TSP, where, say, the cost matrix (c_{ij}), denoted now by $\mathbf{Y} = (Y_{ij})$, is random. Think of Y_{ij} as the random time to travel from city i to city j. The total cost of a tour $\mathbf{x} = (x_1, \ldots, x_n)$ is given by

$$\widehat{S}(\mathbf{x}) = \sum_{i=1}^{n-1} Y_{x_i, x_{i+1}} + Y_{x_n, x_1} \ . \tag{8.47}$$

We assume that $\mathbb{E}[Y_{ij}] = c_{ij}$.

The main CE optimization Algorithm 8.3.1 for deterministic functions $S(\mathbf{x})$ is also valid for noisy ones $\widehat{S}(\mathbf{x})$. Extensive numerical studies [45] with the noisy version of Algorithm 8.3.1 show that it works nicely because, during the course of the optimization, it *filters out* efficiently the noise component from $\widehat{S}(\mathbf{x})$. However, to get reliable estimates of the optimal solution of combinatorial optimization problems, one is required to increase the sample size N by a factor 2 to 5 in each iteration of Algorithm 8.3.1. Clearly, this factor increases with the "power" of the noise.

■ EXAMPLE 8.13 Noisy TSP

Suppose that in the first test case of Table 8.11, `burma14`, some uniform noise is added to the cost matrix. In particular, suppose that the cost of traveling from i to j is given by $Y_{ij} \sim \mathsf{U}(c_{ij} - 8, c_{ij} + 8)$, where c_{ij} is the cost for the deterministic case. The expected cost is thus $\mathbb{E}[Y_{ij}] = c_{ij}$, and the total cost $\widehat{S}(\mathbf{x})$ of a tour $\mathbf{x}$ is given by (8.47). The CE algorithm for optimizing the unknown $S(\mathbf{x}) = \mathbb{E}[\widehat{S}(\mathbf{x})]$ remains exactly the same as in the deterministic case, except that $S(\mathbf{x})$ is replaced with $\widehat{S}(\mathbf{x})$ and a different stopping criterion than (8.21) needs to be employed. A simple rule is to stop when the transition probabilities $\widehat{p}_{t,ij}$ satisfy $\min(\widehat{p}_{t,ij}, 1 - \widehat{p}_{t,ij}) < \varepsilon$ for *all* i and j, similar to Remark 8.4.1. We repeated the experiment 10 times, taking a sample size twice as large as for the deterministic case, that is, $N = 10 \cdot n^2$. For the stopping criterion given above we took $\varepsilon = 0.02$. The other parameters remained the same as those described in Section 8.6.3. CE found the optimal solution eight times, which is comparable to its performance in the deterministic case.

Figure 8.11 displays the evolution of the worst performance of the elite samples $(\widehat{\gamma}_t)$ for both the deterministic and noisy case denoted by $\widehat{\gamma}_{1t}$ and $\widehat{\gamma}_{2t}$, respectively.

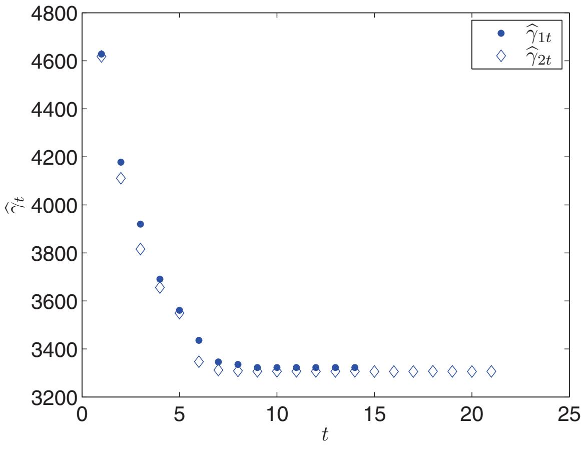

Figure 8.11: Evolution of the worst of the elite samples for a deterministic and noisy TSP.

We see in both cases a similar rapid drop in the level $\widehat{\gamma}_t$. It is important notice that even though here the algorithm in both the deterministic and noisy cases converges to the optimal solution, the $\{\widehat{\gamma}_{2t}\}$ for the noisy case do not converge to $\gamma^* = 3323$, in contrast to the $\{\widehat{\gamma}_{1t}\}$ for the deterministic case. This is because the latter estimates eventually the $(1-\varrho)$-quantile of the deterministic $S(\mathbf{x}^*)$, whereas the former estimates the $(1-\varrho)$-quantile of $\widehat{S}(\mathbf{x}^*)$, which is random. To estimate $S(\mathbf{x}^*)$ in the noisy case, one needs to take the sample average of $\widehat{S}(\mathbf{x}_T)$, where $\mathbf{x}_T$ is the solution found at the final iteration.

8.9 MINXENT METHOD

In the standard CE method for rare-event simulation, the importance sampling density for estimating $\ell = \mathbb{P}_f(S(\mathbf{X}) \geqslant \gamma)$ is restricted to some parametric family, say $\{f(\cdot;\mathbf{v}), \mathbf{v} \in \mathscr{V}\}$, and the optimal density $f(\cdot;\mathbf{v}^*)$ is found as the solution to the *parametric* CE minimization program (8.3). In contrast to CE, we present below a *nonparametric* method called the *MinxEnt* method. The idea is to minimize the CE distance to g^* over *all* pdfs rather than over $\{f(\cdot;\mathbf{v}),\ \mathbf{v} \in \mathscr{V}\}$. However, the program $\min_g \mathcal{D}(g, g^*)$ is void of meaning, since the minimum (zero) is attained at the unknown $g = g^*$. A more useful approach is to first specify a *prior* density h, which conveys the available information on the "target" g^*, and then choose the "instrumental" pdf g as close as possible to h, subject to certain *constraints* on g. If no prior information on the target g^* is known, the prior h is simply taken to be a constant, corresponding to the uniform distribution (continuous or discrete). This leads to the following minimization framework [3], [4], and [22]:

$$(\mathrm{P}_0)\begin{cases} \min_g \mathcal{D}(g,h) = \min_g \int \ln \frac{g(\mathbf{x})}{h(\mathbf{x})}\, g(\mathbf{x})\,\mathrm{d}\mathbf{x} = \min_g \mathbb{E}_g\left[\ln \frac{g(\mathbf{X})}{h(\mathbf{X})}\right] \\ \text{s.t.} \quad \int S_i(\mathbf{x})\,g(\mathbf{x})\,\mathrm{d}\mathbf{x} = \mathbb{E}_g[S_i(\mathbf{X})] = \gamma_i, \quad i = 1,\ldots,m\ , \\ \int g(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1\ . \end{cases} \tag{8.48}$$

Here g and h are n-dimensional pdfs, $S_i(\mathbf{x}),\ i = 1,\ldots,m$, are given functions, and $\mathbf{x}$ is an n-dimensional vector. The program (P_0) is Kullback's *minimum cross-entropy* (MinxEnt) program. Note that this is a *convex* functional optimization problem, because the objective function is a convex function of g, and the constraints are affine in g.

If the prior h is constant, then $\mathcal{D}(g,h) = \int g(\mathbf{x}) \ln g(\mathbf{x})\,\mathrm{d}\mathbf{x} + \text{constant}$, so that the minimization of $\mathcal{D}(g,h)$ in (P_0) can be replaced with the *maximization* of

$$\mathcal{H}(g) = -\int g(\mathbf{x}) \ln g(\mathbf{x})\,\mathrm{d}\mathbf{x} = -\mathbb{E}_g[\ln g(\mathbf{X})]\ , \tag{8.49}$$

where $\mathcal{H}(g)$ is the *Shannon entropy*; see (1.46). (Here we use a different notation to emphasize the dependence on g.) The corresponding program is Jaynes' *MaxEnt* program [20]. Note that the former minimizes the Kullback–Leibler cross-entropy, while the latter maximizes the Shannon entropy [22].

In typical counting and combinatorial optimization problems h is chosen as an n-dimensional pdf with uniformly distributed marginals. For example, in the SAT counting problem, we assume that each component of the n-dimensional random vector $\mathbf{X}$ is $\mathsf{Ber}(1/2)$ distributed. While estimating rare events in stochastic models, like queueing models, h is the fixed underlying pdf. For example, in the $M/M/1$ queue h would be a two-dimensional pdf with independent marginals, where the first marginal is the interarrival $\mathsf{Exp}(\lambda)$ pdf and the second is the service $\mathsf{Exp}(\mu)$ pdf.

The MinxEnt program, which presents a constrained functional optimization problem, can be solved via Lagrange multipliers. The solution for the discrete case is derived in Example 1.19 on page 39. A similar solution can be derived, for example , via calculus of variations [3], for the general case. In particular, the solution of the MinxEnt problem is given [22] by

$$g(\mathbf{x}) = \frac{h(\mathbf{x}) \exp\{\sum_{i=1}^m \lambda_i\, S_i(\mathbf{x})\}}{\mathbb{E}_h\left[\exp\{\sum_{i=1}^m \lambda_i\, S_i(\mathbf{X})\}\right]}\ , \tag{8.50}$$

where $\lambda_i,\ i = 1,\ldots,m$, are obtained from the solution of the following system of equations:

$$\frac{\mathbb{E}_h\left[S_i(\mathbf{X}) \exp\left\{\sum_{j=1}^m \lambda_j\, S_j(\mathbf{X})\right\}\right]}{\mathbb{E}_h\left[\exp\left\{\sum_{j=1}^m \lambda_j\, S_j(\mathbf{X})\right\}\right]} = \gamma_i, \quad i = 1,\ldots,m\ , \tag{8.51}$$

where $\mathbf{X} \sim h(\mathbf{x})$.

Note that $g(\mathbf{x})$ can be written as

$$g(\mathbf{x}) = C(\boldsymbol{\lambda})\, h(\mathbf{x})\, \exp\left\{\sum_{i=1}^m \lambda_i\, S_i(\mathbf{x})\right\}, \tag{8.52}$$

where

$$C^{-1}(\boldsymbol{\lambda}) = \mathbb{E}_h\left[\exp\left\{\sum_{i=1}^{m} \lambda_i\, S_i(\mathbf{X})\right\}\right] \tag{8.53}$$

is the normalization constant. Note also that $g(\mathbf{x})$ is a density function; in particular, $g(\mathbf{x}) \geq 0$.

Consider the MinxEnt program (P_0) with a single constraint, that is,

$$\begin{cases} \quad\ \min_g \mathcal{D}(g,h) = \min_g \mathbb{E}_g\left[\ln \frac{g(X)}{h(X)}\right] \\ \text{s.t.} \quad \mathbb{E}_g[S(X)] = \gamma\ , \\ \quad\ \int g(x)\,\mathrm{d}x = 1\ . \end{cases} \tag{8.54}$$

In this case (8.50) and (8.51) reduce to

$$g(\mathbf{x}) = \frac{h(\mathbf{x})\ \exp\{\lambda\, S(\mathbf{x})\}}{\mathbb{E}_h\left[\exp\{\lambda\, S(\mathbf{X})\}\right]} \tag{8.55}$$

and

$$\frac{\mathbb{E}_h\left[S(\mathbf{X})\exp\left\{\lambda\, S(\mathbf{X})\right\}\right]}{\mathbb{E}_h\left[\exp\left\{\lambda\, S(\mathbf{X})\right\}\right]} = \gamma\ , \tag{8.56}$$

respectively.

In the particular case where $S(\mathbf{x}),\ \mathbf{x} = (x_1, \ldots, x_n)$ is a coordinatewise separable function, that is,

$$S(\mathbf{x}) = \sum_{i=1}^{n} S_i(x_i)\ , \tag{8.57}$$

and the components $X_i,\ i = 1, \ldots, n$ of the random vector $\mathbf{X}$ are independent under $h(\mathbf{x}) = h(x_1)\cdots h(x_n)$, the joint pdf $g(\mathbf{x})$ in (8.55) reduces to the *product of marginal pdfs*. In particular, the i-th component of $g(\mathbf{x})$ can be written as

$$g_i(x) = \frac{h_i(x)\exp\{\lambda S_i(x)\}}{\mathbb{E}_{h_i}\left[\exp\left\{\lambda S_i(x)\right\}\right]}, \quad i = 1, \ldots, n\ . \tag{8.58}$$

Remark 8.9.1 (The MinxEnt Program with Inequality Constraints) It is not difficult to extend the MinxEnt program to contain *inequality* constraints. Suppose that the following M inequality constraints are added to the MinxEnt program (8.48):

$$\mathbb{E}_g[S_i(\mathbf{X})] \geqslant \gamma_i, \qquad i = m+1, \ldots, m+M\ .$$

The solution of this MinxEnt program is given by

$$g(\mathbf{x}) = \frac{h(\mathbf{x})\exp\left\{\sum_{i=1}^{m+M} \lambda_i\, S_i(\mathbf{x})\right\}}{\mathbb{E}_h\left[\exp\left\{\sum_{i=1}^{m+M} \lambda_i\, S_i(\mathbf{X})\right\}\right]}\ , \tag{8.59}$$

where the Lagrange multipliers $\lambda_1, \ldots, \lambda_{m+M}$ are the solutions to the dual convex optimization problem

$$\begin{array}{ll} \displaystyle\max_{\boldsymbol{\lambda},\beta} & \displaystyle\sum_{i=1}^{m+M} \lambda_i\,\gamma_i - \beta - \mathbb{E}_h\left[\exp\left(-1 - \beta + \sum_{i=1}^{m+M} \lambda_i S_i(\mathbf{x})\right)\right] \\ \text{subject to:} & \lambda_i \geqslant 0, \quad i = m+1, \ldots, m+M. \end{array}$$

Thus, only the Lagrange multipliers corresponding to an inequality must be constrained from below by zero. Similar to (1.80), this can be solved in two steps, where β can be determined explicitly as a normalization constant but the $\{\lambda_i\}$ have to be determined numerically.

In the special case of a single inequality constraint (i.e., $m = 0$ and $M = 1$), the dual program can be solved directly (see also Problem 8.30), yielding the following solution:

$$\lambda = \begin{cases} 0 & \text{if } \mathbb{E}_h[S(\mathbf{X})] \geqslant \gamma \,, \\ \lambda^* & \text{if } \mathbb{E}_h[S(\mathbf{X})] < \gamma \,, \end{cases}$$

where λ^* is obtained from (8.56). That is, if $\mathbb{E}_h[S(\mathbf{X})] < \gamma$, then the inequality MinxEnt solution agrees with the equality MinxEnt solution; otherwise, the optimal sampling pdf remains the prior h.

Remark 8.9.2 It is well known [14] that the optimal solution of the single-dimensional single-constrained MinxEnt program (8.54) coincides with the celebrated optimal *exponential change of measure* (ECM). Note that normally in a multidimensional ECM one twists each component separately, using possibly different twisting parameters. In contrast, the optimal solution to the MinxEnt program (see (8.58)) is parameterized by a *single-dimensional* parameter λ.

If not otherwise stated, we consider below only the single-constrained case (8.54). As in the standard CE method we could also use a *multilevel* approach, and apply a sequence of instrumentals $\{g_t\}$ and levels $\{\gamma_t\}$. Starting with $g_0 = f$ and always taking prior $h = f$, we determine γ_t and g_t as follows:

1. Update γ_t as
$$\gamma_t = \mathbb{E}_{g_t}[S(\mathbf{X}) \mid S(\mathbf{X}) \geqslant q_t] \,,$$
where q_t is the $(1-\varrho)$-quantile of $S(\mathbf{X})$ under g_{t-1}.

2. Update g_t as the solution to the above MinxEnt program for level γ_t rather than γ.

The updating formula for γ_t is based on the constraint $\mathbb{E}_g[S(\mathbf{X})] = \gamma$ in the MinxEnt program. However, instead of simply updating as $\gamma_t = \mathbb{E}_{g_{t-1}}[S(\mathbf{X})]$, we take the expectation of $S(\mathbf{X})$ with respect to g_{t-1} *conditional* on $S(\mathbf{X})$ being greater than its $(1-\varrho)$ quantile, here denoted as q_t. In contrast, in the standard CE method the level γ_t is simply updated as q_t.

Note that each g_t is completely determined by its Lagrange multiplier, say λ_t, which is the solution to (8.56) with γ_t instead of γ. In practice, both γ_t and λ_t have to be replaced by their stochastic counterparts $\widehat{\gamma}_t$ and $\widehat{\lambda}_t$, respectively. Specifically, γ_t can be estimated from a random sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ of g_{t-1} as the average of the $N^{\mathrm{e}} = \lceil (1-\varrho)N \rceil$ elite sample performances:

$$\widehat{\gamma}_t = \frac{\sum_{i=N-N^{\mathrm{e}}+1}^{N} S_{(i)}}{N^{\mathrm{e}}} \,, \tag{8.60}$$

where $S_{(i)}$ denotes the i-th order-statistics of the sequence $S(\mathbf{X}_1), \ldots, S(\mathbf{X}_N)$. And λ_t can be estimated by solving, with respect to λ, the stochastic counterpart of (8.56), which is

$$\frac{\sum_{k=1}^{N} \mathrm{e}^{\lambda S(\mathbf{X}_k)} S(\mathbf{X}_k)}{\sum_{k=1}^{N} \mathrm{e}^{\lambda S(\mathbf{X}_k)}} = \widehat{\gamma}_t \,. \tag{8.61}$$

PROBLEMS

8.1 In Example 8.2, show that the true CE-optimal parameter for estimating $\mathbb{P}(X \geqslant 32)$ is given by $v^* = 33$.

8.2 Write a CE program to reproduce Table 8.1 in Example 8.2. Use the final reference parameter $\widehat{v}_3$ to estimate ℓ via importance sampling, using a sample size of $N_1 = 10^6$. Estimate the relative error and give an approximate 95% confidence interval. Check if the true value of ℓ is contained in this interval.

8.3 In Example 8.2, calculate the exact relative error for the importance sampling estimator $\widehat{\ell}$ when using the CE optimal parameter $v^* = 33$ and compare it with the one estimated in Problem 8.2. How many samples are required to estimate ℓ with the same relative error, using CMC?

8.4 Implement the CE Algorithm 8.2.1 for the stochastic shortest path problem in Example 8.5 and reproduce Table 8.3.

8.5 Slightly modify the program used in Problem 8.4 to allow Weibull-distributed lengths. Reproduce Table 8.4 and make a new table for $\alpha = 5$ and $\gamma = 2$ (the other parameters remain the same).

8.6 Make a table similar to Table 8.4 by employing the standard CE method. That is, take $\mathsf{Weib}(\alpha, v_i^{-1})$ as the importance sampling distribution for the i-th component and update the $\{v_i\}$ via (8.6).

8.7 Consider again the stochastic shortest path problem in Example 8.5, but now with nominal parameter $\mathbf{u} = (0.25, 0.5, 0.1, 0.3, 0.2)$. Implement the root-finding Algorithm 8.2.3 to estimate for which level γ the probability ℓ is equal to 10^{-5}. Also, give a 95% confidence interval for γ, for example, using the bootstrap method.

8.8 Suppose $\mathbf{X} = (X_1, \ldots, X_n) \sim \mathsf{Ber}(\mathbf{p})$, with $\mathbf{p} = (p_1, \ldots, p_n)$, meaning that $X_i \sim \mathsf{Ber}(p_i)$, $i = 1, \ldots, n$ independent. Define $S(\mathbf{X}) = X_1 + \cdots + X_n$. We wish to estimate $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$ via importance sampling, using vector $\mathbf{q} = (q_1, \ldots, q_n)$ instead of $\mathbf{p}$. To determine a good importance sampling vector $\mathbf{q}$, we could employ the multilevel CE Algorithm 8.2.1 or the single-level CE Algorithm 8.2.5. The latter requires an approximate sample from the zero-variance pdf $g^*(\mathbf{x}) \propto f(\mathbf{x}; \mathbf{p}) I_{\{S(\mathbf{x}) \geqslant \gamma\}}$, where $f(\mathbf{x}; \mathbf{p})$ is the nominal pdf. In the numerical questions below take $n = 80$, $\gamma = 48$, and $p_i = p = 0.1$, $i = 1, \ldots, n$.

- **a)** Implement Algorithm 8.2.1 using parameters $\varrho = 0.01$ and $N = 10,000$. Note that the CE updating formula is given in (8.10). Plot the empirical distribution function of the parameters $\{q_i\}$ found in the final iteration.
- **b)** Implement a systematic Gibbs sampler to generate a dependent sample $\mathbf{X}_1, \ldots, \mathbf{X}_N$ from the conditional distribution of $\mathbf{X}$ given $S(\mathbf{X}) \geqslant \gamma$. Take $N = 10,000$. Use this sample to obtain

$$q_i = \frac{\sum_{k=1}^{N} X_{ki}}{N}, \quad i = 1, \ldots, n \,.$$

  Plot the empirical distribution function of the $\{q_i\}$ and compare with question **a)**.
- **c)** Compare the importance sampling estimators using the parameters $\{q_i\}$ found in questions **a)** and **b)**. Take a sample size $N_1 = 10^6$.

8.9 Adapt the cost matrix in the max-cut program of Table 8.5 and apply it to the dodecahedron max-cut problem in Example 8.9. Produce various optimal solutions and find out how many of these exist in total, disregarding the fivefold symmetry.

8.10 Consider the following symmetric cost matrix for the max-cut problem:

$$C = \begin{pmatrix} Z_{11} & B_{12} \\ B_{21} & Z_{22} \end{pmatrix}, \tag{8.62}$$

where Z_{11} is an $m \times m$ $(m < n)$ symmetric matrix in which all the upper-diagonal elements are generated from a $\mathsf{U}(a, b)$ distribution (and all the lower diagonal elements follow by symmetry), Z_{22} is an $(n - m) \times (n - m)$ symmetric matrix that is generated in the same way as Z_{11}, and all the other elements are c, apart from the diagonal elements, which are 0.

a) Show that if $c > b(n - m)/m$, the optimal cut is given by $V^* = \{\{1, \dots, m\}, \{m + 1, \dots, n\}\}$.

b) Show that the optimal value of the cut is $\gamma^* = c\, m\, (n - m)$.

c) Implement and run the CE algorithm on this synthetic max-cut problem for a network with $n = 400$ nodes, with $m = 200$. Generate Z_{11} and Z_{22} from the $\mathsf{U}(0, 1)$ distribution and take $c = 1$. For the CE parameters take $N = 1000$ and $\varrho = 0.1$. List for each iteration the best and worst of the elite samples and the Euclidean distance $||\widehat{\mathbf{p}}_t - \mathbf{p}^*|| = \sqrt{(\widehat{p}_{t,i} - p_i^*)^2}$ as a measure of how close the reference vector is to the optimal reference vector $\mathbf{p}^* = (1, 1, \dots, 1, 0, 0, \dots, 0)$.

8.11 Consider a TSP with cost matrix $C = (c_{ij})$ defined by $c_{i,i+1} = 1$ for all $i = 1, 2, \dots, n - 1$, and $c_{n,1} = 1$, while the remaining elements $c_{ij} \sim \mathsf{U}(a, b),\ j \neq i + 1,\ 1 < a < b$, and $c_{ii} \equiv 0$.

a) Verify that the optimal permutation/tour is given by $\mathbf{x}^* = (1, 2, 3, \dots, n)$, with minimal value $\gamma^* = n$.

b) Implement a CE algorithm to solve an instance of this TSP for the case $n = 30$ and make a table of the performance, listing the best and worst of the elite samples at each iteration, as well as

$$p_t^{mm} = \min_{1 \leqslant i \leqslant n} \max_{1 \leqslant j \leqslant n} \widehat{p}_{t,ij}\ ,$$

$t = 1, 2, \dots$, which corresponds to the min max value of the elements of the matrix $\widehat{\mathbf{P}}_t$ at iteration t. Use $d = 3$, $\varrho = 0.01$, $N = 4500$, and $\alpha = 0.7$. Also, keep track of the overall best solution.

8.12 Run Algorithm 8.3.1 on the data from the URL

`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/atsp/`

and obtain a table similar to Table 8.11.

8.13 Select a TSP of your choice. Verify the following statements about the choice of CE parameters:

a) After reducing ϱ or increasing α, the convergence is faster but we can be trapped in a local minimum.

b) After reducing ϱ, we need to decrease simultaneously α, and vice versa, in order to avoid convergence to a local minimum.

c) In increasing the sample size N, we can simultaneously reduce ϱ or (and) increase α.

8.14 Find out how many optimal solutions there are for the Hammersley TSP in Example 8.11.

8.15 Consider a complete graph with n nodes. With each edge from node i to j there is an associated cost c_{ij}. In the *longest path problem* the objective is to find the longest self-avoiding path from a certain *source* node to a *sink* node.

a) Assuming the source node is 1 and the sink node is n, formulate the longest path problem similar to the TSP. (The main difference is that the paths in the longest path problem can have different lengths.)

b) Specify a path generation mechanism and the corresponding CE updating rules.

c) Implement a CE algorithm for the longest path problem and apply it to a test problem.

8.16 Write a CE program that solves the eight-queens problem using the same configuration representation $\mathbf{X} = (X_1, \ldots, X_8)$ as in Example 6.13. A straightforward way to generate the configurations is to draw each X_i independently from a probability vector $(p_{i1}, \ldots, p_{i8})$, $i = 1, \ldots, 8$. Take $N = 500$, $\alpha = 0.7$, and $\varrho = 0.1$.

8.17 In the *permutation flow shop problem* (PFSP), n jobs have to be processed (in the same order) on m machines. The objective is to find the permutation of jobs that will minimize the *makespan*, that is, the time at which the last job is completed on machine m. Let $t(i,j)$ be the processing time for job i on machine j, and let $\mathbf{x} = (x_1, x_2, ..., x_n)$ be a job permutation. Then the completion time $C(x_i, j)$ for job i on machine j can be calculated as follows:

$$\begin{aligned}
C(x_1,1) &= t(x_1,1)\ , \\
C(x_i,1) &= C(x_{i-1},1) + t(x_i,1), \forall i = 2,...,n\ , \\
C(x_1,j) &= C(x_1,j-1) + t(x_1,j), \forall j = 2,...,m\ , \\
C(x_i,j) &= \max\{C(x_{i-1},j), C(x_i,j-1)\} + t(x_i,j)\ , \\
&\quad \text{for all } i = 2,...,n; \quad j = 2,...,m\ .
\end{aligned}$$

The objective is to minimize $S(\mathbf{x}) = C(x_n, m)$. The trajectory generation for the PFSP is similar to that of the TSP.

a) Implement a CE algorithm to solve this problem.

b) Run the algorithm for a benchmark problem from the Internet, for example `http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html`.

8.18 Verify the updating formulas (8.45) and (8.46).

8.19 Plot Matlab's `peaks` function and verify that it has three local maxima.

8.20 Use the CE program in Section A.5 of the Appendix to maximize the function $S(x) = \mathrm{e}^{-(x-2)^2} + 0.8\,\mathrm{e}^{-(x+2)^2}$. Examine the convergence of the algorithm by plotting in the same figure the sequence of normal sampling densities.

8.21 Use the CE method to minimize the *trigonometric function*

$$S(\mathbf{x}) = 1 + \sum_{i=1}^{n} 8\sin^2(\eta(x_i - x_i^*)^2) + 6\sin^2(2\eta(x_i - x_i^*)^2) + \mu(x_i - x_i^*)^2 \ , \quad (8.63)$$

with $\eta = 7$, $\mu = 1$, and $x_i^* = 0.9, i = 1, \ldots, n$. The global minimum $\gamma^* = 1$ is attained at $\mathbf{x}^* = (0.9, \ldots, 0.9)$. Display the graph and density plot of this function and give a table for the evolution of the algorithm.

8.22 A well-known test case in continuous optimization is the *Rosenbrock* function (in n dimensions):

$$S(\mathbf{x}) = \sum_{i=1}^{n-1} 100\,(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \ . \quad (8.64)$$

The function has a global minimum $\gamma^* = 0$, attained at $\mathbf{x}^* = (1, 1, \ldots, 1)$. Implement a CE algorithm to minimize this function for dimensions $n = 2, 5, 10$, and 20. Observe how injection (Remark 8.7.1) affects the accuracy and speed of the algorithm.

8.23 Suppose that $\mathscr{X}$ in (8.15) is a (possibly nonlinear) region defined by the following system of inequalities:

$$G_i(\mathbf{x}) \leqslant 0, \quad i = 1, \ldots, L \ . \quad (8.65)$$

The *proportional penalty* approach to constrained optimization is to modify the objective function as follows:

$$\widetilde{S}(\mathbf{x}) = S(\mathbf{x}) + \sum_{i=1}^{L} P_i(\mathbf{x}) \ , \quad (8.66)$$

where $P_i(\mathbf{x}) = C_i \max(G_i(\mathbf{x}), 0)$ and $C_i > 0$ measures the importance (cost) of the i-th penalty. It is clear that as soon as the constrained problem (8.15), (8.65) is reduced to the unconstrained one (8.15) — using (8.66) instead of S — we can again apply Algorithm 8.3.1.

Apply the proportional penalty approach to the constrained minimization of the Rosenbrock function of dimension 10 for the constraints below. List for each case the minimal value obtained by the CE algorithm (with injection, if necessary) and the CPU time. In all experiments, use $\varepsilon = 10^{-3}$ for the stopping criterion (stop if all standard deviations are less than ε) and $C = 1000$. Repeat the experiments 10 times to check if indeed a global minimum is found.

a) $\sum_{j=1}^{10} x_j \leqslant -8$

b) $\sum_{j=1}^{10} x_j \geqslant 15$

c) $\sum_{j=1}^{10} x_j \leqslant -8, \quad \sum_{j=1}^{10} x_j^2 \geqslant 15$

d) $\sum_{j=1}^{10} x_j \geqslant 15, \quad \sum_{j=1}^{10} x_j^2 \leqslant 22.5$

8.24 Use the CE method to minimize the function

$$S(\mathbf{x}) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3 \ ,$$

subject to the constraints $x_j \geqslant 0, j = 1,2,3$, and

$$\begin{aligned} 8\,x_1 + 14\,x_2 + 7\,x_3 - 56 &= 0\,, \\ x_1^2 + x_2^2 + x_3^2 - 25 &= 0\,. \end{aligned}$$

First, eliminate two of the variables by expressing x_2 and x_3 in terms of x_1. Note that this gives *two* different expressions for the pair (x_2, x_3). In the CE algorithm, generate the samples $\mathbf{X}$ by first drawing X_1 according to a truncated normal distribution on $[0, 5]$. Then choose either the first or the second expression for (X_2, X_3) with equal probability. Verify that the optimal solution is approximately $\mathbf{x}^* = (3.51, 0.217, 3.55)$, with $S(\mathbf{x}^*) = 961.7$. Give the solution and the optimal value in seven significant digits.

8.25 Add $\mathsf{U}(-0.1, 0.1)$, $\mathsf{N}(0, 0.01)$, and $\mathsf{N}(0, 1)$ noise to the objective function in Problem 8.20. Formulate an appropriate stopping criterion, for example, based on $\widehat{\sigma}_t$. For each case, observe how $\widehat{\gamma}_t$, $\widehat{\mu}_t$, and $\widehat{\sigma}_t$ behave.

8.26 Add $\mathsf{N}(0, 1)$ noise to the Matlab `peaks` function and apply the CE algorithm to find the global maximum. Display the contour plot and the path followed by the mean vectors $\{\widehat{\boldsymbol{\mu}}_t\}$, starting with $\widehat{\boldsymbol{\mu}}_0 = (1.3, -2.7)$ and using $N = 200$ and $\varrho = 0.1$. Stop when all standard deviations are less than $\varepsilon = 10^{-3}$. In a separate plot, display the evolution of the worst and best of the elite samples ($\widehat{\gamma}_t$ and S_t^*) at each iteration of the CE algorithm. In addition, evaluate and plot the noisy objective function in $\widehat{\boldsymbol{\mu}}_t$ for each iteration. Observe that, in contrast to the deterministic case, the $\{\widehat{\gamma}_t\}$ and $\{S_t^*\}$ do not converge to γ^* because of the noise, but eventually $S(\widehat{\boldsymbol{\mu}}_t)$ fluctuates around the optimum γ^*. More important, observe that the means $\{\widehat{\boldsymbol{\mu}}_t\}$ do converge to the optimal $\mathbf{x}^*$.

8.27 Select a particular instance (cost matrix) of the synthetic TSP in Problem 8.11. Make this TSP *noisy* by defining the random cost Y_{ij} from i to j in (8.47) to be $\mathsf{Exp}(c_{ij}^{-1})$ distributed. Apply the CE Algorithm 8.3.1 to the noisy problem and compare the results with those in the deterministic case. Display the evolution of the algorithm in a graph, plotting the maximum distance, $\max_{i,j} |\widehat{p}_{t,ij} - p_{ij}^*|$, as a function of t.

8.28 Let $X_1, \ldots, X_n$ be independent random variables, each with marginal pdf f. Suppose that we wish to estimate $\ell = \mathbb{P}_f(X_1 + \cdots + X_n \geqslant \gamma)$ using MinxEnt. For the prior pdf, we could choose $h(\mathbf{x}) = f(x_1)f(x_2)\cdots f(x_n)$, that is, the joint pdf. We consider only a single constraint in the MinxEnt program, namely, $S(\mathbf{x}) = x_1 + \cdots + x_n$. As in (8.55), the solution to this program is given by

$$g(\mathbf{x}) = c\,h(\mathbf{x})\,\mathrm{e}^{\lambda S(\mathbf{x})} = c\prod_{j=1}^{n} \mathrm{e}^{\lambda x_j} f(x_j)\,,$$

where $c = 1/\mathbb{E}_h[\mathrm{e}^{\lambda S(\mathbf{X})}] = (\mathbb{E}_f[\mathrm{e}^{\lambda X}])^{-n}$ is a normalization constant and λ satisfies (8.56). Show that the new marginal pdfs are obtained from the old ones by an *exponential twist*, with twisting parameter $-\lambda$; see also (A.13).

8.29 Problem 8.28 can be generalized to the case where $S(\mathbf{x})$ is a coordinatewise separable function, as in (8.57), and the components $\{X_i\}$ are independent under the prior pdf $h(\mathbf{x})$. Show that also in this case the components under the optimal MinxEnt pdf $g(\mathbf{x})$ are independent and determine the marginal pdfs.

8.30 Write the Lagrangian dual problem for the MinxEnt problem with constraints discussed in Remark 8.9.1.

Further Reading

The CE method was pioneered in [39] as an adaptive algorithm for estimating probabilities of rare events in complex stochastic networks. Originally it was based on variance minimization. It was soon realized [40, 41] that the same technique (using CE rather than VM) could be used not only for estimation but also for optimization purposes.

A gentle tutorial on the CE method is given in [16] and a more comprehensive treatment can be found in [45]. In 2005 a whole volume (134) of the *Annals of Operations Research* was devoted to the CE method. The CE home page, featuring many links, articles, references, tutorials, and computer programs on CE, can be found at `http://www.cemethod.org`.

The CE method has applications in many areas, including buffer allocation [1], queueing models of telecommunication systems [15, 17], control and navigation [18], signal detection [31], DNA sequence alignment [23], scheduling and vehicle routing [11], reinforcement learning [32, 35], project management [12] and heavy-tail distributions [2], [27]. Applications to more classical combinatorial optimization problems are given in [41], [42], and [43]. The continuous counterpart is discussed in [26], and applications to clustering analysis are given in [6] and [28]. Various CE estimation and noisy optimization problems for reliability systems and network design can be found in [19], [24], [25], [36], [37], and [38]. Convergence issues are discussed in [13], [34], and Section 3.5 of [45]. More recent references may be found in [8] and [29].

An approach closely related to CE is the *probability collectives* work of Dr. David Wolpert and his collaborators. This approach uses information theory as a bridge to relate game theory, statistical physics, and distributed optimization; see, for example, [48, 49].

Since the pioneering work of Shannon [47] and Kullback [30], the relationship between statistics and information theory has become a fertile area of research. The work of Kapur and Kesavan, such as [21, 22], has provided great impetus to the study of entropic principles in statistics. Rubinstein [44] introduced the idea of updating the probability vector for combinatorial optimization problems and rare events using the marginals of the MinxEnt distribution. For some fundamental contributions to MinxEnt, see [3, 4]. In [5, 9, 7] a powerful generalization and unification of the ideas behind the MinxEnt and CE methods is presented under the name *generalized cross-entropy* (GCE).

REFERENCES

1. G. Alon, D. P. Kroese, T. Raviv, and R. Y. Rubinstein. Application of the cross-entropy method to the buffer allocation problem in a simulation-based environment. *Annals of Operations Research*, 134:137–151, 2005.
2. S. Asmussen, D. P. Kroese, and R. Y. Rubinstein. Heavy tails, importance sampling and cross-entropy. *Stochastic Models*, 21(1):57–76, 2005.

3. A. Ben-Tal, D. E. Brown, and R. L. Smith. Relative entropy and the convergence of the posterior and empirical distributions under incomplete and conflicting information. Manuscript, University of Michigan, 1988.
4. A. Ben-Tal and M. Teboulle. Penalty functions and duality in stochastic programming via ϕ divergence functionals. *Mathematics of Operations Research*, 12:224–240, 1987.
5. Z. I. Botev. *Stochastic Methods for Optimization and Machine Learning.* ePrintsUQ, http://eprint.uq.edu.au/archive/00003377/, BSc (Hons) Thesis, Department of Mathematics, School of Physical Sciences, The University of Queensland, 2005.
6. Z. I. Botev and D. P. Kroese. Global likelihood optimization via the cross-entropy method, with an application to mixture models. In R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, editors, *Proceedings of the 2004 Winter Simulation Conference*, pages 529–535, Washington, DC, December 2004.
7. Z. I. Botev and D. P. Kroese. The generalized cross entropy method, with applications to probability density estimation. *Methodology and Computing in Applied Probability*, 13(1):1–27, 2011.
8. Z. I. Botev, D. P. Kroese, R.Y. Rubinstein, and P. LEcuyer. The cross-entropy method for optimization. In V. Govindaraju and C.R. Rao, editors, *Handbook of Statistics*, volume 31: Machine Learning, pages 19–34. Elsevier, Chennai, 2013.
9. Z. I. Botev, D. P. Kroese, and T. Taimre. Generalized cross-entropy methods for rare-event simulation and optimization. *Simulation: Transactions of the Society for Modeling and Simulation International*, 83(11):785–806, 2008.
10. J. C. C. Chan and D. P. Kroese. Improved cross-entropy method for estimation. *Statistics and Computing*, 22(5):1031–1040, 2012.
11. K. Chepuri and T. Homem de Mello. Solving the vehicle routing problem with stochastic demands using the cross entropy method. *Annals of Operations Research*, 134(1):153–181, 2005.
12. I. Cohen, B. Golany, and A. Shtub. Managing stochastic finite capacity multi-project systems through the cross-entropy method. *Annals of Operations Research*, 134(1):183–199, 2005.
13. A. Costa, J. Owen, and D. P. Kroese. Convergence properties of the cross-entropy method for discrete optimization. *Operations Research Letters*, 35(5):573–580, 2007.
14. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* John Wiley & Sons, New York, 1991.
15. P. T. de Boer. *Analysis and Efficient Simulation of Queueing Models of Telecommunication Systems.* PhD thesis, University of Twente, 2000.
16. P. T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005.
17. P. T. de Boer, D. P. Kroese, and R. Y. Rubinstein. A fast cross-entropy method for estimating buffer overflows in queueing networks. *Management Science*, 50(7):883–895, 2004.
18. B. E. Helvik and O. Wittner. Using the cross-entropy method to guide/govern mobile agent's path finding in networks. In S. Pierre and R. Glitho, editors, *Mobile Agents for Telecommunication Applications: Third International Workshop, MATA 2001, Montreal*, pages 255–268, New York, 2001. Springer-Verlag.
19. K-P. Hui, N. Bean, M. Kraetzl, and D.P. Kroese. The cross-entropy method for network reliability estimation. *Annals of Operations Research*, 134:101–118, 2005.
20. E. T. Jaynes. *Probability Theory: The Logic of Science.* Cambridge University Press, Cambridge, 2003.

21. J. N. Kapur and H. K. Kesavan. The generalized maximum entropy principle. *IEEE Transactions on Systems, Man, and Cybernetics*, 19:1042–1052, 1989.

22. J. N. Kapur and H. K. Kesavan. *Entropy Optimization Principles with Applications.* Academic Press, New York, 1992.

23. J. Keith and D. P. Kroese. Sequence alignment by rare event simulation. In *Proceedings of the 2002 Winter Simulation Conference*, pages 320–327, San Diego, 2002.

24. D. P. Kroese and K. P. Hui. In: *Computational Intelligence in Reliability Engineering*, chapter 3: Applications of the Cross-Entropy Method in Reliability. Springer-Verlag, New York, 2006.

25. D. P. Kroese, S. Nariai, and K. P. Hui. Network reliability optimization via the cross-entropy method. *IEEE Transactions on Reliability*, 56(2):275–287, 2007.

26. D. P. Kroese, S. Porotsky, and R. Y. Rubinstein. The cross-entropy method for continuous multi-extremal optimization. *Methodology and Computing in Applied Probability*, 8:383–407, 2006.

27. D. P. Kroese and R. Y. Rubinstein. The transform likelihood ratio method for rare event simulation with heavy tails. *Queueing Systems*, 46:317–351, 2004.

28. D. P. Kroese, R. Y. Rubinstein, and T. Taimre. Application of the cross-entropy method to clustering and vector quantization. *Journal of Global Optimization*, 37:137–157, 2007.

29. D.P. Kroese, R. Y. Rubinstein, and P. W. Glynn. The cross-entropy method for estimation. In V. Govindaraju and C.R. Rao, editors, *Handbook of Statistics*, volume 31: Machine Learning, pages 35–59. Elsevier, Chennai, 2013.

30. S. Kullback. *Information Theory and Statistics.* John Wiley & Sons, New York, 1959.

31. Z. Liu, A. Doucet, and S. S. Singh. The cross-entropy method for blind multiuser detection. In *IEEE International Symposium on Information Theory*, Chicago, 2004. Piscataway.

32. S. Mannor, R. Y. Rubinstein, and Y. Gat. The cross-entropy method for fast policy search. In *The 20th International Conference on Machine Learning (ICML-2003)*, Washington, DC, 2003.

33. L. Margolin. *Cross-Entropy Method for Combinatorial Optimization.* Master's thesis, The Technion, Israel Institute of Technology, Haifa, July 2002.

34. L. Margolin. On the convergence of the cross-entropy method. *Annals of Operations Research*, 134(1):201–214, 2005.

35. I. Menache, S. Mannor, and N. Shimkin. Basis function adaption in temporal difference reinforcement learning. *Annals of Operations Research*, 134(1):215–238, 2005.

36. S. Nariai and D. P. Kroese. On the design of multi-type networks via the cross-entropy method. In *Proceedings of the Fifth International Workshop on the Design of Reliable Communication Networks (DRCN)*, pages 109–114, 2005.

37. S. Nariai, D. P. Kroese, and K. P. Hui. Designing an optimal network using the cross-entropy method. In *Intelligent Data Engineering and Automated Learning*, Lecture Notes in Computer Science, pages 228–233, New York, 2005. Springer-Verlag.

38. A. Ridder. Importance sampling simulations of Markovian reliability systems using cross-entropy. *Annals of Operations Research*, 134(1):119–136, 2005.

39. R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99:89–112, 1997.

40. R. Y. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 2:127–190, 1999.

41. R. Y. Rubinstein. Combinatorial optimization, cross-entropy, ants and rare events. In S. Uryasev and P. M. Pardalos, editors, *Stochastic Optimization: Algorithms and Applications*, pages 304–358, Dordrecht, 2001. Kluwer.

42. R. Y. Rubinstein. Combinatorial optimization via cross-entropy. In S. Gass and C. Harris, editors, *Encyclopedia of Operations Research and Management Sciences*, pages 102–106. Kluwer, 2001.

43. R. Y. Rubinstein. The cross-entropy method and rare-events for maximal cut and bipartition problems. *ACM Transactions on Modelling and Computer Simulation*, 12(1):27–53, 2002.

44. R. Y. Rubinstein. The stochastic minimum cross-entropy method for combinatorial optimization and rare-event estimation. *Methodology and Computing in Applied Probability*, 7:5–50, 2005.

45. R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte Carlo Simulation and Machine Learning.* Springer-Verlag, New York, 2004.

46. R. Y. Rubinstein and B. Melamed. *Modern Simulation and Modeling.* John Wiley & Sons, New York, 1998.

47. C. E. Shannon. The mathematical theory of communications. *Bell Systems Technical Journal*, 27:623–656, 1948.

48. D. H. Wolpert. Information theory: the bridge connecting bounded rational game theory and statistical physics. In D. Braha and Y. Bar-Yam, editors, *Complex Engineering Systems.* Perseus Books, New York, 2004.

49. D. H. Wolpert and S. R. Bieniawski. Distributed control by Lagrangian steepest descent. In *IEEE Conference on Decision and Control*, volume 2, pages 1562–1567, 2004.

CHAPTER 9

SPLITTING METHOD

9.1 INTRODUCTION

The *splitting method* is a highly useful and versatile Monte Carlo method that uses a sequential sampling plan to decompose a "difficult" estimation problem into a sequence of "easy" problems. The method has been rediscovered and extended many times since its basic idea was proposed as far back as the 1950s, and it is closely related to particle Monte Carlo and sequential importance resampling techniques.

Splitting is particularly suitable for rare-event simulation. In contrast to importance sampling, the splitting method does not involve a change of probability law; instead, it replicates successful realizations of the simulated process in order to make the rare event more likely to occur. For this reason splitting can be used for many different purposes, including rare-event probability estimation, sampling from arbitrary high-dimensional distributions, Monte Carlo counting, and randomized optimization. The purpose of this chapter is to explain the fundamental ideas behind the splitting method, demonstrate a variety of splitting algorithms, and showcase a wide range of applications.

The rest of this chapter is organized as follows. Section 9.2 introduces the splitting idea as a Monte Carlo method for counting the number of self-avoiding walks. Section 9.3 builds on these ideas, and presents a general splitting algorithm involving a fixed splitting factor. Section 9.4 presents an important modification, called fixed effort splitting. In Section 9.5 we show that the splitting method not only can deal with *dynamic* simulation models (involving random processes that evolve over time) but also with *static* (time-independent) models, thus leading to the *generalized splitting method*. An adaptive version of the (generalized) splitting method is given in Section 9.6. The splitting method is used in Section 9.7 for the efficient estimation of the reliability of complex networks. Section 9.8 explains how splitting can be used as a powerful Monte Carlo method for counting combinatorial objects, and Section 9.9 demonstrates the simplicity and effectiveness of the splitting method for counting objects such as the number of solutions to a satisfiability (SAT) problem, independent sets, binary contingency tables, and vertex colorings. In Section 9.10 we clarify how the splitting method can be viewed as a generalization of the acceptance–rejection method to (approximately) sample from arbitrary probability distributions. Similar to the CE method, the splitting method for rare-event simulation can be conveniently converted into a randomized optimization method. This is explained in Section 9.11.

9.2 COUNTING SELF-AVOIDING WALKS VIA SPLITTING

As a motivating example for the splitting method, consider the estimation of the number of self-avoiding walks of a fixed length on the two-dimensional lattice. We discussed this counting problem previously in Example 5.17, and used sequential importance sampling to obtain an unbiased estimator for the number of self-avoiding walks. Assuming that each walk starts at the origin $(0,0)$, a walk of length T can be represented by a vector $\mathbf{X} = (X_1, \ldots, X_T)$, where X_t is the t-th position visited. An alternative is to let X_t be the *direction* taken. (Note that for consistency with the rest of the notation in this chapter, we denote the length of the walk by T rather than n.) The partial walk $(X_1, \ldots, X_t)$ of length t is denoted by $\mathbf{X}_t$. For example, two possible outcomes of $\mathbf{X}_2$ are $((0,1),(1,1))$ and $((0,1),(0,0))$. The first is a self-avoiding walk of length 2, and the second is a walk of length 2 that intersects itself. Let $\mathscr{X}_t$ be the set of all possible self-avoiding walks of length t, $t = 1, \ldots, T$. We denote the set of all 4^T walks by $\mathscr{X}$. We are interested in estimating the cardinality of the set $\mathscr{X}_T$; that is, the number of self-avoiding walks of length T. Note that if $\mathbf{X} = (X_1, \ldots, X_T)$ is drawn uniformly from $\mathscr{X}$, then the number of self-avoiding walks can be written as

$$|\mathscr{X}_T| = \underbrace{\mathbb{P}(\mathbf{X} \in \mathscr{X}_T)}_{\ell}\ |\mathscr{X}|,$$

where $\ell = \mathbb{P}(\mathbf{X} \in \mathscr{X}_T)$ is the probability of uniformly drawing a self-avoiding walk from the set of all possible walks — which is very small for large T.

The idea behind splitting is to sequentially split sample paths of the random walk into multiple copies during the simulation, so as to make the occurrence of the rare event more frequent. The key observation is that the events $A_1 = \{\mathbf{X}_1 \in \mathscr{X}_1\}$, $A_2 = \{\mathbf{X}_2 \in \mathscr{X}_2\}, \ldots, A_T = \{\mathbf{X}_n \in \mathscr{X}_T\}$ form a decreasing sequence: $A_1 \supset A_2 \supset \cdots \supset A_T = \{\mathbf{X} \in \mathscr{X}_T\}$, since for a path of length t to be self-avoiding, the path of length $t-1$ must also be self-avoiding. As a consequence, using the product rule (1.4), we can write $\ell = \mathbb{P}(\mathbf{X} \in \mathscr{X}_T)$ as

$$\ell = \underbrace{\mathbb{P}(A_1)}_{p_1}\ \underbrace{\mathbb{P}(A_2 \mid A_1)}_{p_2} \cdots \underbrace{\mathbb{P}(A_T \mid A_{T-1})}_{p_T}\ . \tag{9.1}$$

Even though direct estimation of ℓ might be meaningless when $\{\mathbf{X} \in \mathscr{X}_T\}$ is a rare event, estimating each p_t separately is viable, even in a rare-event setting, provided that the $\{p_t\}$ are not too small, such as $p_t = 10^{-2}$ or larger.

This can be achieved by dividing, at each stage t, the sample paths that have reached set $\mathscr{X}_t$ (thus the first t-steps are self-avoiding) into a fixed number, r (e.g., $r = 2$), identical copies and then continuing the simulation in exactly the same way that a single trajectory is generated. For example, starting with $N_1 = N$ trajectories of length 1, at stage $t = 1$ there are $r\,N$ copied/split trajectories. Of these, $N_2 \sim \mathsf{Bin}(r\,N, 3/4)$ paths will reach stage $t = 2$, as there is a 1 in 4 chance that a path will return to $(0, 0)$. These N_2 successful paths will be split into rN_2 copies, and so on. The number r is called the *splitting factor*. Denote the multi-set[1] of successful (self-avoiding) paths of length t by $\mathcal{X}_t$. Note that each element of $\mathcal{X}_t$ lies in $\mathscr{X}_t$. Further, let $\mathbf{e}_1 = (1, 0)$ and $\mathbf{e}_2 = (0, 1)$ be the unit vectors in the plane. The SAW splitting algorithm is summarized as follows:

Algorithm 9.2.1: SAW Splitting Algorithm

```
input  : Initial sample size $N$, splitting factor $r$.
output: Estimator of $|\mathscr{X}_T|$: the number of self-avoiding walks of length $T$.
1  $\mathcal{X}_1 \leftarrow \emptyset$
2  for $i = 1$ to $N$ do
3      Draw $\mathbf{X}_1 = X_1$ uniformly from $\{\mathbf{e}_1, \mathbf{e}_2, -\mathbf{e}_1, -\mathbf{e}_2\}$ and add $\mathbf{X}_1$ to $\mathcal{X}_1$.
4  for $t = 1$ to $T - 1$ do
5      $\mathcal{X}_{t+1} \leftarrow \emptyset$
6      for $\mathbf{X}_t \in \mathcal{X}_t$ do
7          for $i = 1$ to $r$ do
8              Draw $X_{t+1}$ uniformly from $\{\mathbf{e}_1, \mathbf{e}_2, -\mathbf{e}_1, -\mathbf{e}_2\}$. // next component
9              $\mathbf{X}_{t+1} \leftarrow (\mathbf{X}_t, X_{t+1})$
10             if $\mathbf{X}_{t+1} \in \mathscr{X}_{t+1}$ then add $\mathbf{X}_{t+1}$ to $\mathcal{X}_{t+1}$.
11 return $\widehat{\mathcal{X}_T} = 4^T |\mathcal{X}_T| / (N\, r^{T-1})$
```

We now show why Algorithm 9.2.1 returns a good estimator for $\mathcal{X}_T$. Having generated the sets $\{\mathcal{X}_t\}_{t=1}^T$, we can estimate the conditional probabilities $\{p_t\}$ in (9.1) as follows: Let N_t denote the number of elements in $\mathcal{X}_t$, $t = 1, \ldots, T$. At stage t there are N_t successful paths. These are split into rN_t copies, of which N_{t+1} reach the next level (are self-avoiding of length $t+1$). Hence an estimator of p_{t+1} is $N_{t+1}/(r\,N_t)$, and a natural estimator of ℓ is

$$\widehat{\ell} = \frac{N_1}{N}\frac{N_2}{rN_1}\cdots\frac{N_T}{rN_{T-1}} = \frac{N_T}{N\,r^{T-1}} \ .$$

Multiplying $\widehat{\ell}$ by 4^T, we obtain an unbiased estimator of the number of self-avoiding paths $|\mathscr{X}_T|$. It turns out (see Remark 9.3.1) that this is an unbiased estimator of ℓ.

A standard way to derive confidence intervals is to run the algorithm independently for M iterations, which obtains independent copies $\widehat{\ell}_1, \ldots, \widehat{\ell}_M$. The sample mean of these provides the final estimate, and the relative error and confidence

[1] A multi-set can have duplicate elements

interval are determined in the usual way via (4.1) and (4.6). Note that if many independent replications are used, N may be taken to be small; for example, $N = 1$.

■ **EXAMPLE 9.1**

As a numerical illustration, consider the estimation of the number of self-avoiding walks of length $T = 19$. Using Algorithm 9.2.1 with splitting factor $r = 2$, an initial sample size of $N = 1$, and $M = 100$ independent replications, we obtain an estimate of $3.46 \cdot 10^8$ with an estimated relative error of 0.073. The 95% confidence interval is $(2.97, 3.95) \cdot 10^8$, which contains the true value 335116620.

Starting with one path at the first level, some of the 100 replications yield no full-length paths at the final level; others had more than 1000 such paths. Figure 9.1 shows the histogram of the number of paths of length T obtained at the final level of the splitting algorithm.

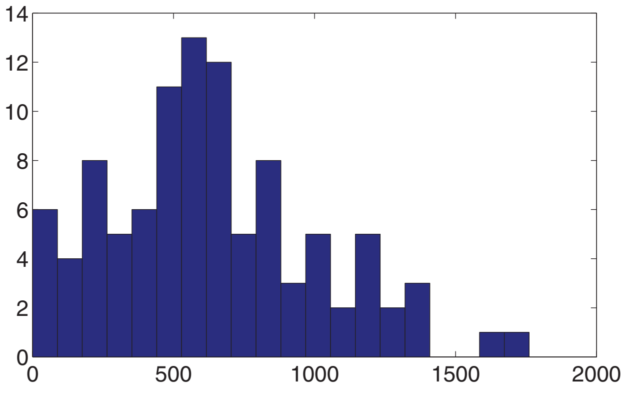

Figure 9.1: Histogram of N_T: the number of paths in the final level of the splitting algorithm.

9.3 SPLITTING WITH A FIXED SPLITTING FACTOR

In this section we generalize the splitting example of Section 9.2. Let $\mathbf{X} = \mathbf{X}_T = (X_1, \ldots, X_T)$ be a random vector and let $\mathbf{X}_t = (X_1, \ldots, X_t)$, $t = 1, 2, \ldots, T$, be the corresponding partial vectors. Note that the stochastic process $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T$, may be viewed as a Markov chain. It is assumed that it is easy to sample from

1. the initial distribution (i.e., the distribution of $\mathbf{X}_1 = X_1$) and

2. the distribution of X_{t+1} conditional on $\mathbf{X}_t$.

Suppose that we wish to estimate the probability

$$\ell = \mathbb{P}(\mathbf{X}_T \in \mathscr{X}_T)$$

for some set $\mathscr{X}_T$. Let $\mathscr{X}_1, \ldots, \mathscr{X}_T$ be sets such that $\{\mathbf{X}_1 \in \mathscr{X}_1\}$, $\{\mathbf{X}_2 \in \mathscr{X}_2\}, \ldots, \{\mathbf{X}_T \in \mathscr{X}_T\}$ form a decreasing sequence of events. Let $\{r_t\}_{t=1}^{T-1}$ be a set of integers, fixed in advance, known as the *splitting factors*. We take N independent samples $X_1^{(1)}, \ldots, X_1^{(N)}$ according to the distribution of X_1, which we refer to as *particles*. The particles for which $X_1^{(i)} \in \mathscr{X}_1$ are copied r_1 times, and the particles for which $X_1^{(i)} \notin \mathscr{X}_1$ are discarded. For each retained copy $X_1^{(i)}$ we then construct $\mathbf{X}_2^{(i)}$ by simulating its second component according to the conditional distribution of X_2 given X_1. Particles for which $\mathbf{X}_2^{(i)} \in \mathscr{X}_2$ are copied r_2 times and particles for which $\mathbf{X}_2^{(i)} \notin \mathscr{X}_2$ are discarded. This process repeats for T steps. The collection (multi-set) of successful particles at step t is denoted by $\mathcal{X}_t$, with size N_t. The final estimate is $N^{-1}\prod_{t=1}^{T-1} r_t^{-1}$ multiplied by N_T. This leads to the following generalization of Algorithm 9.2.1:

Algorithm 9.3.1: Splitting with Fixed Splitting Factors

```
   input  : Initial sample size $N$, splitting factors $r_1, \ldots, r_{T-1}$.
   output: Estimator of $\ell = \mathbb{P}(\mathbf{X}_T \in \mathscr{X}_T)$.
1  $\mathcal{X}_1 \leftarrow \emptyset$
2  for $i = 1$ to $N$ do
3      Simulate $X_1$.                                   // simulate first component
4      if $X_1 \in \mathscr{X}_1$ then add $X_1$ to $\mathcal{X}_1$.      // retain values in $\mathscr{X}_1$
5  for $t = 1$ to $T - 1$ do
6      $\mathcal{X}_{t+1} \leftarrow \emptyset$
7      for $\mathbf{X}_t \in \mathcal{X}_t$ do
8          for $i = 1$ to $r_t$ do
9              Simulate $\mathbf{X}_{t+1}$ conditional on $\mathbf{X}_t$.
10             if $\mathbf{X}_{t+1} \in \mathscr{X}_{t+1}$ then add $\mathbf{X}_{t+1}$ to $\mathcal{X}_{t+1}$.
11 return $\widehat{\ell} = \dfrac{N_T}{N\prod_{t=1}^{T-1} r_t}$
```

■ **EXAMPLE 9.2 Estimating Hitting Probabilities**

Many rare-event estimation problems are of the following form. Let $\{Z_u, u \geqslant 0\}$ be a Markov process taking values in some state space $\mathscr{Z}$. The time index can be discrete or continuous. Consider two sets $E = \{z : S(z) \geqslant \gamma\}$ and $F = \{z : S(z) = 0\}$ for some positive function S on $\mathscr{Z}$, and $\gamma > 0$. Let τ be the first time that the Markov process enters either E or F, assuming, for simplicity, that it starts outside both of these sets. Suppose that E is much more difficult to enter than F and that we wish to estimate the rare-event probability that this happens:

$$\ell = \mathbb{P}(S(Z_\tau) \geqslant \gamma) .$$

This problem can be formulated in the splitting framework by using a sequence of levels $0 < \gamma_1 \leqslant \gamma_2 \leqslant \cdots \leqslant \gamma_T = \gamma$. Define nested sets $E_t = \{z : S(z) \geqslant \gamma_t\}$, $t = 1, \ldots, T$, and let τ_t be the first time that the Markov process hits either F or E_t. Let $\mathbf{X}_t$ be the path of the Markov process up to time τ_t, so that $\mathbf{X}_t = (Z_u, u \leqslant \tau_t)$, $t = 1, \ldots, T$. Some or all the $\{\tau_t\}$ can be infinite if some or all of the sets $\{E_t\}$ and F are not hit. Let $\mathscr{X}_t$ be the set of possible paths $\mathbf{x}_t = (z_u, u \leqslant \tau_t)$ for which $S(z_{\tau_t}) \geqslant \gamma_t$; that is, paths that enter set E_t before F. Then, ℓ can be written as $\mathbb{P}(\mathbf{X}_T \in \mathscr{X}_T)$ and can be estimated via Algorithm 9.3.1. Notice that, in order to continue a path $\mathbf{x}_t = (z_u, u \leqslant \tau_t)$, it suffices to only know (record) the final value z_{τ_t} at which it entered set E_t.

■ EXAMPLE 9.3 Gambler's Ruin

As an instance of the framework of Example 9.2, consider the random walk $Z = \{Z_u, u = 0, 1, \ldots\}$ on the integers, in Example 5.16. Let p and $q = 1 - p$ be the probabilities for upward and downward jumps, respectively. The goal is to estimate the rare-event probability ℓ that state K is reached before 0, starting from some intermediate state k. In the framework of Example 9.2, we have $\mathscr{X} = \{0, \ldots, K\}$ and $S(z) = z$, and for the levels we can take $\gamma_1 = k + 1, \ldots, \gamma_{K-k} = K$, giving $T = K - k$ levels. The problem is known as the *gambler's ruin* problem, where Z_u represents the fortune of a gambler at time u, and ℓ is the probability that the gambler's fortune disappears before hitting the goal K. At any time when the gambler's fortune is $0 < z < K$, his/her next fortune is $z - 1$ or $z + 1$ with probability p or $q = 1 - p$, respectively.

At each stage t of Algorithm 9.3.1, there is only one way in which the process Z can enter the set $E_t = \{k + t, \ldots, K\}$ — namely via state $k + t$. Hence, in order to continue the paths of the Markov chain once it has reached level γ_t, it suffices to only know (record) the total number of paths that reached that level, N_t. Consequently, at each stage t, we run $r_t N_t$ independent Markov chains Z, starting from state $k + t$, and count how many of those (i.e., N_{t+1}) hit either level 0 or $k + t + 1$.

As a numerical illustration, we applied Algorithm 9.3.1 to the gambler's ruin problem with parameters, $p = 0.2$, $k = 8$, and $K = 20$. The exact probability is (e.g., see [32])

$$\ell = \frac{1 - (q/p)^k}{1 - (q/p)^K} \approx 5.9604 \cdot 10^{-8} \ .$$

Using a fixed splitting factor $r = 5$ for all levels, an initial sample size of $N = 100$, and $M = 100$ independent replications of the algorithm, we found the estimate $\widehat{\ell} = 5.8 \cdot 10^{-8}$ with an estimated relative error of 0.04, and a 95% confidence interval $(0.53, 0.62) \cdot 10^{-8}$.

Remark 9.3.1 (Splitting and Sequential Importance Resampling) The splitting method in Algorithm 9.3.1 can be seen as a sequential importance resampling (SIR) procedure with enrichment resampling. To see this, compare the algorithm with SIR Algorithm 5.9.2. Note that the notation is slightly different in Section 5.9. For example, d was used instead of T and $\mathbf{X}_{1:t}$ instead of $\mathbf{X}_t$. Also, here $H(\mathbf{X})$ is simply the indicator $I_{\{\mathbf{X}_T \in \mathscr{X}_T\}}$. The sequential construction of $\mathbf{X}_t$ is evident. The importance sampling density g is here simply the original density f. Let f_t be the density defined by $c_t f_t(\mathbf{x}_t) = I_{\{\mathbf{x}_t \in \mathscr{X}_t\}} g(\mathbf{x}_t)$. Note that $c_T = \ell$. If the densities $\{f_t\}$ are used as auxiliary densities, then the definition of the importance weight W_t becomes

$$\begin{aligned} W_t &= W_{t-1}\frac{c_t f_t(\mathbf{x}_t)}{c_{t-1}f_{t-1}(\mathbf{x}_{t-1})g(x_t \,|\, \mathbf{x}_{t-1})} \\ &= W_{t-1}\frac{I_{\{\mathbf{x}_t \in \mathscr{X}_t\}} f(\mathbf{x}_t)}{I_{\{\mathbf{x}_{t-1} \in \mathscr{X}_{t-1}\}} f(\mathbf{x}_{t-1}) f(x_t \,|\, \mathbf{x}_{t-1})} = \frac{W_{t-1}\, I_{\{\mathbf{x}_t \in \mathscr{X}_t\}}}{I_{\{\mathbf{x}_{t-1} \in \mathscr{X}_{t-1}\}}}. \end{aligned}$$

That is, the importance weights of the particles are either unchanged or are set to zero. If enrichment resampling is applied with splitting factors $\{r_t\}$, then at step t the weight of every particle is either 0 or $\prod_{k=1}^{t-1} r_k^{-1}$. The probability ℓ can be estimated as the average of the importance weights. This gives exactly the estimator returned by Algorithm 9.3.1. There are N_T non-zero importance weights and all the non-zero weights are equal to $\prod_{t=1}^{T-1} r_t^{-1}$. As a consequence, unbiasedness results for splitting estimators follow directly from such results for SIR estimators.

As discussed in Section 5.9, the splitting factors and levels must be judiciously chosen to avoid either having a very large number of particles at the final level or none at all. This may not always be easy or feasible. The next section provides an alternative where the number of particles at each level is kept fixed.

9.4 SPLITTING WITH A FIXED EFFORT

Algorithm 9.3.1 produces a branching tree of "particles" where $N_t = |\mathcal{X}_t|$, the random number of successful particles at stage t, can be difficult to control. An alternative is to resample/split, at each stage t, the N_t successful particles into a fixed number, say N, particles. This ensures a fixed simulation effort at every stage. The difference between this fixed effort splitting and splitting with a fixed splitting factor is described next.

Let $\mathscr{X}_1, \ldots, \mathscr{X}_T$ be as given in Section 9.3. We begin, as previously, by simulating iid copies $X_1^{(1)}, \ldots, X_1^{(N)}$ of the first component X_1. Of these, a random collection $\mathcal{X}_1$ of size N_1 will be successful (those $X_1^{(i)}$ that belong to $\mathscr{X}_1$). From these we construct a new sample $Y_1^{(1)}, \ldots, Y_1^{(N)}$ of size N via some sampling mechanism, two common choices being bootstrap sampling and stratified sampling (both discussed in Section 5.9).

Bootstrap sampling is simply uniform sampling with replacement from $\mathcal{X}_1$, since all the weights are equal. A stratified way of distributing the total effort N over N_t different elite paths is to first split each path $\left\lfloor \frac{N}{N_t} \right\rfloor$ times and then assign the remaining effort randomly without replacement over the N_t paths. That is, we choose

$$R^{(i)} = \left\lfloor \frac{N}{N_t} \right\rfloor + B^{(i)} \ ,$$

where $(B^{(1)}, \ldots, B^{(N_t)})$ is uniformly distributed over the set of binary vectors with exactly $(N \mod N_t)$ unities. This approach ensures that the elements in $\mathcal{X}_1$ are "evenly" resampled. For example, if $N_1 = 10$ and $N = 100$, each sample in $\mathcal{X}_1$ is copied exactly 10 times.

Having constructed the sample $\{Y_1^{(i)}\}$, we simulate for each $Y_1^{(i)}$ a random vector $\mathbf{X}_2^{(i)}$ according to the distribution of $(\mathbf{X}_2 \,|\, X_1 = Y_1^{(i)})$. The subset $\mathcal{X}_2$ of $\{\mathbf{X}_2^{(i)}\}$ for which $\mathbf{X}_2^{(i)} \in \mathscr{X}_2$ is used to construct a sample $\mathbf{Y}_2^{(1)}, \ldots, \mathbf{Y}_2^{(N)}$. This process repeats for T steps. The full algorithm is given as Algorithm 9.4.1. The final estimator is a product of the $\{N_t/N\}_{t=1}^T$. Note that for fixed effort splitting N cannot just be set to 1, as is common in Algorithm 9.3.1.

Algorithm 9.4.1: Splitting with Fixed Effort

```
   input  : Sample size $N$.
   output: Estimator of $\ell = \mathbb{P}(\mathbf{X}_T \in \mathscr{X}_T)$.
1  $\mathcal{X}_1 \leftarrow \emptyset$
2  for $i = 1$ to $N$ do
3      Simulate $X_1$.                                  // simulate first component
4      if $X_1 \in \mathscr{X}_1$ then add $X_1$ to $\mathcal{X}_1$.          // retain values in $\mathscr{X}_1$
5  $N_1 \leftarrow |\mathcal{X}_1|$
6  for $t = 1$ to $T-1$ do
7      $\mathbf{Y}_t^{(1)}, \ldots, \mathbf{Y}_t^{(N)} \leftarrow$ sample of size $N$ from $\mathcal{X}_t$.      // resample
8      $\mathcal{X}_{t+1} \leftarrow \emptyset$
9      for $i = 1$ to $N$ do
10         Simulate $\mathbf{X}_{t+1}$ conditional on $\mathbf{X}_t = \mathbf{Y}_t^{(i)}$.
11         if $\mathbf{X}_{t+1} \in \mathscr{X}_{t+1}$ then add $\mathbf{X}_{t+1}$ to $\mathcal{X}_{t+1}$. //retain values in $\mathscr{X}_{t+1}$
12
13     $N_{t+1} \leftarrow |\mathcal{X}_{t+1}|$
14 return $\prod_{t=1}^T \frac{N_t}{N}$
```

9.5 GENERALIZED SPLITTING

Thus far we have assumed that $\mathbf{X}_T$ is a random vector of the form $\mathbf{X}_T = (X_1, X_2, \ldots, X_T)$, whose components can be simulated sequentially. But all that is really needed to apply a splitting approach is that the random object $\mathbf{X}$ be constructed via a sequence of T intermediate objects $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_T = \mathbf{X}$. These objects can have different or the same dimensions. Let f_t be the known or unknown pdf of $\mathbf{X}_t$, $t = 1, \ldots, T$.

Given subsets $\{\mathscr{X}_t\}$ such that $\{\mathbf{X}_1 \in \mathscr{X}_t\}, \ldots, \{\mathbf{X}_T \in \mathscr{X}_T\}$ form a decreasing sequence of events, a very generic framework for splitting is as follows:

Algorithm 9.5.1: A Generic Splitting Algorithm

```
   input  : $\mathscr{X}_1, \ldots, \mathscr{X}_T$.
   output: Estimator of $\ell = \mathbb{P}(\mathbf{X}_T \in \mathscr{X}_T)$.
1  Create a multi-set $\mathcal{X}_1$ of samples from $f_1$.
2  for $t = 1$ to $T-1$ do
3      Create a multi-set $\mathcal{Y}_t$ of samples from $f_t$ by splitting the elements of $\mathcal{X}_t$.
4      $\mathcal{X}_{t+1} \leftarrow \mathcal{Y}_t \cap \mathscr{X}_{t+1}$
5  return Estimator $\widehat{\ell}$ of $\ell$.
```

This framework, which is similar to that in Gilks and Berzuini [18], includes all splitting scenarios in Sections 9.2–9.4. For instance, to obtain the collection $\mathcal{Y}_t$ of paths in Example 9.2, each path in $\mathcal{X}_t = (Z_u, u \leqslant \tau_t)$ is split by *continuing* the

Markov process (Z_u) until time τ_{t+1}, using the standard generation algorithm for the Markov process. If the splitting factor is fixed, r_t, this gives r_t dependent paths where the first part of the paths (up to time τ_t) are identical.

The preceding splitting framework also includes the *generalized splitting (GS) method* of Botev and Kroese [7], which deals with static (i.e., not dependent on a time parameter) rare-event estimation problems. The setting for the GS algorithm follows.

Let S be a positive function of a random vector $\mathbf{X}$ with some known or unknown pdf f and let γ be a positive number. Suppose that we are interested in estimating the quantity

$$\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma) \ .$$

For large γ this is a rare-event probability, and hence this may be difficult to estimate. The splitting approach involves constructing a nested sequence of events

$$\{S(\mathbf{X}_1) \geqslant \gamma_1\} \supset \{S(\mathbf{X}_2) \geqslant \gamma_2\} \supset \cdots \supset \{S(\mathbf{X}_T) \geqslant \gamma_T\} = \{S(\mathbf{X}) \geqslant \gamma\},$$

where $\gamma_1 \leqslant \gamma_2 \leqslant \cdots \leqslant \gamma_T = \gamma$ and $\mathbf{X}_T$ is constructed sequentially via intermediate objects $\mathbf{X}_1, \ldots, \mathbf{X}_T$. Typically, these all take values in the same space. Let $\mathscr{X}_t = \{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma_t\}$, and let f_t be the conditional pdf of $\mathbf{X}$ given $S(\mathbf{X}) \geqslant \gamma$; that is,

$$f_t(\mathbf{x}) = \frac{f(\mathbf{x}) I_{\{S(\mathbf{x}) \geqslant \gamma_t\}}}{\ell_t},$$

where ℓ_t is the normalization constant. Note that $\ell_T = \ell$.

The key insight is that if a sample $\mathcal{X}_t$ from density f_t is given, a sample $\mathcal{Y}_t$ from the same pdf can be obtained via any Markov chain whose invariant distribution is f_t. Moreover, if $\mathbf{X}$ is distributed according to f_t, then $(\mathbf{X} \mid \mathbf{X} \in \mathscr{X}_{t+1})$ is distributed according to f_{t+1}; this is simply the acceptance–rejection principle. This leads to the following algorithm:

Algorithm 9.5.2: Generalized Splitting with Fixed Splitting Factors

```
   input : Sample size $N$, splitting factors $r_1, \ldots, r_{T-1}$, Markov transition
           density $K_t$ whose invariant pdf is $f_t$, $t = 1, \ldots, T-1$.
   output: Estimator of $\ell = \mathbb{P}(\mathbf{X} \in \mathscr{X}_T)$.
 1 $\mathcal{X}_1 \leftarrow \emptyset$
 2 for $i = 1$ to $N$ do
 3     Draw $\mathbf{X} \sim f$
 4     if $\mathbf{X} \in \mathscr{X}_1$ then add $\mathbf{X}$ to $\mathcal{X}_1$.
 5 for $t = 1$ to $T-1$ do
 6     $\mathcal{X}_{t+1} \leftarrow \emptyset$
 7     for $\mathbf{X} \in \mathcal{X}_t$ do
 8         $\mathbf{Y} \leftarrow \mathbf{X}$
 9         for $i = 1$ to $r_t$ do
10             Draw $\mathbf{Y}' \sim K_t(\mathbf{y} \mid \mathbf{Y})$.
11             if $\mathbf{Y}' \in \mathscr{X}_{t+1}$ then add $\mathbf{Y}'$ to $\mathcal{X}_{t+1}$.
12             $\mathbf{Y} \leftarrow \mathbf{Y}'$
13 return $\widehat{\ell} = |\mathcal{X}_T| N^{-1} \prod_{t=1}^{T-1} r_t^{-1}$
```

Algorithm 9.5.2 can be easily modified to a fixed effort splitting algorithm by replacing lines 7–12 with lines 7–15 below, and returning the estimator $\widehat{\ell} = \prod_{t=1}^{T} \frac{N_t}{N}$. Note that we use here stratified sampling.

Algorithm 9.5.3: Generalized Splitting with Fixed Effort

```
input  : Sample size $N$, Markov transition density $K_t$ whose invariant pdf is
         $f_t$, $t = 1, \ldots, T-1$.
output: Estimator of $\ell = \mathbb{P}(\mathbf{X} \in \mathscr{X}_T)$.
1  $\mathcal{X}_1 \leftarrow \emptyset$
2  for $i = 1$ to $N$ do
3      Draw $\mathbf{X} \sim f$
4      if $\mathbf{X} \in \mathscr{X}_1$ then add $\mathbf{X}$ to $\mathcal{X}_1$.
5  for $t = 1$ to $T-1$ do
6      $\mathcal{X}_{t+1} \leftarrow \emptyset$
7      Denote the elements of $\mathcal{X}_t$ by $\mathbf{X}_t^{(1)}, \ldots, \mathbf{X}_t^{(N_t)}$.
8      Draw $B^{(i)} \sim \mathsf{Bernoulli}(\frac{1}{2})$, $i = 1, \ldots, N_t$, such that $\sum_{i=1}^{N_t} B^{(i)} = N \bmod N_t$.
9      for $i = 1$ to $N_t$ do
10         $R^{(i)} \leftarrow \left\lfloor \frac{N}{N_t} \right\rfloor + B^{(i)}$
11         $\mathbf{Y} \leftarrow \mathbf{X}_t^{(i)}$
12         for $j = 1$ to $R^{(i)}$ do
13             Draw $\mathbf{Y}' \sim K_t(\mathbf{y} \mid \mathbf{Y})$
14             if $\mathbf{Y}' \in \mathscr{X}_{t+1}$ then add $\mathbf{Y}'$ to $\mathcal{X}_{t+1}$.
15             $\mathbf{Y} \leftarrow \mathbf{Y}'$
16 return $\widehat{\ell} = \prod_{t=1}^{T} \frac{N_t}{N}$
```

There is considerable freedom in the choice of the transition density K_t. Ideally, $K_t(\mathbf{y} \mid \mathbf{x}) \approx f_t(\mathbf{y})$ for all $\mathbf{x}$. Note that a single transition according to K_t could consist of M transitions of a Markov chain with a transition density whose invariant distribution is also f_t. This amounts to carrying out an MCMC step with burn-in period M. For large M, if the chain is irreducible (and aperiodic), the sample would approximately be distributed according to f_t. However, this may be unnecessarily time-consuming. It is important to realize that, in order to apply the GS algorithm, the only requirement is that K_t have an invariant pdf f_t, not that it be close to f_t. Often, in practice, only a *single* ($M = 1$) MCMC move is performed, and the easiest way to ensure invariance is to sample from the conditional distribution of one or a few of the components of $\mathbf{X} \sim f_t$. In our applications, we typically use the *Gibbs move* described in Algorithm 9.5.4.

Algorithm 9.5.4: Gibbs Move for Generalized Splitting

```
input  : A performance function $S$, a threshold level $\gamma_t$ and a vector
         $\mathbf{x} = (x_1, \ldots, x_n)$ satisfying $S(\mathbf{x}) \geqslant \gamma_t$.
output: A random vector $\mathbf{Y} = (Y_1, \ldots, Y_n)$ with $S(\mathbf{Y}) \geqslant \gamma_t$.
1  Draw $Y_1$ from the conditional pdf $f_t(y_1 \mid x_2, \ldots, x_n)$.
2  for $i = 2$ to $n$ do
3      Draw $Y_i$ from the conditional pdf $f_t(y_i \mid Y_1, \ldots, Y_{i-1}, x_{i+1}, \ldots, x_n)$.
4  return $\mathbf{Y} = (Y_1, \ldots, Y_n)$
```

There are several variants of the Gibbs move possible. One modification is to update the components in a random order. Another is to update two or more of them simultaneously.

■ EXAMPLE 9.4 Stochastic Shortest Path

In Example 8.1 we used the CE method to estimate the probability, ℓ, that the shortest path between points A and B in the bridge network of Figure 8.1 exceeds 6. Here, the lengths $X_1, \ldots, X_5$ of the links are exponentially distributed (and independent) with means $(u_1, \ldots, u_5) = (1, 1, 0.3, 0.2, 0.1)$. We wish to apply the GS method to this problem, using the same levels as obtained in the CE example; that is, $(\gamma_1, \ldots, \gamma_5) = (1.1656, 2.1545, 3.1116, 4.6290, 6)$.

For the transition step in the GS algorithm, we apply the Gibbs move in Algorithm 9.5.4, which in this case consists of 5 steps. At the first step Y_1 is drawn from the conditional pdf

$$\begin{aligned} f_t(y_1 \,|\, x_2, \ldots, x_5) &\propto f((y_1, x_2, \ldots, x_5)) I_{\{S((y_1, x_2, \ldots, x_5)) \geqslant \gamma\}} \\ &\propto \mathrm{e}^{-y_1/u_1} I_{\{S((y_1, x_2, \ldots, x_5)) \geqslant \gamma\}} \,, \end{aligned}$$

where

$$S((y_1, x_2, \ldots, x_5)) = \min\{y_1 + x_4, y_1 + x_3 + x_5, x_2 + x_5, x_2 + x_3 + x_4\} \,.$$

It follows that Y_1 is drawn from an $\mathsf{Exp}(1/u_1)$ distribution truncated to the set $\{y_1 \geqslant 0 : y_1 + x_4 \geqslant \gamma \text{ and } y_1 + x_3 + x_5 \geqslant \gamma\} = (y_1^*, \infty)$, where $y_1^* = \max\{\gamma - x_4, \gamma - x_3 - x_5,\ 0\}$. Similarly, $Y_i, i = 2, \ldots, 5$ is drawn from an $\mathsf{Exp}(1/u_i)$ distribution truncated to the interval (y_i^*, ∞), with

$$\begin{aligned} y_2^* &= \max\{\gamma - x_5, \gamma - x_3 - x_4,\ 0\} \,, \\ y_3^* &= \max\{\gamma - Y_1 - x_5, \gamma - Y_2 - x_4,\ 0\} \,, \\ y_4^* &= \max\{\gamma - Y_1, \gamma - Y_2 - Y_3,\ 0\} \,, \\ y_5^* &= \max\{\gamma - Y_2, \gamma - Y_1 - Y_3,\ 0\} \,. \end{aligned}$$

We ran 800 independent runs of Algorithm 9.5.3, each with a sample size of $N = 100$. A typical outcome was $\widehat{\ell} = 8.0 \cdot 10^{-6}$ with an estimated relative error of 0.03, giving a 95% confidence interval $(7.6, 8.5) \cdot 10^{-6}$. This is comparable to what was obtained via the CE method.

Figure 9.2 shows the histogram of the 800 estimates. We also carried out the Gibbs move in a random order (rather than 1,2,. . . ,5). This did not lead to further improvements in accuracy.

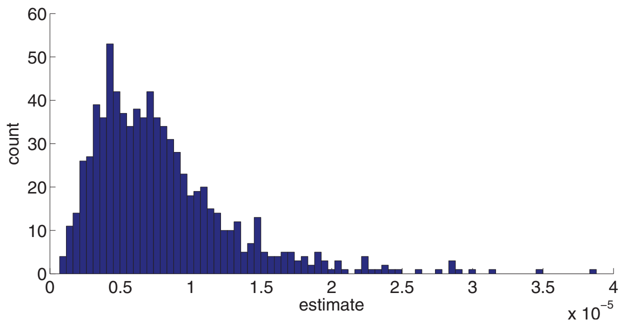

Figure 9.2: Histogram of 800 estimates returned by the splitting algorithm for the stochastic shortest path example.

9.6 ADAPTIVE SPLITTING

In the previous section we did not specify how the levels $\gamma_1, \dots, \gamma_T$ should be chosen, or indeed how many levels are desired. A general rule of thumb is to choose level γ_t such that the conditional probability $p_t = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma_{t+1} \mid S(\mathbf{X}) \geqslant \gamma_t)$ is not too small. Theoretical investigations [16, 19] indicate that the choices $T = \lceil \ln(\gamma)/2 \rceil$ and $p_t = \mathrm{e}^{-2} \approx 0.135$ are optimal, under highly simplified assumptions, where $\lceil \cdot \rceil$ denotes rounding to the largest integer. Still this does not solve the problem of how the levels should be chosen.

A more practical approach is to determine the levels *adaptively*, similar to what is done in the CE method. The adaptive splitting algorithm presented in this section is quite similar to the CE algorithm. In particular, the adaptive splitting algorithm generates a sequence of pairs

$$(\gamma_1, \mathcal{X}_1),\ (\gamma_2, \mathcal{X}_2), \dots, (\gamma_T, \mathcal{X}_T),$$

where $\mathcal{X}_t$ is a collection of dependent particles that are approximately distributed according to $f_t(\mathbf{x}) \propto f(\mathbf{x}) I_{\{S(\mathbf{x}) \geqslant \gamma_t\}}$, whereas the CE algorithm generates a sequence of pairs

$$(\gamma_1, \mathbf{v}_1),\ (\gamma_1, \mathbf{v}_1), \dots, (\gamma_T, \mathbf{v}_T),$$

where each $\mathbf{v}_t$ is a parameter of a parametric pdf $f(\mathbf{x}; \mathbf{v}_t)$ that is close (in cross-entropy sense) to f_t. The CE method has the desirable property that the elite samples at stage t (those samples that are drawn from $f(\mathbf{x}; \mathbf{v}_{t-1})$ and have a performance greater than or equal to γ_t) are *independent*, whereas in splitting the elite samples (the particles in $\mathcal{X}_t$) are not. The advantage of the splitting method is that it enables approximate sampling from f_T rather than direct sampling from a good approximation of f_t in some specified parametric family $f(\mathbf{x}; \mathbf{v})$; the latter of course also requires an additional importance sampling step.

Below we present the main steps of the adaptive splitting algorithm; see also [7] and [15]. The principal idea is, as mentioned before, to choose the thresholds such that the conditional probabilities $\{p_t\}$ of reaching the "next" level are not too small; for example, such that $p_t \approx \varrho, t = 1, \dots, T-1$ for a given *rarity parameter* ϱ. Typically, ϱ is chosen between 0.1 and 0.3. Other inputs to the algorithm are the number of samples N at each level and the final level γ. The resulting splitting algorithm is also called *splitting with a fixed probability of success* [26].

The workings of the algorithm are illustrated via Figure 9.3. Let $\varrho = 0.15$. As the figure shows, we start ($t = 1$) by generating an iid sample of $N = 20$ points from the nominal pdf f. The collection of these initial N particles is denoted by $\mathcal{Y}_0$. Choose γ_1 to be the sample $(1 - \varrho)$-quantile of the S-values of the samples in $\mathcal{Y}_0$. That is, γ_1 is the worst of best $N\varrho = 3$ function values. Let $\mathcal{X}_1$ be the set of elite samples: those samples in $\mathcal{Y}_0$ that have a function value greater than or equal to γ_1. In Figure 9.3 the 3 elite samples (indicated by red circles), are denoted by $\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3$, and here $\gamma_1 = S(\mathbf{X}_1)$. Let N_1 be the number of elite samples at stage $t = 1$.

At the next stage ($t = 2$) each of the elite particles is the starting point of an independent Markov chain whose invariant pdf is f_t. Because each starting point is distributed according to f_1 (by acceptance–rejection), all consecutive points of each Markov chain are so as well. The length of these Markov chains is determined by the total effort N. In particular, distributing this effort evenly among the Markov chains, the length of chain is at least $\lfloor N/N_1 \rfloor = 6$ and at most $\lfloor N/N_1 \rfloor + 1 = 7$. Which of the Markov chains receive the $(N \mod N_1) = 2$ extra steps can be decided by a random lottery. For example, in Figure 9.3 the Markov chains starting from $\mathbf{X}_1$ and $\mathbf{X}_2$ have length 7 and the Markov chain starting from $\mathbf{X}_3$ has length 6. Let $\mathcal{Y}_1$ be the set of all N particles thus generated, and let γ_2 be the worst of the best $N\varrho$ function values of the particles in $\mathcal{Y}_1$. Let $\mathcal{X}_2$ be the set of elite samples at stage $t = 2$ (indicated by blue dashed circles in Figure 9.3). From these, create multi-set $\mathcal{Y}_2$ via Markov sampling, then determine γ_3 and $\mathcal{X}_3$, and so on. The algorithm continues until some γ_t exceeds the target γ, at which point γ_t is reset to γ and the final level $T = t$ is returned, as well as the estimator $\widehat{\ell} = \prod_{t=1}^{T}(N_t/N)$ of $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$. The full specification is given in Algorithm 9.6.1.

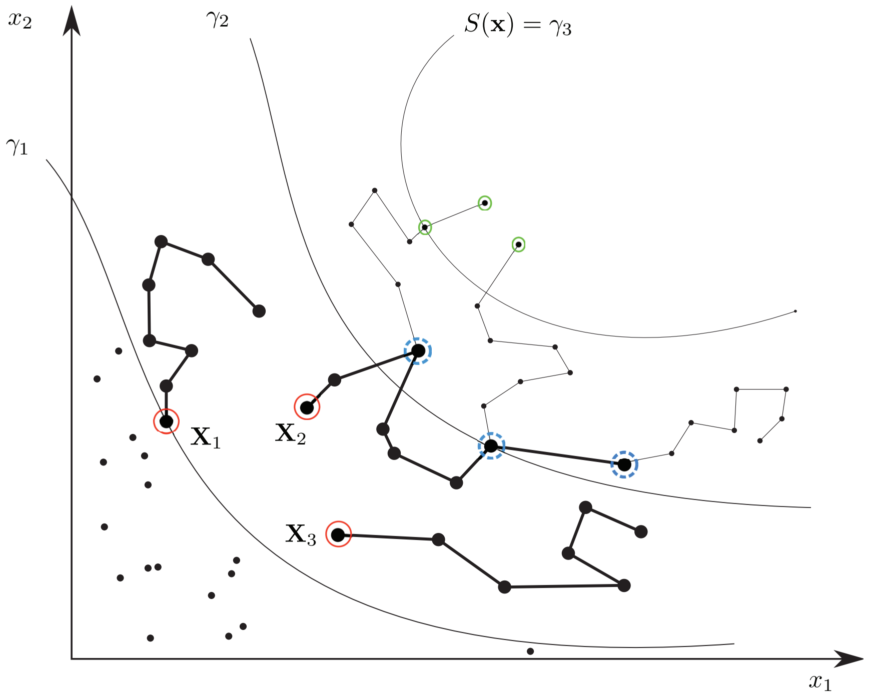

Figure 9.3: Illustration of adaptive generalized splitting with fixed effort.

Remark 9.6.1 When the levels are determined in an adaptive way, we will use the notation $\widehat{\gamma}_t$ instead of γ_t to emphasize that the $\{\widehat{\gamma}_t\}$ are random.

Despite the advantage of not having to determine the levels beforehand, the use of the adaptive splitting algorithm can have some disadvantages. Unlike the fixed-level splitting algorithms, the estimator $\widehat{\ell}$ is *biased*; see, for example, [6]. However, it is asymptotically unbiased as $N \to \infty$. To avoid complications, we could use the adaptive splitting method in a pilot run, to determine appropriate levels, and then use fixed splitting (repeated several times, using the same levels), to obtain an unbiased estimate and confidence interval. The alternative is to use a significantly larger sample size in the adaptive case than in the fixed-level case, but this could make the computation of confidence intervals more costly if many repetitions are required.

Algorithm 9.6.1: Adaptive Generalized Splitting with Fixed Effort

```
input  : Sample size $N$, rarity parameter $\varrho$, and final level $\gamma$.
output: Estimator of $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma)$, level number $T$, and levels $\widehat{\gamma}_1, \ldots, \widehat{\gamma}_T$.
1  $\mathcal{Y}_0 \leftarrow \emptyset$; $\mathcal{X}_1 \leftarrow \emptyset$
2  for $j = 1$ to $N$ do draw $\mathbf{Y} \sim f$ and add $\mathbf{Y}$ to $\mathcal{Y}_0$.
3  $s \leftarrow$ sample $(1-\varrho)$-quantile of $\{S(\mathbf{Y}), \mathbf{Y} \in \mathcal{Y}_0\}$.
4  $\widehat{\gamma}_1 \leftarrow \min\{\gamma, s\}$
5  for $\mathbf{Y} \in \mathcal{Y}_0$ do
6      if $S(\mathbf{Y}) \geqslant \widehat{\gamma}_1$ then add $\mathbf{Y}$ to $\mathcal{X}_1$.
7  $N_1 \leftarrow |\mathcal{X}_1|$. Denote the elements of $\mathcal{X}_1$ by $\mathbf{X}_1^{(1)}, \ldots, \mathbf{X}_1^{(N_1)}$.
8  if $\widehat{\gamma}_1 = \gamma$ then return estimator $\widehat{\ell} = N_1/N$, $T = 1$, $\widehat{\gamma}_1 = \gamma$
9  $t \leftarrow 1$
10 while $\widehat{\gamma}_t < \gamma$ do
11     $\mathcal{Y}_t \leftarrow \emptyset$; $\mathcal{X}_{t+1} \leftarrow \emptyset$
12     Draw $B^{(i)} \sim \mathsf{Bernoulli}(\frac{1}{2})$, $i = 1, \ldots, N_t$, such that $\sum_{i=1}^{N_t} B^{(i)} = N \bmod N_t$.
13     for $i = 1$ to $N_t$ do
14         $R^{(i)} \leftarrow \left\lfloor \frac{N}{N_t} \right\rfloor + B^{(i)}$                // random splitting factor
15         $\mathbf{Y} \leftarrow \mathbf{X}_t^{(i)}$
16         for $j = 1$ to $R^{(i)}$ do
17             Draw $\mathbf{Y}' \sim K_t(\mathbf{y} \mid \mathbf{Y})$ and add $\mathbf{Y}'$ to $\mathcal{Y}_t$.
18             $\mathbf{Y} \leftarrow \mathbf{Y}'$
19     $s \leftarrow$ sample $(1-\varrho)$-quantile of $\{S(\mathbf{Y}), \mathbf{Y} \in \mathcal{Y}_t\}$
20     $\widehat{\gamma}_{t+1} \leftarrow \min\{\gamma, s\}$
21     for $\mathbf{Y} \in \mathcal{Y}_t$ do
22         if $S(\mathbf{Y}) \geqslant \widehat{\gamma}_{t+1}$ then add $\mathbf{Y}$ to $\mathcal{X}_{t+1}$.
23     $N_{t+1} \leftarrow |\mathcal{X}_{t+1}|$. Denote the elements of $\mathcal{X}_{t+1}$ by $\mathbf{X}_{t+1}^{(1)}, \ldots, \mathbf{X}_{t+1}^{(N_{t+1})}$.
24     $t \leftarrow t + 1$
25 return Estimator $\widehat{\ell} = \prod_{t=1}^{T}(N_t/N)$, $T = t$, and $\widehat{\gamma}_1, \ldots, \widehat{\gamma}_T = \gamma$.
```

■ EXAMPLE 9.5 Stochastic Shortest Path (Continued)

We continue Example 9.4 by applying the adaptive splitting algorithm to the estimation of ℓ. We first employ Algorithm 9.6.1 as a pilot algorithm to determine the levels. Using $\varrho = 0.1$ and $N = 1000$, we find the levels $(\widehat{\gamma}_1, \ldots, \widehat{\gamma}_5) = (1.2696, 2.4705, 3.6212, 4.8021, 6.0000)$. Repeating the fixed level Algorithm 9.5.3 800 times with these levels and $N = 100$ produces very similar results to those in Example 9.4: $\widehat{\ell} = 7.9 \cdot 10^{-6}$ with an estimated relative error of 0.03.

In contrast, repeating Algorithm 9.6.1 800 times with $N = 100$ gives an average estimate of $1.5 \cdot 10^{-5}$ with an estimated relative error of 0.025. Clearly, the estimator is biased in this case. However, using 80 repetitions with $N = 1000$ gives reliable estimates, with an overall estimate of $8.1 \cdot 10^{-6}$ and an estimated relative error of 0.025.

9.7 APPLICATION OF SPLITTING TO NETWORK RELIABILITY

In Example 4.2 and Section 5.4.1 we discussed the network (un)reliability estimation problem and showed how permutation Monte Carlo (PMC) can be used to greatly reduce the computational effort when the unreliability is very small. In this section we demonstrate that the generalized splitting method can provide a powerful (and simpler) alternative.

To recapitulate the network reliability estimation problem, consider a network (graph) consisting of n unreliable links (edges). The random states of the links are represented by a binary vector $\mathbf{X} = (X_1, \ldots, X_n)$ of independent Bernoulli random variables with $\mathbb{P}(X_i = 1) = p_i$, $i = 1, \ldots, n$. The system state is $H(\mathbf{X})$ for some *structure function* $H : \{0,1\}^n \to \{0,1\}$. The objective is to estimate the system unreliability $\bar{r} = \mathbb{P}(H(\mathbf{X}) = 0)$, which is difficult to estimate if the components are highly reliable (all $\{p_i\}$ close to 1).

Just as in the PMC method, consider the link states vector $\mathbf{X}_t, t \geqslant 0$, of a *dynamic* network process, where the network starts with all links down, and each link is repaired independently according to an $\mathsf{Exp}(-\ln(1-p_i))$ distribution. Then $\mathbf{X}$ has the same distribution as $\mathbf{X}_1$, and hence can be viewed as a snapshot of the dynamic process at time $t = 1$.

Let $\mathbf{Y} = (Y_1, \ldots, Y_n)$ be the vector of repair times of the dynamic network, and let $S(\mathbf{Y})$ be the first time that the network becomes operational. Then,

$$\bar{r} = \mathbb{P}(S(\mathbf{Y}) > 1)$$

is a rare-event probability (for large $\{p_i\}$) and is amenable to the generalized splitting method. For example, given a sequence of levels $0 < \gamma_1 < \ldots < \gamma_T = 1$, we can apply Algorithm 9.5.3 with $\mathscr{X}_t = \{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma_t\}$. Good choices for the levels can be obtained via a pilot run with the adaptive splitting Algorithm 9.6.1. It remains to specify the Markov move. For this, we can use the Gibbs move in Algorithm 9.5.4. In particular, each Y_i is drawn from a (truncated) exponential distribution. The specific form of the Gibbs move is given in Algorithm 9.7.1 below. Note that the components are updated in a random order.

Algorithm 9.7.1: Gibbs Move for Network Reliability

```
  input : A performance function $S$, a threshold level $\gamma$ and a vector
          $\mathbf{y} = (y_1, \ldots, y_n)$ satisfying $S(\mathbf{y}) \geqslant \gamma$.
  output: A random vector $\mathbf{Y} = (Y_1, \ldots, Y_n)$ with $S(\mathbf{Y}) \geqslant \gamma$.
1 $\mathbf{Y} \leftarrow \mathbf{y}$
2 Draw a random uniform permutation $(\pi_1, \ldots, \pi_n)$ of $(1, \ldots, n)$.
3 for $i = 1$ to $n$ do
4     $k \leftarrow \pi_i$
5     if $S(Y_1, \ldots, Y_{k-1}, 0, Y_{k+1}, \ldots, Y_n) \leqslant \gamma$ then
6         $Y_k \leftarrow \gamma + Z, \quad Z \sim \mathsf{Exp}(-\ln(1-p_k))$
7     else
8         $Y_k \leftarrow Z, \quad Z \sim \mathsf{Exp}(-\ln(1-p_k))$
9 return $\mathbf{Y}$
```

■ EXAMPLE 9.6 Dodecahedron Network

Figure 9.4 depicts a dodecahedron network, with 30 links (not labeled in the figure) and 20 nodes. Suppose that the network functions if all highlighted nodes (1, 4, 7, 10, 13, 16, 20) are connected via paths of functioning links.

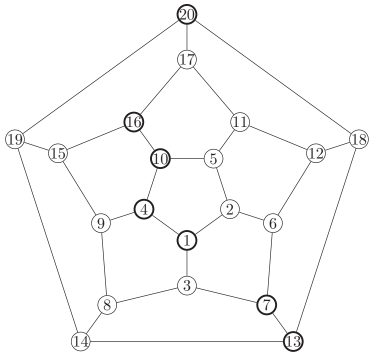

Figure 9.4: Dodecahedron network.

Consider the case where all link failure probabilities are $q = 1 - p = 0.01$. We ran Algorithm 9.6.1 with $N = 10,000$ and $\varrho = 0.5$ to obtain the 18 levels given in Table 9.1.

Table 9.1: Levels of the splitting algorithm for the dodecahedron reliability problem.

0.229623	0.421039	0.606999	0.801514	0.996535
0.284589	0.465779	0.655211	0.84999	1.000000
0.331599	0.511747	0.703615	0.898574	
0.376894	0.559533	0.752544	0.947693	

Using these levels and the same N and ϱ, 10 independent runs with Algorithm 9.5.3 were performed, giving the following estimates: $(6.89, 7.11, 7.07, 7.72, 7.16, 7.30, 7.38, 6.94, 7.70, 7.59) \cdot 10^{-6}$, and giving an average estimate of $7.29 \cdot 10^{-6}$ with an estimated relative error of 0.013.

9.8 APPLICATIONS TO COUNTING

In Section 9.2 we saw how the problem of counting self-avoiding walks can be viewed as an estimation problem, to which Monte Carlo techniques such as importance sampling and splitting can be applied. In this section we show how many other counting problems, such as the satisfiability counting problem and many graph-related counting problems, can be dealt with using the splitting method. It is interesting to note [31, 44] that in many cases the counting problem is hard to solve, while the associated decision or optimization problem is easier to solve. For example, finding the shortest path between two fixed vertices in a graph is easy, whereas finding the total number of paths between the two vertices is difficult.

The latter problem, for example, belongs to the *#P-complete* complexity class — a concept related to the familiar class of NP-hard problems [30].

The key to applying the splitting method is to decompose a difficult counting problem — counting the number of elements in some set $\mathscr{X}^*$ — into a sequence of easier ones, using the following steps. First, find a sequence of decreasing sets

$$\mathscr{X} = \mathscr{X}_0 \supset \mathscr{X}_1 \supset \cdots \supset \mathscr{X}_T = \mathscr{X}^* \,, \tag{9.2}$$

where the elements in $\mathscr{X}$ are easy to count. Next, consider the problem of estimating

$$\ell = \mathbb{P}(\mathbf{X} \in \mathscr{X}^*) \,,$$

where $\mathbf{X}$ is uniformly distributed over $\mathscr{X}$. Note that $|\mathscr{X}^*| = \ell\,|\mathscr{X}|$ and hence an accurate estimate of ℓ will lead to an accurate estimate of $|\mathscr{X}^*|$. We can use any Monte Carlo algorithm to estimate ℓ. Similar to (9.1), we can write

$$\ell = \mathbb{P}(\mathbf{X} \in \mathscr{X}_1 \,|\, \mathbf{X} \in \mathscr{X}_0) \cdots \mathbb{P}(\mathbf{X} \in \mathscr{X}_T \,|\, \mathbf{X} \in \mathscr{X}_{T-1}) = \prod_{t=1}^{T} \frac{|\mathscr{X}_t|}{|\mathscr{X}_{t-1}|}. \tag{9.3}$$

The above decomposition of ℓ is motivated by that idea that estimating each $p_t = |\mathscr{X}_t|/|\mathscr{X}_{t-1}|$ may be much easier than estimating ℓ directly.

In general, to deliver a meaningful estimator of $|\mathscr{X}^*|$, we have to address the following two problems:

1. Construct the sequence $\mathscr{X}_0 \supset \mathscr{X}_1 \supset \cdots \supset \mathscr{X}_T = \mathscr{X}^*$ such that each $p_t = |\mathscr{X}_t|/|\mathscr{X}_{t-1}|$ is not a rare-event probability.

2. Obtain a low variance unbiased estimator $\widehat{p}_t$ of each p_t.

The first task can often be resolved by introducing a positive performance function $S(\mathbf{x})$ and levels $\gamma_1, \ldots, \gamma_T$ such that

$$\mathscr{X}_t = \{\mathbf{x} \in \mathscr{X} \,:\, S(\mathbf{x}) \geqslant \gamma_t\}, \quad t = 0, 1, \ldots, T.$$

The second task, in contrast, can be quite complicated, and involves sampling *uniformly* from each subset $\mathscr{X}_t$. Note that if we could sample uniformly from each subset, we could simply take the proportion of samples from $\mathscr{X}_{t-1}$ that fall in $\mathscr{X}_t$ as the estimator for p_t. For such an estimator to be efficient (have low variance), the subset $\mathscr{X}_t$ must be relatively "dense" in $\mathscr{X}_{t-1}$. In other words, p_t should not be too small. The generalized splitting method in Section 9.5 is tailor-made for this situation. Namely in this case the target pdf $f(\mathbf{x})$ is the uniform pdf on $\mathscr{X}_0$ and each pdf f_t corresponds to the uniform pdf on $\mathscr{X}_t$, $t = 1, \ldots, T$. A typical generalized splitting procedure could involve the adaptive version in Algorithm 9.6.1, the fixed-level version in Algorithm 9.5.3, or a combination of the two. For example, the former can be used in a pilot run to determine the levels for the latter, whereas the latter provides an unbiased estimator.

For convenience, we repeat the main steps of the adaptive version for counting, allowing for level-dependent rarity parameters $\varrho_t, t = 1, 2, \ldots$ and adding one extra "screening" step. Screening simply deletes duplicate particles in the elite multisets. Such a modification enriches the elite sets in subsequent iterations. Of course, duplications are only an issue when the sample space is discrete (as in counting problems).

Algorithm 9.8.1 (Adaptive Splitting Algorithm for Counting)

- Input: performance function S, final level γ, rarity parameters $\varrho_t,\ t = 1, 2, \ldots$, sample size N.

- Output: estimator $\widehat{\ell}$ of $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant \gamma) = \mathbb{P}(\mathbf{X} \in \mathscr{X}^*)$, estimator $\widehat{|\mathscr{X}^*|}$ of $|\mathscr{X}^*| = |\{\mathbf{x} \in \mathscr{X} : S(\mathbf{x}) \geqslant \gamma)\}|$, level number T and levels $\widehat{\gamma}_1, \ldots, \widehat{\gamma}_T$.

1. **Acceptance–Rejection.** Set a counter $t = 1$. Generate a sample $\mathcal{Y}_0 = \{\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(N)}\}$ uniformly on $\mathscr{X} = \mathscr{X}_0$. Compute the threshold $\widehat{\gamma}_1$ as the sample $(1 - \varrho_1)$-quantile of the values of $S(\mathbf{Y}^{(1)}), \ldots, S(\mathbf{Y}^{(N)})$. Let $\mathcal{X}_1 = \{\mathbf{X}_1^{(1)}, \ldots, \mathbf{X}_1^{(N_1)}\}$ be the collection (multi-set) of elements $\mathbf{X}_1 \in \mathcal{Y}_0$ for which $S(\mathbf{X}_1) \geqslant \widehat{\gamma}_1$ (the elite points). Set $\widehat{p}_1 = N_1/N$ as an unbiased estimator of $p_1 = |\mathscr{X}_1|/|\mathscr{X}_0|$. Note that $\mathbf{X}_1, \ldots, \mathbf{X}_{N_1} \sim \mathsf{U}(\mathscr{X}_1)$, the uniform distribution on the set $\mathscr{X}_1 = \{\mathbf{x} : S(\mathbf{x}) \geqslant \widehat{\gamma}_1\}$. If $\widehat{\gamma}_1 > \gamma$, return $\widehat{\ell} = N_1/N, T = 1$, and $\widehat{\gamma}_1 = \gamma$ and stop.

2. **Screening.** Reset $\mathcal{X}_t$ and N_t by screening out (removing) any duplicate particles.

3. **Splitting.** To each particle in the elite multi-set $\mathcal{X}_t$ apply a Markov chain sampler with $\mathsf{U}(\mathscr{X}_t)$ as its invariant (stationary) distribution, and length $\lfloor N/N_t \rfloor$. Apply the Markov chain sampler to $N - \lfloor N/N_t \rfloor = N \mod N_1$ randomly chosen endpoints for one more period. Denote the new entire sample (of size N) by $\mathcal{Y}_t$.

4. **Selecting elites.** Compute level $\widehat{\gamma}_{t+1}$ as the sample $(1 - \varrho_t)$ quantile of the values of $S(\mathbf{Y}), \mathbf{Y} \in \mathcal{Y}_t$. If $\widehat{\gamma}_{t+1} > \gamma$, reset it to γ. Determine the elite sample, i.e., the largest subset $\mathcal{X}_{t+1} = \{\mathbf{X}_{t+1}^{(1)}, \ldots, \mathbf{X}_{t+1}^{(N_{t+1})}\}$ of $\mathcal{Y}_t$ consisting of N_{t+1} points for which $S(\mathbf{X}_{t+1}) \geqslant \widehat{\gamma}_{t+1}$.

5. **Estimating p_{t+1}.** Take $\widehat{p}_{t+1} = N_{t+1}/N$ as an estimator of $p_{t+1} = |\mathscr{X}_{t+1}|/|\mathscr{X}_t|$.

6. **Stopping rule and estimation.** Increase the counter $t = t+1$. If $\widehat{\gamma}_{t+1} < \gamma$, repeat from Step 2; otherwise return $T = t$, $\widehat{\gamma}_1, \ldots, \widehat{\gamma}_T$ and the estimators $\widehat{\ell} = \prod_{t=1}^{T}(N_t/N)$ and $\widehat{|\mathscr{X}^*|} = \widehat{\ell}|\mathscr{X}|$.

The main ingredient that remains to be specified is the Markov move in Step 3. A convenient choice is to use a Gibbs move, one variant of which is described in Algorithm 9.5.4. That is, given a vector $\mathbf{x} = (x_1, \ldots, x_n)$ and an intermediate threshold γ_t, the transition to $\mathbf{Y} = (Y_1, \ldots, Y_n)$ is carried out by drawing one or more components Y_i from the uniform distribution of $\mathbf{Y}$ conditional on the remaining components *and* conditional on $S(\mathbf{Y}) \geqslant \gamma_t$.

Remark 9.8.1 (Adaptive Choice of ϱ_t) Choosing in advance a fixed rarity parameter $\varrho_t = \varrho$ for all iterations t may not always be desirable, especially in discrete problems. The reason is that in Step 3 of Algorithm 9.8.1 many points $\mathbf{y} \in \mathcal{Y}_t$ could have identical function values. As a consequence, in Step 4 the number of elite samples, N_{t+1}, could be much larger than ϱN. For example, if at iteration t the ordered performance values are $1, 1, 2, 2, 3, 3, 3, 3$, and $\varrho = 0.1$, and $N = 10$, then $\widehat{\gamma}_{t+1} = 3$

and $N_{t+1} = 4$, whereas $N\varrho = 1$. In an extreme case this could even lead to $N_t = N$ and the algorithm would lock. To eliminate such undesirable behavior, Step 4 in Algorithm 9.8.1 can be modified as follows: Choose a fixed ϱ; say $\varrho = 0.1$. Next, let a_1 and a_2 be such that $a_1 \leqslant \varrho \leqslant a_2$; say $a_1 = 0.1$ and $a_2 = 0.3$. Initially, let $\widehat{\gamma}_{t+1}$, $\mathcal{X}_{t+1}$ and N_{t+1} be exactly as in Step 4 of Algorithm 9.8.1. If $N_{t+1} \leqslant a_2 N$ (not too many elite samples), then continue with Step 5. Otherwise, increase $\widehat{\gamma}_{t+1}$ by some fixed amount, say 1, and reset $\mathcal{X}_{t+1}$ and N_{t+1} *unless* the new number of elite samples is too small, $N_{t+1} < a_1 N$, in which case the original choices are used.

Remark 9.8.2 (Other Counting Strategies) Algorithm 9.8.1 can be combined with various other counting strategies. An obvious one is to count directly how many unique particles are encountered in the final level, giving the *direct estimator*

$$\widehat{|\mathscr{X}^*|}_{\text{dir}} = |\texttt{unique}(\mathcal{X}_T)| \;, \tag{9.4}$$

where `unique` screens out the duplicate elements of $\mathcal{X}_T$. To increase further the accuracy of $\widehat{|\mathscr{X}^*|}_{\text{dir}}$ we can take a larger sample after reaching the final level γ, or pool the results of several independent runs of Algorithm 9.8.1. We found numerically that the direct estimator is very useful and accurate, as compared to the estimator

$$\widehat{|\mathscr{X}^*|} = |\mathscr{X}| \prod_{t=1}^{T} \frac{N_t}{N},$$

when $|\mathscr{X}^*|$ is not too large and less than the sample size N.

Another possibility is to combine Algorithm 9.8.1 with the classic *capture-recapture* (CAP-RECAP) method [36], by running the algorithm twice and counting how many unique final-level particles in the second run also appear in the first run. For example, if the first run has n_1 unique particles and the second has n_2 unique particles, R of whom are also in the first set, then (because $n_2/|\mathscr{X}^*| \approx R/n_1$), a (biased) estimator of $|\mathscr{X}^*|$ is $\frac{n_1 n_2}{R}$.

9.9 CASE STUDIES FOR COUNTING WITH SPLITTING

In this section we illustrate the splitting method for counting via a selection of difficult counting problems. For each case we (1) describe the problem and indicate how it fits into the framework of Section 9.8, (2) specify how the Markov transition step is carried out, and (3) provide numerical examples.

9.9.1 Satisfiability (SAT) Problem

The Boolean satisfiability (SAT) problem plays a central role in combinatorial optimization and computational complexity. Any NP-complete problem, such as the max-cut problem, the graph-coloring problem, and the TSP, can be translated *in polynomial time* into a SAT problem. The SAT problem plays a central role in solving large-scale computational problems, such as planning and scheduling, integrated circuit design, computer architecture design, computer graphics, image processing, and finding the folding state of a protein.

There are different formulations for the SAT problem, but the most common one comprises the following two components [22]:

- A vector of n Boolean variables $\mathbf{x} = (x_1, \ldots, x_n)^\top$, called a *truth assignment*, representing statements that can either be `TRUE` (=1) or `FALSE` (=0). The negation (the logical `NOT`) of a variable x_i is denoted by $\overline{x}_i$. A variable or its negation is called a *literal*.
- A set of m distinct *clauses* $\{C_1, C_2, \ldots, C_m\}$ of the form $C_i = l_{i_1} \vee l_{i_2} \vee \cdots \vee l_{i_k}$, where the l's are literals and $\vee$ denotes the logical `OR` operator. For example, $0 \vee 1 = 1$.

The simplest SAT problem can now be formulated as: find a truth assignment $\mathbf{x}$ such that *all* clauses are true. Denoting the logical `AND` operator by $\wedge$, we can represent the above SAT problem via a single formula as

$$F = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{|C_i|} l_{i_j} \ .$$

The SAT formula is said to be in *conjunctive normal form* (CNF).

As an illustration of the SAT problem and the corresponding SAT counting problem, consider the following toy example of coloring the nodes of the graph in Figure 9.5. Is it possible to color the nodes either black or white in such a way that no two adjacent nodes have the same color? If so, how many such colorings are there?

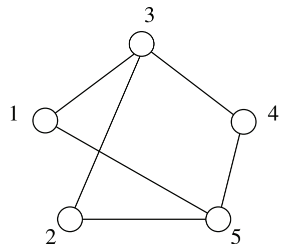

Figure 9.5: Can the graph be colored with two colors so that no two adjacent nodes have the same color?

We can translate this graph-coloring problem into a SAT problem in the following way: Let x_j be the Boolean variable representing the statement "the j-th node is colored black". Obviously, x_j is either `TRUE` or `FALSE`, and we wish to assign truth to either x_j or $\overline{x}_j$, for each $j = 1, \ldots, 5$. The restriction that adjacent nodes cannot have the same color can be translated into a number of clauses that must all hold. For example, "node 1 and node 3 cannot both be black" can be translated as clause $C_1 = \overline{x}_1 \vee \overline{x}_3$. Similarly, the statement "node 1 and node 3 cannot both be white" is translated as $C_2 = x_1 \vee x_3$. The same holds for all other pairs of adjacent nodes. The clauses can now be conveniently summarized as in Table 9.2. Here, in the left-hand table, for each clause C_i a 1 in column j means that the clause contains x_j; a -1 means that the clause contains the negation $\overline{x}_j$; and a 0 means that the clause does not contain either of them. Let us call the corresponding $(m \times n)$ matrix $A = (a_{ij})$ the *clause matrix*. For example, $a_{75} = -1$ and $a_{42} = 0$. A common representation of a SAT in CNF form is to list for each clause only the indexes of all Boolean variables present in that clause. In addition, each index that corresponds to a negation of a variable is preceded by a minus sign. This is given in the right-hand panel of Table 9.2.

Table 9.2: SAT table and an alternative representation of the clause matrix.

	1	2	3	4	5	
$\mathbf{x}^\top$	0	1	0	1	0	
C_1	-1	0	-1	0	0	1
C_2	1	0	1	0	0	0
C_3	-1	0	0	0	-1	1
C_4	1	0	0	0	1	0
C_5	0	-1	-1	0	0	1
C_6	0	1	1	0	0	1
C_7	0	-1	0	0	-1	1
C_8	0	1	0	0	1	1
C_9	0	0	-1	-1	0	1
C_{10}	0	0	1	1	0	1
C_{11}	0	0	0	-1	-1	1
C_{12}	0	0	0	1	1	1

C_1	-1	-3
C_2	1	3
C_3	-1	-5
C_4	1	5
C_5	-2	-3
C_6	2	3
C_7	-2	-5
C_8	2	5
C_9	-3	-4
C_{10}	3	4
C_{11}	-4	-5
C_{12}	4	5

Now let $\mathbf{x} = (x_1, \ldots, x_n)^\top$ be a truth assignment. The question is whether there exists an $\mathbf{x}$ such that all clauses $\{C_k\}$ are satisfied. Define the clause value $C_i(\mathbf{x}) = 1$ if clause C_i is `TRUE` with truth assignment $\mathbf{x}$ and $C_i(\mathbf{x}) = 0$ if it is `FALSE`. For example, for truth assignment $(0, 1, 0, 1, 0)$ the corresponding clause values are given in the rightmost column of the left-hand panel in Table 9.2. We see that the second and fourth clauses are violated. However, the assignment $(1, 1, 0, 1, 0)$ does indeed render all clauses true, and this therefore gives a way in which the nodes can be colored: 1 = black, 2 = black, 3 = white, 4 = black, 5 = white. It is easy to see that $(0, 0, 1, 0, 1)$ is the only other assignment that renders all the clauses true.

In general, let $n_i = \sum_{j=1}^n I_{\{a_{ij}=-1\}}$ be the total number of (-1)- entries in the i-th row of the clause matrix A. If $x_j = 0$ for all $(+1)$-entries a_{ij} and $x_j = 1$ for all (-1)-entries, then $C_i(\mathbf{x}) = 0$ and $\sum_{i=1}^n a_{ij} x_j = -n_i$; otherwise, this sum is at least $b_i = 1 - n_i$ and $C_i(\mathbf{x}) = 1$. Consequently, the SAT problem can be formulated as a search problem in a linearly constrained set:

$$\text{Find} \quad \mathbf{x} \in \{0,1\}^n \quad \text{such that} \quad A\mathbf{x} \geqslant \mathbf{b}\ ,$$

where $\mathbf{b} = (b_1, \ldots, b_n)^\top$.

The problem of deciding whether there *exists* a valid assignment, and indeed providing such a vector, is called the *SAT-assignment* problem. A SAT-assignment problem in which each clause contains exactly K literals is called a *K-SAT* problem. Finding the total number of such valid assignments is called the *SAT-counting* problem. The latter is more difficult and is the focus our our case study.

In the framework of Section 9.8, let $\mathscr{X} = \{0,1\}^n$ be the set of all truth vectors, and define for each $\mathbf{x} \in \mathscr{X}$,

$$S(\mathbf{x}) = \sum_{i=1}^m I_{\{\sum_{j=1}^n a_{ij} x_j \geqslant b_i\}}\ ,$$

which is the total number of clauses that are satisfied. Our objective is to estimate the number of elements in the set $\mathscr{X}^* = \{\mathbf{x} \in \mathscr{X} : S(\mathbf{x}) \geqslant m\}$.

We will use the following Markov move for the splitting algorithm. Note that this algorithm can be applied to any counting problem involving binary vectors.

Algorithm 9.9.1: Gibbs Move for Binary Vector Counting

```
   input  : A performance function $S$, a threshold level $\gamma$ and a binary vector
            $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ satisfying $S(\mathbf{x}) \geqslant \gamma$.
   output: A random binary vector $\mathbf{Y} = (Y_1, \ldots, Y_n)$ with $S(\mathbf{Y}) \geqslant \gamma$.
 1 $\mathbf{Y}^0 \leftarrow (0, x_2, \ldots, x_n)$; $\mathbf{Y}^1 \leftarrow (1, x_2, \ldots, x_n)$
 2 if $S(\mathbf{Y}^0) \geqslant \gamma$ and $S(\mathbf{Y}^1) \geqslant \gamma$ then
 3     Choose $Y_1 \in \{0,1\}$ with probability $\frac{1}{2}$.
 4 else
 5     $Y_1 \leftarrow x_1$
 6 for $i = 2$ to $n$ do
 7     $\mathbf{Y}^0 \leftarrow (Y_1, \ldots, Y_{i-1}, 0, x_{i+1} \ldots, x_n)$; $\mathbf{Y}^1 \leftarrow (Y_1, \ldots, Y_{i-1}, 1, x_{i+1} \ldots, x_n)$
 8     if $S(\mathbf{Y}^0) \geqslant \gamma$ and $S(\mathbf{Y}^1) \geqslant \gamma$ then
 9         Choose $Y_i \in \{0,1\}$ with probability $\frac{1}{2}$.
10     else
11         $Y_i \leftarrow x_i$
12 return $\mathbf{Y} = (Y_1, \ldots, Y_n)$
```

9.9.1.1 Numerical Experiments Although K-SAT counting problems for $K \geqslant 2$ are NP-hard, numerical studies nevertheless indicate that most K-SAT problems are easy to solve for certain values of n and m. To study this phenomenon, Mézard and Montanari [29] define a family of *random* K-SAT problems, where each clause is drawn uniformly from the set of $\binom{n}{K} 2^K$ clauses, independently of the other clauses. It has been observed empirically that a crucial parameter characterizing this problem is the *clause density* $\beta = m/n$, and that, depending on K, the most difficult problems occur around a critical clause density β_K. For $K = 3$, for example, $\beta_3 \approx 4.3$.

We present data from experiments with two different instances of random 3-SAT models, both of which can be found at the SATLIB website `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`. In the tables we use the following notation.

- N_t and N_t^{scr} denote the number of elites before and after screening, respectively;
- $\overline{S}_t$ and $\underline{S}_t$ denote the upper and the lower elite levels reached, respectively (the $\underline{S}_t$ levels are the same as the $\widehat{\gamma}_t$ levels in the description of the algorithm);
- $\widehat{p}_t = N_t/N$ is the estimator of the t-th conditional probability;
- (intermediate) product estimator after the t-th iteration $\widehat{|\mathscr{X}_t^*|} = |\mathscr{X}| \prod_{i=1}^{t} \widehat{p}_i$;
- (intermediate) direct estimator after the t-th iteration $\widehat{|\mathscr{X}_t^*|}_{\text{dir}}$, which is obtained by counting directly the number of distinct points satisfying all clauses.

First Model: 3-SAT (20 × 91) model. This instance of the 3-SAT problem (file: `uf20-01.cnf` on the SATLIB website) consists of $n = 20$ variables and $m = 91$ clauses. The number of solutions is $|\mathscr{X}^*| = 8$.

A typical dynamics of a run of the adaptive splitting Algorithm 9.8.1 with $N = 1000$ and $\varrho = 0.1$ is given in Table 9.3.

Table 9.3: Dynamics of Algorithm 9.8.1 for 3-SAT (20×91) model.

t	$\widehat{\lvert \mathscr{X}_t^* \rvert}$	$\widehat{\lvert \mathscr{X}_t^* \rvert}_{\text{dir}}$	N_t	N_t^{scr}	$\overline{S}_t$	$\underline{S}_t$	$\widehat{p}_t$
1	1.69E+05	0	161	161	89	84	0.161
2	2.00E+04	0	118	117	89	87	0.118
3	5.50E+03	1	276	268	91	88	0.276
4	8.85E+02	1	161	144	91	89	0.161
5	9.03E+01	5	102	61	91	90	0.102
6	7.40E+00	8	82	8	91	91	0.0820

We ran the algorithm 10 times with $N = 1000$ and $\varrho = 0.1$. The average estimate was 8.40 with an estimated relative error (calculated for the 10 independent runs) of 0.07. The direct estimator $\widehat{\lvert \mathscr{X}^* \rvert}_{\text{dir}}$ gave always the exact result $|\mathscr{X}|^* = 8$.

Second model: 3-SAT (75×325) model. This instance of the 3-SAT problem (file: `uf75-01.cnf` on the SATLIB website) consists of $n = 75$ variables and $m = 325$ clauses. The number of solutions is $|\mathscr{X}^*| = 2258$.

Table 9.4 presents the dynamics of the adaptive Algorithm 9.8.1 for this case, using $N = 10,000$ and $\varrho = 0.1$.

Table 9.4: Dynamics of Algorithm 9.8.1 for the random 3-SAT (75×325) model.

t	$\widehat{\lvert \mathscr{X}_t^* \rvert}$	$\widehat{\lvert \mathscr{X}_t^* \rvert}_{\text{dir}}$	N_t	N_t^{scr}	$\overline{S}_t$	$\underline{S}_t$	$\widehat{p}_t$
1	3.82E+21	0	1011	1011	305	292	0.101
2	5.00E+20	0	1309	1309	306	297	0.131
3	5.00E+19	0	1000	1000	309	301	0.100
4	6.08E+18	0	1216	1216	310	304	0.122
5	1.12E+18	0	1843	1843	314	306	0.184
6	1.63E+17	0	1455	1455	312	308	0.146
7	1.94E+16	0	1188	1188	315	310	0.119
8	5.97E+15	0	3083	3083	317	311	0.308
9	1.68E+15	0	2808	2808	317	312	0.281
10	4.45E+14	0	2654	2654	318	313	0.265
11	1.10E+14	0	2473	2473	319	314	0.247
12	2.57E+13	0	2333	2333	319	315	0.233
13	5.09E+12	0	1983	1983	319	316	0.198
14	9.19E+11	0	1806	1806	320	317	0.181
15	1.63E+11	0	1778	1778	321	318	0.178
16	2.49E+10	0	1523	1523	322	319	0.152
17	3.17E+09	0	1273	1273	322	320	0.127
18	3.73E+08	0	1162	1162	323	321	0.116
19	3.56E+07	0	967	967	324	322	0.097
20	2.73E+06	1	766	766	325	323	0.077
21	1.37E+05	0	501	500	324	324	0.050
22	2.35E+03	168	172	168	325	325	0.017

Running the splitting algorithm 10 times independently (using sample size $N = 10,000$ and $\varrho = 0.1$), we obtained an average estimate 2343.1 with an estimated relative error of 0.03. All 2258 solutions were found by pooling the unique particles of these 10 runs.

9.9.2 Independent Sets

Consider a graph $G = (V, E)$ with m edges and n vertices . Our goal is to count the number of independent vertex sets of the graph. A vertex set is called *independent* if no two vertices are connected by an edge, that is, if no two vertices are adjacent; see Figure 9.6 for an illustration of this concept.

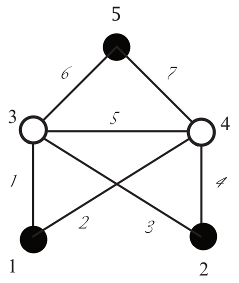

Figure 9.6: Black vertices form an independent set, since they are not adjacent to each other.

Label the vertices and edges as $1, \ldots, n$ and $1, \ldots, m$, respectively, and let $A = (a_{ij})$ be the $m \times n$ matrix with $a_{ij} = 1$ if edge i is connected to vertex j; and $a_{ij} = 0$ otherwise. The i-th row of A thus has exactly two nonzero elements that identify the vertices of the i-th edge. For the graph in Figure 9.6 we have

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

The transpose of A is known as the *incidence matrix* of the graph. Any set $\mathscr{V}$ of vertices can be represented by a binary column vector $\mathbf{x} = (x_1, \ldots, x_n)^\top \in \{0, 1\}^n$, where $x_j = 1$ means that vertex j belongs to $\mathscr{V}$. For this set to be an independent set, it cannot contain both end vertices of any edge. In terms of matrix A and vector $\mathbf{x}$, this can be written succinctly as $A\mathbf{x} \leqslant \mathbf{1}$, where $\mathbf{1}$ is the $n \times 1$ vector of 1s.

For a vertex set representation $\mathbf{x}$, let $S(\mathbf{x})$ be the number of edges whose end vertices do not *both* belong to the vertex set represented by $\mathbf{x}$; that is,

$$S(\mathbf{x}) = \sum_{i=1}^{m} I_{\{\sum_{j=1}^{n} a_{ij} x_j \leqslant 1\}} \ .$$

We wish to estimate the number of independent sets $|\mathscr{X}^*| = |\{\mathbf{x} \in \{0,1\}^n : S(\mathbf{x}) \geqslant m\}|$. As this is a counting problem involving fixed-length binary vectors, we can use Algorithm 9.9.1 for the Markov move in splitting Algorithm 9.8.1.

9.9.2.1 Numerical Experiment We wish to estimate the number of independent sets in a hypercube graph of order n. The vertices of such a graph can be identified with a binary vector of length n — there are thus 2^n vertices. Pairs of vertices are connected by an edge if and only if the corresponding binary vectors differ by one component. Each vertex therefore has n neighbors. Figure 9.7 depicts the hypercube graphs of order $n = 3$ and $n = 4$. It is known that there are 2, 3, 7, 35, 743, 254475, 19768832143, independent sets for hypergraphs of order $n = 0, 1, 2 \ldots, 6$, respectively. For example, for $n = 2$ (the square graph), the independent sets are $\{(0,0),(1,1)\}, \{(1,0),(0,1)\}, \{(0,0)\}, \{(0,1)\}, \{(1,0)\}, \{(1,1)\}$, and the empty set $\emptyset$. Exact results are not known for $n \geqslant 7$.

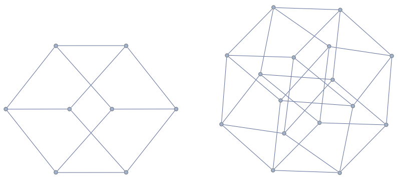
Figure 9.7: Three- and four-dimensional hypercube graphs.

Table 9.5 gives the dynamics of the splitting algorithm for the 6-dimensional hypercube graph. To ensure unbiasedness of the estimator, we used *fixed levels*, which were determined by a pilot run of Algorithm 9.8.1 with parameters $N = 19200$ (100 times the number of edges) and $\varrho = 0.2$. The same sample size was used in the fixed-level run.

Table 9.5: Dynamics of Algorithm 9.5.3 for counting the number of independent sets of the 6-dimensional hypercube graph.

t	$\widehat{\lvert\mathscr{X}_t^*\rvert}$	$\widehat{\lvert\mathscr{X}_t^*\rvert}_{\text{dir}}$	N_t	N_t^{scr}	$\overline{S}_t$	$\underline{S}_t$	$\widehat{p}_t$
1	3.75E+18	0	3908	3907	184	155	0.204
2	8.61E+17	0	4403	4402	183	164	0.229
3	1.92E+17	0	4283	4282	187	170	0.223
4	5.15E+16	0	5151	5150	187	174	0.268
5	1.50E+16	0	5579	5578	189	177	0.291
6	3.38E+15	0	4339	4338	189	180	0.226
7	1.05E+15	2	5943	5942	192	182	0.310
8	2.72E+14	0	4995	4994	191	184	0.260
9	1.26E+14	1	8869	8868	192	185	0.462
10	5.43E+13	4	8286	8285	192	186	0.432
11	2.13E+13	4	7516	7515	192	187	0.391
12	7.60E+12	18	6866	6865	192	188	0.358
13	2.37E+12	38	5977	5976	192	189	0.311
14	6.15E+11	153	4987	4986	192	190	0.260
15	1.23E+11	643	3826	3825	192	191	0.199
16	1.23E+11	3182	19200	19183	192	191	1.000
17	1.99E+10	3116	3116	3115	192	192	0.162

Repeating the fixed-level algorithm 15 times (with the same levels) gave an average estimate of $2.00 \cdot 10^{10}$ with an estimated relative error of 0.014.

For the 7-dimensional case we found an estimate of the number of independent sets of $7.87 \cdot 10^{19}$ with an estimated relative error of 0.001, using 1577 replications of the fixed-level splitting Algorithm 9.5.3, with $N = 10,000$ samples. The levels were determined by a pilot run of the adaptive splitting algorithm, with $N = 10,000$ and $\varrho = 0.2$.

9.9.3 Permanent and Counting Perfect Matchings

The *permanent* of a general $n \times n$ matrix $A = (a_{ij})$ is defined as

$$\mathrm{per}(A) = |\mathscr{X}^*| = \sum_{\mathbf{x} \in \mathscr{X}} \prod_{i=1}^{n} a_{ix_i} \,, \tag{9.5}$$

where $\mathscr{X}$ is the set of all permutations $\mathbf{x} = (x_1, \ldots, x_n)$ of $(1, \ldots, n)$. It is well known that the calculation of the permanent of a *binary* matrix is equivalent to the calculation of the number of perfect matchings in a certain bipartite graph. A *bipartite graph* $G = (V, E)$ is a graph in which the node set V is the union of two disjoint sets V_1 and V_2, and in which each edge joins a node in V_1 to a node in V_2. A *matching* of size m is a collection of m edges in which each node occurs at most once. A *perfect matching* is a matching of size n.

To see the relation between the permanent of a binary matrix $A = (a_{ij})$ and the number of perfect matchings in a graph, consider the bipartite graph G, where V_1 and V_2 are disjoint copies of $\{1, \ldots, n\}$ and $(i, j) \in E$ if and only if $a_{ij} = 1$, for all i and j. As an example, let A be the 3×3 matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}. \tag{9.6}$$

The corresponding bipartite graph is given in Figure 9.8. The graph has three perfect matchings, one of which is displayed in the figure with bold lines. Each such matching corresponds to a permutation $\mathbf{x}$ for which the product $\prod_{i=1}^{n} a_{ix_i}$ is equal to 1.

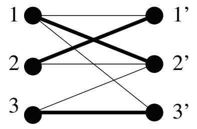

Figure 9.8: Bipartite graph. The bold edges form a perfect matching.

For a general $n \times n$ matrix A, we can apply the splitting algorithm to find the permanent of A by way of the performance function

$$S(\mathbf{x}) = \sum_{i=1}^{n} a_{ix_i} \,.$$

Let $\mathscr{X}_j$ denote the set of permutations with a performance (number of matchings) greater than or equal to t, $t = 1, \ldots, n$. We are interested in calculating $|\mathscr{X}_n| = \text{per}(A)$. Taking into account that $\mathscr{X}_0$ is the set of all permutations of length n, we have the product form (9.3). We apply the following Gibbs move:

Algorithm 9.9.2: Gibbs Move for Permutation Sampling

```
   input  : A permutation $\mathbf{x} = (x_1, \ldots, x_n)$ with $S(\mathbf{x}) \geqslant \gamma$.
   output: A random permutation $\mathbf{Y}$ with the same property.
1  $\mathbf{Y} \leftarrow \mathbf{x}$
2  for $i = 1$ to $n - 1$ do
3      for $j = i + 1$ to $n$ do
4          Let $\mathbf{Y}'$ be the equal to $\mathbf{Y}$ with the $I$-th and $J$-th elements swapped.
5          if $S(\mathbf{Y}') \geqslant \gamma$ then
6              Draw $U \sim \mathsf{U}(0, 1)$.
7              if $U < 1/2$ then $\mathbf{Y} \leftarrow \mathbf{Y}'$

8  return $\mathbf{Y}$
```

9.9.3.1 Numerical Experiment We wish to compute the permanent of the following 30×30 sparse matrix A:

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

Table 9.6 presents the performance of 10 runs of splitting Algorithm 9.8.1 for this problem. We chose $N = 100,000$ and $\varrho = 0.1$. In this case the direct estimator always returns the true value, 266.

Table 9.6: Performance of splitting Algorithm 9.8.1 for the permanent of the 30×30 matrix A.

Run	Iterations	$\widehat{\lvert\mathscr{X}^*\rvert}$	$\lvert\widehat{\mathscr{X}^*}\rvert_{\text{dir}}$	CPU
1	21	261.14	266	115.68
2	21	254.45	266	115.98
3	21	268.04	266	115.65
4	21	272.20	266	117.68
5	21	261.50	266	118.38
6	21	255.03	266	117.10
7	21	261.36	266	116.58
8	21	266.82	266	115.82
9	21	264.76	266	115.84
10	21	254.13	266	116.13
Average	21	261.94	266	116.48

9.9.4 Binary Contingency Tables

Given are two vectors of positive integers $\mathbf{r} = (r_1, \ldots, r_m)$ and $\mathbf{c} = (c_1, \ldots, c_n)$ such that $r_i \leqslant n$ for all i, $c_j \leqslant n$ for all j, and $\sum_{i=1}^m r_i = \sum_{j=1}^n c_j$. A binary contingency table with row sums $\mathbf{r}$ and column sums $\mathbf{c}$ is an $m \times n$ matrix $\mathbf{x} = (x_{ij})$ of zero-one entries satisfying $\sum_{j=1}^n x_{ij} = r_i$ for every row i and $\sum_{i=1}^m x_{ij} = c_j$ for every column j. The problem is to count all such contingency tables. Wang and Zhang [43] give explicit expressions for the number of (0,1)-tables with fixed row and column totals (containing the class of binary contingency tables), which can be used to exactly compute these numbers for small matrices (say $m, n < 15$) and small row/column totals. We wish to apply instead splitting Algorithm 9.8.1.

We define the configuration space $\mathscr{X}$ as the space of binary $m \times n$ matrices where all column sums are satisfied:

$$\mathscr{X} = \left\{ \mathbf{x} \in \{0,1\}^{m+n} : \sum_{i=1}^m x_{ij} = c_j,\ j = 1, \ldots, n \right\}.$$

Clearly, sampling uniformly on $\mathscr{X}$ is straightforward. For the performance function $S : \mathscr{X} \to \mathbb{Z}_-$, we take

$$S(\mathbf{x}) = -\sum_{i=1}^m \left| \sum_{j=1}^n x_{ij} - r_i \right| .$$

That is, we take the negative difference of the row sums $\sum_{j=1}^n x_{ij}$ with the target r_i if the column sums are right.

The Markov (Gibbs) move for the splitting algorithm is specified in Algorithm 9.7. In particular, for each element $x_{ij} = 1$ we check if the performance function does not become worse if we set x_{ij} to 0 and x_{kj} to 1 for any 0-entry x_{kj} (in the same column). We then uniformly select among these entries (including i) and make the swap. Note that in this way we keep the column sums correct.

Algorithm 9.9.3: Gibbs Move for Random Contingency Table Sampling

```
input : A matrix $\mathbf{x} = (x_{ij})$ with performance $S(\mathbf{x}) \geqslant \gamma$.
output: A random matrix $\mathbf{Y} = (Y_{ij})$ with the same property.
1  $\mathbf{Y} \leftarrow \mathbf{x}$
2  for $j = 1$ to $n$ do
3      $\mathcal{I} \leftarrow \{i : Y_{ij} = 1\}$
4      for $i \in \mathcal{I}$ do
5          $\mathcal{K} \leftarrow \{k : Y_{kj} = 0\} \cup \{i\}$.
6          $\mathcal{M} \leftarrow \emptyset$
7          for $k \in \mathcal{K}$ do
8              $\mathbf{Y}' \leftarrow \mathbf{Y}$;  $Y'_{ij} \leftarrow 0$;  $Y'_{kj} \leftarrow 1$
9              if $S(\mathbf{Y}') \geqslant \gamma$ then $\mathcal{M} \rightarrow \mathcal{M} \cup \{k\}$
10         Choose $M$ uniformly from $\mathcal{M}$.
11         $Y_{ij} \leftarrow 0$; $Y_{Mj} \leftarrow 1$
12 return $\mathbf{Y}$
```

9.9.4.1 Numerical Experiments In Chen et al. [10] a delicate sequential importance sampling procedure is used to estimate the number of binary contingency tables. To demonstrate the efficacy and simplicity of the splitting algorithm, we apply the splitting algorithm to two of their numerical examples.

Model 1. Consider the case $m = 12, n = 12$ with row and column sums

$$\mathbf{r} = (2,2,2,2,2,2,2,2,2,2,2,2), \ \mathbf{c} = (2,2,2,2,2,2,2,2,2,2,2,2).$$

The true count value is known to be $21,959,547,410,077,200$ (reported in [43]). Table 9.7 presents a typical dynamics of the adaptive splitting Algorithm 9.8.1 for Model 1 using $N = 50,000$ and $\varrho = 0.5$.

Table 9.7: Typical dynamics of the splitting Algorithm 9.8.1 for Model 1 using $N = 50,000$ and $\varrho = 0.5$.

t	$\widehat{\lvert\mathscr{X}^*\rvert}$	N_t	N_t^{scr}	$\overline{S}_t$	$\underline{S}_t$	$\widehat{p}_t$
1	4.56E+21	13361	13361	−2	−24	0.6681
2	2.68E+21	11747	11747	−2	−12	0.5874
3	1.10E+21	8234	8234	−2	−10	0.4117
4	2.76E+20	5003	5003	−2	−8	0.2502
5	3.45E+19	2497	2497	0	−6	0.1249
6	1.92E+18	1112	1112	0	−4	0.0556
7	2.08E+16	217	217	0	−2	0.0109

The average of the 10 such estimates was $2.17 \cdot 10^{16}$ with an estimated relative error 0.0521. The average CPU time was around 4 seconds.

Model 2. Consider the problem of counting tables with Darwin's finch data as marginal sums given in Chen et al. [10]. The data are $m = 12, n = 17$, with row and columns sums

$$\mathbf{r} = (14, 13, 14, 10, 12, 2, 10, 1, 10, 11, 6, 2),\ \mathbf{c} = (3, 3, 10, 9, 9, 7, 8, 9, 7, 8, 2, 9, 3, 6, 8, 2, 2).$$

The true count value [10] is 67, 149, 106, 137, 567, 600. The average estimate based on 10 independent experiments using sample size $N = 200,000$ and rarity parameter $\varrho = 0.5$, was $6.71 \cdot 10^{16}$ with an estimated relative error of 0.08.

9.9.5 Vertex Coloring

Given a graph $G = (V, E)$ consisting of $n = |V|$ nodes and $m = |E|$ edges, a *q-coloring* is a coloring of the vertices with q colors is such that no two adjacent vertices have the same edge. We showed in Section 9.9.1 how a 2-coloring problem can be translated into a SAT problem. In this section we apply the splitting algorithm to counting the number of q-colorings of a graph.

Let $\mathscr{X} = \{1, \ldots, q\}^n = \{(x_1, \ldots, x_n) : x_i \in \{1, \ldots, q\}, i = 1, \ldots, n\}$ be the set of possible vertex colorings. For a vertex coloring $\mathbf{x} \in \mathscr{X}$, let $S(\mathbf{x})$ be minus the number of adjacent vertices that share the same color. The objective is to count the number of elements in the set $\mathscr{X}^* = \{\mathbf{x} \in \mathscr{X} : S(\mathbf{x}) \geqslant 0\}$.

We use the following Gibbs move in Algorithm 9.8.1:

Algorithm 9.9.4: Gibbs Move for Vertex Coloring

```
   input : A graph $G = (V, E)$, the number of available colors $q$, a threshold
           value $\gamma$, and a vertex coloring $\mathbf{x} = (x_1, \ldots, x_n)$ with performance
           $S(\mathbf{x}) \geqslant \gamma$.
   output: A random vertex coloring $\mathbf{Y}$ with $S(\mathbf{Y}) \geqslant \gamma$.
1  for $i = 1$ to $n$ do
2      $\mathcal{M} \leftarrow \emptyset$
3      for $c = 1$ to $q$ do
4          if $S\big((Y_1, \ldots, Y_{i-1}, c, x_{i+1} \ldots, x_n)\big) \geqslant \gamma$ then $\mathcal{M} \to \mathcal{M} \cup \{c\}$
5      Choose $Y_i$ uniformly at random from $\mathcal{M}$.
6  return $\mathbf{Y} = (Y_1, \ldots, Y_n)$
```

9.9.5.1 Numerical Experiment Consider the 3D order-4 grid graph in Figure 9.9. This graph has $n = 64$ vertices and $m = 3 \times 4^2 \times 3 = 144$ edges.

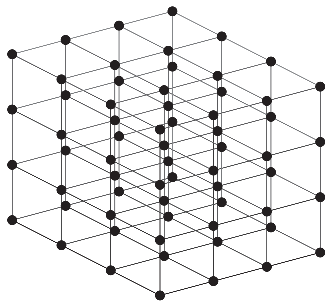

Figure 9.9: 3D grid graph of order 4.

How many 5-colorings are there? We used a pilot run (Algorithm 9.8.1) with sample size $N = 1440$ and $\varrho = 0.2$. The pilot run resulted in 27 levels, which were subsequently used in multiple independent repetitions of the fixed-level Algorithm 9.5.3. In particular, we kept repeating until an estimated relative error of 0.03 was obtained. This took 92 repetitions in 40 seconds. The final estimate was $4.19 \cdot 10^{31}$.

9.10 SPLITTING AS A SAMPLING METHOD

In this section we investigate how the splitting algorithm can be used to sample from complicated distributions. We focus on the fixed-level generalized splitting algorithms in Section 9.5.

We can reason inductively as follows: By acceptance–rejection, each element in $\mathcal{X}_1$ is distributed according to f_1; that is, according to the conditional pdf of $\mathbf{X} \sim f$ given $\mathbf{X} \in \mathscr{X}_1$. Consequently, all elements $\mathbf{Y}'$ generated during the Markov sampling phase at $t = 1$ also have the same density, because this Markov chain has invariant density f_1. It follows, again by acceptance–rejection, that all elements in $\mathcal{X}_2$ (those $\mathbf{Y}'$ that fall in $\mathscr{X}_2$) have density f_2. Markov sampling at phase $t = 2$ creates other particles from f_2, and those that fall in $\mathscr{X}_3$ have density f_3, and so on. As a result, the elements in $\mathcal{X}_T$ are all distributed according to f_T — the conditional pdf of $\mathbf{X} \sim f$ given $\mathbf{X} \in \mathscr{X}_T$. In particular, for applications to counting (Section 9.8), f is typically the uniform pdf on a set $\mathscr{X}$ and f_T the uniform distribution on a much smaller set $\mathscr{X}_T$. In this way the splitting method can be viewed as a multi-level generalization of the acceptance–rejection method.

There is, however, a subtle but important point to make about sampling. The reasoning above does not hold for randomly selected points from the final set, as the distribution of the points in $\mathcal{X}_T$ points depends on the size of the final set N_T: particles corresponding to a small N_T may be quite differently distributed to particles corresponding to a large N_T. So, if we select at random one of the returned points, its distribution differs in general from the conditional distribution given the rare event. Nevertheless, by repeating the algorithm n times, the empirical distribution of the set of all points returned over all n replicates converges to the conditional distribution given the rare event. The following example illustrates this illusive behavior of the splitting algorithm.

■ **EXAMPLE 9.7 A Bivariate Discrete Uniform Example**

We want to apply a generalized splitting algorithm to estimate $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant 8)$, where $\mathbf{X} = (X_1, X_2)$ has a discrete uniform distribution on the square grid $\{1, \ldots, 8\}^2$ and $S(\mathbf{x}) = \max\{x_1, x_2\}$. In addition we want to compare the conditional distribution of the samples returned by a GS algorithm to the true distribution of $\mathbf{X}$ given $S(\mathbf{X}) \geqslant 8$, being the uniform distribution on the set $\{(1,8), \ldots, (8,8), (8,1), \ldots, (8.7)\}$, which has 15 elements.

For our algorithm we take Algorithm 9.5.2 with fixed splitting factors, using two levels: the final level $\gamma_2 = 8$ and the intermediate level $\gamma_1 = 7$; see Figure 9.10. Note that in Algorithm 9.5.2, $\mathscr{X}_t = \{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma_t\}, t = 1, 2$.

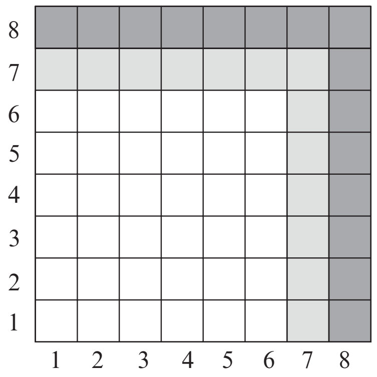

Figure 9.10: Apply the splitting algorithm to sample uniformly from the level set $\{\mathbf{x} : S(\mathbf{x}) \geqslant 8\}$ (dark shading) using the intermediate level set $\{\mathbf{x} : S(\mathbf{x}) \geqslant 7\}$ (light shading).

A point $\mathbf{x} = (x_1, x_2)$ with $S(\mathbf{x}) \geqslant 7$ is split into $r = 2$ points, $\mathbf{Y}^{(1)} = (Y_1^{(1)}, x_2)$ and $\mathbf{Y}^{(2)} = (Y_1^{(2)}, x_2)$, where $Y_1^{(1)}$ and $Y_1^{(2)}$ are drawn independently from the uniform distribution of X_1 conditional on $S((X_1, x_2)) \geqslant 7$. Note that the second coordinate is kept the same in a deliberate attempt to bias the distribution of the (one or two) final states. However, this is a valid Markov move for the splitting algorithm.

The splitting algorithm thus simplifies to the following. We first generate $\mathbf{X} = (X_1, X_2)$ according to the uniform distribution on $\{1, \ldots, 8\}^2$. If $\max\{X_1, X_2\} \geqslant 7$, we resample X_1 twice, conditional on $\max(X_1, X_2) \geqslant 7$, to obtain two states $\mathbf{Y}^{(1)}$ and $\mathbf{Y}^{(2)}$. Among these two states, we retain those for which $S(\mathbf{Y}) \geqslant 8$, if any. Let $M = N_2$ be the number of particles in the final level. So the algorithm returns either $M = 0$ (no points), $M = 1$ and one point $\mathbf{X}^*$, or $M = 2$ and two points $\mathbf{X}_1^*$ and $\mathbf{X}_2^*$.

Simple calculations give the following possible outcomes of this scheme. With probability 1/8, we have $X_2 = 8$, in which case the splitting algorithm outputs $M = 2$ states $\mathbf{X}_1^*$ and $\mathbf{X}_2^*$ that are independent and uniformly distributed over $\{(1, 8), \ldots, (8, 8)\}$. With probability 1/8, we have $X_2 = 7$, in which case the algorithm outputs two copies of state $(8, 7)$ with probability 1/64 and one copy of $(8, 7)$ with probability 14/64. For $j \in \{1, \ldots, 6\}$ with probability 1/32, we have $X_2 = j$ and $X_1 \geqslant 7$. Conditional on this happening, the splitting algorithm outputs two copies of $(8, j)$ with probability 1/4, and one copy with probability 1/2.

Let $\mathbf{Z}$ be a randomly selected state returned by the splitting algorithm (if any). $\mathbf{Z}$ takes values in the set $\mathscr{X}_2$, augmented by an extra state $\emptyset$. Each state $(i, 8)$ has probability $1/64 = 8/512$ of being selected, state $(8, 7)$ has probability $(1/8)(15/64) = 15/512$, state $(8, j)$ has probability $(1/8)(3/4)(1/4) = 3/128 = 12/512$ for $1 \leqslant j \leqslant 6$, and with the remaining probability 361/512 nothing is returned. Table 9.8 summarizes the situation. We see, on one hand, that conditional on $M > 0$, the states do not have the same probability.

Table 9.8: Sampling probabilities for the splitting algorithm.

$\mathbf{x}$	$\mathbb{P}(\mathbf{Z}=\mathbf{x})$	$\mathbb{P}(\mathbf{Z}=\mathbf{x}\,\|\,M>0)$	$\mathbb{P}(\mathbf{X}=\mathbf{x}\,\|\,S(\mathbf{X})\geqslant 8)$
$(1,8),\ldots,(8,8)$	$8/512$	$8/151 \approx 0.053$	$1/15 \approx 0.0667$
$(8,1),\ldots,(8,6)$	$12/512$	$12/151 \approx 0.079$	$1/15$
$(8,7)$	$15/512$	$15/151 \approx 0.1$	$1/15$
$\emptyset$	$361/512$	0	0

On the other hand, $\mathbb{P}(M=2)=89/512$ and $\mathbb{P}(M=1)=62/512$, so that

$$\mathbb{E}[M]/2 = 15/64.$$

It follows that the probability $\ell = \mathbb{P}(S(\mathbf{X}) \geqslant 8)$ can be estimated in an unbiased way via the splitting estimator $\widehat{\ell} = M/2$.

Suppose now that we run the preceding splitting algorithm $n = 100$ times, collect the set (multiset) of all states returned over these n replications, and pick a state at random from it. The second panel of Table 9.8 gives the probabilities of the 15 different states (estimated by simulation) in that case. Now they are all much closer to $1/15 \approx 0.066666667$. These probabilities actually converge exponentially fast to $1/15$ when $n \to \infty$.

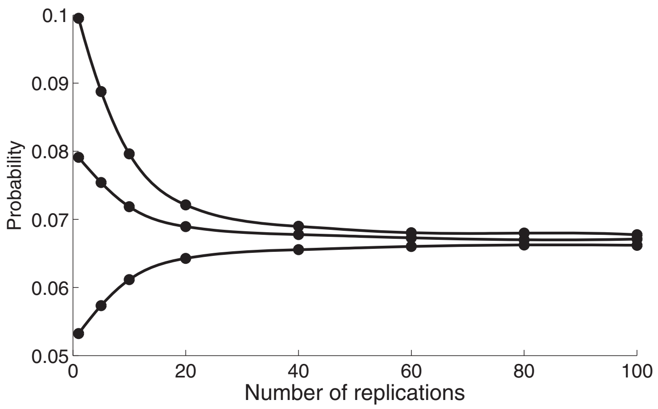

Figure 9.11: Let $\mathbf{Z}$ be a particle selected at random from all particles produced by n independent replications of the splitting algorithm. The graphs show how $\mathbb{P}(\mathbf{Z}=(1,8))$ (bottom), $\mathbb{P}(\mathbf{Z}=(8,1))$ (middle), $\mathbb{P}(\mathbf{Z}=(8,7))$ (top) all converge to $1/15$ as n increases.

9.10.0.2 Numerical Experiment We test the uniformity of the sample returned by the splitting algorithm for the 3-SAT (20×91) model in Section 9.9.1.1. By uniformity we mean that the sample passes the standard χ^2 test. In particular, let $C_1,\ldots,C_8$ be the counts for the 8 solutions, out of $C_1+\cdots+C_8 = m$ samples.

Under the hypothesis of uniformity, the statistic

$$\sum_{i=1}^{8} \frac{(C_i - m/8)^2}{m/8},$$

has approximately a χ^2_7 distribution for large m.

We repeated Algorithm 9.8.1 (with $N = 1000$ and $\varrho = 0.1$) a number of times, pooling the particles in the final multi-set. Table 9.9 shows the results of four χ^2 tests. The first column gives the number of independent replications of the splitting algorithm, and the last column the total (pooled) number of particles in the final multi-set. The middle columns give the value of the test statistic and the corresponding p-value: the probability that under the null-hypothesis (uniformity) the test statistic takes a value greater than or equal to the one observed. Since all p-values are large, the null-hypothesis is not rejected. A histogram of the counts for the last case (10 replications) is given in Figure 9.12.

Table 9.9: Results of the χ^2 test for uniformity.

Replications	Test statistic	p-Value	Total number of samples
1	3.62	0.82	853
2	3.70	0.84	1729
5	5.50	0.60	4347
10	10.10	0.14	8704

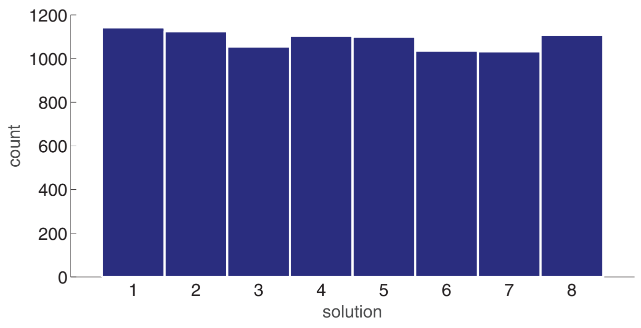

Figure 9.12: Histogram of the number of times, out of 8704, that each of the 8 solutions in the 3-SAT (20 × 91) problem was found.

9.11 SPLITTING FOR OPTIMIZATION

The adaptive splitting algorithm for counting the number of elements in sets of the form $\{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma\}$ can be modified, without much effort, to an optimization algorithm for the performance function S. The resulting algorithm is suitable for solving both discrete and continuous optimization problems, and thus can be considered as an alternative to the standard cross-entropy and MinxEnt methods.

The main difference with the counting version is that the splitting method for optimization does not require knowledge of the final level γ or estimation of the level crossing probabilities $\{p_t\}$.

Algorithm 9.11.1 (Splitting Algorithm for Optimization)

- Input: performance function S, rarity parameter ϱ, sample size N.
- Output: estimators $\widehat{\gamma}^*$ and $\widehat{\mathbf{x}}^*$ of the maximal performance γ^* and the corresponding optimal solution $\mathbf{x}^*$.

1. **Acceptance–Rejection.** Set a counter $t = 1$. Generate a sample $\mathcal{Y}_0 = \{\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(N)}\}$ uniformly on $\mathscr{X} = \mathscr{X}_0$. Compute the threshold $\widehat{\gamma}_1$ as the sample $(1 - \varrho_1)$ quantile of the values of $S(\mathbf{Y}^{(1)}), \ldots, S(\mathbf{Y}^{(N)})$. Let $\mathcal{X}_1 = \{\mathbf{X}_1^{(1)}, \ldots, \mathbf{X}_1^{(N_1)}\}$ be the collection (multi-set) of elements $\mathbf{X}_1 \in \mathcal{Y}_0$ for which $S(\mathbf{X}_1) \geqslant \widehat{\gamma}_1$ (the elite points).
2. **Screening.** Reset $\mathcal{X}_t$ and N_t by screening out (removing) any duplicate particles.
3. **Splitting.** To each particle in the elite multi-set $\mathcal{X}_t$ apply a Markov chain sampler with $\mathsf{U}(\mathscr{X}_t)$ as its invariant (stationary) distribution, and length $\lfloor N/N_t \rfloor$. Apply the Markov chain sampler to $N - \lfloor N/N_t \rfloor = N \mod N_t$ randomly chosen endpoints for one more period. Denote the new entire sample (of size N) by $\mathcal{Y}_t$.
4. **Selecting elites.** Compute level $\widehat{\gamma}_{t+1}$ as the sample $(1 - \varrho_t)$ quantile of the values of $S(\mathbf{Y}), \mathbf{Y} \in \mathcal{Y}_t$. Determine the elite sample, that is, the largest subset $\mathcal{X}_{t+1} = \{\mathbf{X}_{t+1}^{(1)}, \ldots, \mathbf{X}_{t+1}^{(N_{t+1})}\}$ of $\mathcal{Y}_t$ consisting of N_{t+1} points for which $S(\mathbf{X}_{t+1}) \geqslant \widehat{\gamma}_{t+1}$.
5. **Stopping rule.** If for some $t \geqslant d$, say $d = 5$, the overall best performance $\widehat{\gamma}^*$ has not changed in the last d iterations, then **stop** and deliver $\widehat{\gamma}^*$ as the estimator for the optimal function value, and also return the corresponding solution $\widehat{\mathbf{x}}^*$; otherwise, increase the counter $t = t + 1$ and repeat from Step 2.

The main ingredient to be specified is the Markov move in Step 3. As in the rare-event and counting cases, Gibbs moves are convenient; that is, one or more components are chosen from the uniform distribution on a level set $\{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma\}$ while the other components are kept fixed.

■ EXAMPLE 9.8 Knapsack Problem

An intrepid explorer has to decide which of n possible items to pack into his/her knapsack. Associated with each item are m attributes: volume, weight, and so on. Let a_{ij} represent the i-th attribute of item j, and let $A = (a_{ij})$ be the corresponding $m \times n$ matrix. Associated with each attribute i is a maximal capacity c_i; for example, c_1 could be the maximum volume of the knapsack and c_2 the maximum weight the explorer can carry. Similarly, associated with each item j is a reward r_j if this item is taken into the knapsack. Let $\mathbf{c} = (c_i)$ and $\mathbf{r} = (r_j)$ denote the $m \times 1$ vector of capacities and

$n \times 1$ vector of rewards, respectively. Finally, let $\mathbf{x} = (x_j)$ be the $n \times 1$ decision vector of the explorer: $x_j = 1$ if item j is packed and $x_j = 0$ otherwise, $j = 1, \ldots, n$.

The binary knapsack problem can be formulated as: find which items can be packed so as to optimize the total reward, while all attribute constraints are satisfied. In mathematical terms this becomes

$$\max_{\mathbf{x} \in \{0,1\}^n} \quad \mathbf{r}^\top \mathbf{x}$$

subject to

$$A\mathbf{x} \leqslant \mathbf{c}\,.$$

To solve this constrained minimization problem, we can maximize the function

$$S(\mathbf{x}) = \sum_{j=1}^{n} r_j\, x_j + \Big(- \sum_{j=1}^{n} r_j \Big) \sum_{i=1}^{m} I_{\{\sum_{j=1}^{n} a_{ij} x_j > c_i\}},$$

where the second term in $S(\mathbf{x})$ is a penalty term. As a numerical example, consider the Sento1.dat knapsack problem given in `http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/mknap2.txt`. The problem has 30 constraints and 60 variables. The 30×60 attribute matrix A has to be read from the data file (180 rows of 10 numbers) in row format. In particular, the first row of A starts with 47, 774, ...and ends with ..., 898, 37.

We ran the splitting Algorithm 9.11.1 with $\varrho = 0.1$ and $N = 1000$. In 10 out of 10 trials the optimal value 7772 was found. A typical dynamics is given in Table 9.10. The notation in the table is the same as in Section 9.9.1.1.

Table 9.10: Typical dynamics of the splitting Algorithm 9.11.1 for the knapsack problem, using $\varrho = 0.1$ and $N = 1000$.

t	N_t	N_t^{scr}	$\overline{S}_t$	$\underline{S}_t$	p_t
1	100	100	−62496	−202462	0.100
2	100	100	−15389	−109961	0.100
3	100	100	3535	−43463	0.100
4	100	100	4082	−7648	0.100
5	100	100	5984	2984	0.100
6	100	100	6017	4407	0.100
7	100	100	6730	5245	0.100
8	100	100	6664	5867	0.100
9	100	100	7086	6366	0.100
10	100	100	7277	6798	0.100
11	101	101	7475	7114	0.101
12	100	100	7495	7337	0.100
13	100	100	7592	7469	0.100
14	103	101	7695	7563	0.103
15	104	97	7739	7639	0.104
16	108	59	7772	7697	0.108
17	107	8	7772	7739	0.107
18	157	1	7772	7772	0.157

9.11.1 Continuous Optimization

For continuous problems, sampling a component of $\mathbf{X}$ uniformly from a level set $\{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma\}$ may not be easy or fast. An alternative is to replace at some stage t the uniform sampling step with a simpler sampling mechanism; for example, sampling from a normal distribution with standard deviation that depends on the positions of the particles in $\mathcal{X}_t$. Such an approach is taken in [14], giving a very fast and accurate splitting algorithm for continuous optimization. The detailed description of this algorithm for minimizing a continuous function follows.

Algorithm 9.11.2: Splitting for Continuous Optimization

```
input : Objective function $S$, sample size $N$, rarity parameter $\varrho$, scale factor
        $w$, and maximum number of attempts MaxTry.
output: Final iteration number $t$ and sequence $(\mathbf{X}_{\text{best},1}, b_1), \ldots, (\mathbf{X}_{\text{best},t}, b_t)$
        of best solutions and function values at each iteration.
1  Generate $\mathcal{Y}_0 = \{\mathbf{Y}_1, \ldots, \mathbf{Y}_N\}$ uniformly. Set $t \leftarrow 0$ and $N^e \leftarrow \lceil N\varrho \rceil$.
2  while stopping condition is not satisfied do
3      Determine the $N^e$ smallest values, $S_{(1)} \leqslant \cdots \leqslant S_{(N^e)}$, of $\{S(\mathbf{X}), \mathbf{X} \in \mathcal{Y}_t\}$,
         and store the corresponding vectors, $\mathbf{X}_{(1)}, \ldots, \mathbf{X}_{(N^e)}$, in $\mathcal{X}_{t+1}$. Set
         $b_{t+1} \leftarrow S_1$ and $\mathbf{X}_{\text{best},t+1} \leftarrow X_{(1)}$.
4      Draw $B_i \sim \mathsf{Bernoulli}(\frac{1}{2})$, $i = 1, \ldots, N^e$, with $\sum_{i=1}^{N^e} B_i = N \bmod N^e$.
5      for $i = 1$ to $N^e$ do
6          $R_i \leftarrow \left\lfloor \frac{N}{N^e} \right\rfloor + B_i$                    // random splitting factor
7          $\mathbf{Y} \leftarrow \mathbf{X}_{(i)}$; $\mathbf{Y}' \leftarrow \mathbf{Y}$
8          for $j = 1$ to $R_i$ do
9              Draw $I \in \{1, \ldots, N^e\} \setminus \{i\}$ uniformly and let $\boldsymbol{\sigma}_i \leftarrow w|\mathbf{X}^{(i)} - \mathbf{X}^{(I)}|$.
10             Generate a uniform permutation $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$ of $(1, \ldots, n)$.
11             for $k = 1$ to $n$ do
12                 for Try = 1 to MaxTry do
13                     $\mathbf{Y}'(\pi_k) \leftarrow \mathbf{Y}(\pi_k) + \boldsymbol{\sigma}_i(r_k)Z$,    $Z \sim \mathsf{N}(0,1)$
14                     if $S(\mathbf{Y}') > S(\mathbf{Y})$ then $\mathbf{Y} \leftarrow \mathbf{Y}'$ and break.
15             Add $\mathbf{Y}$ to $\mathcal{Y}_{t+1}$
16     $t \leftarrow t + 1$
17 return $\{(\mathbf{X}_{\text{best},k}, b_k), k = 1, \ldots, t\}$
```

In Line 9, $\boldsymbol{\sigma}_i$ is a n-dimensional vector of standard deviations obtained by taking the component-wise absolute difference between the vectors $\mathbf{X}^{(i)}$ and $\mathbf{X}^{(I)}$, multiplied by a constant w. The input variable `MaxTry` governs how much computational time is dedicated to updating a component. In most cases the choices $w = 0.5$ and `MaxTry` $= 5$ work well. It was found that a relatively high ϱ value works well, such as $\varrho = 0.4, 0.8$, or even $\varrho = 1$. The latter case means that at each stage t *all* samples from $\mathcal{Y}_{t-1}$ carry over to the elite set $\mathcal{X}_t$.

A possible stopping condition is to stop if the overall best found function value does not change over d (e.g., $d = 5$) consecutive iterations.

■ **EXAMPLE 9.9 Minimizing the Rastrigin Function**

The n-dimensional *Rastrigin* function is defined by

$$S(\mathbf{x}) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right], \quad \mathbf{x} \in \mathbb{R}^n .$$

It has served as a benchmark function for many optimizers (e.g., see [14]). In most test cases the function is minimized over the n-dimensional cube $[-5.12, 5.12]^n$.

For the case $n = 50$, Figure 9.13 shows the maximum and minimum function values obtained in each iteration of Algorithm 9.11.2, using the parameters $N = 50$, $\varrho = 0.8$, and $w = 0.5$. We see an exponential rate of convergence to the true minimum, 0.

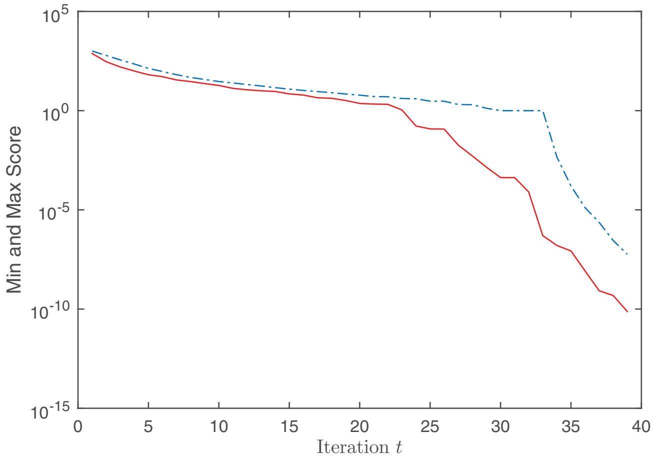

Figure 9.13: Performance of the continuous splitting algorithm for the 50-dimensional Rastrigin function.

PROBLEMS

9.1 Implement Algorithm 9.2.1 and verify the results in Example 9.1.

9.2 For the self-avoiding walk counting problem in Example 9.1 compare the efficiency of splitting Algorithm 9.2.1 with that of Algorithm 5.8.3 on Page 164, which uses sequential importance sampling. Improve both algorithms by combining them into a sequential importance resampling method.

9.3 Apply Algorithm 9.3.1 to the gambler's ruin problem in Example 9.3 and verify the numerical results given in the example.

9.4 Implement and run a splitting algorithm on a synthetic max-cut problem described in Problem 8.10 on Page 299, for a network with $n = 400$ nodes, with $m = 200$. Compare with the CE algorithm.

9.5 Let X_1 and X_2 be independent exponentially distributed random variables with mean 1. Implement the generalized splitting Algorithm 9.5.3 to estimate the probability $\ell = \mathbb{P}(X_1 + X_2 \geqslant 30)$. Use $T = 10$, with levels $\gamma_i = 3i$ for $i = 1, \ldots, 10$, and sample size (fixed effort) $N = 100$. For the Markov transition use the Gibbs move of Algorithm 9.5.4. Generate 800 independent runs of the splitting algorithm, and compare the overall estimate with the true probability. Also estimate the relative error of the overall estimator. Finally, plot all points obtained in the 800 replications whose sum is greater than or equal to 30.

9.6 Repeat Problem 9.5, but now determine the levels adaptively via a pilot run with Algorithm 9.6.1, using $\varrho = 0.1$ and $N = 10^3$. Is there any improvement in the relative error compared with using the levels $1, 2, \ldots, 5$? Also run Algorithm 9.6.1 100 times with an effort of $N = 10^4$ and compare with the previous results.

9.7 Suppose in the bridge network of Example 4.2 on Page 110 the link states $X_1, \ldots, X_5$ all have failure probability $q = 1-p = 0.001$. Implement Algorithm 9.5.3 with $N = 10^2$ using the Gibbs move in Algorithm 9.7.1. Run the algorithm 1000 times to obtain a 95% confidence interval for the system unreliability. Specify which intermediate levels were used.

9.8 Let $\{A_i\}$ be an arbitrary collection of subsets of some finite set $\mathscr{X}$. Prove the following *inclusion–exclusion* principle, which can be useful for decomposing a complicated counting problem into possibly simpler ones.

$$|\cup_i A_i| = \sum |A_i| - \sum_{i<j} |A_i \cap A_j| + \sum_{i<j<k} |A_i \cap A_j \cap A_k| - \cdots .$$

9.9 A famous problem in combinatorics is the *distinct representatives* problem, which is formulated as follows: Given a set $\mathscr{A}$ and subsets $\mathscr{A}_1, \ldots, \mathscr{A}_n$ of $\mathscr{A}$, is there a vector $\mathbf{x} = (x_1, \ldots, x_n)$ such that $x_i \in \mathscr{A}_i$ for each $i = 1, \ldots, n$ and the $\{x_i\}$ are all distinct (that is, $x_i \neq x_j$ if $i \neq j$)?

a) Suppose, for example, that $\mathscr{A} = \{1, 2, 3, 4, 5\}$, $\mathscr{A}_1 = \{1, 2, 5\}$, $\mathscr{A}_2 = \{1, 4\}$, $\mathscr{A}_3 = \{3, 5\}$, $\mathscr{A}_4 = \{3, 4\}$, and $\mathscr{A}_5 = \{1\}$. Count the total number of distinct representatives.

b) Argue why the total number of distinct representatives in the problem above is equal to the *permanent* of the following matrix A:

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

9.10 Let $X_1, \ldots, X_n$ be independent Bernoulli random variables, each with success probability 1/2. Devise a splitting algorithm to efficiently estimate $\ell = \mathbb{P}(X_1 + \cdots + X_n \geqslant \gamma)$ for large γ (e.g., $\gamma = 0.9n$). For the Markov move, use Algorithm 9.9.1. Show the dynamics of the algorithm similar to the tables in Section 9.9.

9.11 Let $\mathscr{X}$ be the set of permutations $\mathbf{x} = (x_1, \ldots, x_n)$ of the numbers $1, \ldots, n$, and let

$$S(\mathbf{x}) = \sum_{j=1}^{n} j\, x_j \; . \tag{9.7}$$

Let $\mathscr{X}^* = \{\mathbf{x} : S(\mathbf{x}) \geqslant \gamma\}$, where γ is chosen such that $|\mathscr{X}^*|$ is very small relative to $|\mathscr{X}| = n!$. Implement splitting Algorithm 9.8.1 to estimate $|\mathscr{X}^*|$, for $n = 8$ and $\gamma = 201$, using $N = 10^4$ and rarity parameter $\varrho = 0.1$ for all levels. Repeat the algorithm 1000 times and report the overall estimate $\widehat{|\mathscr{X}^*|}$, its estimated relative error, and also the direct estimator (9.4). Use the following Markov move at each iteration t. Given a permutation $\mathbf{x} = (x_1, \ldots, x_n)$, select I and J uniformly from $\{1, \ldots, n\}$, swap x_I and x_J to obtain a new permutation $\mathbf{x}'$, and check whether $S(\mathbf{x}') \geqslant \gamma_t$. If so, accept the move to $\mathbf{x}'$; otherwise, repeat (with the same $\mathbf{x}$). Note that $I = J$ results in $\mathbf{x}' = \mathbf{x}$.

9.12 The volume V_n of the n-dimensional simplex $\mathscr{Y}_n = \{(y_1, \ldots, y_n) : y_i \geqslant 0, i = 1, \ldots, n, \sum_{i=1}^{n} y_i \leqslant 1\}$ is equal to $1/n!$. Devise a splitting algorithm to estimate V_n for $n = 5$, $n = 10$, and $n = 20$. Compare the method with the crude Monte Carlo (acceptance–rejection) approach.

9.13 Write a splitting algorithm for the TSP, using the 2-opt neighborhood structure. For various test cases, obtain a table similar to Table 8.11 on Page 291.

9.14 How many *Hamiltonian cycles* exist in a given graph? That is, how many different tours (cycles) does the graph contain that visit each node (vertex) exactly once, apart for the beginning/end node? Note that the problem of finding a particular Hamiltonian cycle is a special case of the TSP in which the distances between adjacent nodes are 1, and other distances are ∞, and the objective is to find a tour of length n.

A possible way to solve the Hamiltonian cycles counting problem via splitting is as follows:

1. Run Algorithm 9.8.1 to estimate the permanent of the adjacency matrix $A = (a_{ij})$ of the graph, as in Section 9.9.3. Let $\mathcal{X}_T$ be the screened elite samples (permutations of $(1, \ldots, n)$) at the final level, and let $\widehat{P}$ be the estimator of the permanent. Note that, by construction, for each such permutation $(\pi_1, \ldots, \pi_n)$ it holds that $\sum_{i=1}^{n} a_{ix_i} = n$.

2. Proceed with one more iteration of Algorithm 9.8.1 to produce a set $\mathcal{Y}_T$ of permutations with $\sum_{i=1}^{n} a_{ix_i} = n$. Each such permutation corresponds to an incidence matrix $B = (b_{ij})$ defined by $b_{ij} = a_{ij}$ for $j = \pi_i$, $i = 1, \ldots, n$ and $b_{ij} = 0$ otherwise. This matrix B can be viewed as the one-step transition matrix of a Markov chain in which from each state only one transition is possible. If, starting from 1, the Markov chain returns to 1 in exactly n steps, then the corresponding path is a Hamiltonian cycle; otherwise, it is not.

   Let ζ be the proportion of Hamiltonian cycles in $\mathcal{Y}_T$.

3. Deliver $\widehat{P}\zeta$ as an estimator of the number Hamiltonian cycles in the graph.

Implement this method for the graph in Figure 9.14, using $N = 100{,}000$ and $\varrho = 0.1$. Also provide the direct estimator of the number of Hamiltonian cycles.