Estimating the Reliability of Networks

Bogdan Sitaru
Supervisor: Andreas Galanis

3rd Year Project Report
Honour School of Computer Science (Part B)

Trinity Term 2022

Abstract

Networks are fundamental structures studied across sciences. The components of these systems are prone to failures, potentially altering substantially the functionality and structure of the networks. One important property of a network is the so-called all-terminal reliability, which captures the probability that the network remains connected when the connections inside it fail independently. The problem of exactly computing the all-terminal reliability is hard, being in the #P-complexity class, so approximation algorithms have been designed to instead estimate the reliability of a network, or, the closely related quantity of unreliability.

Here we study the problem through the lens of algorithm design. Recently, there have been major breakthroughs from a theoretical perspective on the problem: Karger introduced an almost quadratic-time algorithm for unreliability, and Guo and Jerrum an efficient algorithm for the reliability. Despite the strong theoretical guarantees of these algorithms, the empirical performance of these algorithms has not been thoroughly studied, and their performance in practical applications is not very well understood.

The goal of this project is to fill in this gap by studying these state-of-the-art algorithms, and give a comparative analysis of them relative to other heuristic approaches used in application domains. To do this, we first study the theoretical foundations of the algorithms and give an overview of the relevant ideas. Then, our first contribution is to give a fast implementation of both algorithms, and compare it against other methods for computing the reliability of networks; for Karger's algorithm, we are in fact able to do various optimisations which speed up its performance substantially. Our second contribution is to explore the practical limitations of the algorithms against large instances; here, Karger's algorithm appears to scale better overall, while the algorithm of Guo and Jerrum has some bottlenecks that slow it down on larger networks. Our third and final contribution is to analyse the empirical approximation accuracy of the two algorithms, where our results indicate that the theoretical bounds are not tight and therefore suggest there is room for even faster algorithms.

Contents

1. Introduction

Networks represent groups of interconnected entities and therefore are fundamental structures in our modern society, being used in diverse fields of study to model systems of interest, such as utility networks (for delivery of gas, electricity, or water), transportation networks (rail lines or airline routes), and communication networks (the Internet, telephone networks, or special communications). For instance, the road structure of a country can be represented by a network where each city is represented as a node in the network and roads connecting the cities are edges in the graph. Some areas such as biology or social sciences use networks in a different way, to model the development of dynamic processes, one relevant example being the spread of a disease in a pandemic, which can be modelled as a network where nodes represent people and edges are interactions or contacts between them. A third example is the power-transmission grid of a city, where power plants, generator sites and consumers (e.g. houses) are represented as nodes in the network, and transmission lines are edges in the network.

In the real world, the elements of a network (either nodes or edges) are subject to failure. For example, roads in a road system might become unavailable because of accidents or natural disasters, or generator sites in a power-transmission grid can break. It is important to have an understanding of the impact of these failures on the network, which determines the ability of the network to preserve its structure and properties. The existence of failures led to the problem of quantifying the reliability of a network, that is the probability of the network being functional, which in most cases means that the network (or some part of it) is connected. We begin by defining the problem more precisely.

1.1 Definition of the Network Reliability Problem

We will model a network as an undirected graph $G = (V, E)$, and denote by $n = |V|$ the number of nodes and $m = |E|$ the number of edges. Here, we will always assume that the graph is connected (otherwise, we can look at each component separately).

In the reliability problem, each edge $e \in E$ of the network can fail with probability $p_e \in [0, 1]$. Here, we consider the most-widely studied setting of the problem where all edge-failures are independent. Failure of an edge results in the deletion of the corresponding edge from the graph. The all-terminal reliability of the network $rel_G(p)$ is the probability of the resulting graph remaining connected. The complementary probability, i.e., the

quantity $u_G(p) = 1 - rel_G(p)$, is the unreliability of the network, the probability of the graph getting disconnected. For example, for the road structure of a country, we are interested in the probability that all cities remain connected (there is at least a way to travel from any city to any other one), or for the pandemic case, we might be interested in the probability that the whole population is infected if at least one of them is infected initially.

There are some other variants of the network reliability problem, the general one being the K-terminal instance, where we check if a set of (important) nodes K is still connected, while another popular instance is the two-terminal one, where we are interested if there is at least a path between two given nodes. Moreover, there are many less popular variations for the problem, some of them considering failures of the vertices instead of the edges (e.g. in communication networks, where a pole might break), failures of both vertices and edges, or other variations capturing reliability on directed graphs (e.g. chain supply networks).

We will focus on the all-terminal reliability problem for this project, since it is the most well-studied instance of the problem, and has the most applications in practice. More precisely, we study the instance of the all-terminal reliability problem when all edges have the same probability of failing $p_e = p$, which does not reduce the generality, as one can model distinct p_e values by having a multigraph instead of a simple graph, using multiple edges between two nodes (with lower p_e), instead of a single edge. While this might seem as a simplification, in most cases the algorithms generalise to the non-uniform setting, and in applications it was revealed that most network structures present small variation in reliability between the two instances.

1.2 Algorithms for the Network Reliability Problem

There is a huge literature of heuristic and practical algorithms that solve the problem exactly (computing the exact reliability probability). To briefly overview the main approaches, these involve cut-set or path-set enumeration [1], inclusion-exclusion principle [19], or reducing the network followed by complementary subgraphs solving processes [25]. Some of them achieve polynomial running time on particular graphs, using heuristics and reductions, but are still exponential in the worst case, being inefficient for large general networks.

This limitation of exact algorithms is inherent, since, from a complexity point of view, the all-terminal network reliability problem is known to be #P-complete, as proven by Valiant [28, 4]. This class consists of computationally hard problems, roughly containing the counting version of problems in the NP class.[1] Thus, the #P class is at least as intractable as the NP class, and in fact, it is believed to be much harder than it. It is widely believed that no exact algorithm solving the network reliability problem that runs

[1]For example, some well-known #P-complete problems are: counting solutions for boolean formulas (#SAT), number of k-colourings of a graph, and number of perfect matchings in bipartite graphs.

in polynomial time is possible, unless a major complexity breakthrough.

Because of these inherent limitations of the exact algorithms for the network reliability problem, the focus turned to approximation algorithms, leading to the problems of approximating network reliability and unreliability. Somewhat surprisingly, these two approximation tasks are different, since an approximate solution to one of them does not necessarily entail a good approximation for the other. Here, following the literature, we are interested in relative error approximation, that is, given an $\epsilon > 0$, we require some value $\hat{u}$ that is within a factor of $1 \pm \epsilon$ of the unreliability u ($1 - \epsilon < \frac{\hat{u}}{u} < 1 + \epsilon$), or a value $\widehat{rel}$ for the reliability probability rel. In other words, we want to efficiently produce an ϵ-approximation[2] for the solution to our problem To illustrate this with an example, if a graph has $rel_G(p) = 0.9$ (and $u_G(p) = 0.1$), taking $\widehat{rel} = 1$ is a good (0.2)-approximation for the reliability, but the converse, $\hat{u} = 1 - \widehat{rel} = 0$ is not a (0.2)-approximation for $u_G(p)$.

1.2.1 State-of-the-art for Approximation Algorithms

Reliable networks are more common in practice, as highly unreliable networks are easy to observe, thus, approximating unreliability is a more well studied problem. Currently, the best algorithms for approximating network reliability and unreliability are:

- for estimating reliability, the reliability algorithm [10] proposed by Guo and Jerrum, making use of connected graph sampling, with a run time of $O((1-p)^{-3}m^2n^3)$;
- for estimating network unreliability, Karger's unreliability algorithm [12], which uses cut-set counting, having a run time of $\tilde{O}(n^2)$.

An overview of these two algorithms is presented in Chapter 3, specifying the important procedures and giving intuition for their analysis, because they represent a significant part of our study.

It is worth mentioning that other approximation heuristic algorithms techniques include Convolutional Neural Networks [6] and Deep Neural Networks [5], but their performance is not comparable to traditional algorithms, as these neural networks do not come with guaranteed bounds, and their accuracy is done on an instance by instance basis.

1.3 Our Contribution

Here is a breakdown of this project, highlighting also our most important contributions. Given the huge number of heuristic algorithms and approximation algorithms, there are a lot of options for real-world applications to choose from. The state-of-the-art approximation algorithms (Karger's unreliability algorithm [12] and Guo and Jerrum's reliability algorithm [10]) have promising theoretical running time bounds, and also offer guarantees about the produced estimations, which is not necessarily the case for most heuristics. Prior

[2]See Definition 2.2.1.

to our work, there are no known implementations for either of these algorithms, so their practicality is underexplored. In this project, we focus on the theoretical and empirical analysis of the two algorithms, and place emphasis on the latter where the understanding is lacking.

We start by giving an overview of the theoretical analysis of some approximation algorithms (in Chapter 3), focusing on the two mentioned above. We present strong intuition about the way these algorithms produce the estimations, concentrating on the important procedures.

Then, our first contribution (in Chapter 4) is to offer an efficient implementation for Karger's $\tilde{O}(n^2)$ unreliability algorithm, which we further optimise using practical considerations to achieve better run time performance. To test our implementation, we compare our results in practical experiments against other approaches that have been used in the field (parallel algorithms, heuristic algorithms). Following the trials, we assess that Karger's unreliability algorithm, and in particular our implementation, is very efficient, outperforming most of the other compared methods.

In Chapter 5, which describes our second contribution, we study the practical limitations of the two approximation algorithms, providing the results for the largest graphs these procedures can be run on, considering appropriate run time limit. Consequently, using Karger's unreliability algorithm, we present the estimations for the reliability of large graphs (up to 2500 vertices), results that were not yielded before, as other methods are not able to compute reliability for such big networks. Here it is important to emphasise that for almost all previous work the sizes of the instances studied were significantly smaller (of order 20 nodes), so this further demonstrates the efficiency of Karger's algorithm. The reliability algorithm, however, was not fast enough to obtain good guarantees for such large instances. For this method, our experiments suggest that getting samples that represent the typical structure of the system is fast (even for large networks), but estimating from these the reliability quantity has an extra step that comes with a rather big overhead.

For our third and final contribution (in Chapter 6), we analyse the empirical approximation bounds for the estimate produced by both approximation algorithms (Karger's unreliability algorithm and Guo and Jerrum's reliability algorithm). Since these methods have theoretical guarantees for the relative error and the failure rate in terms of the number of trials, we empirically test these values. Our hypothesis here is that these bounds are not tight, and we conclude that there is room to improve on the theoretical analysis of these algorithms.

2. Preliminaries

2.1 Graphs

We will use undirected graphs to represent networks in our study. For notation purposes, recall that a *graph* is a pair $G = (V, E)$, where V is a finite set of vertices (nodes) and $E \subseteq \{(x, y) \mid x, y \in V, x \neq y\}$ is a set of unordered pairs called edges. A *multigraph* is a graph that allows self-loops (edges that have the same node at both ends) and multiple edges between two vertices.

Since these notions are common in further analysis, whenever the context contains a single graph (there is no ambiguity to which graph we refer), we will assume that $G = (V, E)$ is the graph, n is the number of nodes in the graph, and m is the number of edges in the graph.

2.2 Approximation Schemes

Recall that we are going to study approximation algorithms, so here we formalise what an approximation represents.

Definition 2.2.1. For $\epsilon > 0$, An *ϵ-approximation* for a quantity q is an estimate value $\hat{q}$, such that $(1 - \epsilon)q \leq \hat{q} \leq (1 + \epsilon)q$.

Our algorithms are going to use randomness, so their output to reliability will frequently be a random variable. We allow approximations to be valid with high probability.

Definition 2.2.2. For $\epsilon > 0$ and $\delta \in (0, 1)$, an (ϵ, δ)-approximation for a quantity q is a random variable that is an ϵ-approximation for q with probability at least $1 - \delta$.

Now that these are in place, we introduce more precisely the most used type of approximation schemes, that we are focusing on in the analysis of the algorithms, in Section 3.

Definition 2.2.3. A *Fully Polynomial-time Randomised Approximation Scheme (FPRAS)* is an algorithm that takes an instance of a problem and a parameter $\epsilon > 0$, and produces an ϵ-approximation with high probability (usually greater than $\frac{3}{4}$), having run time polynomial in both input size and ϵ^{-1}.

2.3 Estimators and their Relative Variance

To study the success probability of randomised algorithms, we need to study their output, viewed as an estimator of the quantities of interest (in our case the reliability and unreliability of a network, depending on the failure probability).

Definition 2.3.1. An *unbiased estimator* for a random variable X is a random variable Y, such that $\mathbb{E}[Y] = \mathbb{E}[X]$.

In order to assess the spread of an estimator, we need to study their relative variance, one of their fundamental properties.

Definition 2.3.2. The *relative variance* of a random variable X that is distributed with mean μ and variance σ^2 is equal to $r = \frac{\sigma^2}{\mu^2}$.

In various scenarios, we will have an unbiased estimator that approximates the value of another estimator, thus, it is important to see how to compose unbiased estimators.

Lemma 2.3.3. ([14]) *Let X be some random variable, Y be an unbiased estimator for X with relative variance $r_{Y,X}$, and Z be an unbiased estimator for Y with relative variance $r_{Z,Y}$, then Z is an unbiased estimator for X having relative variance $r_{Z,X} = (r_{Y,X} + 1)(r_{Z,Y} + 1) - 1$.*

Unbiased estimators are fundamental pieces in the field of approximation and randomised algorithms, being used to create ϵ-approximations by repeated samplings.

Lemma 2.3.4. *Let X be a random variable distributed with mean μ and variance σ^2, having relative variance r, then the average of $pr\epsilon^{-2}$ independent samples of X produces an ϵ-approximation of μ with probability $(1 - \frac{1}{p})$.*

Proof. Let $X_1, \ldots, X_t$ be independent and identically distributed random variables representing the samples, $t = pr\epsilon^{-2}$ be the number of samples. The average $\overline{X} = \sum_{i=1}^{t} X_i$ is a random variable with mean μ and variance $\frac{\sigma^2}{t} = \frac{\sigma^2}{pr\epsilon^{-2}} = \frac{\mu^2\epsilon^2}{p}$. Using Chebyshev's Inequality on $\overline{X}$, it results:

$$\Pr\left[\left|\overline{X} - \mu\right| > \epsilon\mu\right] \leq \frac{\frac{\mu^2\epsilon^2}{p}}{\epsilon^2\mu^2} = \frac{1}{p}$$

Thus, $\overline{X}$ is an ϵ-approximation for μ with probability at least $1 - \frac{1}{p}$. □

A well-known way to reduce the failure probability of an approximation algorithm to an arbitrary low failure probability is by using the Median Trick.

Lemma 2.3.5. (Median Trick) *Given an algorithm A that fails with probability $\frac{1}{p} < 0.5$, running A for $t = \Omega(\log(\delta^{-1}))$ times and taking the median results in an algorithm that fails with probability δ (is correct with probability $1 - \delta$).*

The two previous lemmas are powerful tools that, combined, yield a method for producing an (ϵ, δ)-approximation.

Lemma 2.3.6. *Let X be a random variable distributed with mean μ and relative variance r, then we can obtain an (ϵ, δ)-approximation of μ using $O(r\epsilon^{-2}\log(\delta^{-1}))$ samples.*

Proof. Using Lemma 2.3.4 with any $p > 2$ (e.g. $p = 4$, so failure probability is $\frac{1}{4}$) as the algorithm for Lemma 2.3.5, it results that we need $O(pr\epsilon^{-2}\log(\delta^{-1}))$ samples. By ignoring p, which is a constant, it follows that $O(r\epsilon^{-2}\log(\delta^{-1}))$ samples are required for an (ϵ, δ)-approximation of μ. □

2.4 Asymptotic Notations

In this part, we define a less common asymptotic notation that we use in the analysis of the algorithms.

Definition 2.4.1. The $\tilde{O}$ *(soft-O)* notation is equivalent to Big-O notation ignoring logarithmic factors. That is, we say that $f = \tilde{O}(g)$ if there is some positive integer k such that $f = O(g\log^k n)$

In most approximation schemes complexity functions the ϵ and δ factors are present, mostly in form of ϵ^{-2} and $\log(\delta^{-1})$, but we will omit these for most of the analysis, as it makes the notation easier to read.

3. Analysis of Approximation Algorithms

In this chapter we describe and analyse the state-of-the-art algorithms for approximating network reliability and unreliability, giving strong intuition about the way they work and their running time. We focus on the analysis of Karger's unreliability algorithm and Guo and Jerrum's reliability algorithm since they are the most efficient ones for the respective problems, also being the main subject of this project.

3.1 Unreliability Approximation Algorithms

3.1.1 Naive Monte Carlo

One of the first attempts for an algorithm approximating network unreliability (u_G) was made by Karp and Luby [18], who investigated the Naive Monte Carlo (NMC) algorithm. The idea is to simulate edge failures by randomly removing each edge (with the given probability) from the graph, and then check if the network is still connected. A good estimation results from running the algorithm multiple times, and take the fraction of runs in which the graph becomes disconnected as a result.

Lemma 3.1.1. *The Naive Monte Carlo algorithm approximates $u_G(p)$ within a factor of $1 \pm \epsilon$ with high probability using $O(u_G(p)^{-1})$ trials.*

Proof. Let X be the random variable representing the result of a trial of NMC, which is Bernoulli distributed, with the mean $\mu = u_G(p)$ and variance $\sigma^2 = u_G(p)(1 - u_G(p))$. The relative variance of X is $r = \frac{1-u_G(p)}{u_G(p)} = \frac{1}{u_G(p)} - 1 \leq u_G(p)^{-1}$. Using Lemma 2.3.6 it results that NMC returns a ϵ-approximation using $O(u_G(p)^{-1})$ trials with high probability. □

Each sample takes $O(m)$ time, leading to a total running time of $O(m \cdot u_G(p)^{-1})$. Therefore, NMC is efficient for unreliable networks (where $u_G(p)$ is large), but not for highly reliable graphs.

3.1.2 Karger's First FPRAS

Definition 3.1.2. A *cut* is a partition of the nodes in a graph into two disjoint sets. The *cut-set* determined by a cut is the set of edges that do not have the endpoints in the same set of the partition. The size of a cut (usually denoted by c) is equal to the number of edges in the cut-set.

Karger introduced the first FPRAS [13] for the network unreliability problem. The main observation is that, when a network is reliable (u_G is small), the probability of a large cut-set failing is insignificant, so $u_G(p)$ can be approximated as the probability of any small cut-set failing. This can be computed by enumerating small cut-sets, and then counting the number of subgraphs where at least one of these cut-sets fails, which can be done using various techniques (for example, using counting for boolean DNF formulas).

Since this approach only works when the unreliability is small, we need a method to roughly assess if a network is reliable or not. This can be accomplished using the following lemma, which gives a lower bound for the unreliability of a network.

Lemma 3.1.3. *For any graph G having minimum cut size c, and failure probability p, it holds that $p^c \leq u_G(p)$.*

Proof. The graph becomes disconnected when the minimum cut-set fails, which has probability p^c. Since the unreliability represents the probability of at least one cut-set failing, it results that $p^c \leq u_G(p)$. □

For the current purposes, using Lemma 3.1.3, we designate that a graph is reliable if $u_G(p) \leq n^{-4}$. It follows that Karger's first FPRAS is a hybrid that uses NMC for unreliable graphs (when $n^{-4} \leq p^c \leq u_G(p)$), and the cut-set enumeration procedure otherwise (for reliable graph), leading to a running time of $\tilde{O}(n^5)$.

3.1.3 Karger's $\tilde{O}(n^2)$ algorithm

One other idea that is used in minimum cut-set size computation is the Recursive Contraction Algorithm (RCA) [17], likewise introduced by Karger. A contraction in a graph is the action of uniting two adjacent nodes, removing the edges between them, but keeping all the resulted parallel edges. This algorithm contracts an edge sampled uniformly random until only two nodes remain in the graph, the resulting number of edges between them being a cut-set in the initial graph.

Recently, Karger presented a new method [11] to estimate unreliability, adapting the RCA procedure for the network unreliability problem. The algorithm does not involve cut-set enumeration, but uses a variation of RCA to reduce the size of the graph, creating an unbiased estimator for the problem. This method was further improved [14, 12], not by changing the structure of the procedure, but by better analysis of the running time and bounding of the relative variance of the unbiased estimator. We study this algorithm in more detail because it is the main focus of the project.

Let $k \leq \frac{1}{2}$ be the limit between a reliable and an unreliable graph (considered unreliable if $k \leq p^c \leq u_G(p)$), which will be determined later, depending on the time complexity needs. Similar to the previous one, this algorithm is also a hybrid: if n is smaller than a constant fixed value, compute $u_G(p)$ by brute-force in constant time, otherwise compute the minimum cut value c. Use NMC for unreliable graphs ($p^c \geq k$), or solve the problem

recursively using RCA for reliable graphs. For the latter case, choose $q = k^{\frac{1}{c}}$ (i.e. $q^c = k$), and contract each edge of G with probability $1 - q$, producing a (probably smaller) graph H, then compute $u_H(\frac{p}{q})$ and return as a result. We can think about the contraction phase similarly to a NMC run, but instead of sampling a network, we sample a set of cuts that might fail, adjusting the weight accordingly by using probability $\frac{p}{q}$.

The first iteration of this method [11] sets $k = \frac{\log^2 n}{n^2}$, yielding an $\tilde{O}(n^3)$ algorithm, but the following one [14] presented a simpler version, having $k = \frac{1}{2}$, resulting in an $\tilde{O}(n^{2.78})$ algorithm with easier analysis, that is further improved [12] (just by deeper bound analysis) to $\tilde{O}(n^2)$. We will set $k = q^c = \frac{1}{2}$ for the rest of the section, as it leads to a more intuitive analysis of the algorithm. The pseudocode in Algorithm 1 describes the high-level implementation idea for the approach that is analysed further.

Algorithm 1 `Unreliability`(G, p)

```
(V, E) ← G
if |V| ≤ 3 then
    return Brute_Unreliability(G, p)
end if
c ← Min_Cut(G)
if p^c > 0.5 then
    return Monte_Carlo(G, p)
end if
q ← ∛0.5                                   ▷ q^c = 0.5
h_1 ← Contract(G, 1 − q)
u_1 ← Unreliability(h_1, p/q)
h_2 ← Contract(G, 1 − q)
u_2 ← Unreliability(h_2, p/q)
return (u_1 + u_2)/2
```

Lemma 3.1.4. (Cycle Lemma in [11, 14]) *Let C be a cycle of n nodes where each pair of adjacent nodes is connected by $\frac{c}{2}$ edges, then C is the most unreliable graph having n nodes and minimum cut c.*

Let $R(G, p)$ be the graph resulted by taking G and contracting each edge from it with probability $1 - p$. $R(C, q)$ is expected to have at most $\frac{n}{\sqrt{2}}$ with high probability ([14]). For any graph G with n nodes and minimum cut c, more contractions are probable to happen for G, being more reliable than C, so $R(G, q)$ is expected to have fewer nodes than $R(C, q)$. Hence, by taking C as a bound for G, we expect that $R(G, q)$ will have at most $\frac{n}{\sqrt{2}}$ nodes with high probability.

Lemma 3.1.5. *Let $H = R(G, q)$. $u_H(\frac{p}{q})$ is an unbiased estimator for $u_G(p)$.*

Proof. We look at the probability of an edge failing in this scenario. Initially, the edge is not contracted with probability $1 - (1 - q) = q$, then it fails with probability $\frac{p}{q}$, which leads to a total probability of $q \cdot \frac{p}{q} = p$. Each edge still has the same failure probability,

so, since every cut of H is also a cut of G (by construction), the probability that the edges that fail in H contain a cut of G is the same. $u_H(\frac{p}{q})$ and $u_G(p)$ have the same mean, so the former is an unbiased estimator for network unreliability in G. □

For this estimator to produce an efficient algorithm, we need to bound the relative variance. The following lemma was first conjectured in [14] and then proved in [12], using a new technique that involves conditioning the structure of the graph on the failure of one of its small cuts.

Lemma 3.1.6. *Let $u_H(\frac{p}{q})$ has relative variance $O(q^{-c}-1)$.*

Note that for this lemma, we use the infinitesimal asymptotic version of the big-O notation (here q varies between $\frac{1}{2}$ and 1) to bound the relative variance of the estimator.

Using Lemma 3.1.6 with $q^c = 2^{-1}$, we get that $u_H(\frac{p}{q})$ has relative variance $O(1)$. Now we inductively prove that the relative variance of the estimator produced by the algorithm is $O(1)$. In the base case of n being less than a constant value, we compute $u_G(p)$ exactly, so the relative variance is 0. In the other base case, when $p^c > \frac{1}{2}$, the relative variance is $u_G(p)^{-1} - 1 \leq p^{-c} - 1 \leq 2 - 1 = 1$ (as in the proof of Lemma 3.1.1). In the induction step case, when two contractions are made, let H be one of the contracted graphs, and assume (by the induction hypothesis) that we have an estimator E for $u_H(\frac{p}{q})$ having relative variance $O(1)$. By composing the two unbiased estimators (Lemma 2.3.3) and Lemma 3.1.5, we get that E is an unbiased estimator for $u_G(p)$ with relative variance $(1+1)(1+1)-1 = 3$. Since two independent samples are performed, the relative variance in this case is $\frac{3}{2} = O(1) + \frac{1}{2}$. The total relative variance of the algorithm is thus bounded by $\tilde{O}(1)$.

Finally, we inductively analyse the time complexity for the algorithm, in a similar manner to the relative variance. The analysis assumes that the bound is deterministic, and each contraction of a graph with n nodes yields a graph having at most $\frac{n}{\sqrt{2}}$ nodes (which happens with high probability). The probabilistic bound is proved in [14], but is not the subject of this project. In the base cases, when n is constant, it takes constant work, and when $p^c > \frac{1}{2}$, using one Monte Carlo iteration takes linear amount of time. In the induction step, let $T(n)$ be an upper bound for the complexity of an instance of the problem with n nodes. Then, since a contraction is $O(n^2)$, we get $T(n) = O(n^2) + 2T(\frac{n}{\sqrt{2}})$. Using Master Theorem, $T(n) = O(n^2 \log n) = \tilde{O}(n^2)$. The total complexity is dominated by the induction step, so this algorithm for approximating network unreliability is $\tilde{O}(n^2)$.

3.2 Reliability Approximation Algorithm

Unlike the complementary problem, approximating network unreliability (u_G), approximating network reliability (rel_G) in polynomial time remained unsolved until recently, when Guo and Jerrum [10] introduced the first FPRAS for this task, being the only one known to date.

In directed graphs, another relevant problem, similar to the network reliability one, is the network reachability problem. In the simplest version of this problem, we are given a directed graph $G = (V, E)$, a special node r (named the root), the edge failure probability p, and we want to compute the probability ($reach_G(p, r)$) that the graph is root-connected (for each node $v \in V$, there is a directed path from v to r), after each edge fails independently. Recall that a bi-directed graph is a directed graph with the property that for each edge $(u, v) \in E$, the anti-parallel edge is also in the graph ($(v, u) \in E$).

Lemma 3.2.1. ([2]) *Let G be an undirected graph, and H be the equivalent bi-directed graph, then $rel_G(p) = reach_H(p, r)$ for any p and any $r \in V_G$.*

The network reliability algorithm presented by Guo and Jerrum is based on the equivalence of the network reliability problem and the reachability problem (Lemma 3.2.1). If we have a procedure that computes $reach_G(p, r)$, we can create one for computing $rel_G(p)$ by first building the equivalent bi-directed graph for G (let it be H), and choosing a random vertex as the root, then computing $reach_H(p, r)$. The rest of this section will focus on highlighting an algorithm for solving the reachability problem, using root-connected graphs sampling in order to approximate the solution.

3.2.1 Exact Sampler for Root-Connected Subgraphs

A sampler that draws (edge-weighted) root-connected subgraphs is an algorithm that, given a directed graph $G = (V, E)$, and a root node r, returns a subgraph $G' = (V, E')$, such that for every node $u \in V$ there is a path from u to r in G'. The sampler is an exact sampler if every subgraph is chosen with probability proportional to the weight of the selected edges (equal to the product of failure probabilities of erased edges and the not-disappearing probabilities of non-erased edges).

One of the simplest, but still efficient samplers of this type was introduced by Gorodezky and Pak [9], and is based on the Cluster-Popping algorithm described in their article.

Definition 3.2.2. A *cluster* C in a directed graph $G = (V, E)$, with root r, is a subset of V, such that $r \notin C$ and there is no edge $(u, v) \in E$ that has $u \in C$ and $v \notin C$. A cluster is minimal if there is no cluster C' with $C' \subset C$.

Intuitively, a cluster is a maximal set of nodes that has no outgoing edges to the outside of it and does not contain the root. It follows that every minimal cluster is strongly connected, and every two distinct minimal clusters are disjoint (Claims 2 and 3 in [10]). Note that a graph without any clusters is root-connected. With these observations, we introduce the following sampler (Algorithm 2), presented by Gorodezky and Pak [9], that re-randomises all outgoing edges of minimal clusters, until there is no cluster left.

This is an exact sampler (as proven in [9]), where the expected number of popped clusters before the graph is root-connected is $O((1 - p)^{-1}m^2n)$ (shown in [10]). Note that this sampler can be modified to sample from a graph where different edges can have different failure probabilities, as the graph structure is not modified (by contractions).

Algorithm 2 `Cluster_Popping`(G, p, r)

$(V, E) \leftarrow G$
$S \leftarrow$ subset of E, each edge being added with probability $(1 - p)$
$C \leftarrow$ minimal clusters in (V, S)
while $|C| > 0$ **do**
    $S' \leftarrow S$ without outgoing edges in clusters in C
    $T \leftarrow$ re-randomise all outgoing edges in C
    $S \leftarrow S' \cup T$
    $C \leftarrow$ minimal clusters in (V, S)
end while
return (V, S)

3.2.2 Approximation Algorithm for the Reachability Problem

Here we present the algorithm introduced by Guo and Jerrum [10] for approximating reachability, that uses the sampler previously described and the telescopic product technique. Consider an instance of the problem, having graph G with n vertices, root r and failure probability p, and construct a sequence of graphs $G_n, \ldots, G_1$, such that $G_n = G$. Having G_i, we create G_{i-1} by contracting a node adjacent to the root u and the root r (remove all edges connected r and u_i, and unify the nodes, keeping node r, preserving possible parallel edges). Hence, G_1 is the graph containing only node r and no edges, having $reach_{G_1}(p, r) = 1$.

Let $G_i = (V_i, E_i)$, then observe that $V_i \subset V_{i+1}$ and $E_i \subset E_{i+1}$. Consider (V_i, S_i) a random (exactly sampled) root-connected subgraph of G_i and create set T_{i+1} by starting from S_i and independently adding the edges between r and u_{i+1} with probability $1 - p$, producing $H_{i+1} = (V_{i+1}, T_i)$, which is a subgraph of G_{i+1}. Then, let R_{i+1} be the random variable that takes value 1 if H_{i+1} is root-connected in G_{i+1}, or 0 otherwise. We can see that $\mathbb{E}[R_{i+1}] = \frac{reach_{G_{i+1}}(p,r)}{reach_{G_i}(p,r)}$, so by taking the telescopic product

$$R = \prod_{i=2}^{n} \mathbb{E}[R_i] = \frac{reach_{G_n}(p, r)}{reach_{G_{n-1}}(p, r)} \cdot \ldots \cdot \frac{reach_{G_2}(p, r)}{reach_{G_1}(p, r)} = \frac{reach_{G_n}(p, r)}{reach_{G_1}(p, r)}$$

we conclude that $reach_G(p, r) = R$.

Using this procedure, we have an algorithm for the reachability problem, produced by using the exact sampler described in Section 3.2.1. When repeating the process for $5(1 - p)^{-2}(n - 1)\epsilon^{-2}$ times, and taking the average result, it was proven [9] that we obtain an ϵ-approximation with probability of failure $\frac{1}{4}$. Then, by applying the Median Trick (Lemma 2.3.5), we get an algorithm that approximates reliability using $O((1 - p)^{-2}(n - 1)\epsilon^{-2} \log(\delta^{-1}))$ samples, leading to an asymptotic time complexity of $O(\epsilon^{-2} \log(\delta^{-1})(1 - p)^{-3}m^2n^3)$.

4. Karger's Unreliability Algorithm in Practice

This chapter contains our first contribution, a practical analysis of Karger's $\tilde{O}(n^2)$ Unreliability Algorithm [12], introducing our implementation for it, and comparing results with other algorithms that solve the problem.

4.1 Algorithm Implementation

First, we describe an implementation of the algorithm based on Karger's paper [14], following to improve it in terms of empirical run time in the next section. The main procedure is described in Algorithm 1. When the graph is small, the `Brute_Unreliability` method computes the network unreliability exactly in $O(2^m n)$ time, using an exhaustive enumeration technique to get all possible failure outcomes and their probability, then a depth first search routine for checking if the network is connected. We implemented the `Min_Cut` procedure, which returns the size of the minimum cut in the graph, using Karger's Near-Linear Time Minimum Cut algorithm [15]. The `Monte_Carlo` routine represents a Naive Monte Carlo sample, which first fails each edge with probability p, and then checks if the resulted graph is connected, similarly to the brute-force procedure. Since we represent graphs as an edge list, linear-time work is achieved by deleting each edge with probability p and then checking if the graph is connected using a depth first search algorithm. The `Contract` procedure contracts each edge of a graph with probability $1 - q$, and we use a Disjoint-Set structure that keeps track of contracted-nodes sets for producing a linear time algorithm.

For completeness, the main `Unreliability` procedure is called $O(\epsilon^{-2})$ ($4\epsilon^{-2}$ in practice) times for each experiment, where we return the average as the answer, and we run $O(\log(\delta^{-1}))$ experiments, then get the median value as our final answer (based on the Median Trick method).

4.2 Improvements

4.2.1 Brute-force Improvements

We tested the correctness of our implementation by running it against the brute-force procedure (that gives the exact value of unreliability) on graphs of sizes 3 and 4, then compared the results. When tested on different (ϵ, δ) pairs, and multiple p values, the algorithm always yielded an ϵ-approximation for unreliability, confirming the theoretical bounds.

In order to test on larger graphs, we improved the brute-force routine using some heuristics. The first one is to check at each step in the exhaustive enumeration procedure if the graph is connected, and not continue the solution. The second one consists of not considering edges that unite two nodes that are already connected by previously selected edges in our search, leading to a maximum depth of $n - 1$ for each solution. These optimisations produce a faster algorithm, that we used to compare our implementation on graphs of size up to 10 nodes.

4.2.2 Min-cut Improvements

To assess the time efficiency of our algorithm, we run[1] it on complete graphs having sizes from 20 nodes to 25 nodes, with $\epsilon = 0.2$ and $\delta = 0.05$ (values which are mostly used in benchmarks [22]). The main observation was that the implementation was running very slow for these sizes of graphs, considering the theoretical complexity. We decided to do code profiling to see which parts are taking the most time, and noticed that the computation of minimum cut (`Min_Cut` procedure) was dominating over all the others.

Since the minimum cut algorithm was the bottleneck for our implementation, we tried changing it. The considered alternative algorithms are: Karger-Stein Minimum Cut Algorithm [16], Edmonds-Karp Maximum Flow Algorithm [8], Dinic Maximum Flow Algorithm [7] (the last two had to be run n times in order to find the global min-cut). We tested Karger's unreliability algorithm using all these implementations, on three different types of graphs that are used to model real world networks in practice (complete graphs, grid graphs, Erdős-Rényi random graphs[2]), using $\epsilon = 0.2$, $\delta = 0.1$, and multiple values for p. Each experiment was run 3 times, and we report the average time for each procedure over all trials conducted for a graph.

Tables 4.1, 4.2 and 4.3 present the running times for the tested algorithms (the sum of time periods for minimum cuts computed in `Unreliability`, without counting the other procedures), together with the ones for `Brute_Unreliability` and `Contract`, for complete graphs of size from 10 to 25, for square grid graph having side length from 2 to 8, and

[1] All experiments in this project were run on a single core of an Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz, and 16GB of RAM.

[2] In the Erdős-Rényi model, given a probability p, the graph is generated by taking all possible edges and adding each one to the graph with probability p.

n	Brute	Contract	Edmonds-Karp	Dinic	Karger-Stein	Karger Near-Linear
5	4	13	6	8	135	76616
10	17	102	60	115	2857	182766
15	26	245	145	310	15262	-
20	29	390	240	574	43103	-
25	38	628	390	1029	104530	-

Table 4.1: Time (ms) for procedures, run on complete graphs

n	Brute	Contract	Edmonds-Karp	Dinic	Karger-Stein	Karger Near-Linear
2	2	4	1	1	37	16920
4	13	77	95	136	7853	111761
6	15	172	296	437	102761	-
8	16	308	748	1091	694404	-

Table 4.2: Time (ms) for procedures, run on $n \times n$ square grid graphs

the average running times from 100 random connected Erdős-Rényi graphs, and number of nodes from 15 to 20. These tables show only some of the important experiments, full tables are available in Appendix A.

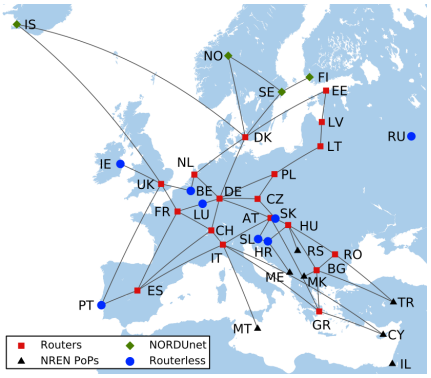
The Edmonds-Karp Algorithm was the best in all experiments, having running time similar to the contract procedure in some cases, while Dinic Algorithm was the next one, having similar results as the former one. Surprisingly, the Near-Linear Time algorithm was the slowest, and therefore it was very inefficient to run it on larger graphs (that is why some run times are missing from the given tables for this algorithm). We suspect that this algorithm is inefficient in practice because of the complex implementation and high constant factor. Having in mind the empiric results for these minimum cut algorithms, we decided to replace the current minimum cut algorithm implementation with the Edmonds-Karp one, as it is the most efficient.

4.3 Benchmarks

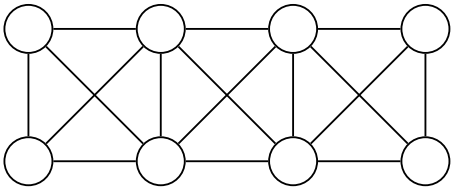
The following is a comparison of our implementation for Karger's unreliability algorithm with the results of other algorithms that can be used in practice. Most of the published algorithms are focused on experiments with small networks, having up to 20 nodes, but we present some methods that are tested on both small networks (for answer accuracy) and large networks (for performance).

Brute	Contract	Edmonds-Karp	Dinic	Karger-Stein	Karger Near-Linear
14	161	207	422	18385	-

Table 4.3: Average time (ms) for procedures, run on 100 random Erdős-Rényi graphs



(a) GEANT Network topology in 2009



(b) K_4^4 Graph Representation

Figure 4.1: Networks used in experiments

4.3.1 Comparison with Parallel Algorithms

Parallel algorithms are popular in the field, as they use the full processing power of a computer by processing independent subroutines at the same time, reducing the total execution time. The network reliability problem can be approached for parallel solving in different ways, either using a Monte Carlo algorithm [20], or by other methods (Master-Slave scheme [24]), all of them making use of either supercomputers with a large number of cores, or GPUs.

We compare our implementation against the parallel Monte Carlo one presented in [20] that uses GPU to approximate network reliability with absolute precision (instead of the relative one used in most approximation algorithms). Note that since Karger's unreliability algorithm solves the complementary problem, we need to complement both the failure probabilities and the results described in the paper.

The first network experimented by the parallel algorithm is the GEANT network structure in 2009[3] (Figure 4.1a), with the assumption that all edges have the same failure probability. The results for the two algorithms are represented in Table 4.4a.

The second graph type taken into study is the longitudinal graph K_4^n created by joining $n-1$ complete graphs with 4 nodes (K_4). A representation for K_4^4 can be seen in Figure 4.1b. The unreliability values produced by our implementation are complementary to the ones described in the paper. Since Karger's unreliability algorithm produces results with relative precision, and the tested edges are reliable (failure probability less than $\frac{1}{2}$), our approximation is better than the one described in the paper. The run times for the two algorithms on longitudinal graphs are shown in Table 4.4b

Observing the run times for the two algorithms, we can see that our implementation performs better than the parallel algorithm (even though it uses better hardware than our implementation) on lower failure probabilities for edges, as Karger's algorithm is not dependent on these probabilities.

[3]Dataset: http://topology-zoo.org/files/Geant2009.gml

p	Karger	GPU Parallel
0.25	0.28	2.09
0.1	0.35	2.11
0.01	0.35	2.16
0.001	0.35	2.16

(a) Time (s) for GEANT network

Graph	p	Karger	GPU Parallel
K_4^{50}	0.25	4.55	1.33
K_4^{50}	0.1	4.58	1.41
K_4^{100}	0.25	15.27	24.27
K_4^{100}	0.1	15.34	24.91

(b) Time (s) for K_4^n longitudinal graphs

Table 4.4: Comparison between Karger's algorithm and the GPU Parallel algorithm

n	`Karger`	$\mathcal{K}$-RelNet
2	0.03	0.09
3	0.13	0.34
4	0.32	0.89
5	0.63	2.43
6	0.96	5.78
7	1.24	13.18
8	1.77	27.59
9	2.34	53.90
10	2.90	119.55

Table 4.5: Time (s) for $n \times n$ square grid graphs

4.3.2 Comparison with other Approximation Algorithms

Another approximation algorithm for the network unreliability problem is $\mathcal{K}$-RelNet, introduced by Paredes [22]. This method reduces approximation of unreliability to counting of satisfiable boolean formulas, using a relatively small number of propositional variables (smaller than the ones used by Karger in his first FPRAS [13]), and then approximating this number using the state-of-the-art algorithm for approximate CNF model counting, ApproxMC [27]. An important advantage of $\mathcal{K}$-RelNet over Karger's algorithm is that the former one solves the general K-terminal network unreliability problem.

Based on the experiments conducted by Paredes, we reproduced[4] them by testing our implementation and $\mathcal{K}$-RelNet on $n \times n$ square grids having side lengths from 2 to 10. Accordingly, we use failure probabilities of 2^{-1}, 2^{-3}, 2^{-5}, and $\epsilon = 0.2$, $\delta = 0.05$ for the approximation parameters. Each experiment is run 3 times (for reducing variance), and for each grid size we report the average run time of all experiments (9 total trials for each n) in Table 4.5.

Following the experimental results, Karger's unreliability algorithm runs in the same magnitude of time as $\mathcal{K}$-RelNet on small grids, but outperforms it in the long run (on larger grids). It is worth mentioning that, even though our implementation uses randomness, it had almost no variation in the runs for the same grid size, contrary to $\mathcal{K}$-RelNet, which had much lower run times for bigger failure probabilities (this being expected, as the

[4]Used implementation made available by author (https://github.com/meelgroup/RelNet) to run both algorithms on the same system, eliminating differences created by different hardware.

number of trials in Paredes' algorithm is dependent on the failure probability). The better results achieved by our implementation also come from the fact that Karger's algorithm is specialised on all-terminal network reliability, while $\mathcal{K}$-RelNet solves the general K-terminal problem.

4.3.3 Comparison with Heuristic Algorithms

There is huge interest in heuristic algorithms in the literature, as they provide a faster way to compute the exact reliability of a network. Most of the heuristics used for the network reliability problem are based on graph reductions, which are transformations of the graph (that roughly preserve structure and properties, reducing the size of the graph) done before computing the answer. Two of the best heuristic procedures are compared in [23]. The first one [26] uses paths and cut-sets enumeration after applying reductions to find lower and upper bounds for the network reliability. The second one [3] uses a graph decomposition technique to compute the exact reliability value (also after applying graph reductions). Based on the results of the comparison between the two algorithms, we will take the decomposition algorithm for testing, as it has better run time performance.

For the comparison of Karger's unreliability algorithm with the decomposition algorithm we used the same networks tested in [23]. These are 11 networks from SNDlib [21], which is a library containing realistic models of telecommunication networks, most of them being based on the structures of either continents, countries or big cities (for example, "polska" network, shown in Figure 4.2, is based on the telecommunication network of Poland). Because of the realism of these networks, the failure probabilities of the edges are proportional to the distance between the endpoints, meaning that there is no global failure probability. In order to test Karger's unreliability algorithm, we use the average failure probability (computed in [26]) for all edges. This difference unleashes the potential of empirically studying the assumption presented in the introduction, that having the same edge failure probability does not vary the reliability of a network by much.

We run our implementation using $\epsilon = 0.2$ and $\delta = 0.05$ as approximation parameters, and present the results in Table 4.6, comparing them with the ones in [23]. Contrary to the other benchmarks that we have done, Karger's unreliability algorithm is not much faster than the decomposition algorithm, in fact being slower on some tested networks. Overall, both algorithms run in roughly the same magnitude of time, each of them being faster than the other in approximately half of the experiments.

We can now test how much having a unique failure probability affects the reliability of a network. In Table 4.7 we present the exact unreliability of the tested networks[5] (from [23]), the solution computed by our implementation, and the actual ϵ (relative precision of our estimation compared to the unreliability probability). Observing the outcomes of the trials, the solution produced by Karger's unreliability algorithm in half of the

[5]In this experiment we ignore all networks that have unreliability less than 10^{-8}, as the results provided in the paper have 8 decimal places of precision. precision.

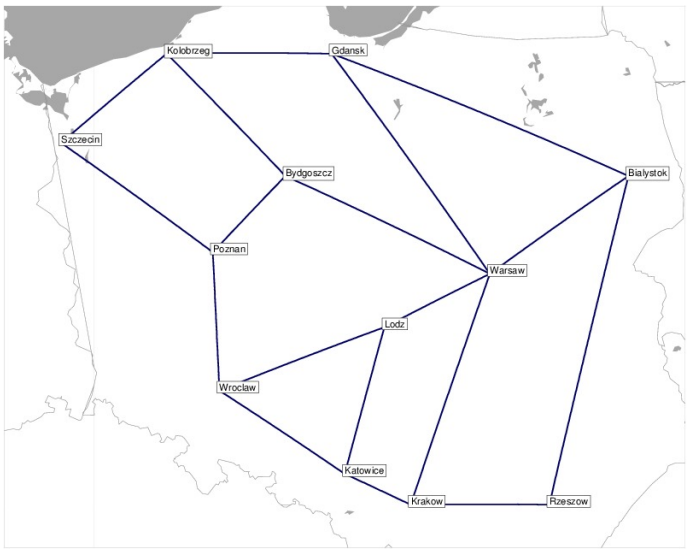


Figure 4.2: "polska" network from SNDlib

Network	Nodes	Edges	`Karger`	`Decomposition`
polska	12	18	0.09	0.08
atlanta	15	22	0.12	0.06
newyork	16	49	0.08	0.22
nobel-germany	17	26	0.11	0.06
geant	22	36	0.16	0.08
france	25	45	0.25	0.10
nobel-eu	28	41	0.32	0.13
pioro40	40	89	1.37	1.11
germany50	50	88	0.59	0.77
ta2	65	108	0.63	0.62
india35	35	80	0.33	0.45

Table 4.6: SNDlib networks properties and run time (s) for the two compared algorithms

Network	Unreliability	`Karger` Solution	Actual ϵ
atlanta	0.00004659	0.00006592	0.4150
newyork	0.00001199	0.00001124	-0.0621
geant	0.00000531	0.00000428	-0.1939
france	0.00007443	0.00006865	-0.0775
nobel-eu	0.00000047	0.00000026	-0.4329
ta2	0.00139541	0.00211905	0.5185

Table 4.7: Exact unreliability and the estimate of our solution, on SNDlib networks

experiments is valid, being a (0.2)-approximation for the exact unreliability. Nonetheless, the maximum relative error is 0.5185 on the "ta2" network. This error is a combination of the imprecision created by the approximation algorithm and the error produced by the assumption. Even though we tested on a relatively small number of networks, it results that, in realistic networks, assuming a uniform failure probability (equal to the average) over all edges instead of different failure probabilities does not change the result by much. It can obviously happen that changing the failure probabilities results in a far different outcome, but we expect that case to be quite rare in realistic networks.

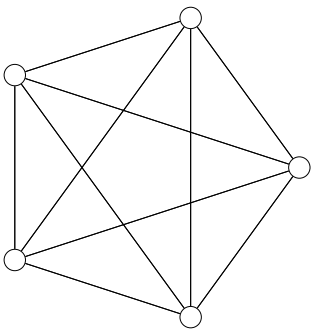
5. Limit Testing

In this part of the project, we show the practical limits of Karger's unreliability algorithm and Guo and Jerrum's reliability algorithm, testing both algorithms on multiple graph structures. We test each algorithm on networks having size appropriate for the theoretical complexity bound of it. That is, Karger's algorithm is tested on large graphs, up to 2500 vertices, while experiments on the reliability algorithm are done on much smaller graphs, having at most 20 vertices.

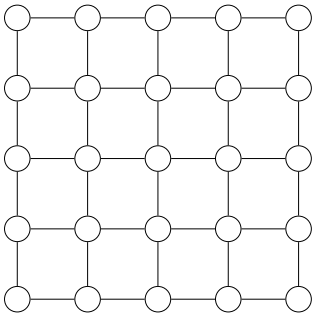
5.1 Karger's Unreliability Algorithm on Big Networks

Currently, no algorithm is as fast as Karger's unreliability algorithm, so this section presents some results on larger networks, using the optimised implementation shown in Section 4. These graphs could not be tested against any exact algorithms to confirm that the solution is a valid approximation (but there are theoretical guaranteed bounds), and the run time could not be compared with any other algorithm to compare the performance. All the further experiments are run using $\epsilon = 0.2$ and $\delta = 0.05$ as approximation parameters.

Firstly, we consider complete graphs (Figure 5.1a), as they are the biggest possible graphs with a given number of nodes, being appropriate for our worst case testing. Since this type of graphs are very reliable, we consider some high failure probabilities in our experiments. Table 5.1 shows the run times and approximation results on large complete



(a) Complete graph with 5 nodes



(b) 5×5 square grid graph

Figure 5.1: Graph types

n	Time (s)	$\hat{u}_G(0.1)$	$\hat{u}_G(0.5)$	$\hat{u}_G(0.9)$	$\hat{u}_G(0.95)$
100	30	9.62e−98	1.65e−28	2.99e−3	0.464
200	183	2.01e−197	2.47e−58	1.58e−7	7.23e−3
300	606	3.05e−297	3.06e−88	6.26e−12	6.53e−5
400	1404	4.04e−397	3.18e−118	2.17e−16	5.16e−7
500	3495	5.06e−497	3.00e−148	7.29e−21	3.82e−9
1000	45629	9.92e−997	1.87e−298	1.96e−43	5.58e−20

Table 5.1: Complete graphs results

n	Time (s)	$\hat{u}_G(0.01)$	$\hat{u}_G(0.05)$	$\hat{u}_G(0.1)$
10	3	4.64e−4	1.54e−2	8.38e−2
20	34	4.81e−4	2.24e−2	0.148
30	186	5.21e−4	2.98e−2	0.223
40	617	5.21e−4	3.90e−2	0.305
50	1514	6.20e−5	4.91e−2	0.392

Table 5.2: $n \times n$ square grids results

graphs, with number of nodes from 100 to 1000.

Another relevant type of networks are grid networks (Figure 5.1b), being irreducible by the most known heuristics. By structure, these graphs are relatively unreliable, so we run the experiments using smaller failure probabilities. Table 5.2 presents the performance of our implementation and approximation results on big square grid graphs, having 10 to 50 nodes on a side.

Based on the observed outcomes, our implementation of Karger's unreliability algorithm is very efficient in practice. On small and medium sized networks (less than 100 nodes), it is able to run in at most 30 seconds, making it suitable for real-time software that tracks dynamic networks. On the other hand, the algorithm can estimate the unreliability for large networks (complete graph of 1000 nodes, and square grid of 2500 nodes). Since some of these results are very small, exceeding double precision in C++, we had to use the extended double precision for all computations By our estimates, the unreliability of complete graphs with more than 5000 nodes and failure probability of 0.1 exceed the extended double precision, so further floating-point numbers improvements need to be implemented in order to use this algorithm. In terms of magnitudes, an interesting (but not necessarily surprising) result confirms that complete networks are highly reliable, even when the failure probability is close to 1 (0.95 in our experiments), while the square grid structure of networks is quite unreliable, having a relatively high unreliability even with a small failure probability (0.1 in our tests).

n	$p = 0.5$	$p = 0.75$	$p = 0.9$
4	0.3	1.0	7.7
5	0.5	2.2	17
6	1.0	4.2	34
7	1.9	6.9	57
8	3.3	11	88
9	5.4	18	136
10	8.7	27	203

(a) Time (s) for complete graphs

n	$p = 0.5$	$p = 0.75$	$p = 0.9$
2	0.2	0.9	9.1
3	3.7	21	227
4	24	150	1919
5	126	802	10163

(b) Time (s) for $n \times n$ square grids

Table 5.3: The reliability algorithm run time results

5.2 Practical Limits and Bottlenecks of the Reliability Algorithm

In terms of run time complexity, the reliability algorithm is $O((1-p)^{-3}m^2n^3)$ per sample, which roughly represents $O(n^5)$ for sparse graphs, or $O(n^7)$ for dense (or complete) ones. Thus, we test this method on small graphs, with at most 20 nodes. For the experiments, we implemented Guo and Jerrum's algorithm, as described in the paper, without using any additional optimisations, and used $\epsilon = 0.2$ and $\delta = 0.05$ as approximation parameters in all the following trials.

In the same manner as for the practical limit testing for the unreliability algorithm, we test this method on complete graphs and square grid graphs. Table 5.3a contains run times and approximation results for complete graphs having between 4 and 10 vertices, while Table 5.3b shows results for $n \times n$ square grid graphs having n from 2 to 5 (between 4 and 25 vertices in total). For all these experiments we used failure probabilities of 0.5, 0.75 and 0.9 that ensure a relative small reliability of the graph, while still being appropriate in terms of running time, since this algorithm runs in time proportional to $(1-p)^{-3}$.

In order to find the bottleneck in this algorithm, we tested the sampler separately from the rest of the procedure. We observed that, on average, sampling root-connected graphs using the Cluster-Popping algorithm (Algorithm 2) is fast, taking way less iterations than the theoretical expected bound. We were able to run the sampler on larger graphs, up to 2000 nodes, and noticed that the number of popped clusters is proportional to the density of the graph and inverse proportional to the probability of non-failure $(1-p)$, but has a very low constant. In particular, with a failure probability of $p = 0.999$, roughly 3000 cluster popping steps were needed for a sparse graph with 1000 nodes, and around 50 steps for a dense graph having the same number of nodes. Setting a smaller failure probability significantly lowers the number of cluster popping steps completed. Thus, the main bottleneck of the reliability algorithm is the big number of samples needed in order to approximate the reliability. We tried, for practical purposes, to reduce the number of samples required (giving up the theoretical guarantees), but our experiments did not

produce valid approximations with high probability, as needed. The big number of samples required for the algorithm shows that, even with a relatively fast sampler, it is hard to produce a good-enough estimation of the reliability of the network.

Considering the results, the reliability algorithm proposed by Guo and Jerrum seems inefficient in applied settings, as it shows performance limitations when run on relatively small graphs, more exactly, on complete graphs with 10 vertices (dense graph) or grid graphs with 25 vertices (sparse graph). This outcome is not a surprise considering also the big asymptotic run time complexity. Since this algorithm was introduced rather recently, there is place for improvement by either practical means or by procedure structure changes, from a theoretical point of view.

6. Empirical Approximation Bounds

Some approximation schemes produce an estimate for the reliability (or unreliability) of a network, but do not necessarily offer any guarantees in terms of the relative error (this kind of methods are called guarantee-less procedures). Both algorithms that we focused on for this project, Karger's unreliability algorithm and Guo and Jerrum's reliability algorithm, offer theoretical approximation bounds, producing an (ϵ, δ)-approximation of the solution. In this chapter, we present our third contribution, a study of the empirical approximation values for these two algorithms, in order to see if their theoretical analysis is tight.

6.1 Empirical Bounds for Karger's Algorithm

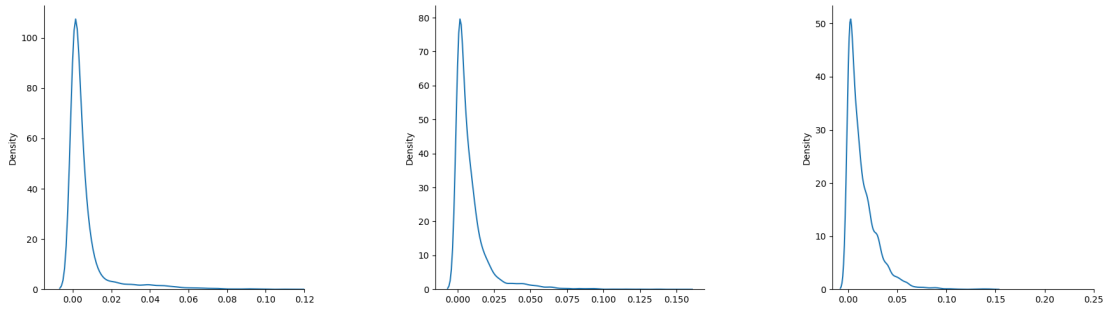
In our experiments, we are interested in finding the actual approximation errors when testing Karger's unreliability algorithm using $(0.05, 0.05)$, $(0.1, 0.05)$ and $(0.2, 0.05)$ as approximation parameters. We find the actual approximation error ϵ_0, by running both the brute-force algorithm (which produces the exact unreliability value) and our implementation on the same input network, and then computing $\epsilon_0 = \left|\frac{\hat{u_G}}{u_G} - 1\right|$. Since the run time complexity of the brute-force algorithm is big, small graphs, having at most 10 vertices, are utilized in our trials. For each of the tested networks, we use 8 different failure probability values ($p \in \{0.001,\ 0.01, 0.1,\ 0.2, 0.25,\ 0.5, 0.75, 0.9\}$) and report the maximum empiric error between these 8 trials.

First, we evaluated the algorithm on complete graphs of sizes $8, 9$ and 10, and square grid graphs of sizes 4 and 9. Table 6.1 contains the results for these graphs, using $p = 0.1$ as failure probability. Furthermore, we tested our implementation on 1000 uniformly random generated connected graphs, having between 6 and 10 vertices. The graphs in Figure 6.1 show the distribution density of empirical approximation values, for each of the initially used ϵ parameters. Note that the graphs have different scales on both `x` and `y` axis.

Based on the results, all densities are dominated by some interval in the first part: for $\epsilon = 0.05$, most values are less than 0.025, for $\epsilon = 0.1$, the majority of values are less than 0.04, and for $\epsilon = 0.2$ most values are less than 0.07. In each of these cases, we compute the empirical approximation error ϵ_0 by considering $\delta = 0.05$ as a tight bound, and report the results in Table 6.2a. In a similar manner, we define the empirical failure rate δ_0 as the rate of valid approximations, considering the given ϵ as approximation parameter, and

Graph	u_G	$\epsilon = 0.05$		$\epsilon = 0.1$		$\epsilon = 0.2$	
		$\hat{u}_G$	ϵ_0	$\hat{u}_G$	ϵ_0	$\hat{u}_G$	ϵ_0
complete 8	8.00e−7	8.02e−7	0.0031	8.12e−7	0.0159	7.82e−7	0.0225
complete 9	9.00e−8	9.003e−8	0.0001	8.96e−8	0.0044	8.74e−8	0.0288
complete 10	1.00e−8	9.99e−9	0.0009	9.96e−9	0.0037	9.863e−9	0.0140
grid 2	5.23e−2	5.22e−2	0.0019	5.21e−2	0.0023	5.20e−2	0.0041
grid 3	5.30e−2	5.307e−2	0.0010	5.22e−2	0.0149	5.363e−2	0.0183

Table 6.1: Approximation values for $p = 0.1$



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$

Figure 6.1: Distribution density of the empirical approximation errors

present these results in Table 6.2b.

These outcomes are incredibly good, since both the empirical error and empirical failure rate are way better than the theoretical guarantees. The experiments suggest that the error is at least 2 to 3 times smaller in practice, while the failure rate becomes almost insignificant. The best improvements are made for $\epsilon = 0.2$, where the empirical error is 5 times smaller, but there is no trial that did not produce a valid approximation (meaning that $\delta_0 = 0$). These observations hint that the guaranteed bounds are not tight, and further refinement in the theoretical analysis could be done.

6.2 Empirical Bounds for the Reliability Algorithm

In a similar manner, we test the reliability algorithm using $(0.05, 0.05)$, $(0.1, 0.05)$ and $(0.2, 0.05)$ as approximation parameters, and compute the empirical approximation ration ϵ_0 and empirical failure rate δ_0 in the same way as described in the previous section. As

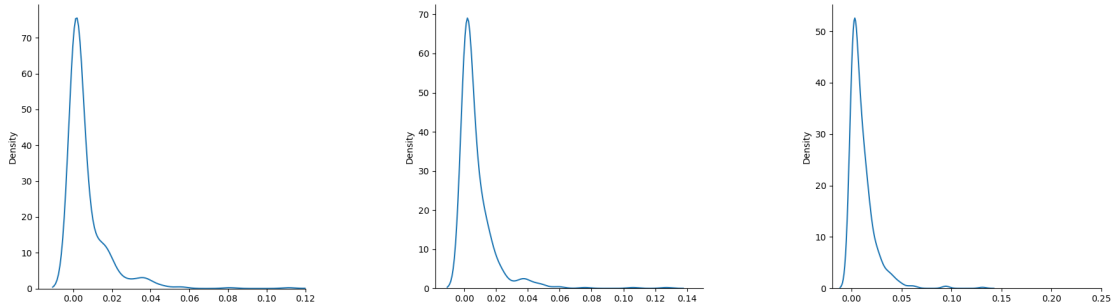
ϵ	ϵ_0
0.05	3.103e−2
0.1	3.469e−2
0.2	4.427e−2

(a) Empirical error

ϵ	δ_0
0.05	0.023
0.1	0.0022
0.2	0.0

(b) Empirical failure rate

Table 6.2: Empirical approximation parameters for Karger's unreliability algorithm



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$

Figure 6.2: Distribution density of the empirical approximation errors

ϵ	ϵ_0
0.05	2.983e−2
0.1	3.019e−2
0.2	3.482e−2

(a) Empirical error

ϵ	δ_0
0.05	0.010
0.1	0.004
0.2	0.0

(b) Empirical failure rate

Table 6.3: Empirical approximation parameters for the reliability algorithm

this algorithm is much slower, we base our trials on even smaller networks.

We use a data set of 200 uniformly random generated connected graphs, having between 4 and 7 vertices. The graphs in Figure 6.2 show the distribution density of empirical approximation values, for each of the initial ϵ parameters. Having these outcomes, we observe that here is also the case that the distributions are heavily dominated by smaller values. Then, the resulted empirical approximation error ϵ_0 and empirical failure rate δ_0 for each initial ϵ value are presented in Table 6.3a, and Table 6.3b respectively.

The reliability algorithm achieves even better results than Karger's unreliability algorithm in practice, yielding smaller experimental approximation factors in all cases. The empirical error is much smaller than the theoretical one, whereas the empirical failure rate is less than 0.01 for all three ϵ values. As in the previous analysis, the best improvements are made for $\epsilon = 0.2$, where the error is almost 6 times smaller, and $\delta_0 = 0$, as there is no experiment yielding a non-valid estimation. Thus, these results suggest that the analysis of this algorithm can be further tightened, resulting in both a faster and more precise procedure.

7. Conclusions and Future Directions

We investigated the theoretical and empirical analysis of the state-of-the-art approximation algorithms (Karger's unreliability algorithm [12] and Guo and Jerrum's reliability algorithm [10]) for the network reliability and unreliability problems, focusing on their practical performance. We provide appropriate network size limits for which these algorithms are effective, and measure the empirical approximation error in this regime.

For Karger's unreliability algorithm, we provide[1] an implementation of the algorithm which to the best of our knowledge has not been done before, even without our optimisations. Furthermore, we analyse the bottlenecks of the relevant procedures, coming up with several optimisations in order to speed up the performance of the algorithm in applications, namely by improving the brute-force procedure and using a procedure for minimum-cut size computation that is more efficient in practice. We then show an in-depth empirical comparison between our implementation and other methods for computing network reliability, such as parallel algorithms and heuristic algorithms. We test these algorithms on edge-case graphs and models of realistic networks, where Karger's algorithm outperforms all the competitor methods. Additionally, we run our implementation on large networks and provide for the first time estimations for the unreliability of graphs having up to 2500 vertices, further showing the efficiency of this procedure.

For Guo and Jerrum's Reliability algorithm, we present the limitations that arise in practice when run on medium sized graphs (around 20 vertices), as the algorithm is not a good fit for usage in real life software. We find that, even though the sampler used in the algorithm is fast (since, from our experiments, it runs efficiently on large instances), computing a valid estimation for the network reliability needs an overhead given by the large number of samples needed. Since this is a relatively new algorithm (the FPRAS for the network reliability problem) further improvements in terms of run-time complexity can be made such that it becomes more attractive for practical matters.

We analyse both algorithms in terms of empirical approximation parameters, yielding a somewhat surprising, but encouraging result, that both algorithms have rather conservative theoretical guarantees. In both cases, the actual approximation error was far smaller than the theoretical bound, while the failure rate was insignificant compared to the theoretical one. These results suggest that further thorough analysis of the algorithms might

[1] All code and resources for this project are in a private GitHub repository, available upon request.

yield tighter theoretical guaranteed bounds.

The subject of an interesting future study is implementing as parallel approximation algorithms, either Karger's unreliability algorithm, or Guo and Jerrum's reliability algorithm, which can improve the running time by a significant constant factor. It appears that both methods should be relatively easy to parallelise, since they use multiple independent trials of the same procedure, both in the median trick step, and in the main procedure (contraction procedure for Karger's algorithm, and the sampling procedure for the reliability algorithm). We could not conduct this experiment because of the lack of parallel hardware.

From a theoretical point of view, one interesting direction is to verify Karger's conjecture [12], stating that the analysis of the $\tilde{O}(n^2)$ unreliability algorithm can be further improved to a near-linear time bound. On our experiments, this conjecture seems to hold, at least on models of realistic networks. Furthermore, our considerations suggest that tighter bounds also hold for the reliability algorithm, by refining the existing analysis.

7.1 Reflection

The project was open-ended, with the purpose of studying the state-of-the-art algorithms for approximating network reliability and unreliability, and extending the theoretical analysis, while exploring the practical nature of the two algorithms. Perhaps surprisingly, the empirical performance of the two algorithms has not been considered before. This lack of a rigorous practical analysis represented a significant gap to bridge in the current understanding of the aforementioned procedures.

I[2] started reading on Karger's unreliability algorithm stated in 2017 [14] and Guo and Jerrum's reliability algorithm [10], paying particular attention to understanding the analysis and its existing limitations. For the theoretical aspect, my supervisor suggested studying the conjectures stated in the future work section of Karger's paper, and I subsequently tried proving or disproving some of them. Later on, I came across the fairly recent paper of Karger [12], published in 2020, which addressed these conjectures, therefore improving the analysis of the unreliability algorithm. At this point, extending or improving the reliability algorithm seemed relatively hard, and there were no further promising directions for the theoretical analysis of Karger's algorithm.

On the practical side, I began by implementing the aforementioned algorithms, following the respective papers [12, 10]. Then, I considered the bottlenecks of these procedure, and came up with practical optimisations that sped up these implementations by at least an order of magnitude. In order to get an appropriate feeling for the efficiency of these algorithms, I tested them on graphs that capture the most difficult cases, as well as instances occurring as part of realistic networks. A natural next step was to search for other heuristic methods for computing the network reliability and compare them to our algorithms.

[2] In this part, the author uses the singular pronoun "I" to refer to themselves.

Here we found that Karger's algorithm is effective in real world large network applications, while Guo and Jerrum's algorithm becomes impractical even for medium-sized graphs. Finally, I discovered another fundamental metric for approximation algorithms, called the empirical error, which I then practically studied in relation to Karger's algorithm and the reliability algorithm.

On the engineering side, the project involved implementing and testing multiple procedures for computing the reliability and unreliability of a network. I had to implement this in a modular way, with a disciplined approach to analysing the results.

Retrospectively, I think that my biggest achievement during this project was to get a comprehensive understanding of the field of approximate randomised algorithms and probabilistic computation. Even though the theoretical analysis of the state-of-the-art algorithms required highly technical machinery, in this report, I condense these details into a concise form, focusing on providing a good intuition rather than diving deep into the involved technical details.

Bibliography

[1] S. H. Ahmad. "Simple enumeration of minimal cutsets of acyclic directed graph". In: *IEEE Transactions on Reliability* 37.5 (1988), pp. 484–487. DOI: 10.1109/24.9868.

[2] Michael O. Ball. "Complexity of network reliability computations". In: *Networks* 10.2 (1980), pp. 153–165. DOI: https://doi.org/10.1002/net.3230100206.

[3] Jacques Carlier and Corinne Lucet. "A decomposition algorithm for network reliability evaluation". In: *Discrete Applied Mathematics* 65.1 (1996). First International Colloquium on Graphs and Optimization, pp. 141–156. ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(95)00032-M.

[4] Charles J. Colbourn. *The Combinatorics of Network Reliability*. USA: Oxford University Press, Inc., 1987. ISBN: 0195049209.

[5] Alex Davila-Frias, Saeed Salem, and Om Prakash Yadav. "Deep Neural Networks for All-Terminal Network Reliability Estimation". In: *2021 Annual Reliability and Maintainability Symposium (RAMS)*. 2021, pp. 1–7. DOI: 10.1109/RAMS48097.2021.9605767.

[6] Alex Davila-Frias and Om Prakash Yadav. "All-terminal network reliability estimation using convolutional neural networks". In: *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 0.0 (0), p. 1748006X20969465. DOI: 10.1177/1748006X20969465.

[7] Yefim Dinitz. "Dinitz' Algorithm: The Original Version and Even's Version". In: Jan. 2006, pp. 218–240. ISBN: 978-3-540-32880-3. DOI: 10.1007/11685654_10.

[8] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *J. ACM* 19.2 (1972), pp. 248–264. ISSN: 0004-5411. DOI: 10.1145/321694.321699.

[9] Igor Gorodezky and Igor Pak. "Generalized Loop-Erased Random Walks and Approximate Reachability". In: *Random Struct. Algorithms* 44.2 (2014), pp. 201–223. ISSN: 1042-9832. DOI: 10.1002/rsa.20460.

[10] Heng Guo and Mark Jerrum. "A Polynomial-Time Approximation Algorithm for All-Terminal Network Reliability". In: *SIAM Journal on Computing* 48.3 (2019), pp. 964–978. DOI: 10.1137/18m1201846.

[11] David R. Karger. “A Fast and Simple Unbiased Estimator for Network (Un)reliability”. In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 635–644. DOI: 10.1109/FOCS.2016.96.

[12] David R. Karger. “A Phase Transition and a Quadratic Time Unbiased Estimator for Network Reliability”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 485–495. ISBN: 9781450369794.

[13] David R. Karger. “A Randomized Fully Polynomial Time Approximation Scheme for the All Terminal Network Reliability Problem”. In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*. STOC ’95. Las Vegas, Nevada, USA: Association for Computing Machinery, 1995, pp. 11–17. ISBN: 0897917189. DOI: 10.1145/225058.225069.

[14] David R. Karger. “Faster (and Still Pretty Simple) Unbiased Estimators for Network (Un)reliability”. In: *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*. Ed. by Chris Umans. IEEE Computer Society, 2017, pp. 755–766. DOI: 10.1109/FOCS.2017.75.

[15] David R. Karger. “Minimum Cuts in Near-Linear Time”. In: *J. ACM* 47.1 (2000), pp. 46–76. ISSN: 0004-5411. DOI: 10.1145/331605.331608.

[16] David R. Karger and Clifford Stein. “A New Approach to the Minimum Cut Problem”. In: *J. ACM* 43.4 (1996), pp. 601–640. ISSN: 0004-5411. DOI: 10.1145/234533.234534.

[17] David R. Karger and Clifford Stein. “An $\tilde{O}(n^2)$ Algorithm for Minimum Cuts”. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. STOC ’93. San Diego, California, USA: Association for Computing Machinery, 1993, pp. 757–765. ISBN: 0897915917. DOI: 10.1145/167088.167281.

[18] Richard M. Karp and Michael Luby. “Monte-Carlo algorithms for the planar multiterminal network reliability problem”. In: *Journal of Complexity* 1.1 (1985), pp. 45–64. ISSN: 0885-064X. DOI: https://doi.org/10.1016/0885-064X(85)90021-4.

[19] Ruiying Li, Ning Huang, and Rui Kang. “A new parameter and its algorithm for network connection reliability: k/N-terminal reliability”. In: *2009 First International Conference on Future Information Networks*. 2009, pp. 259–262. DOI: 10.1109/ICFIN.2009.5339609.

[20] Denis A. Migov, Tatyana V. Snytnikova, Alexey S. Rodionov, and Vyacheslav I. Kanevsky. “Network Reliability Calculation with Use of GPUs”. In: Kaliningrad, Russia: Springer-Verlag, 2021, pp. 210–219. ISBN: 978-3-030-86358-6. DOI: 10.1007/978-3-030-86359-3_16.

[21] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. "SNDlib 1.0–Survivable Network Design Library". English. In: *Networks* 55.3 (2010), pp. 276–286. DOI: 10.1002/net.20371.

[22] R. Paredes, L. Dueñas-Osorio, K.S. Meel, and M.Y. Vardi. "Principled network reliability approximation: A counting-based approach". In: *Reliability Engineering & System Safety* 191 (2019), p. 106472. ISSN: 0951-8320. DOI: https://doi.org/10.1016/j.ress.2019.04.025.

[23] Willem Pino, Teresa Gomes, and Robert Kooij. "A Comparison between Two All-Terminal Reliability Algorithms". In: *Journal of Advances in Computer Networks* 3 (Jan. 2015), pp. 284–290. DOI: 10.18178/JACN.2015.3.4.183.

[24] Kirill Sergeev and Denis Migov. "On Parallel Calculation of All-Terminal Network Reliability". In: *2021 17th International Asian School-Seminar "Optimization Problems of Complex Systems (OPCS)*. 2021, pp. 104–107. DOI: 10.1109/OPCS53376.2021.9588720.

[25] A. M. Shooman. "Algorithms for network reliability and connection availability analysis". In: *Proceedings of Electro/International 1995*. 1995, pp. 309–333. DOI: 10.1109/ELECTR.1995.471030.

[26] Jaime Silva, Teresa Gomes, David Tipper, Lúcia Martins, and Velin Kounev. "An Effective Algorithm for Computing All-Terminal Reliability Bounds". In: *Netw.* 66.4 (2015), pp. 282–295. ISSN: 0028-3045. DOI: 10.1002/net.21634.

[27] Mate Soos and Kuldeep S. Meel. "BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting". In: *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*. Jan. 2019.

[28] Leslie G. Valiant. "The Complexity of Enumeration and Reliability Problems". In: *SIAM Journal on Computing* 8.3 (1979), pp. 410–421. DOI: 10.1137/0208032.

Appendix A. Unreliability Experimental Results

This appendix contains complete tables for the experiments conducted using Karger's Unreliability algorithm in Section 4.2.2

Experiments in Tables A.1 and A.2 are run using $\epsilon = 0.2$, $\delta = 0.05$ and the following different values for $p \in \{0.01, 0.1, 0.2, 0.5, 0.9\}$. For every combination of graph structure and p, 3 trails were run, and we report the average over all experiments conducted for a given graph (15 trials in total for each graph).

n	`Brute`	`Contract`	`Edmonds-Karp`	`Dinic`	`Karger-Stein`	`Karger Near-Linear`
5	4	13	6	8	135	76616
6	6	24	12	21	303	95995
7	9	37	20	35	579	104888
8	10	54	31	56	1047	119498
9	12	70	40	76	1666	164431
10	17	102	60	115	2857	182766
11	19	125	75	147	4218	-
12	23	157	95	191	6382	-
13	22	173	105	218	8364	-
14	24	201	122	257	11480	-
15	26	245	145	310	15262	-
16	24	262	154	337	18505	-
17	31	332	196	439	26014	-
18	32	355	212	481	31181	-
19	30	366	219	519	36098	-
20	29	390	240	574	43103	-
21	32	449	272	659	52427	-
22	34	484	300	739	63142	-
23	35	527	325	836	75626	-
24	37	582	357	926	89301	-
25	38	628	390	1029	104530	-

Table A.1: Time (ms) for procedures, run on complete graphs

n	Brute	Contract	Edmonds-Karp	Dinic	Karger-Stein	Karger Near-Linear
2	2	4	1	1	37	16920
3	6	27	30	39	1083	42603
4	13	77	95	136	7853	111761
5	17	126	184	269	32528	-
6	15	172	296	437	102761	-
7	13	224	450	661	277420	-
8	16	308	748	1091	694404	-

Table A.2: Time (ms) for procedures, run on $n \times n$ square grid graphs