An Approximation Algorithm for the Coefficients of the Reliability Polynomial*

Isabel Beichl and Brian Cloteaux
National Institute of Standards and Technology
Gaithersburg, Maryland, USA
{isabel.beichl,brian.cloteaux}@nist.gov

Francis Sullivan
Institute for Defense Analyses Center for Computational Sciences
Bowie, Maryland, USA
fran@super.org

Abstract

The reliability polynomial gives the probability that a graph remains connected given that each edge in it can fail independently with an identical probability. While in general determining the coefficients of this polynomial is #*P*-complete, we give a randomized algorithm for approximating its coefficients. When compared to the known approximation method of Colbourn, Debroni and Myrvold, our method empirically shows a much faster rate of convergence.

1 The reliability polynomial

A classic problem in network research, the *all-terminal reliability* problem, looks to provide a simplified measure of the reliability of a network by giving the probability that a network stays connected given that each edge in it can fail with some identical probability. More formally, a network is modeled as an undirected graph $G = (V,E)$ where V is a set of vertices and E is a set of edges. The problem assumes that the vertices of the network are always reliable, while each edge can independently fail with a probability $1-p$. For this model, the all-terminal reliability $R(G;p)$ is the probability that G is connected when the edges are allowed to fail.

Computation of the all-terminal reliability problem has a simple recursive form. For any $e \in E$, we can express the probability of remaining connected as

$$R(G;p) = (1-p) \cdot R(G - \{e\};p) + p \cdot R(G \backslash \{e\};p)$$

where $G - \{e\}$ is the graph where the edge e is deleted from G, and $G \backslash \{e\}$ represents the graph (not necessarily simple) where the edge e is contracted in G. From this form, it is easy to show inductively that the probability over all p is a polynomial (often called the *reliability polynomial*). It is also easy to see that this recursive formulation potentially requires an exponential number of evaluations. Unfortunately, there is strong evidence that no polynomial time algorithm for computing the reliability polynomial exists since the computational complexity of the all-terminal reliability problem has been shown to be #*P*-complete [7]. Thus if we are interested in computing the values of the reliability polynomial, the best we can expect to do, in general, is to approximate the values of the polynomial.

Much of the research in approximating the reliability polynomial has been in providing point estimates of the polynomial. In a breakthrough result, Karger produced a fully polynomial randomized approximation scheme (fpras) for estimating the point values of the reliability polynomial [3, 4]. However, for directly estimating the coefficients of the reliability polynomial, there is a much smaller body of literature. The principal result is from Colbourn, Debroni, and Myrvold [2] who provided a routine which reduces the variance over simple sampling in estimating the coefficients. Nel and Colbourn [6] then went on to explore how to combine theoretical bounds with approximation methods. We extend this line of research by providing a new algorithm for directly estimating the coefficients. We then compare the speed of our algorithm to the method of Colbourn, Debroni, and Myrvold.

2 Computing the reliability polynomial by counting nodes in a tree

The reliability polynomial has been long studied and a number of equivalent forms of the polynomial that have been discovered [1]. One form of reliability polynomial which we will focus on to describe our result is

$$R(G;p) = \sum_{i=0}^{|E|} C_i p^i (1-p)^{|E|-i}$$

where C_i is the number of spanning connected subgraphs of G having exactly i edges. Since the all-terminal reliability problem is #*P*-complete, we can deduce that computing the number of spanning connected subgraphs must also be #*P*-complete for certain values of i. Throughout the remainder of this paper, when

we speak of estimating the coefficients of the reliability polynomial, we are actually speaking of estimating the C_i values for the above form of the reliability polynomial.

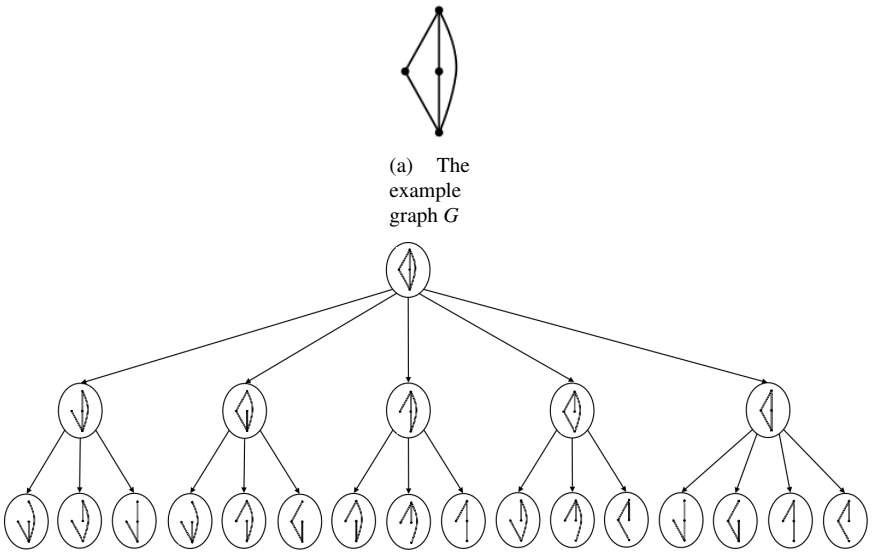
Our approach to approximating the reliability polynomial is based on the observation that we can convert the original problem into an equivalent problem of counting the number of nodes at each level of a certain type of rooted tree. To show this, we define a structure that we call the *connected subgraph tree*. For any simple connected graph G, there is an associated connected subgraph tree T. This tree T is a rooted directed tree and every node in the tree is a connected subgraph in G. The root node is the graph G itself. Each directed edge in the tree T goes from the node G_1 to G_2 if and only if there exists an edge e such that $e \in G_1$ and $G_1 - \{e\} = G_2$. An example of the construction of this tree is shown in Figure 1.

The reason we are interested in this formulation is because of the following important property of the connected subgraph tree: Starting with level 0 as the root node, each level i is composed of exactly $i!$ copies of each of the connected subgraphs having exactly $|E| - i$ edges. Thus each level of the tree has exactly $i! \cdot C_{|E|-i}$ nodes. We can prove this observation using induction starting by noticing that the base case trivially follows from the definition of its root node. If we assume that the statement holds for level n of the tree, then we can consider the nodes for level $n+1$. From the induction hypothesis and the definition of the connected subgraph tree, every node on level $n+1$ must be a connected subgraph with exactly $|E| - n - 1$ edges. Since every connected subgraph is a subgraph of G, then for any connected graph G' with $|E| - n - 1$ edges there are $n+1$ edges in G that are not in G'. Every permutation of those $n+1$ edges defines a path from the root node G, to a node G' on level $n+1$. Thus there are $(n+1)!$ different paths from G to G' defining $(n+1)!$ copies of G' on level $n+1$ and establishing our premise.

This formulation reduces the problem of estimating each of the coefficients to estimating the number of nodes on each level of a connected subgraph tree. Knuth gave a randomized algorithm for counting the number of nodes in a tree [5] which we use in order to estimate the number of nodes on each level of the connected subgraph tree. This resulting scheme is detailed in Algorithm 1. The proof that our algorithm converges to the number of connected subgraphs follows directly from the correctness proof in Knuth's paper.

In examining the time complexity for Algorithm 1, we see that iterating through all the coefficients requires $O(|E|)$ steps. For each iteration, the time is dominated by the check for edges in the graph that are bridges (i.e. edges whose removal disconnect the graph). By decomposing the graph into its biconnected components, we can check if any component has exactly one edge and thus is a bridge. Using Tarjan's algorithm [9], we can check for bridges in $O(|V| + |E|)$ time which gives a total running time for the algorithm of $O(|E|^2)$.

A more difficult question and one that is still open is exactly how fast our algorithm is able to converge. In Knuth's original paper, he showed a large variance



(a) The example graph G

(b) The connected subgraph tree for the graph G

Figure 1: An example of the connected subgraph tree for the graph G

for his scheme for counting the nodes of general trees. Stockmeyer later expanded this result by showing that for some highly unbalanced trees the number of samples using Knuth's algorithm needed to estimate the tree size is $\Omega(\sqrt{n})$ where n is the number of nodes [8]. In other words, for some trees an exponential number of samples in the height of the tree is required for the method to converge. We point out that the lower bound of Stockmeyer does not necessarily apply to our problem since the class of the trees we are estimating are highly structured. Most noticeably, all connected subgraph trees are balanced, in other words every path from the root to a leaf node has equal length. From empirical evidence presented in the next section, we speculate that our algorithm may even be a fully polynomial randomized approximation scheme.

3 Comparison with the Colbourn, Debroni, and Myrvold algorithm

Since the variance of our sampling method is still open, we look to compare the rate of convergence of our method to the established method of Colbourn, Debroni, and Myrvold (CDM). We performed a series of computational experiments over three classes of random graphs. Our experiments consisted of generating a random graph and computing a set of 20 point estimates with .99 confidence in-

Input: a graph $G = (V, E)$
Output: vector of coefficients C

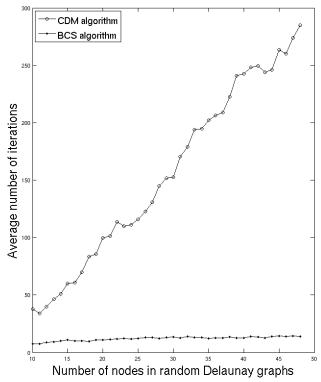
$a_0 \leftarrow 1$
for $k \leftarrow 1$ *to* $|E| - |V| + 1$ **do**
  D is the set of all edges which if removed do not disconnect G
  $a_k \leftarrow |D|$
  Uniformly select an edge e from the set D
  $G \leftarrow G - \{e\}$
end
for $k \leftarrow 0$ *to* $|E| - |V| + 1$ **do**
  $C_{|E|-k} \leftarrow \frac{\prod_{0 \le i \le k} a_i}{k!}$
end
for $k \leftarrow 0$ *to* $|V| - 2$ **do**
  $C_k \leftarrow 0$
end
return C

Algorithm 1: Scheme for estimating the coefficients for the graph G

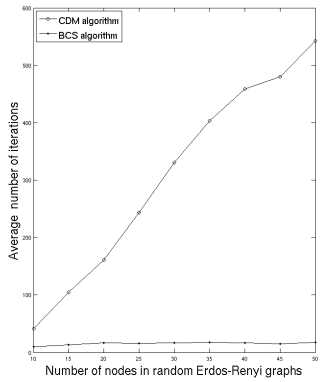
tervals of the reliability polynomial. We then iterated our algorithm and the CDM algorithm until both produced a reliability polynomial where the 20 points were within the confidence interval we had computed. We repeated this test over a series of graphs and averaged the results.

Two of the three classes of random graphs that we compared against were suggested by Karger and Tai [4]. The first class is a set of Delaunay graphs which are created by randomly placing n points on the unit square and then computing the Delaunay triangulation of those points. The second class, which is related to the Delaunay graphs, are the near neighbor graphs. These are also produced by placing n random points on the unit square, but now each point is randomly connected to d out of its r nearest neighbors using Euclidean distance. For our tests, we used the values $r = 8$ and $d = 4$ as suggested in the Karger and Tai paper. Finally, since the first two classes produce sparse graphs, we decided to generate a third class of dense graphs. We generated a series of Erdős-Renyi random graphs with edge probabilities of 0.75. The results of these experiments are shown in Figure 2.

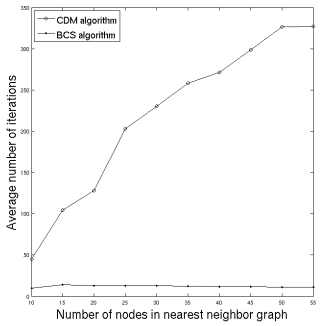
The results of our tests seem to suggest that our algorithm converges to within the confidence interval in a logarithmic number of samples to the number of nodes in the graph, while the CDM algorithm requires a linear number of samples. This observation seems to hold whether the graphs are sparse or dense.



(a) Delaunay graphs



(b) Erdős-Renyi graphs



(c) Nearest neighbor graphs

Figure 2: Comparison of the average number of iterations required for both algorithms

4 Handling large coefficients

When computing the number of connected subgraphs using the above approximation scheme, the sizes of these values, even for moderately sized graphs, can quickly overflow the word size of a computer. In order to maintain these values efficiently, we present a simple method for storing the coefficient values as logarithms while still being able to average the values between samples of Knuth's method.

While descending the connected subgraph tree, we are estimating the value $i! \cdot C_i$ with the product series $a_1 \cdot a_2 \cdot \ldots \cdot a_i$. In computing the average, we denote

the sth sample of a_r as $a_{r,s}$ while the average of s samples of a_r is $\langle a_r \rangle_s$, i.e.

$$\langle a_r \rangle_s = \frac{\sum_{i=1}^{s} a_{r,i}}{s}$$

What we are interested in is a scheme to compute $\log \langle a_1 \cdot a_2 \cdot a_3 \cdot \ ... \ \cdot a_n \rangle$. To accomplish this, we can convert this problem to the easier problem of saving the running averages of the values $\langle a_1 \rangle, \langle a_2 \rangle, ..., \langle a_n \rangle$ throughout the computation. These running averages can be updated after every sample with the formula

$$\langle a_r \rangle_{n+1} = \langle a_r \rangle_n \cdot \frac{n}{n+1} + \frac{a_{r,n+1}}{n+1}$$

We then can rewrite the computation as

$$\log \langle a_1 \cdot a_2 \cdot ... \cdot a_n \rangle = \log \langle a_1 \rangle + \log \langle a_2 \rangle + \log \langle a_n \rangle + X$$

Thus the main idea is to find an efficient method to compute X. Towards this, for each a_r, we compute an additional value S_r where

$$S_r = \begin{cases} 1 & \text{if } r = 1, \\ \frac{\langle a_1 \cdot a_2 \cdot ... \cdot a_r \rangle}{\langle a_1 \rangle \langle a_2 \rangle ... \langle a_r \rangle} & \text{otherwise} \end{cases}$$

Thus in general the nth iteration of S_r is

$$S_{r,n} = \frac{\langle a_1 \cdot a_2 \cdot ... \cdot a_r \rangle_n}{\langle a_1 \rangle_n \cdot \langle a_2 \rangle_n \cdot ... \cdot \langle a_r \rangle_n}$$

We can compute $S_{r,n}$ in terms of $S_{r,n-1}$ and the values for $a_{i,n}$ for $1 \leq i \leq r$.

$$T = \frac{k}{k+1} \cdot S_{i,k} + \frac{1}{k+1} \left(\frac{a_{1,k+1} \cdot a_{2,k+1} \cdot ... \cdot a_{i,k+1}}{\langle a_1 \rangle_k \cdot \langle a_2 \rangle_k \cdot ... \cdot \langle a_i \rangle_k} \right)$$

$$S_{i,k+1} = T \cdot \frac{\langle a_1 \rangle_k \cdot \langle a_2 \rangle_k \cdot ... \cdot \langle a_i \rangle_k}{\langle a_1 \rangle_{k+1} \cdot \langle a_2 \rangle_{k+1} \cdot ... \cdot \langle a_i \rangle_{k+1}}$$

In order to simplify the computation of T and S, we use the intermediate values P and Q where

$$Q_{l,i} = \begin{cases} 1 & \text{if } l = 0, \\ Q_{l-1,i} \cdot \frac{a_{l,i}}{\langle a_l \rangle_{i-1}} & \text{otherwise} \end{cases}$$

$$P_{l,i} = \begin{cases} 1 & \text{if } l = 0, \\ P_{l-1,i} \cdot \frac{\langle a_l \rangle_{i-1}}{\langle a_l \rangle_i} & \text{otherwise} \end{cases}$$

We can compute both values as we increment through the values during the sampling process. Thus computing $S_{i,k+1}$ becomes

$$T = \frac{k}{k+1} \cdot S_{i,k} + \frac{1}{k+1} \cdot Q_{i,k+1}$$

$$S_{i,k+1} = T \cdot P_{i,k+1}$$

We also note that for the first sample, $S_{i,1} = 1$ for all i. By treating this sample as a special case, it allows us to avoid overflow problems with Q and P when sampling extremely large instances.

Now, by setting $X = \log S_n$, we can compute the logarithm of the average

$$\log \langle a_1 \cdot a_2 \cdot \ldots \cdot a_n \rangle = \log \langle a_1 \rangle + \log \langle a_2 \rangle + \ldots + \log \langle a_n \rangle + \log S_n$$

By integrating this method for handling large coefficients into our approximation algorithm, we have been able to approximate the reliability polynomial for networks with well over 1000 nodes and 6000 edges.

5 Conclusion

We have outlined a new method to estimate the coefficients of the reliability polynomial. This approach has shown several advantages. The first is that while there are several algorithms which provide point approximations of the polynomial, our approach directly approximates the actual coefficients of the polynomial itself. Secondly, when compared to the known approximation method of Colbourn et al., our method has shown a much faster rate of convergence.

The major open question left from our investigation is establishing the variance of our sampling routine. Because of the structure of the connected subgraph trees, we have pointed out that known lower bounds do not seem to apply in this instance. We speculate that this algorithm may even be the first fully polynomial randomized approximation scheme to directly compute the coefficients of the reliability polynomial.

6 Acknowledgments

We would like to thank Javier Bernal and Desh Ranjan for their helpful comments on our manuscript.

References

[1] C. J. COLBOURN, *The Combinatorics of Network Reliability*, Oxford University Press, Inc., New York, NY, USA, 1987.

[2] C. J. COLBOURN, B. M. DEBRONI, AND W. J. MYRVOLD, *Estimating the coefficients of the reliability polynomial*, Congressus Numerantium, 62 (1988), pp. 217–223.

[3] D. R. KARGER, *A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem*, SIAM Journal on Computing, 29 (1999), pp. 492–514.

[4] D. R. KARGER AND R. P. TAI, *Implementing a fully polynomial time approximation scheme for all terminal network reliability*, in SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 1997, Society for Industrial and Applied Mathematics, pp. 334–343.

[5] D. E. KNUTH, *Estimating the efficiency of backtrack programs*, Mathematics of Computation, 29 (1975), pp. 121–136.

[6] L. D. NEL AND C. J. COLBOURN, *Combining Monte Carlo estimates and bounds for network reliability*, Networks, 20 (1990), pp. 277–298.

[7] J. S. PROVAN AND M. O. BALL, *The complexity of counting cuts and of computing the probability that a graph is connected*, SIAM Journal on Computing, 12 (1983), pp. 777–788.

[8] L. J. STOCKMEYER, *On approximation algorithms for #P*, SIAM Journal on Computing, 14 (1985), pp. 849–861.

[9] R. TARJAN, *Depth-first search and linear graph algorithms*, SIAM Journal on Computing, 1 (1972), pp. 146–160.