Implementing a Fully Polynomial Time Approximation Scheme for All Terminal Network Reliability

David R. Karger[*] Ray P. Tai

March 3, 1997

1 Abstract

The classic all-terminal network reliability problem posits a graph, each of whose edges fails (disappears) independently with some given probability. The goal is to determine the probability that the network becomes disconnected due to edge failures. The practical applications of this question to communication networks are obvious, and the problem has therefore been the subject of a great deal of study. Since it is $\sharp\mathcal{P}$-complete, and thus believed hard to solve exactly, a great deal of research has been devoted to *estimating* the failure probability. A comprehensive survey can be found in [Col87].

The first author recently presented an algorithm for approximating the probability of network disconnection under random edge failures. In this paper, we report on our experience implementing this algorithm. Our implementation shows that the algorithm is practical on networks of moderate size, and indeed works better than the theoretical bounds predict. Part of this improvement arises from heuristic modifications to the theoretical algorithm, while another part suggests that the theoretical running time analysis of the algorithm might not be tight.

Based on our observation of the implementation, we were able to devise analytic explanations of at least some of the improved performance. As one example, we formally prove the accuracy of a simple heuristic approximation for the reliability. We also discuss other questions raised by the implementation which might be susceptible to analysis.

1.1 The Problem. Formally, a network is modeled as a a graph G, each of whose edges e is presumed to fail (disappear) with some probability p_e, and thus to survive with probability $q_e = 1 - p_e$ (a simplified version that we will focus on assumes each $p_e = p$, but our techniques apply to the general case). Network reliability is concerned with determining the probabilities of certain connectivity-related events in this network. The most basic question of *all-terminal network reliability* is determining the probability that the network becomes disconnected.

The all-terminal network reliability problem is $\sharp\mathcal{P}$-complete [Val79, PB83]. That is, it is in a complexity class at least as intractable as $\mathcal{NP}$ and therefore seems unlikely to have a polynomial time solution. Attention therefore turned to approximation algorithms.

Perhaps the best approximation algorithm that can be hoped for is a *polynomial time approximation scheme* (PTAS). In this definition, the interesting measure is the running time of the approximation algorithm as a function of the problem size n and the error parameter ϵ, and the goal is for a running time which is polynomial in n for each fixed ϵ (e.g., $O(2^{1/\epsilon}n)$). If the running time is also polynomial in $1/\epsilon$, the algorithm is said to be a *fully polynomial time approximation scheme (FPTAS)*.

Recently, the first author gave a randomized FPTAS for the all terminal network reliability problem [Kar95]. Given a failure probability p for the edges, the algorithm, in time polynomial in n and $1/\epsilon$, returns a number P that estimates the probability FAIL(p) that the graph becomes disconnected. With high probability, P is in the range $(1 \pm \epsilon)$FAIL(p). The algorithm is Monte Carlo, meaning that the approximation is correct with high probability but that it is not possible to verify its correctness.

1.2 Our Results. In this paper, we report on an implementation of the algorithm presented in [Kar95]. Our first observation is that the algorithm is indeed practical. On graphs of moderate size (up to 60 nodes) the algorithm ran in under an hour on a mid-range workstation (60MHz SPARC 20). On all our input families, we observed a roughly $O(n^3)$ running time with reasonable constants. This is significantly better than the claimed bound of the theoretical algorithm.

Part of this improved running time is due to heuristic modifications (discussed below) that we made while implementing the algorithm. Part, however, cannot be

[*]MIT Laboratory for Computer Science, Cambridge, MA 02139.
email: `karger@lcs.mit.edu`
URL: `http://theory.lcs.mit.edu/~karger`

explained by these changes, and instead suggests that the running-time analysis of [Kar95] may not be tight. We will discuss our reasons for this belief.

Examination of some of our results suggested that in many cases, FAIL(p) could be accurately approximated by an easily computed formula involving the number of near-minimum graph cuts. In this paper we give a proof that this is indeed the case.

2 The Algorithm

In this section, we describe the approximation algorithm. Additional details can be found in [Kar95]. An outline of the algorithm is as follows. If the failure probability is large, then it can be estimated via direct *Monte Carlo Algorithm (MCA)* simulation of the edge failures. If the failure probability is small, we find that only small cuts in the graph are likely to fail. We enumerate these cuts using the Recursive Contraction Algorithm (RCA) of [KS95] and then find the probability that one of them fails using the Self Adjusting Coverage Algorithm (SACA) of Karp, Luby, and Madras [KLM89]. We now give some additional details of the algorithm.

2.1 Monte Carlo Simulation. The most obvious way to estimate FAIL(p) is through Monte Carlo simulations. Given the failure probability p for each edge, we can "simulate" edge failures by flipping an appropriately biased random coin for each edge. We can then test whether the network is connected. If we do this many times, then the fraction of trials in which the network becomes disconnected should intuitively provide a good estimate of FAIL(p). Karp and Luby [KL85] investigated this idea formally, and observed (a generalization of) the following.

THEOREM 2.1. *$O((\log n)/(\epsilon^2 FAIL(p)))$ trials suffice to estimate FAIL(p) to within $1 \pm \epsilon$ with high probability.*

COROLLARY 2.1. *FAIL(p) can be estimated to within $(1 \pm \epsilon)$ in $\tilde{O}(m/\epsilon^2 FAIL(p))$ time.*

Proof. Each trial requires random choices for each edge followed by a depth first search. We remark that since FAIL(p) is unknown, our estimate is based on the number of trials that must be performed before a certain number of network failures occurs.

2.2 Restriction to Small Cuts. The flaw of the simulation approach is that it is too slow for small values of FAIL(p). We now describe a scheme that becomes effective precisely when direct simulation fails.

Observe that a graph becomes disconnected precisely when all of the edges in some cut of the graph fail (a *cut* is a minimal-under-inclusion set of edges whose removal partitions the graph into more than one connected component). Throughout this paper, we assume that our graph has *minimum cut* c—that is, that the smallest cut in the graph has exactly c edges. An α-*minimum cut* is a cut with value at most αc. The following are proved in [Kar95]:

FACT 2.1. *If each edge of a graph with minimum cut c fails with probability p, then the probability that the graph becomes disconnected is at least p^c.*

LEMMA 2.1. *There are at most $n^{2\alpha}$ α-minimum cuts.*

THEOREM 2.2. *Suppose a graph has minimum cut c and that each edge of the graph fails independently with probability p, where $p^c = n^{-(2+\delta)}$ for some $\delta > 0$. Then the probability that a cut of value exceeding αc fails is at most $n^{-\alpha\delta}(1 + 2/\delta)$. In particular, if*

$$\alpha > 1 + 2/\delta + \frac{\ln\left(\frac{\delta+2}{\epsilon\delta}\right)}{\delta \ln n}$$

then the probability the network fails is approximated to within ϵ by the probability that a cut of value less than α fails.

We have already argued that when p^c is large, we can use a *Monte Carlo Algorithm (MCA)* that directly simulates edge failures. The previous theorem shows that the when p^c is small, we need only determine the probability that a cut of value near c fails. The *Recursive Contraction Algorithm (RCA)* can be used to generate all cuts of value less than αc in $\tilde{O}(n^{2\alpha})$ time [KS95]. Since the cuts of value near c can be enumerated, we can generate a polynomial size boolean expression (with a variable for each edge, true if the edge has failed) which is true if a small cut has failed. We then need to determine the probability that this boolean expression is true, which can be done using *Self Adjusting Coverage Algorithm (SACA)* of Karp, Luby, and Madras [KL85, KLM89]:

LEMMA 2.2. *For the DNF counting problem, there is a randomized FPTAS running in $\tilde{O}(m/\epsilon^2)$ time on a size m formula.*

COROLLARY 2.2. *When $p^c = n^{-(2+\delta)}$, FAIL(p) can be estimated to within ϵ in $O(cn^{2+4/\delta}\left(\frac{2+\delta}{\epsilon\delta}\right)^{1/\delta})$ time.*

THEOREM 2.3. ([KAR95]) *FAIL(p) can be estimated in $O(\min(mn^{3.5}, cn^{4.7})/\epsilon^2)$ time.*

Proof. If $p^c > 1/n^{3.5}$, run the Monte Carlo Algorithm. Otherwise, use the RCA/SACA method.

Additional tweaking can slightly reduce the running time exponent, but it still appears rather large. In particular, even in the optimistic case where $c = O(1)$ and $m = O(nc) = O(n)$, we see that the theoretical running time bound is roughly $O(n^{4.5})$, which seems unlikely to be practical.

3 Implementation

Implementing the approximation algorithm was relatively straightforward. The MCA simply required a depth first search routine. To implement the scheme for small failures probabilities, we began with an implementation of the RCA written as part of an experimental study of minimum cut algorithms [CGK+]. That algorithm was designed to find a single minimum cut. The implementation of [CGK+] essentially follows [KS95], though with some modifications that sped it up in practice. We had to make further modifications to find all near-minimum cuts instead of finding a single minimum cut. This had a significant impact on performance, since we could no longer use the *Padberg-Rinaldi heuristics* that give significant speedup when looking for one minimum cut [CGK+]. The running time of the RCA was the dominant factor in the running time of our algorithm.

Once we had generated the set of near minimum cuts, we built a boolean formula and passed it to a straightforward implementation of the self adjusting coverage algorithm (SACA) of [KLM89]. The time spent in this part was typically negligible (less than 10%) compared to that spent in the RCA. Thus, when RCA/SACA dominates MCA, it is basically the running time of the RCA that determines the running time of the approximation algorithm.

We now discuss two changes we made to improve in practice on the algorithm's theoretical performance bound.

3.1 Dovetailing. Although the theoretical algorithm specified which option (MCA or RCA/SACA) should be run for any specific probability, we wanted our algorithm to choose the option that was best in practice for the particular input instance. To achieve this, we dovetailed calls to both algorithms on any particular input, and stopped as soon as one of them terminated. This, of course, gave us an algorithm which took twice the time of the faster of the two choices. Rather than performing the dovetailing ourselves, we chose to let the operating system handle it. We forked two processes that separately ran the two algorithms. The operating system interleaved their operation, allocating roughly 50% of the CPU cycles to each process. When one process finished, it sent a signal to the other process. Both processes then reported their CPU time usages; these times were added to give the overall running time. In the (now common) case of a two-processor machine, this implementation will clearly gain a factor of two in speed with no additional programming necessary.

3.2 Optimizing α. In the theoretical algorithm, α is chosen so that the probability a greater than α-minimum cut fails, namely $n^{-\alpha\delta}(1+2/\delta)$ is less than ϵ times the probability that a minimum cut fails, namely p^c. In practice, we can hope to do much better. The goal is simply to get $n^{-\alpha\delta}(1+2/\delta) \leq \epsilon \mathrm{FAIL}(p)$ in order to get an estimate accurate to within ϵ. In theory, since initially we do not know $\mathrm{FAIL}(p)$, we take p^c as lower a bound on $\mathrm{FAIL}(p)$. However, in practice this may be extremely pessimistic. Indeed, in our initial implementation, we would discover after the fact that we had chosen an α significantly larger than we needed, dramatically slowing down the algorithm. We therefore realized that it would help to have an (even coarse) lower bound estimate of $\mathrm{FAIL}(p)$ that could be used to give a better bound on α.

To compute this estimate, we note that for any α, the probability that an at most α-minimum cut fails is a lower bound on $\mathrm{FAIL}(p)$. Therefore, running the RCA with any α and applying the SACA to the resulting set of cuts gives us a usable lower bound. We select our initial α so that in fact,

$$n^{-\alpha\delta}(1+2/\delta) = n^2 p^c = n^{-\delta}.$$

This is a very optimistic choice, as in fact $n^2 p^c$ is nearly an upper bound on the possible value of $\mathrm{FAIL}(p)$. However, it worked well in practice. The resulting α is much smaller than the theoretical algorithm calls for.

To minimize our estimation running time, we also set a goal of $\epsilon = 1/2$. This still gives us a factor of two estimate for $\mathrm{FAIL}(p)$. We run the forked MC and RCA algorithm and use the returned estimate to choose a suitable value of α for the final run. In all of our experiments, the running time of the initial estimation step was negligible compared to that of the more accurate final step.

4 Test Procedure

In order to test our algorithm, we devised several input families parameterized by the number of vertices n. We then ran them with variations in the interesting parameters of the problem, p and ϵ. After discussing these input families, we describe some of their limitations which one might wish to address in future experimental work.

4.1 Input Families.

4.1.1 Cycles.

As has already been observed in [KS95], the cycle is a peculiarly "hard" graph when it comes to minimum cut problems. Lomonosov and Polesskii [LP71] showed that in fact the cycle is the "least reliable graph" in an easily formalized sense. The n-vertex cycle has $\binom{n}{2}$ minimum cuts—the most possible for any n-vertex graph. It also has the maximum number of $(k/2)$-minimum cuts for any integer $k \geq 2$. As observed in [CGK$^+$], it is the input on which the RCA runs slowest without Padberg-Rinaldi tests. On the cycle, the RCA has running time $\Theta(n^2)$—equal to its upper bound. We therefore expected it to take more time to compute the reliability of a cycle than any other graph. We will argue later that this assumption was not actually correct.

Another advantage of the cycle as an input for network reliability computations is that the reliability of a cycle can be determined exactly. The cycle becomes disconnected precisely when at least two edges fail. Thus if each edge fails with probability p, the probability the network becomes disconnected is

$$1 - (1-p)^n - np(1-p)^{n-1}.$$

Comparing the output of the our algorithm to the analytic bound lets us determine the accuracy our algorithm achieves in practice.

4.1.2 Delaunay Graphs (DEL).

We chose random Delaunay graphs as a natural model of network building. We begin by placing n points randomly in the unit square. We find a Delaunay triangulation of these points. We then construct the dual graph if this triangulation; this connects two of our original points if they share an edge in the triangulation. This construction yields planar graphs in which nearby vertices are connected to each other and all edges are thus relatively short. This seemed a plausible model of networks that are constructed with an aim towards creating "short" (near straight line) paths between points while also creating reasonable connectivity. The Delaunay graphs tend to have minimum cuts in the range 3-5.

We constructed our Delaunay graphs using NETPAD, a publicly available network design tool distributed by Bellcore.

4.1.3 Near Neighbor Graphs (NN).

Milena Mihail of Bellcore suggested our third model to us; she stated that it captured many of the characteristics of current telecommunication networks. These graphs are parameterized by two quantities, a radius r and a degree d. We begin by placing n random points in the unit square. Then, for each point x, we find its r nearest neighbors. From among those r neighbors, we choose d neighbors at random and connect them to x. Note that since each edge can be picked two ways, we might end up with parallel edges—these correspond to "doubly reliable" links.

In our experiments, we set $r = 8$ and $d = 4$, parameters suggested by Dr. Mihail.

An extension of this model adds *supernodes* that serve as "central exchanges." We chose 4 such supernodes, and attached each to half the nodes among the $n/3$ nodes nearest to it.

4.1.4 Weighted Near Neighbor Graphs (WNN).

The previous families all assign to every edge a fixed failure probability p. The algorithm of [Kar95] can also handle the case of varying failure probabilities. To study this case, consider the following model of link failures. We begin with the Near Neighbor graphs, but we give each link a length equal to the Euclidean distance of its endpoints. We then assign to each infinitesimal unit dl of length on the link an independent failure probability λdl, implying that a link of length l survives with probability $e^{-\lambda l}$. Varying λ is analogous to varying p in the uniform failure model.

Our motivation for this model was knowledge that many link failures are caused by utility workers accidentally digging through a communication link.

4.2 Discussion of Input Families.

It should be noted that our test families fail to explore possible variations in several input features. All of our graphs:

- are sparse,
- have low degree (in the range 2–8),
- have their minimum cuts at individual vertices, and
- have $\Theta(n)$ minimum cuts (at individual vertices).

Further work should clearly explore graphs which are dense and which have more interesting minimum cut structures. However, we believe our input families are reasonable models of the kinds of communication networks that will arise in practice. For example, if a network designer is attempting to minimize the cost of the network, then typically every edge will cross a near-minimum cut. If an edge crosses no near-minimum cut, the designer can reduce cost be leaving out the edge without significantly decreasing the network's reliability. Thus, a well designed network will tend to have many minimum cuts. Similarly, since link failures are quite unlikely events, we do not need large minimum cuts to ensure high reliability. By the same cost-based argument as above, this would seem to imply that networks will have small minimum cuts and will therefore be sparse.

We also believe that the good performance of our algorithms on our input families indicates that they will also perform well on other input families. For example, since the running time of the RCA is not particularly dependent on the number of edges in a graph (see [KS95]) we expect essentially the same performance on dense graphs as on sparse ones.

4.3 Parameters. For our experiments, we have 3 input parameters to consider: the error parameter ϵ, the number of vertices n, and the failure probability p. For any one graph family, several natural questions are raised:

- How does the actual error in approximation compare to the specified ϵ?
- For a given input, what is the running time of the approximation as a function of the edge failure rate p? As a function of FAIL(p)?
- For a given input, how does FAIL(p) depend on p?
- For a given family, as a function of n, what is the worst case running time over all p?

We will raise other questions later.

5 Running Time Measurements

We now begin describing the results of our tests. We use logarithmic axis scales for running times, so that polynomial running times correspond to straight line graphs. We also use logarithmic axis scales for p and FAIL(p), as these are typically polynomially related. We use linear scales for n and α. All times are reported in (CPU) seconds.

In many cases, our data is qualitatively the same for all of our test inputs. Therefore, we place one figure in place for ease of reference and give the others in the appendix. Additional data can be found in the full paper. We discard one parameter immediately: we found that running time as a function of ϵ was essentially as predicted. Thus, for the remainder of this paper, we work with a fixed $\epsilon = 0.1$.

As we discuss our results, we will note that they often diverge significantly from the theoretical predictions. We give some tentative explanations of these divergences, and believe that future work will be able to extend these rationalizations to sound theoretical results.

5.1 Running Time vs. p. We now consider a representative outcome, a plot of the running time as a function of p for the cycle family. Figure 1 shows this plot. The same overall behavior can be seen for Delaunay graphs and near neighbor graphs (see Figure 8).

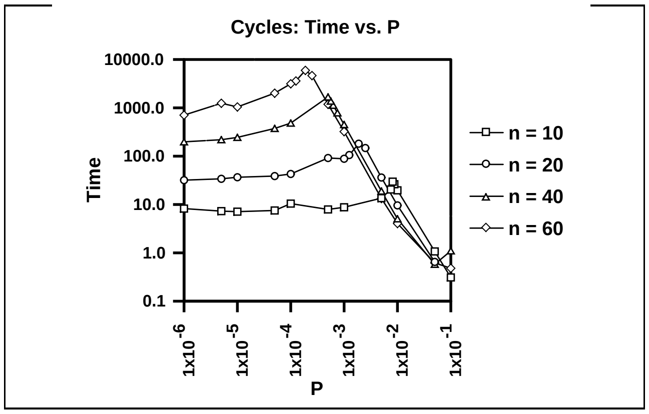

Figure 1: Running time vs. p for cycles

As can be seen in this plot, the running time at very small p is small. It rises superpolynomially to a peak as p increases, and then decays polynomially in $1/p$ as p continues to increase. All of our graph families exhibit this behavior macroscopically. The interpretation is as follows. At very small values of p, the MCA is useless. However, we can get a good approximation simply by examining the α-minimum cuts for α very near 1. Indeed, for p infinitesimal, all of the failure probability comes from the probability that a minimum cut fails. Enumerating the α-minimum cuts takes $O(n^{2\alpha})$ time using the RCA, and this is the dominant time in the computation. As p grows, we must consider larger and larger values α; this causes the running time increase. Eventually, the RCA is so slow that the MCA becomes the faster of the two. The MCA has running time inversely proportional to the failure probability, which on the cycle is proportional to $O((np)^2)$. Thus for large p the running time decays quadratically as p increases.

5.2 Running time versus α. We next consider the running time of the RCA as a function of the parameter α. Since the RCA dominates the running time of the algorithm, the running time of our approximation scheme is determined by the α needed for a good approximation. The theoretical time bound of the RCA is $\tilde{O}(n^{2\alpha})$. Figure 2 shows that the running time bound does indeed have the form $n^{k\alpha}$ for some k, since we get a running time dependence whose logarithm is near-linear in α. Surprisingly, however, the constant k is not the theoretically predicted 2 (except in the case of the cycle). Indeed, for the Delaunay and near-neighbor constructions, the running time appears to be roughly $O(n^{\alpha-1})$.

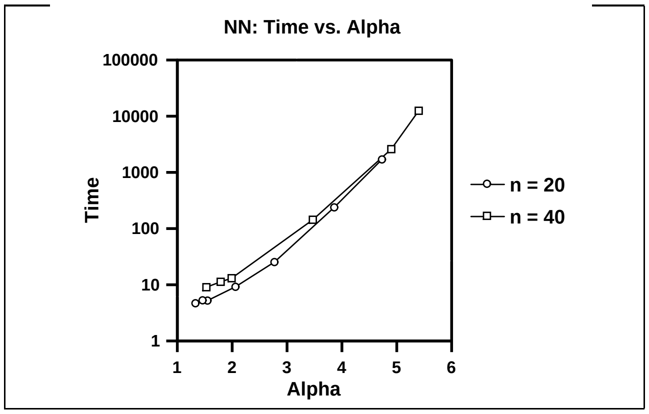

Figure 2: Running time vs. α for near neighbors

A possible explanation for this phenomenon is the following. The Delaunay and Near-Neighbor graphs have some degree of "expansion" in that the number of edges leaving a set of vertices tends to be proportional to the number of vertices in a set. Thus in these families, all minimum cuts tend to be isolated vertices, while larger sets of vertices tend to have at least $2c$ edges leaving them. In many ways, then, these graph act like graphs with minimum cut $2c$, implying that the cuts of value αc have value $\alpha/2$ times the "effective" minimum cut. Finding such cuts will only take $O(n^{2(\alpha/2)}) = O(n^{\alpha})$ time.

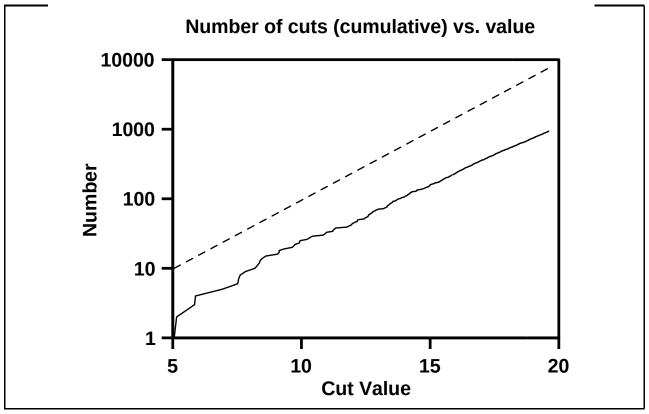

Figure 3: Running time vs. α for near neighbors, compared to 10^{α}

As evidence for this conjecture, consider Figure 3 counting the (cumulative) number of minimum cuts less than a given value for a particular 40 node weighted near-neighbor graph. The minimum cut is roughly 5. As can be seen by the fitted line, the number of minimum cuts increases as 10^{α} rather than as $40^{2\alpha}$.

5.3 Worst case running time. Since we cannot predict what failure probability might actually be needed by a user, we are of course interested in the worst-case p. This is the running time at the "peak" of the time-versus-p graph of Section 5.1. In Figure 4 we plot for cycles the worst case running time (over all p) as a function of the size n (graphs for other families are given in the appendix).

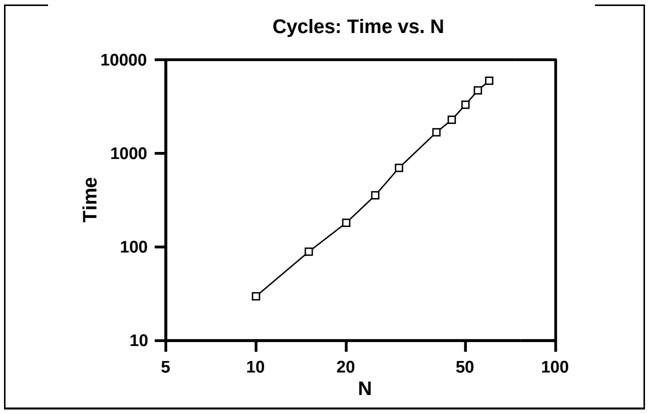

Figure 4: Running time vs. n for Cycles

The roughly $O(n^{2.8})$ running time we experience is significantly less than the $O(n^{4.5})$ running time claimed in the theoretical algorithm. This can be explained as follows. In a cycle, $\text{FAIL}(p) \approx \binom{n}{2}p^c \approx n^{-\delta}$. It follows that the RCA chooses an α such that

$$n^{-\alpha\delta}(1 + 2/\delta) \leq \epsilon n^{-\delta}.$$

We therefore deduce that the running time of the RCA,

$$\tilde{O}(n^{2\alpha}) = \tilde{O}(n^2/(\epsilon\delta)^{2/\delta})$$

This is $\tilde{O}(n^3)$ unless $\delta \approx 0$. Once $\delta \approx 0$, so that $\text{FAIL}(p) \approx 1/n^2$, we find that the MCA runs in $\tilde{O}(n^3)$ time. This observation is validated by the plot in Figure 5 of the value of $\text{FAIL}(p)$ that corresponds to the worst case running time; as claimed it is roughly $1/n^2$.

We initially chose to study the cycle because it is the graph for which the RCA takes longest to find α-minimum cuts, but this was on the assumption that α was fixed. Since the necessary α for a cycle is small, it

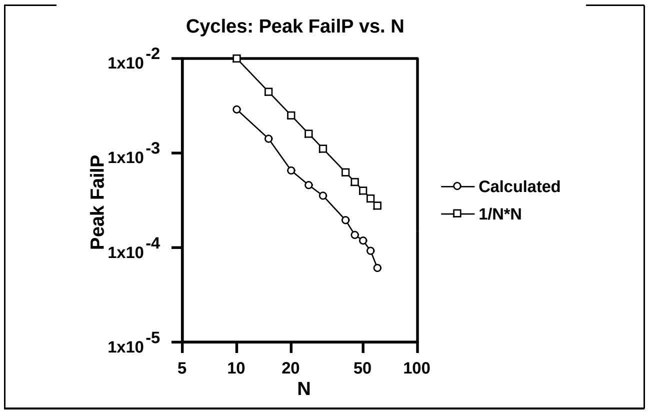

Figure 5: FAIL(p) at worst runtime vs. n for Cycles

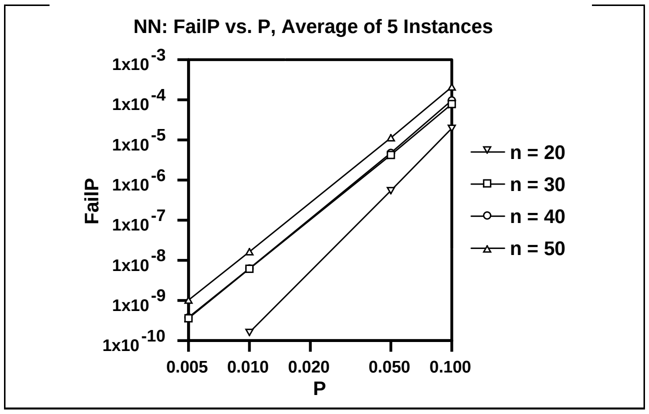

Figure 6: FAIL(p) vs. p for Near Neighbors

is not clear that the cycle graph whose reliability our algorithm finds it hardest to estimate.

While the running time on the cycle is easily explained, what is more surprising is that the algorithm had a similar bound on our other classes of graphs. We found the $\tilde{O}(n^3)$ worst-case running time to be typical, as can be seen in Figures 9 and 10 for our other graph families. What this is essentially saying is that RCA dominates until δ is very close to 0—that is, the probability that a minimum cut fails will be very close to n^{-2} at crossover to MCA. To verify this fact, consider Figure 11 which shows α increasing to 5 or 6—much larger than the theoretical bound suggests.

One possible explanation for this phenomenon is a generalization of our argument for the cycle. In the full version of [CGK+], using ideas from [Kar95], a weak connection is drawn between the running time of the RCA and the reliability of the graph it is being run on. Roughly speaking, the more reliable a graph, the faster the RCA solves it (for a given α). We have just argued that when graphs are unreliable, our approximation algorithm uses a smaller α in the RCA. Our theoretical analysis assumes simultaneously that the RCA takes its worst-case time (implying that the input is an unreliable graph) but that the graph is so reliable that we must use our worst-case α. Perhaps these two assumptions cannot be true simultaneously.

6 Reliability Results

A user of the program is probably less concerned with the implementation details than with the actual reliabilities. In Figures 6 and 12, we plot the failure probability FAIL(p) as a function of p for each graph family.

It can be seen that the reliability graph was basically a straight line. In each case, examining the input showed that the slope was determined by the minimum cut. That is, it appeared that with other quantities held constant,

$$\mathrm{FAIL}(p) = \Theta(p^c)$$

This put us in mind of a well known heuristic estimate of FAIL(p) (see [Col87]). Let us write the graph cuts as C_i, $i = 1, \ldots, 2^{n-1}$. Let F_i denote the event that cut C_i fails. We can use inclusion exclusion to write FAIL(p) as

$$\begin{aligned}\Pr[\cup F_i] &= \sum_{i_1} \Pr[F_{i_1}] - \sum_{i_1<i_2} \Pr[F_{i_1} \cap F_{i_2}] + \\ &\quad \sum_{i_1<i_2<i_3} \Pr[F_{i_1} \cap F_{i_2} \cap F_{i_3}] + \cdots .\end{aligned}$$

A heuristic argument says that the first term in this sum is a reasonable estimate (it is an upper bound). That is, we might as well sum the probabilities that each cut fails. If we let n_k denote the number of cuts with k edges, then this heuristic says that we should approximate FAIL(p) by $\sum n_k p^k$. The analysis of Theorem 2.2 [Kar95] shows that this sum in turn is accurately approximated by $\sum_{k \le \alpha c} n_k p^k$ when p is sufficiently small.

The advantage of the heuristic is that it does not require running the SACA. This does not simply save on implementation overhead. With the heuristic, all we need is a count of the number of cuts of each size. Such a count can be built without storing any cuts—all we need to store is a (hash) key for each cut we encounter so that we can ignore it if we encounter it a second time. This is a significant benefit: on our larger problems, we began running out of space as α became large. Therefore, an optimization like this one to conserve space would increase the size of problems we could handle.

Figure 7 shows that these estimates are indeed quite good. The graph plots the relative error between our original RCA/SACA estimate and that determined by summing the small-cut failure probabilities.

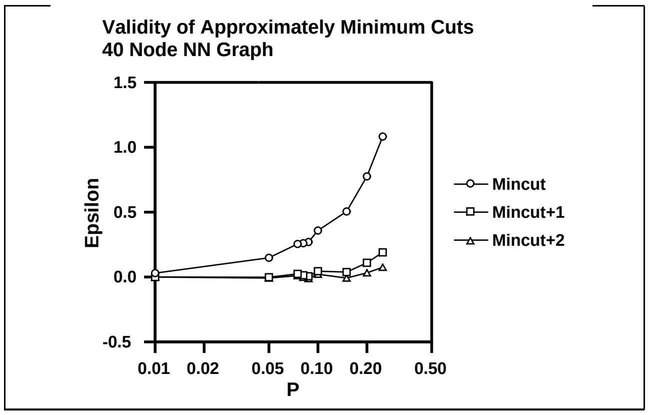

Figure 7: Error in heuristic approximation vs. p

7 Analysis of a Heuristic

In this section, we give an analytic justification for the heuristic presented in the previous section. Our formal theorem is the following:

THEOREM 7.1. *When FAIL*$(p) < n^{-(4)}$*, the sum of the* α*-minimum cuts' failure probabilities is a* $(1 + o(1))$*-approximation to FAIL*(p)*.*

The proof of this theorem runs as follows. Until now, we have relied on the fact that the most likely way for a graph to fail is for some of its near-minimum cuts to fail. We strengthen this argument to observe that most likely, *exactly one* of these near minimum cuts fails. We use this argument to show that that the error which arises from truncating the inclusion-exclusion expansion at the first term is negligible.

To prove the theorem, we argue as follows. As discussed Section 2, [Kar95] proves that it is sufficient to approximate, for the given ϵ, the probability that some α-minimum cut fails, where

$$\alpha \quad = \quad 1 + 2/\delta - (\ln \epsilon)/\delta \ln n$$

Let us write these α-minimum cuts as $C_i, i = 1, \dots, n^{2\alpha}$. Let F_i denote the event that cut C_i fails. We can use inclusion exclusion to write the failure probability as

$$\begin{aligned}\Pr[\cup F_i] \quad &= \quad \sum_{i_1} \Pr[F_{i_1}] - \sum_{i_1 < i_2} \Pr[F_{i_1} \cap F_{i_2}] + \\ &\qquad \sum_{i_1 < i_2 < i_3} \Pr[F_{i_1} \cap F_{i_2} \cap F_{i_3}] + \cdots .\end{aligned}$$

Later terms in this summation measure events involving many cut failures. We show that when many cuts fail, the graph partitions into many pieces, meaning a multiway cut fails. We then argue that this is so unlikely that later terms in the sum can be ignored. This immediately yields Theorem 7.1.

7.1 Inclusion-Exclusion Analysis.

As discussed above, our analyses use a truncation of the inclusion-exclusion expression for $\Pr[\cup F_i]$. Suppose we truncate the inclusion-exclusion, leaving out the k^{th} and later terms. If k is odd the truncated sum yields a lower bound; if k is even it yields an upper bound. We show that this bound is sufficiently tight.

LEMMA 7.1. *Let* S_u *be the event that* u *or more of the events* F_i *occur. If the inclusion-exclusion expansion is truncated at the* k^{th} *term, the error introduced is*

$$\sum_u \binom{u-2}{k-2} S_u .$$

Proof. Let T_u be the event that *exactly* u of the events F_i occur. Consider the first summation $\sum F_{i_1}$ in the inclusion-exclusion expansion. The event that precisely the events $F_{j_1}, \dots, F_{j_u}$ occur contributes to the u terms $\Pr[F_{j_1}], \dots, \Pr[F_{j_u}]$ in the sum. It follows that each sample point contributing to T_u is counted $u = \binom{u}{1}$ times in this summation. Thus,

$$\sum \Pr[F_{i_1}] = \sum_u \binom{u}{1} \Pr[T_u].$$

By the same reasoning,

$$\sum \Pr[F_{i_1} \cap F_{i_2}] = \sum_u \binom{u}{2} \Pr[T_u],$$

and so on. It follows that the error introduced by truncation at term k is

$$\begin{aligned}\sum_{j \geq k} (-1)^{k-j} &\sum_{i_1 < \cdots < i_j} \Pr[F_{i_1} \cap \cdots \cap F_{i_j}] \\ &= \sum_{j \geq k} (-1)^{k-j} \sum_u \binom{u}{j} \Pr[T_u] \\ &= \sum_u \sum_{j \geq k} (-1)^{k-j} \binom{u}{j} \Pr[T_u] \\ &= \sum_u \binom{u-1}{k-1} \Pr[T_u]\end{aligned}$$

Now let us define S_u to be the event that u *or more* of the F_i occur, meaning that $\Pr[T_u] = \Pr[S_u] - \Pr[S_{u-1}]$. Then we can rewrite our bound above as

$$\begin{aligned}
&\sum_u \binom{u-1}{k-1}(\Pr[S_u] - \Pr[S_{u+1}]) \\
&= \sum_u \binom{u-1}{k-1} \Pr[S_u] - \sum_u \binom{u-1}{k-1} \Pr[S_{u+1}] \\
&= \sum_u \binom{u-1}{k-1} \Pr[S_u] - \sum_u \binom{u-2}{k-1} \Pr[S_u] \\
&= \sum_u \left(\binom{u-1}{k-1} - \binom{u-2}{k-1}\right) \Pr[S_u] \\
&= \sum_u \binom{u-2}{k-2} \Pr[S_u]
\end{aligned}$$

7.2 A Simple Approximation. Using the above error bound, we can prove Theorem 7.1. Let F_i denote the event that the i^{th} near-minimum cut fails. Our objective is to estimate $\Pr[\cup F_i]$. Summing the individuals cuts' failure probabilities corresponds to truncating our inclusion-exclusion sum at the second term, giving (by Lemma 7.1) an error of $\sum_{k \geq 2} S_u$. We now bound this error by bounding the quantities S_u using *r-way cuts.* An r-way cut is simply a partition of the graph vertices into r components; the cut edges are those with endpoints in different components.

Lemma 7.2. *If u distinct (2-way) cuts fail then a $\lceil \log(u+1) + 1 \rceil$-way cut fails.*

Proof. Consider a configuration in which u distinct cuts have failed simultaneously. Suppose this induces k connected components. Let us contract each connected component in the configuration to a single vertex. Each failed cut in the original graph corresponds to a distinct failed cut in the contracted graph. Since the contracted graph has k vertices, we know that there are at most $2^{k-1} - 1$ ways to partition its vertices into two nonempty groups, and thus at most this many cuts. In other words, $u \leq 2^{k-1} - 1$. Now solve for u and observe it must be integral.

We now use a fact which is a slight generalization of Theorem 2.2:

Lemma 7.3. *If $p^c = n^{-(2+\delta)}$, then the probability that an r-way cut fails is at most $n^{-\delta r/2}$.*

Proof. Based on work in [Kar95], with details in the full version of that paper.

Thus, for example, S_2 and S_3 are at most the probability that a 3-way cut fails, which by Lemma 7.3 is at most $n^{-3\delta/2}$. More generally, it follows from the above lemma and Lemma 7.3 that all 2^k values $S_{2^k}, \ldots, S_{2^{k+1}-1} \leq n^{-(k+2)\delta/2}$. It follows that the error in our approximation is

$$\begin{aligned}
\sum_{u \geq 2} S_u &\leq \sum_{k \geq 1} 2^k n^{-(k+2)\delta/2} \\
&= n^{-\delta} \sum_{k \geq 1} (2n^{-\delta/2})^k \\
&= 2n^{-3\delta/2}(1 + o(1))
\end{aligned}$$

This is $o(\text{FAIL}(p))$ whenever $n^{-3\delta/2} = o(n^{-(2+\delta)})$, i.e. $\delta > 4$.

8 Conclusion

Our implementation of the approximation algorithm of [Kar95] had a pleasant outcome: an algorithm that works better in practice than claimed theoretically. Our algorithm is practical and has motivated analysis of a well known heuristic for the reliability problem. The implementation has suggested several open problems.

- Is the analysis of the theoretical algorithm tight? We have been unable to identify any input instances that even approach the worst-case running time bounds given.
- In particular, can we prove that networks satisfying certain natural reliability criteria yield a better running time bound for the recursive contraction algorithm? For example, some improvement would follow immediately from the fact that all such graphs have many minimum cuts.
- Can we totally do away with the use of SACA by working only with the inclusion-exclusion representation of cuts?

Naturally, it would also be appropriate to examine other graph families (e.g. dense ones) to learn more about the way the algorithm behaves in practice.

References

[CGK$^+$] Chandra C. Chekuri, Andrew V. Goldberg, David R. Karger, Matthew S. Levine, and Cliff Stein. Experimental study of minimum cut algorithms. Submitted to SODA 1997.

[Col87] Charles J. Colbourn. *The Combinatorics of Network Reliability*, volume 4 of *The International Series of Monographs on Computer Science.* Oxford University Press, 1987.

[Kar94] David R. Karger. Random sampling in cut, flow, and network design problems. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 648–657. ACM, ACM Press, May 1994. Submitted for publication..

[Kar95] David R. Karger. A randomized fully polynomial approximation scheme for the all terminal network reliability problem. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, pages 11–17. ACM, ACM Press, May 1995.

[KL85] Richard M. Karp and Michael G. Luby. Monte carlo algorithms for planar multiterminal network reliability problems. *Journal of Complexity*, 1:45–64, 1985.

[KLM89] Richard M. Karp, Michael Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, September 1989.

[KM96] David R. Karger and Rajeev Motwani. Derandomization through approximation: An $\mathcal{NC}$ algorithm for minimum cuts. *SIAM Journal on Computing*, 1996. To appear.A preliminary version appeared in STOC 1993, p. 497.

[KS93] David R. Karger and Clifford Stein. An $\tilde{O}(n^2)$ algorithm for minimum cuts. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 757–765. ACM, ACM Press, May 1993.

[KS95] David R. Karger and Clifford Stein. A new approach to the minimum cut problem. *Journal of the ACM*, 1995. To appear. Preliminary portions appeared in SODA 1992 and STOC 1993.

[LP71] Micael V. Lomonosov and V. P. Polesskii. Lower bound of network reliability. *Problems of Information Transmission*, 7:118–123, 1971.

[PB83] J. Scott Provan and Michael O. Ball. The complexity of counting cuts and of computing the probability that a network remains connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[Val79] Leslie Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

A Additional Figures

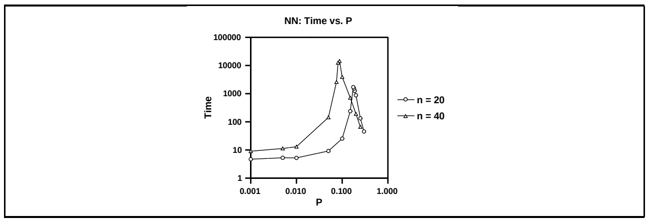

Figure 8: Running time vs. p for Near Neighbor Graphs

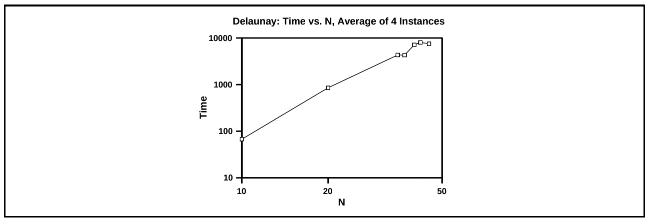

Figure 9: Running time vs. n for Delaunay Graphs

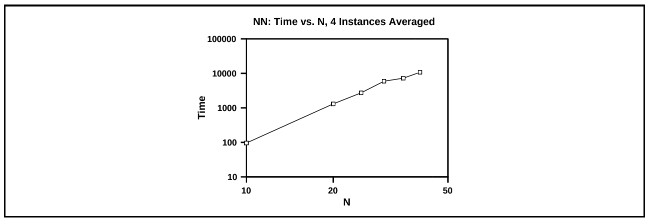

Figure 10: Running time vs. n for Near Neighbor Graphs

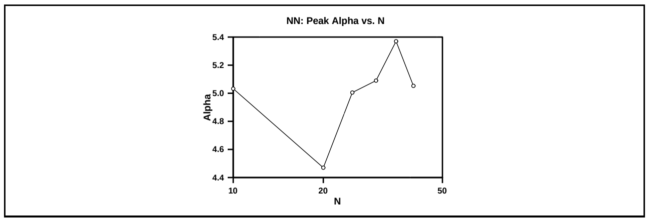

Figure 11: α at worst case runtime for Near Neighbors

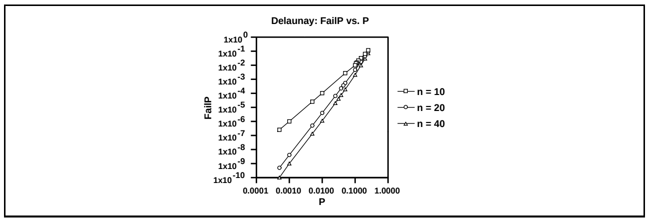

Figure 12: FAIL(p) vs. p for Delaunay graphs