Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

LINEAR ALGEBRA AND SEQUENTIAL IMPORTANCE SAMPLING FOR NETWORK RELIABILITY

David G. Harris

U.S. Department of Defense
17100 Science Drive
Bowie MD 20715, USA

Francis Sullivan

IDA/Center for Computing Sciences
17100 Science Drive
Bowie, MD 20715, USA

Isabel Beichl

National Institute of Standards & Technology
100 Bureau Drive
Gaithersburg, MD 20899

ABSTRACT

The reliability polynomial of a graph gives the probability that a graph is connected as a function of the probability that each edge is connected. The coefficients of the reliability polynomial count the number of connected subgraphs of various sizes. Algorithms based on sequential importance sampling (SIS) have been proposed to estimate a graph's reliability polynomial. We develop a new bottom-up SIS algorithm for estimating the reliability polynomial by choosing a spanning tree and adding edges. This algorithm improves on existing bottom-up algorithms in that it has lower complexity $\approx O(E^2)$ as opposed to $O(EV^3)$, and it uses importance sampling to reduce variance.

1 INTRODUCTION

Let G be a connected graph with vertex set V and edge set E. We define $r(x)$, the reliability polynomial of G, to be the probability that the graph remains connected when edges are removed independently with probability x. This function is a polynomial which can be written as

$$r(x) = \sum_{i=0}^{|E|-|V|+1} N_i x^i (1-x)^{|E|-i}$$

where N_i is the number of connected subgraphs of G with $|E|-i$ edges.

It will be more convenient for us to work with this factored form. We define the reliability generator

$$p(x) = \sum_i N_i x^i$$

In particular, we use the term, "low" order coefficient to refer to N_i for small i, that is, the number of ways of removing just a few edges from G and still have the resulting graph connected. And we use "high" order coefficient to refer to N_i for large i, that is the number of connected graphs that can be formed by adding a few edges to a spanning tree of G. The reliability generator can be used to evaluate the reliability polynomial itself,

$$r(x) = (1-x)^{|E|} p\Big(\frac{x}{1-x}\Big)$$

Computing the reliability of a graph is computationally hard (Ball 1986). A variety of algorithms have been proposed to approximate the reliability of a graph. Some of these algorithms seek to compute $r(x_0)$ or $1-r(x_0)$ for some fixed probability x_0, such as Fishman (1986) or Karger (1996). Other algorithms determine some or all of the coefficients of the generator, such as Colbourn, Debroni, and Myrvold (1988). These problems are related, but are not equivalent especially in terms of measuring error. For example, a fully-polynomial randomized approximation scheme (fpras) for evaluating $r(x_0)$ where x_0 is held constant does not yield a fpras for the coefficients of p. The algorithm of Beichl, Cloteaux, and Sullivan (2010), which we denote TOP-DOWN, is a sequential importance sampling (SIS) algorithm. See Liu (2001) for an explanation of SIS.

TOP-DOWN ALGORITHM

```
Set m ← 1.
For k = 0,...,K do
    Set N̂_k ← m/k!
    Determine D, the set of available non-bridges of G.
    Choose an edge e ∈ D uniformly.
    Set m ← m|D|.
    Set G ← G − e
Endfor
```

This pseudocode is for one estimate. Multiple samples can be averaged to yield an estimate $\bar{N}_k$ where we use the bar to indicate a mean. We draw as many samples as necessary until a reasonable accuracy is achieved. We may describe this as a top-down algorithm, as it starts with the original graph and removes edges until reaching a tree. This algorithm as presented has running time $O(|E|^2)$. However, as described in Harris, Sullivan, and Beichl (2011), the running time can be reduced to approximately $O(E\log(V)\alpha(V))$, where $\alpha()$ is the inverse Ackermann function. To simplify the notation in this paper, we will abuse notation so that E and V can mean $|E|$ and $|V|$. We also let $\tau(G)$ represent the number of spanning trees of a graph G.

While this top-down algorithm is very effective at estimating the low-order coefficients of the reliability generator, its relative variance tends to increase on the high-order coefficients. In many physical situations, only the low-order coefficients of the reliability generator are important. For example, we would likely be interested in knowing when the network has a non-negligible chance of staying connected after a series of disruptions. Later coefficients, corresponding to a large number of broken links and an exponentially small probability of connectivity, would probably not be physically relevant. However, in cases where the high-order coefficients are desired, Colbourn, Debroni, and Myrvold (1988) proposed an alternative SIS algorithm. This algorithm, which we denote BOTTOM-UP-UNIFORM, can be viewed as a "bottom-up" algorithm: it starts with a spanning tree of the graph and inserts edges. We sketch a simplified version of the algorithm described in Colbourn, Debroni, and Myrvold (1988). Their algorithm has some additional optimizations which will not be relevant in this paper.)

BOTTOM-UP-UNIFORM ALGORITHM

```
Choose a spanning tree H of G uniformly at random.
For k = K,...,0 do
    Set N̂_k ← τ(G) (K choose k) / τ(H)
    Choose an edge, e, of G − H uniformly at random.
    Set H ← H ∪ e
Endfor
```

This algorithm tends to have low relative variance on high order coefficients. Choosing the spanning tree uniformly at random can be done with Wilson's algorithm (Wilson 1996).

Some reliability algorithms return a single estimate $\hat{p}(x)$ of the reliability generator. In this paper we are estimating the entire polynomial, not simply evaluating the polynomial at particular value of x, and so our estimate actually consists of all K coefficients.

In this paper, we will analyze and improve the BOTTOM-UP-UNIFORM. We call the new method BOTTOM-UP-NEW. We make two key improvements. First, the BOTTOM-UP-UNIFORM algorithm requires computing the number of spanning trees of certain subgraphs of the graph G. Computing this directly is very expensive, costing $O(EV^3)$ runtime. We will describe a more efficient method, which speeds up BOTTOM-UP-UNIFORM leading to a runtime of $O(E^2\text{polylog}(E,\varepsilon))$. It should be noted that Colbourn, Myrvold, and Neufeld (1996) uses a different but similar linear algebra technique for unranking arboresences.

The second improvement is in the choice of edge e. Algorithm BOTTOM-UP-UNIFORM selects among the possible edges uniformly (all edges are equally likely). We will describe an importance sampling technique for selecting the edges, which greatly improves the accuracy.

In a later paper we will analyze the variance of BOTTOM-UP-NEW, from a theoretical point of view (including worst-case variance). Here we concentrate on practical problems. In a later paper we will also describe how to combine the TOP-DOWN with BOTTOM-UP-NEW, in effect achieving the best of both worlds — low variance on both high-order and low-order coefficients.

2 ESTIMATING HIGH-ORDER COEFFICIENTS

As shown in Beichl, Cloteaux, and Sullivan (2010), Algorithm TOP-DOWN has very good accuracy on the low-order coefficients of the reliability generator. However the algorithm has increasing relative variance for the high-order coefficients. This is a consequence of its top-down design. This is unfortunate because the highest-order coefficient of the reliability generator, which corresponds to the number of subtrees of G, can be computed exactly in polynomial time using the well-known Kirchoff formula. See, for example Harris, Hirst, and Mossinghoff (2008). Let A_G be the adjacency matrix of G, and let D be a diagonal matrix whose ith entry is the degree of vertex v_i. Recall that $\tau(G)$ is the number of spanning trees of G. We have the identity

$$\tau(G) = \det L,$$

where $L = (D - A_G)_{11}$ denotes the minor of $D - A_G$ obtained by removing the first row and column. This can be evaluated in $O(V^3)$ work.

Next, consider how one might seek to estimate the penultimate coefficient of the reliability generator. This coefficient counts the number of graphs with V $(= (V-1)+1)$ edges. To obtain such a graph, one might take a spanning tree T and add any of the edges of other K edges of G. However, this would count graphs multiple times; each such graph $T \cup e$ would be counted with multiplicity $\tau(T \cup e)$. Accounting for this factor, one has that

$$\text{penultimate coefficient of } p_G = \tau(G)(E - V + 1)\mathbf{E}[1/\tau(T \cup e)]$$

where the expectation, $\mathbf{E}$, is taken over all spanning trees T and all edges $e \in G - T$.

To estimate this quantity unbiasedly, one might sample spanning trees T and edges $e \in G - T$, compute the resulting $\tau(T \cup e)$, and extract a sample mean of $1/\tau(T \cup e)$.

In general, coefficient $K - k$ of the reliability generator is given by the formula

$$\tau(G)\mathbf{E}[\sum_{e_1,\ldots,e_k} 1/\tau(T \cup e_1 \cdots \cup e_k)]$$

where T is chosen uniformly among spanning trees of G, $e_1, \ldots, e_k$ are distinct edges of $G - T$.

This formula suggests that there are two search spaces to cover: the space of spanning trees T and the space of edges $e_1, \ldots, e_k$. Our algorithm will handle the first using simple Monte Carlo sampling, and will handle the second using Sequential Importance Sampling. This will be a bottom-up algorithm: we start

from a spanning tree and add edges to it until recovering the original graph. We will denote this Algorithm BOTTOM-UP-UNIFORM:

Algorithm BOTTOM-UP-UNIFORM

Precompute $\tau(G)$
Choose a spanning tree H of G uniformly at random (e.g. with Wilson's algorithm)
For $k = K, \ldots, 0$ do
  Set $\hat{N}_k \leftarrow \frac{\tau(G)\binom{K}{k}}{\tau(H)}$
  Choose an edge of $G - H$ uniformly at random.
  Set $H \leftarrow H \cup e$
Endfor

Algorithm BOTTOM-UP-UNIFORM is equivalent to the algorithm described in Colbourn, Debroni, and Myrvold (1988). In this algorithm as presented, as in many SIS algorithms, variance is introduced when different choices for e at each step lead to estimates of divergent size. (Variance is also introduced with choice of spanning tree, but we will not use the importance to deal with this.) In general, in SIS algorithms, it is best to choose an importance function which equalizes, as far as possible, the contributions from each choice. In our case, this is not really possible, for two reasons. First, we cannot compute the exact contribution — that is precisely the quantity we are trying to estimate in the first place. Second, we are seeking to estimate the entire reliability generator, and so our selection contributes to multiple estimated coefficients, which we cannot equalize simultaneously.

In our case, we will use a simple importance function so that the contribution from each edge *e with respect to the next estimate* $\hat{N}_k$ is equalized. Letting $P(e)$ denote the probability of inserting edge e, the choice of $P(e)$ that achieves this is

$$P(e) \propto \frac{1}{\tau(H \cup e)}$$

which yields the following algorithm:

Algorithm BOTTOM-UP-NEW

Precompute $\tau(G)$.
Set $m \leftarrow 1$, the running total of all importance functions seen so far.
Choose a spanning tree H of G uniformly at random (e.g. with Wilson's algorithm).
For $k = K, \ldots, 0$ do
  Set $\hat{N}_k \leftarrow m\tau(G)\binom{K}{k}$
  Set

$$S = \sum_{d \in G - H} \tau(H)/\tau(H \cup d)$$

  Update

$$m \leftarrow mS/k$$

  Choose an edge $e \in G - H$ with probability $P(e)$ proportional to $1/\tau(H \cup e)$.
  Set $H \leftarrow H \cup e$
Endfor

In practice, Algorithm BOTTOM-UP-NEW has much better variance than Algorithm BOTTOM-UP-UNIFORM on the high-order coefficients.

The results of Algorithm BOTTOM-UP-NEW are almost the opposite of Algorithm TOP-DOWN: high-order coefficients are computed very accurately (in fact, the highest-order coefficient is computed exactly), while the relative variance increases for the low-order coefficients.

3 COMPUTING τ EFFICIENTLY

Algorithms BOTTOM-UP-UNIFORM and BOTTOM-UP-NEW appear to be very expensive because they requires extensive computation of the τ function. Directly computing τ using the determinant formula costs $O(V^3)$ work and $O(V^2)$ memory (for Gaussian elimination). It would appear that at each iteration, we must compute $\tau(G \cup e)$ for each edge; and altogether there are K iterations. In all, this would appear to cost $O(E^2V^3)$ work and $O(V^2)$ memory for Algorithm BOTTOM-UP-NEW.

Instead of recomputing $\tau(G \cup e)$ at each stage, we will instead keep track of the value

$$\lambda(e) = \tau(H \cup e)/\tau(H)$$

at each stage of the algorithm, for each edge $e \in G - H$.

Initially, when H is a spanning tree, we set

$$\lambda\langle i,j\rangle \leftarrow \text{tree-distance } (i,j) + 1$$

for each edge $\langle i,j\rangle$.

Next, when we update H by inserting an edge e, we must update λ. Let d be another edge in H. We need to compute the updated $\lambda'(d) = \tau(H \cup e \cup d)/\tau(H \cup e)$.

We will find the following notation convenient. If $e = \langle i,j\rangle$ is any edge in G, we define the column vector δ_e to be the vector which is $+1$ in coordinate i, is -1 in coordinate j, and is zero elsewhere. Observe that when edge e is added to H, the matrix L changes by $\delta_e\delta_e^T$:

$$L_{H\cup e} = L_H - \delta_e\delta_e^T$$

Now let us examine how to recompute λ':

$$\begin{aligned}
\lambda'(d) &= \tau(H \cup e \cup d)/\tau(H \cup e) \\
&= \det(L_H - \delta_d\delta_d^T - \delta_e\delta_e^T)/\det(L_H - \delta_e\delta_e^T) \\
&= \det\Big(I - (L_H - \delta_e\delta_e^T)^{-1}\delta_d\delta_d^T\Big) \\
&= 1 - \delta_d^T(L_H - \delta_e\delta_e^T)^{-1}\delta_d \\
&= 1 - \delta_d^T(L_H^{-1} + \frac{L_H^{-1}\delta_e\delta_e^T L_H^{-1}}{1 - \delta_e^T L_H^{-1}\delta_e})\delta_d \\
&= 1 - \delta_d^T\big(L_H^{-1} + \frac{uu^T}{1 - \delta_e^T u}\big)\delta_d \qquad \text{where } u = L_H^{-1}\delta_e \\
&= 1 - \delta_d^T L_H^{-1}\delta_d + \frac{(\delta_d^T u)^2}{1 - \delta_e^T u} \\
&= \lambda(d) + \frac{(\delta_d^T u)^2}{\lambda(e)}
\end{aligned}$$

This leads to the following technique for updating λ. Whenever we add edge e to H, we use the conjugate gradient method, a sparse matrix technique, to compute $u = L_H^{-1}\delta_e$. We then update each $\lambda(d)$

$$\lambda(d) \leftarrow \lambda(d) + (\delta_d^T u)^2/\lambda(e)$$

In total, this technique allows us to reduce the memory requirement to $O(E)$ and the complexity to $O(EV)$. Furthermore, in practice the conjugate gradient method needs far less than $O(EV)$ iterations to compute an adequate approximation, so the work requirement per iteration is more like $O(E\text{polylog}(E,\varepsilon))$ where ε is the desired accuracy after averaging multiple samples together.

To make this completely transparent, we present Algorithm BOTTOM-UP-NEW explicitly describing how to use and update λ:

```
Algorithm BOTTOM-UP-NEW (with linear algebra)

   Precompute τ(G).
   Set m ← 1, the running total of all importance functions seen so far.
   Choose a spanning tree H of G uniformly at random (e.g. with Wilson's
   algorithm).
   For each edge ⟨i,j⟩ ∈ G−H, set

                        λ⟨i,j⟩ ← tree-distance (i,j)+1

   For k = K,...,0 do
      Set p̂_k ← mτ(G)(K choose k)
      Set S = Σ_{d∈G−H} 1/λ(d) and update m ← mS/k
      Choose an edge e ∈ G−H with probability P(e) = (1/λ(e))/S.
      Using conjugate gradient method, compute u ≈ L_H^{-1} δ_e
      For each edge d ∈ G−H update

                        λ(d) ← λ(d) + (δ_d^T u)^2/λ(e)

      Endfor
      Set H ← H ∪ e
   Endfor
```

This is mathematically equivalent to the previous Algorithm BOTTOM-UP-NEW. In total, these techniques reduce the work requirement of both Algorithm BOTTOM-UP-NEW and Algorithm BOTTOM-UP-UNIFORM to a maximum of $O(E^2V)$ and memory requirement to $O(E)$; and in practice much less than this if we accept a numerical approximation (more like $O(E^2\text{polylog}(E,\varepsilon))$).

Also, for large values of k, the matrix L is extremely sparse. In particular, it has many density-two rows (corresponding to degree-one vertices in H). For inverting a matrix of this form, a simple application of conjugate gradient is not best. A better strategy is to use a partial Gaussian elimination, eliminating all degree-two rows, and performing a preconditioned conjugate gradient on the resulting smaller matrix. We have found that the inverse diagonal preconditioner works well.

Another trick to speed up this step is to note that the Kirchoff formula works for any minor of the matrix $D-A$. To reduce the density of the resulting matrix L, we chose to omit the row and column i corresponding to the highest-degree vertex (after pruning away degree-one vertices).

In practice, these tricks can speed up Algorithm BOTTOM-UP-NEW by a factor of four or more for large k. For small k, the matrix L is not so sparse and these techniques gain little.

4 RESULTS

Let us turn to some sample graphs to assess the performance of these algorithms. Our first test case is an Erdős-Renyi graph with 100 vertices and 150 edges. Figure 1 depicts the relative variance of the three Algorithms: Algorithm TOP-DOWN, Algorithm BOTTOM-UP-NEW and Algorithm BOTTOM-UP-UNIFORM.

As we have discussed earlier, the relative variance of Algorithm TOP-DOWN increases for large k and the relative variance of Algorithm BOTTOM-UP-NEW increases for small k. Algorithm BOTTOM-

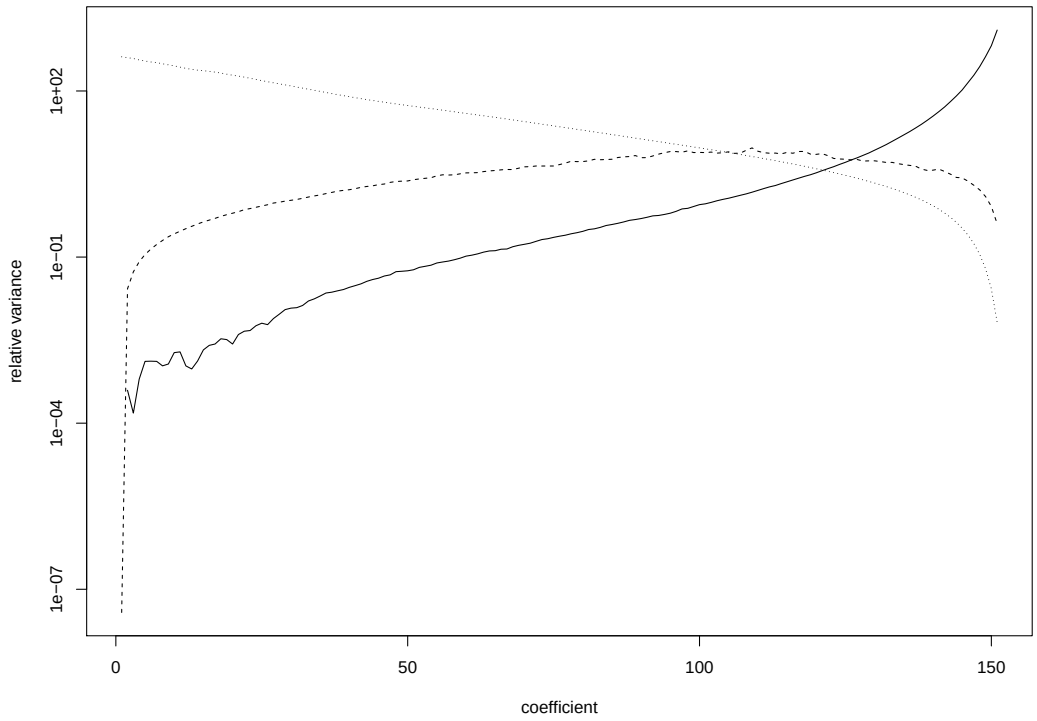

Figure 1: Relative variance of various algorithms on Erdős Renyi graph, 100 vertices 250 edges Key: Solid = TOP-DOWN algorithm, Dashed = BOTTOM-UP-UNIFORM (uniform distribution), Dotted = BOTTOM-UP-NEW (new importance sampling)

UP-UNIFORM, as we have discussed, has better relative variance than Algorithm BOTTOM-UP-NEW on low-order coefficients and worse variance on high-order coefficients. Since Algorithm TOP-DOWN also has low relative variance on these low-order coefficients, Algorithm BOTTOM-UP-UNIFORM seems to be always dominated by either Algorithm TOP-DOWN or Algorithm BOTTOM-UP-NEW.

Test Case 2 comes from a Delaunay triangulation on 100 points, yielding a graph with 100 vertices and 286 edges. Figure 2 depicts the relative variance of Algorithms TOP-DOWN, BOTTOM-UP-UNIFORM, and BOTTOM-UP-NEW.

For any coefficient k, either TOP-DOWN or BOTTOM-UP-NEW has lower variance than BOTTOM-UP-UNIFORM. Furthermore, either TOP-DOWN or BOTTOM-UP-NEW is at least as fast as BOTTOM-UP-UNIFORM.

5 CONCLUSION

We have described a new bottom-up algorithm for estimating the reliability polynomial of a connected graph. This new algorithm achieves substantially better speed and accuracy compared to the algorithm of Colbourn, Debroni, and Myrvold (1988). This improved accuracy is especially beneficial because the top-down algorithm of Beichl, Cloteaux, and Sullivan (2010) has poor accuracy in the high-order coefficients.

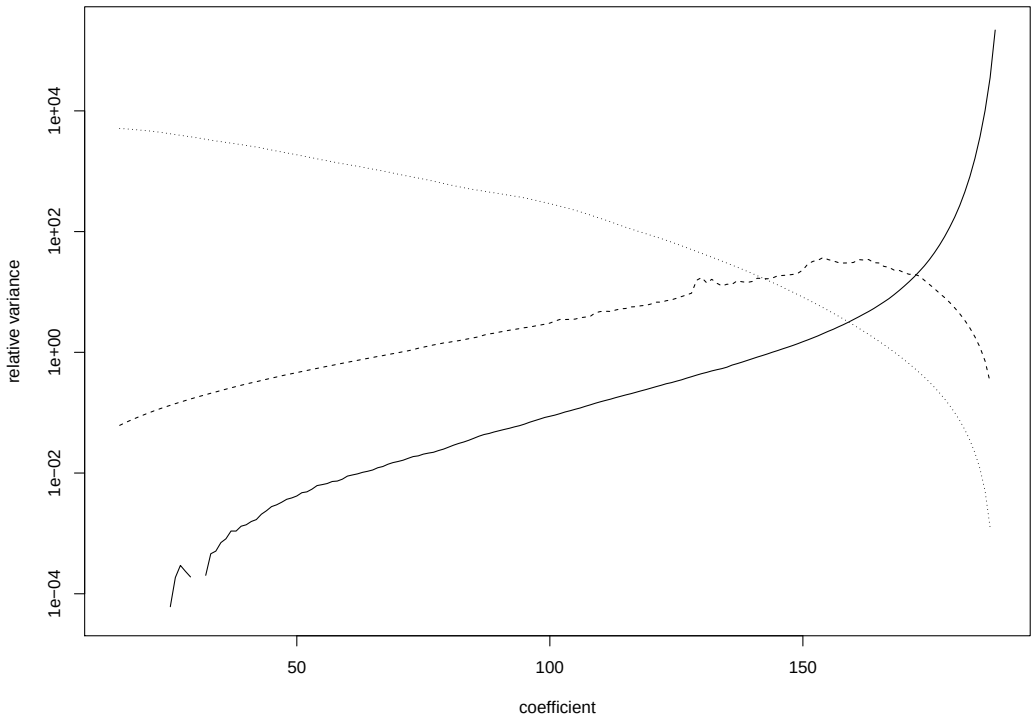

Figure 2: Relative Variance of various algorithms on Delanuay data. Key: Solid = TOP-DOWN algorithm, Dashed = BOTTOM-UP-UNIFORM (uniform distribution), Dotted = BOTTOM-UP-NEW (new distribution)

REFERENCES

Ball, M. 1986. "Computational Complexity of Network Reliability Analysis: An Overview". *IEEE Trans. Reliability* 35:230–239.

Beichl, I., B. Cloteaux, and F. Sullivan. 2010. "An Approximation Algorithm for the Coefficients of the Reliability Polynomial". *Congressus Numerantium* 197:143–151.

Colbourn, C., B. Debroni, and W. Myrvold. 1988. "Estimating the Coefficients of the Reliability Polynomial". *Proceedings of the Seventeenth Manitoba Conference on Numerical Mathematics and Computing* 62:217–223.

Colbourn, C., W. Myrvold, and E. Neufeld. 1996. "Two algorithms for Unranking Arborescences". *Journal of Algorithms* 20:268–281.

Fishman, G. 1986. "A Monte Carlo Sampling Plan for Estimating Network Reliability". *Operations Research* 34:581–594.

Harris, D., F. Sullivan, and I. Beichl. 2011. "Fast Sequential Importance Sampling to Estimate the Graph Reliability Polynomial". *to appear*.

Harris, J. M., J. L. Hirst, and M. J. Mossinghoff. 2008. *Combinatorics and Graph Theory*. New York, New York: Springer Verlag.

Karger, D. 1996. "A Randomized Fully Polynomial Time Approximation Scheme for the All Terminal Network Reliability Problem". *SIAM Journal of Computing* 29:11–17.

Liu, J. S. 2001. *Monte Carlo Strategies in Scientific Computing*. New York, New York: Springer Verlag.

Wilson, D. 1996. "Generating Random Spanning Trees More Quickly than the Cover Time". *Proceedings of the twenty-eighth annual ACM Symposium of Theory of Computing*:296–313.

AUTHOR BIOGRAPHIES

DAVID G. HARRIS is a researcher at the United States Department of Defense. He received a Bachelor of Science in mathematics from Harvard University and is currently enrolled in the Applied Mathematics doctoral program at the University of Maryland. His research interests are in algorithms and cryptography. His email address is davidgharris29@hotmail.com

FRANCIS SULLIVAN is Director of the IDA Center for Computing Sciences in Bowie MD. He received a B.S. degree in Physics from the Pennsylvania State University and a Ph.D. in Mathematics from the University of Pittsburgh. He is the author of a book and papers in algorithm design, functional analysis, Monte Carlo methods, and computational physics. He serves on several editorial boards and is past Editor-in-Chief of *Computing in Science and Engineering*, a joint publication of the IEEE Computer Society and the American Institute of Physics. His email address is fran@super.org.

ISABEL BEICHL is a mathematician in the Information Technology Laboratory at the National Institute of Standards and Technology, Gaithersburg MD . She received a B.A. in Mathematics from the University of Pennsylvania and her Ph.D. in Mathematics from the Cornell University. She is the Editor-in-Chief of *Computing in Science and Engineering*. Her research interests are in probabilistic methods for discrete problems. Her email address is isabel.beichl@nist.gov.