Experimental study of the growth rate of network reliability

Diana-Andreea Tanase

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

This paper presents the all-terminal network reliability of square grids. This #*P*-hard problem was solved using a polynomial-time approximation scheme based on sampling root-connected spanning subgraphs from sequential contractions of the input graph. The algorithm, to the best of our knowledge, has not been practically implemented before and the reliability polynomial of grids is known only for graphs of up to 5 x 5 nodes.

Comprehensive focus is devoted to presenting the individual modules of the approximation algorithm: contraction, sampling and expansion. This paper describes an iterative implementation of the recursive algorithm for sampling root-connected subgraphs, which allowed a performance analysis of the Sampling Module for grids of up to 100 x 100 vertices with 5 different failure probabilities on edges. This paper also provides an approach to generate the samples required by the estimation algorithm in parallel.

The work was mostly devoted to computing the reliability of grids with edges failure probability $p = 0.5$, of up to 15 x 15 nodes. However, estimation of the reliability was also possible for smaller grids with failure probability in the set $\{0.1, 0.25, 0.75, 0.9\}$. In our analysis, we look at the approximated reliability and the logarithmic reliability growth rate of grids, as well as of some family of graphs with known reliability polynomial or recursive expressions: paths, pans, ladders and complete graphs. The behaviour of these values, varying the size of the graph and the failure probability, and the distance from the estimations to their upper bounds is investigated. In addition, the reliability for $p = 0.5$ enables us to approximate the total number of connected spanning subgraphs of square grids.

Acknowledgements

I would like to sincerely thank my project supervisor, Dr. Heng Guo, for his valuable guidance on my MInf I and MInf II projects. I would thank him as well for proposing a project on a topic I found very interesting. I would also like to express my gratitude to my family and friends for believing in me and for showing me their moral support throughout my academic years.

Table of Contents

Chapter 1

Introduction

In the first part of the MInf project, we analysed the roots of graph polynomials resulted from applying the b-matching constraint to regular graphs of up to 20 vertices and degree up to 11. The project was motivated by the work on finding fully polynomial randomised approximation schemes for counting constraint satisfaction problems that are #P-hard, such as counting the number of b-matchings. The existence of approximation algorithms for such hard problems depends on the absence of zeros of graph polynomials in a disk centred at the origin [PR17] or in a strip containing the $[0,1]$ interval [GLLZ]. Hence, we were interested in the distribution of the roots around the unit circle, to establish the possibility of existing approximation algorithms for the b-matching constraint problem. An overview of MInf - Part I project is provided in Section 2.

For the second part of the MInf project, we shifted our focus from establishing the existence of approximation schemes to the practical aspects of implementing and analysing the results of an approximation algorithm for the all-terminal network reliability problem. With a polynomial randomised approximation scheme already existing for this #P-hard problem, we were not interested anymore in the behaviour of the roots of the reliability polynomial. In addition, the location of the roots of the reliability polynomials has been studied before, with research providing upper bounds for the modulus of the roots [BM17], finding the graphs with roots of smallest modulus [BD19] and disproving the conjecture that the roots lie inside the unit circle [RS04].

1.1 Motivation

The study of network reliability began in 1956 when E.F. Moore and C.E. Shannon, inspired by John von Neumann's problem of designing reliable computing circuits, introduced a probabilistic model to describe the reliability of relays [MS56]. By their model, the nodes of a network are perfectly reliable (they do not fail) and the links between them may fail independently with certain probabilities. Under these circumstances, the problem under consideration is finding the probability that the network is still connected, with a path of edges between any two nodes, which is normally referred to as *the network reliability*.

Much research interest has been devoted during the years to understanding and efficiently computing the network reliability. Usually modelled with the helps of graphs that represent relations between instances (nodes), its importance has been established in many areas of study, such as in the field of communication networks that analyses the connections between computers or internet routers ([HMW74], [SF71], [RKP86]). Nonetheless, its great significance has been recognised in other areas such as engineering (for modelling high-reliable networks) or, in particular, electrical engineering (for instance, as in the analysis of redundancy networks in electric circuits with parameters such are impedances, currents and voltages, ([Mos58] and [Min59]), statistics (for example, in epidemiology for the effect of network structure on the spread of diseases [YKE13]) and mathematical graph theory (for instance, for assigning a partial ordering of the edges in the graph according to reliability importance [PP94]).

The reliability measure could also be defined as the probability that given source and sink nodes are connected (for the *2-terminal reliability*) or as the probability that all vertices (nodes) in a subset of size K, belonging to a subgraph of the initial network, are connected with each other by operational edges. In this case, it is known as *K-terminal reliability*. When the subset includes all vertices of the graph, the network reliability is called *all-terminal reliability*, which we were concerned with in this paper. If edges have an equal probability of independently failing, the all-terminal reliability can be computed with an associated *reliability polynomial*. The reliability polynomial is a graph invariant, an expression describing the dependence of the graph's reliability on the failure probability. A graph invariant is a function that depends only on the structure of the graph and not on its various representations.

The reliability polynomial is given by summing over the connected spanning subgraphs of the graph $G = (V,E)$ that models the network. A spanning subgraph is a graph $G' = (V,E')$, with $E' \subseteq E$ and the same vertex set as G. The reliability polynomial weights the edges of the connected subgraph (operational with probability $1-p$) and those failing, outside the spanning set:

$$Z_{rel}(G,p) = \sum_{S \subseteq E,(V,S)\ connected} (1-p)^{|S|} p^{|E|-|S|},$$

where p is the probability of each edge failing independently and E is the set of edges of the graph G.

The reliability expression can also be written using the number of path-sets (subset of edges that makes the graph is operational) or cut-sets (subset of edges without which the graph is not operational) [PR18]. In addition, the reliability polynomial is expressible in terms of the Tutte polynomial [EMM11], a fundamental invariant in graph theory with many applications. For example, it is related to the number of spanning trees, acyclic edge subsets or the chromatic polynomial which gives the number of vertex colourings [WW93]. These representations are also discussed in Section 3.2.1.

With some exceptions, the network reliability belongs to the #*P*-hard class of problems [PB83], introduced by Valiant [Val79] to express the computational complexity of *NP*-hard counting problems, meaning that the time to calculate the reliability of a generic graph grows exponentially. There is no polynomial-time algorithms for computing all the spanning (connected) subgraphs of a graph, required by the reliability polynomial expression. Moreover, the problem of computing the coefficients of the Tutte polynomial is #*P*-hard as well for planar graphs (which can be drawn in the two-dimensional space without edges crossing each other) ([Ver05], [EMM11]) . Hence, extensive work has been focused towards finding more efficient solutions for the network reliability.

For some restricted families of graphs, the existence of exact reliability polynomials or recursive relations based on increasing the size of a graph in the same family have been shown. For example, [CS03] provides an exact solution for the reliability polynomials for lattice strips L_x x L_y of fixed widths, with $L_y \leq 4$, and arbitrarily length L_x, using calculations of the Tutte polynomials for these graphs (such as the Tutte polynomials for recursive family of graphs in [CS04]). The known reliability polynomials, as functions of the failure probability and the size of the graph, have been established for some well-known type of graphs, such as ladder (an L_x x L_2 lattice) [Tan06], cycle [CS03], pan, path and star graphs [Wei08]. Some of these graphs and their polynomials are presented in Section 3.2.5. In addition, a recursive expression for the reliability polynomial of the complete graph has been derived in [SI98] (using Tutte's polynomial) and [Tit99] (using partitions of the vertex set). [SI98] introduces a recursive reliability relation according to a deletion-contraction approach, where the reliability of a graph G depends on the reliability of the graph resulted by deleting an edge, contracting its vertices and keeping all the other edges incident to them, which can be computed in $O(2^{O\sqrt{n}})$ time for planar graph with n vertices. This technique, known as the *factorization method*, can be improved in the average case with appropriate edge selection ([PS86]). Other algorithms, based on enumeration of spanning trees ([BN79]), acyclic subgraphs ([SH81]) or node partitions ([Buz80]) have been developed, but their complexity grows exponentially with the size of the graphs.

To solve the problem of computing the reliability of general graphs with no known solutions, approaches that provide approximations of the real values have been proposed, including determining the lower and upper bounds (first noticed in [Kel65], for failure probabilities close to 0 and 1, and improved by Van Slyke and Frank in [VSF71]) or using Markov Chain Monte Carlo evaluations (for example, presented in [KL85], [Fis86] and [GS16]). Approximating the Tutte polynomial is a difficult task as well, with Goldberg and Jerrum proving in [GJ08] that fully polynomial randomised approximation scheme (*FPRAS*) exists for polynomials $T(G,x,y)$ only for a limited region in the xy plane. [EMM11] provides a good summary on the work on Tutte polynomials.

The problem of finding a polynomial-time randomised approximation scheme for the network reliability was still unresolved before the first efficient approximation algorithm was recently given in [GJ19], using the equivalence of reliability with *reachability* in bi-directed graphs. Reachability, introduced by Ball and Provan in [BP83], is defined for directed graphs with a node chosen as a distinguished root. It measures the probability that if each arc independently fails with a probability, there still is a path in the remaining graph from each node to the root (the graph is *root-connected*). In [Bal80], Ball showed that reliability is equivalent to reachability in bi-directed graphs, obtained by replacing every undirected edge with a pair of anti-parallel arcs with the same failure probability in either direction and choosing the root arbitrarily. However, computing the reachability is also a #*P*-hard for most general graphs [BP83], with exact polynomial-time algorithms known acyclic graphs [PB83] or with a small number of cycles [Hag91].

In [GJ19], reliability is approximated with a FPRAS for reachability, that uses the cluster-popping algorithm introduced by Gorodezky and Pak in [GP14], to sample root-connected subgraphs with probability proportional to their weights (depending on the operational/failure probability of each edge) for a sequence of contractions of the bi-directed graph, similar to the deletion-contraction technique of the factorization method. The efficiency of the approximation scheme in [GJ19] is highly influenced by the cluster-popping algorithm, which belongs to the partial rejection sampling framework [GJL19], an approach to sampling from a product distribution by gradually eliminating "bad" events. For this algorithm, "bad" events have been identified as clusters - subset of vertices not including the root and without any out-going arc. Although the cluster-popping has exponential complexity in general, the algorithm runs in polynomial-time for bi-directed graph [GP14] (conjecture proven in [GJ19]). The polynomial bounds of the cluster-popping algorithm have been improved in [GH18], by finding an upper bound for the expected number of resampled variables on bi-directed graphs and providing an efficient implementation, the *cluster-popping with Tarjan's algorithm* (based on Tarjan's recursive algorithm to find strongly connected components [Tar72]). This approach results in a faster implementation of the all-terminal reliability approximation in [GJ19], from $O(mn^3)$ to $O(mn^2log(n))$ time complexity (with m number of edges and n number of nodes), assuming constant failure probability. The approximation scheme is described in Section 3.5.

So far, the work on the new reliability approximation algorithm has only been theoretical. Thus, in this project, we approached the practical aspects of the algorithm proposed in [GJ19], with the cluster-popping optimisation in [GH18], aiming to analyse the real running time of the sampling algorithm and compute the reliability of a certain family of graphs. We decided to work with square grid graphs (L_n x L_n lattice graphs). Apart from the reliability of lattice graphs of width up to 4 [CS03] and the reliability polynomials of square grids of size up to 5 x 5 presented in [SI98], we are not aware of other results on the reliability of square grid graphs.

1.2 Goals and achievements

Motivated by the introduction of a polynomial-time approximation scheme for the all-terminal network reliability, the project aimed to compute the reliability of square grid graphs. For the reliability approximation algorithm, we implemented 3 main modules: Contraction (Section 4.3), Sampling (Section 4.4) and Expansion (Section 4.5). One of the additional contributions of this project is presenting an iterative implementation of the cluster-popping algorithm, which was only recursively define in [GH18]. In addition, we described the Coupling method (Section 4.7) to generate the equivalent subgraph of a directed graph in its original undirected graph.

In the end, we managed to generate the reliability of grids of size up to 15 x 15 with a failure probability on edges of $p = 0.5$. In addition, we extended our results with the reliability of grids of size up to 14 x 14 with failure probabilities $p \in \{0.1, 0.25, 0.75, 0.9\}$. To improve the real running time, we parallelised the processes involved in the sampling (cluster-popping) algorithm. We run the reliability system on three platforms with different performance specifications, including the Google Cloud Platform, to maximise the data collected. We tested the accuracy of our system against the reliability of graphs with known polynomials and with the exact reliability of square grids of size up to 5 x 5.

We also looked at the logarithmic growth rate of the reliability values and noticed the convergence towards a small constant. Using the average number of connected expansions, we established upper bounds for the reliability and analysed the distance to these bounds. In addition, for $p = 0.5$, we used the computed reliability to determine an approximation of the number of spanning subgraphs of the grid, an *NP*-hard problem in itself.

Furthermore, apart from computing and analysing the reliability value, we defined the auxiliary goals presented below:

- Analysis of the sampling algorithm. The cluster-popping with Tarjan's algorithm [GH18] is the main ingredient of the efficient implementation of the approximation scheme. Since the algorithm has only been theoretically analysed before, we also focused on the its real running time.

- Analysis of the number of arcs resampled in the cluster-popping algorithm, which influences the running time of the sampling module and, consequently, of the approximation algorithm. Together with the number of arcs resampled, we also looked at the number of minimal clusters found during the generation of root-connected subgraphs by the sampling module.

The results for these two goals were generated from the contractions (with the edge-deletion technique) of each grid involved in the approximation scheme, of size up to 15 x 15, with failure probabilities in $\{0.1, 0.25, 0.5, 0.75, 0.9\}$. In addition, we obtained the relevant values from sampling from grid graphs of size up to 100 x 100, using the same range of failure probability.

1.3 Structure of the report

This reports follows the development of the project, with extensive focus on the implementation of the individual modules involved in the reliability algorithm. For completeness, the next chapter provides an overview of the work and results of the MInf Project - Part I. Introductory notions on graph theory and examples of graphs with known reliability polynomials are then provided in Chapter 3. Chapter 3 also presents a detailed description of the reliability approximation algorithm. Chapter 4 explains the decisions on programming language and libraries used and discusses the implementation of each module of the reliability algorithm. Chapter 5 describes the environment that system was run on to generate the results presented in Chapter 6. In the end, Chapter 7 summarises the work undertaken in this project.

Chapter 2

MInf Project Part I - Overview

In the first part of the MInf project (available at [Tan19]), our focus was devoted to analysing the distribution of the roots of *graph polynomials* from the *b-matchings* of *regular graphs*, aiming to acknowledge the existence of a zero-free region containing the $[0,1]$ interval.

Graph polynomials (a type of graph invariants) are functions that depend only on the abstract structure of graphs and not on their representations, mapping graphs to polynomials. The polynomials are associated with the *signature function* of a counting problem, like the b-matching constraint which counts the size of subsets of edges such that every vertex has at most b edges of the subset incident with it. A symmetric function $f : \{0,1\}^n \to \{0,1\}$ with d arguments is associated with a signature $f = [f_0, f_1, f_2.., f_d]$, where $f_i = f(x)$ if $|x| = i$ [GLLZ]. The signature function f for a b-matching is $f = [1,1,..,1,0,0,..,0]$, with $b+1$ 1s. For $i \leq b$, $f_i = 1$ because the constraint permits a vertex v to be matched to at most i edges. Otherwise, $f_i = 0$.

We applied the constraint to (n,d) regular graph, in which each of the n vertices has the same number d of neighbours. Three examples of regular graphs are provided in Figures 2.1, 2.2 and 2.3.

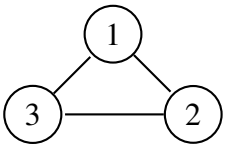

Figure 2.1: 3-2 regular

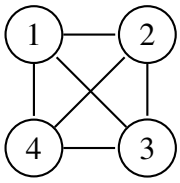

Figure 2.2: 4-3 regular

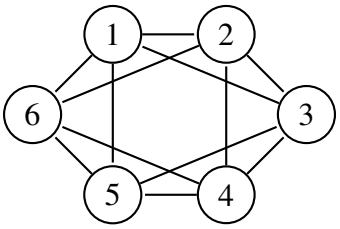

Figure 2.3: 6-4 regular

The graph polynomials are known as special cases of Holant Problems [CGW16], partition functions on graphs, represented as a sum of products, where edges are variables and vertices are constraint functions. Holant is a generalization of *counting* constraint satisfaction problems [CLX09], in which variables must satisfy a number of constraints. The graph polynomial evaluated at 1 expresses a Holant problem. In [GLLZ], the Holant of a graph $G = (V,E)$ with $\pi : V \to F$ an assignment from the set of

vertices V to a set of functions F and $f_v = \pi(v)$ a constraint function $\{0,1\}^{deg(v)} \to \mathbb{C}$ associated with the vertex v is defined as:

$$Z(G,\pi) = \sum_{\sigma \in \{0,1\}^{|E|}} \prod_{v \in V} f_v(\sigma|_{E(v)}),$$

where $E(v)$ is the set of adjacent edges of v, $\sigma|_{E(v)}$is the restriction of σ on $E(v)$ and $deg(v) = |E(v)|$ is the degree of vertex v.

The polynomial of a graph $G = (V,E)$ with signature f is:

$$P_G(x) = \sum_{i=0}^{|E|} Z_i x^i,$$

where

$$Z_i = \sum_{\sigma \in \{0,1\}^{|E|} \text{ and } |\sigma|=i} \prod_{v \in V} f_{|\sigma_{E(v)}|},$$

E(v) is the set of edges adjacent to vertex v and $|\sigma_{E(v)}|$ counts the number of 1s of the substring of σ restricted on $E(v)$.

To illustrate the relation between the graph polynomial and the Holant expression on a short example, consider the 3-2 regular graph in Figure 2.4. Let $V = \{1,2,3\}$ be the set of vertices and $f : \{0,1\}^2 \to \{0,1\}$ a symmetric binary function such that:

$$\pi(v) = f, \text{ for every } v \in V.$$

As f is symmetric,

$$f(01) = f(10).$$

If $E(v) = (e_i, e_j)$ is the set of edges of vertex v, with $i \neq j$ and $i,j \in \{0,1,2\}$, and $\sigma = \sigma_0\sigma_1\sigma_2 \in \{0,1\}^3$, then:

$$\sigma|_{E(v)} = \sigma_i\sigma_j \in \{0,1\}^2.$$

The Holant value is:

$$Z(G,\pi) = \sum_{\sigma \in \{0,1\}^3} f(\sigma|_{E(0)}) f(\sigma|_{E(1)}) f(\sigma|_{E(2)}) \Rightarrow$$

$$Z(G,\pi) = f(000|_{E(0)}) f(000|_{E(1)}) f(000|_{E(1)}) + \sum_{\sigma \in \{001,010,100\}} f(\sigma|_{E(0)}) f(\sigma|_{E(1)}) f(\sigma|_{E(2)}) +$$

$$+ \sum_{\sigma \in \{011,101,110\}} f(\sigma|_{E(0)}) f(\sigma|_{E(1)}) f(\sigma|_{E(2)}) + f(111|_{E(0)}) f(111|_{E(1)}) f(111|_{E(1)}) \Rightarrow$$

$$Z(G,\pi) = f(00)^3 + 3f(00)^2 f(01) + 3f(01)^2 f(11) + f(11)^3.$$

For a 2-matching with $f = [1,1,0]$, $f(00) = f_0 = 1$, $f(01) = f_1 = 1$, $f(11) = f_2 = 0$. Thus, the Holant value is:

$$Z(G,\pi) = 1^3 + 3 \cdot 1^2 \cdot 1 + 3 \cdot 1^2 \cdot 0 + 0^3 = 4.$$

Using the same graph and signature function, we now compute its graph polynomial. We need to find the value of the Z_i coefficients, for $i \in \{0,1,2,3\}$.

The vertices of the graph are $\{1,2,3\}$ with the following set of edges:

$$E(1) = \{e_0, e_2\},$$
$$E(2) = \{e_0, e_1\},$$
$$E(3) = \{e_1, e_2\}.$$
$$\sigma \in \{000, 001, 010,$$
$$100, 011, 101, 110, 111\}.$$

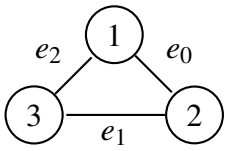

Figure 2.4: 4-3 regular

$$i = 0, \ \sigma \in \{000\}, \ Z_0 = f_0 * f_0 * f_0 \Rightarrow Z_0 = 1.$$
$$i = 1, \ \sigma \in \{001, 010, 100\}, \ Z_1 = f_0 * f_1 * f_1 + f_1 * f_0 * f_1 + f_1 * f_1 * f_0 \Rightarrow Z_1 = 3.$$
$$i = 2, \ \sigma \in \{011, 101, 110\}, \ Z_2 = f_2 * f_1 * f_1 + f_1 * f_2 * f_1 + f_1 * f_1 * f_2 \Rightarrow Z_2 = 0.$$
$$i = 3, \ \sigma \in \{111\}, \ Z_3 = f_2 * f_2 * f_2 \Rightarrow Z_3 = 0.$$

The graph polynomial of the 3-2 regular graph with a 2-matching constraint is:

$$P_G(x) = 1 + 3x.$$

Indeed, $P_G(1) = 4$, the Holant value.

The project was motivated by the work towards finding efficient approximation algorithms for solving hard constraint satisfaction problems, with no known efficient (polynomial-time) algorithm. For problems that are #P-hard, the associated Holant is, in general, #P-hard to compute as well, where the complexity class denoted by #P was introduced by Valiant in [Val79] for expressing the computational complexity of computing the number of solutions of a counting problem associated with an NP search problem. Papers such as [GLLZ] and [PR17] present polynomial-time algorithms that approximate Boolean Holant problems for a various constraints, such as matchings, even subgraphs and edge covers. The existence of these approximation algorithms depends on the absence of zeros of graph polynomials in a certain disk centred at the origin [PR17]. If the graph polynomial is zero-free in a strip containing $[0,1]$, then, by [GLLZ], a series of transformations can be applied to obtain the necessary polynomial that is zero-free in an origin-centred disk, implying the existence of an approximation algorithm for the Holant problem it represents. Hence, we decided to analyse the distribution of roots of b-matchings graph polynomials of regular graphs and decide if they suggest the existence of the corresponding approximation algorithm.

There was no knowledge of any previous experimental studies to generate the polynomials and analyse their roots. In the project, comprehensive focus was devoted to the algorithms to generate regular graphs and the algorithms to compute graph polynomials. One of the main contributions of the projects was a recursive algorithm for generating graph polynomials. The roots of polynomials of regular graphs of up to 20 vertices of degree at most 11 have been generated for up to 8-matchings and empirical

observations have been made on the pattern of the distribution of the roots. In order to achieve the goal of analysing the roots of polynomials, the following major steps were completed:

- **Generate random regular graphs**: 4 different approaches, based on Bollobás pairing model [Bol01] and Steger and Wormald's faster refinements [SW99] were implemented and compared to decide which approach is the most efficient. In the end, the polynomials of 295 non-isomorphic graphs have been generated and analysed.
- **Generate global polynomials**: 4 algorithms were implemented and compared, including a contraction of multivariate polynomials based on Kronecker substitution [Pan94]. In the end, we found a more efficient approach with a recursive relation on the coefficients of the polynomial, which was used to generate a total of 800 different polynomials.

Analysis of the generated global polynomials confirmed the existence of a zero-free region, by comparing their real arguments with the bounds suggested by Lebowitz, Pittel, Ruelle and Speer in [LPRS16] (such as the θ angle in Figure 2.5), proving the existence of deterministic polynomial-time approximation algorithms for these Holant problems. We found examples of polynomials with complex roots in the right-half plane, disproving Lebowitz, Pittel, Ruelle and Speer's assumption that the roots lie in the left plane only. We discovered as well non-isomorphic (n,d) regular graphs having the same associated global polynomials and polynomials with roots outside the unit circle. We also noticed that the degrees of the polynomial comply with a general expression. In Figure 2.5, we provide an example of the distribution of roots for 5-matchings. Figure 2.6 presents the distribution for the same graph, with different constraints.

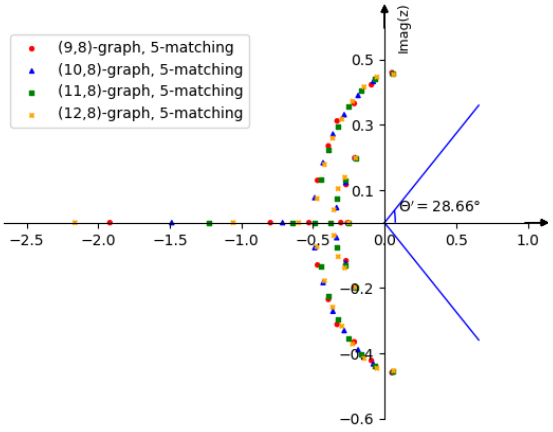

Figure 2.5: Roots of global polynomials for the same (d,b) configuration

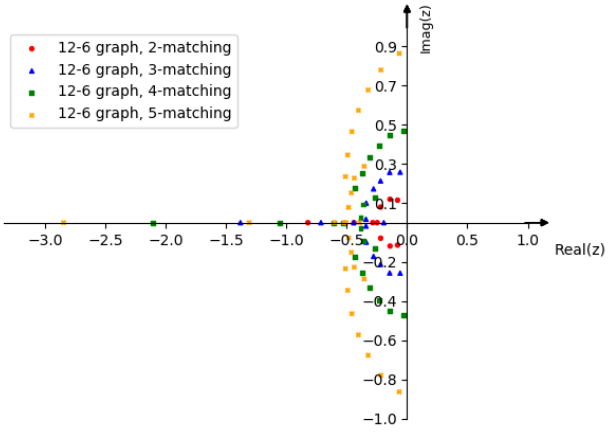

Figure 2.6: Roots of global polynomials for the same graph, increasing the constraint

Chapter 3

MInf Project Part II - Background

3.1 Introductory graph theory

Graphs represent a widely researched topic of high importance in many areas, due to their ability to model relationships between objects. In technology, in particular, graphs are applied to a variety of tasks, such as resource allocation in Operating Systems [MCOD06] or nodes communication in Computer Networks [Kur05]. This section presents some definitions related to the graph data structure known in Computer Science.

Definition 3.1.1. *A **graph** G is a pair (V, E), where V is a finite set and E is a binary relation on V. The set V is called the vertex set of G, and its elements are called **vertices** (singular: vertex). The set E is called the edge set of G, and its elements are called **edges*** [RS02].

In this paper, we refer to both undirected and directed graphs. Hence, it is necessary to clearly differentiate between the two notions.

Definition 3.1.2. *In an **undirected graph** G = (V, E), the edge set E consists of unordered pairs of vertices. That is, an edge is a set $\{u, v\}$, where $u, v \in V$ and $u \neq v$* [RS02]. An undirected graph is a graph in which edges have no orientation.

Definition 3.1.3. *A **directed graph** (or **digraph**) G=(V, E) is a graph in which edges have orientations.* An arrow *(u,v)* is considered to be a directed edge from *u* to *v*.

Definition 3.1.4. *If (u, v) is an edge in a graph G = (V, E), then the vertex v is **adjacent** to vertex u and u and v are **neighbours*** [RS02]. When the graph is undirected, the adjacency relation is symmetric.

Note that the terms *vertex* and *node* can be used interchangeably. By convention, the pair notation *(u,v)* is used for an edge, rather than the set notation $\{u, v\}$, and *(u,v)* and *(v,u)* are considered to be the same edge in the case of undirected graphs. Moreover, the term *arc* is usually applied when referring to the edges of a directed graph. Lastly, to describe the size of the two sets V and E, the notations $|V| = n$ and $|E| = m$ have been generally adopted.

3.2 Network reliability

The term *network* is often associated with the academic fields of telecommunication networks [LGW03], computer networks [Kur05] or social networks [Wel01]. It is strongly connected with the study of graph theory, considering distinct elements or actors represented by nodes and the connections between them as links (or edges), sometimes with certain properties. An area of particular research interest is the *network reliability*.

Definition 3.2.1. *The **reliability** $Z_{rel}(G,p)$ of an undirected graph $G=(V,E)$ is the probability that G is connected, given that each edge e is independently operational with probability $1-p_e$ and fails with probability p_e [BC92], where $p=\{p_e\}_{e\in E}$.*

Before we give the expression of the network reliability polynomial, we should firstly understand what it means for a graph to be connected.

Definition 3.2.2. *A **path** of length k from a vertex u to a vertex u' in a graph G = (V, E) is a sequence $< v_0, v_1, v_2, ..., v_k >$ of vertices such that u = v_0, u' = v_k and $(v_{i-1}, v_i) \in E$ for i = 1, 2,...,k* [RS02].

Definition 3.2.3. *An pair of vertices $\{x, y\}$ of a graph is called **connected (operational)** if there exists a path that leads from x to y.*

Definition 3.2.4. *An undirected graph is **connected (operational)** if every pair of vertices is connected by a path.*

3.2.1 Different expressions for the reliability polynomial

3.2.1.1 Reliability with subsets of connected edges

Assuming that edges' failure probability is p for each edge, then the reliability polynomial is obtained summing over the probability of all subsets of connected edges:

$$Z_{rel}(G,p) = \sum_{S\subseteq E,(V,S)\ connected} (1-p)^{|S|} p^{|E|-|S|},$$

where E is the set of edges of the graph G.

Other expressions have been adopted for the reliability polynomial, with coefficients depending on different attributes of the graphs.

3.2.1.2 Reliability with the number of path-sets

Definition 3.2.5. *For a graph $G = (V,E)$, a **path-set** is a subset $E' \subseteq E$ of edges, such that $G' = (V,E')$ is an operational graph.*

Since for the all-terminal reliability we are interested in the graph's connectivity, a path set is any connected spanning subgraph of G.

Definition 3.2.6. *A **subgraph** of a graph $G = (V,E)$ is a graph $G' = (V',E')$ with $V' \subseteq V$ and $E' \subseteq E$.*

Definition 3.2.7. *A **spanning** subgraph is a subgraph that contains all the vertices of the original graph.*

If we denote with N_i the number of path-sets with i edges, the probability of obtaining a set of i operational edges is $(1-p)^i p^{m-i}$, with p edges' failure probability. Hence, the reliability polynomial is:

$$Z_{rel}(G,p) = \sum_{i=0}^{m} N_i (1-p)^i p^{m-i} \quad [\text{PR18}].$$

3.2.1.3 Reliability with the number of cut-sets

Definition 3.2.8. *For a graph $G = (V,E)$, a **cut-set** is a subset $E' \subseteq E$ of edges such that $G' = (V,E-E')$ is not an operational graph.*

If we denote with C_i the number of cut-sets with i edges and $|E| = m$, the reliability polynomial is:

$$Z_{rel}(G,p) = 1 - \sum_{i=0}^{m} C_i p^i (1-p)^{m-i} \quad [\text{PR18}].$$

3.2.1.4 Reliability with the Tutte polynomial

The Tutte polynomial is a graph polynomial in two variables with an important role in graph theory, giving information about how the graph is connected.

Definition 3.2.9. *For a graph $G = (V,E)$, with k the number of connected components of G, $k(E')$ the number of connected components of the subgraph on (V,E'), with $E' \subseteq E$, and $n(E')$ the number of vertices induced by E', the **Tutte polynomial** is:*

$$T(G,x,y) = \sum_{E' \subseteq E} (x-1)^{k(E')-k} (y-1)^{k(E')+n(E')-|V|} \quad [\text{Jae88}].$$

The reliability polynomial is a well-known application of the Tutte polynomial. For a graph $G = (V,E)$, with $|V| = n$, $|E| = m$ and p failure probability, the reliability polynomial is:

$$Z_{rel}(G,p) = (1-p)^{n-1} p^{m-n+1} T\left(G, 1, \frac{1}{1-p}\right) \quad [\text{EMM11}].$$

3.2.2 General properties of the reliability polynomial

- If edges fail with $p = 0$ (all edges are perfectly operational) and the graph is connected, then $Z_{rel}(G,0) = 1$.
- If edges fail with $p = 1$ (no edges are operational), then $Z_{rel}(G,1) = 0$.
- For $p \in (0,1)$, $Z_{rel}(G,p) \in (0,1)$.
- If $p = 0.5$,

$$Z_{rel}(G,0.5) = \sum_{S \subseteq E,(V,S)\ connected} 0.5^{|S|}(1-0.5)^{|E|-|S|} =$$

$$= \sum_{S \subseteq E,(V,S)\ connected} 0.5^{|S|}0.5^{|E|-|S|} = \sum_{S \subseteq E,(V,S)\ connected} 0.5^{|E|} = \sum_{S \subseteq E,(V,S)\ connected} \frac{1}{2^{|E|}}.$$

  Thus, if K is the total number of connected spanning subgraphs of $G = (V,E)$, then $Z_{rel}(G,0.5) = K \cdot \frac{1}{2^{|E|}}$. Similarly, using the reliability expression with the number of cut-sets, $Z_{rel}(G,0.5) = 1 - C \cdot \frac{1}{2^{|E|}}$, where C is the total number of cut-sets of G.
- It is easy to see $Z_{rel}(G,0.5)$ as a function on the number of edges of the graph with constant vertices is monotonically decreasing with the increase in the number of edges.

The observations for $p = 0.5$ allow us to introduce a useful application of the reliability. With $p = 0.5$, if we can approximate Z_{rel}, it is easy to compute $K = 2^{|E|}Z_{rel}$ and $C = 2^{|E|}(1 + Z_{rel})$ as approximations for other well-known #P-hard problems: the total number of connected spanning subgraphs and the total number of cut-sets.

3.2.3 Reliability growth rate and logarithmic reliability

In this project, we were interested in two reliability-related values:

- The behaviour of the network reliability with the increase in the size of the graph and its convergence. For example, for most graphs with known polynomials (with some presented in Section 3.2.5), it is easy to prove that

$$\lim_{n \to \infty} Z_{rel}(G_n, p) = 0,$$

  where n denotes the vertex size of the graph and p the failure probability of edges.
- The behaviour of the logarithmic reliability and its convergence. For a graph $G = (V,E)$, with $|V| = n$, we define the logarithmic reliability for an edges' failure probability p with the following relation:

$$L_{rel}(G_n, p) = \frac{1}{n} \log_2 Z_{rel}(G_n, p).$$

An interesting topic is its convergence to a constant c, with the increase in the size of the graph:

$$c = \lim_{n\to\infty} L_{rel}(G_n, p) = \lim_{n\to\infty} \frac{1}{n} \log_2 Z_{rel}(G_n, p).$$

For $p = 0.5$, an observation can be made:

$$c = \lim_{n\to\infty} L_{rel}(G_n, p) = \lim_{n\to\infty} \frac{1}{n} \log_2 \frac{K}{2^{|E|}} = \lim_{n\to\infty} \frac{|E| + \log_2 K}{n}.$$

When $|E|$ is a function of n, we can define a constant $c' = \frac{|E|}{n}$ and hence:

$$c'' = c - c' = \lim_{n\to\infty} \frac{\log_2 K}{n}.$$

For large enough n, the last relation allows us to approximate K, the total number of connected spanning subgraphs as $K \to 2^{nc''}$..

3.2.4 Reliability of square grids

For the experiments presented in this paper, we decided to compute and analyse the reliability growth rate and logarithmic reliability of *square grid graphs*.

Definition 3.2.10. *A **square grid graph** (or an **n x n rectangular grid graph**) is the graph whose vertices correspond to the points in the plane with integer coordinates, with both x-coordinates and y-coordinates being in the range 1,..., n, and two vertices connected by an edge whenever the corresponding points are at distance 1. In other words, it is a unit distance graph for the described point set [Wei].*

Observation 3.2.1. An n x n rectangular grid graph has n^2 nodes.

Observation 3.2.2. The number of edges of an undirected square grid graph with n x n nodes is

$$|E| = 2n(n-1).$$

The first observation follows trivially from the definition of the square grid. The second observation can be easily proven. If we picture the grid on the 2-dimensional space with vertical and horizontal edges, it is easy to see that there are n vertical sets of nodes, each consisting of n nodes and $n-1$ edges. This gives $n(n-1)$ vertical edges. Similarly, we have $n(n-1)$ horizontal edges. Hence, the total is $n(n-1)+n(n-1) = 2n(n-1)$ edges.

For simplicity, the nodes of the grid graphs used in this papers have been notated with integers from 1 to n^2.

In the rest of the paper, the short notation $n-grid$ refers to an n x n grid graph. Three examples of square graphs are provided in Figures 3.1, 3.2 and 3.3.

- The first graph is a 2 x 2 grid, with 4 nodes and 4 edges.
- The second graph is a 3 x 3 grid, with 9 nodes and 12 edges.
- The third graph is a 4 x 4 grid, with 16 nodes and 24 edges.

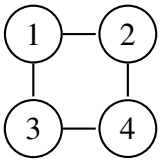

Figure 3.1: 2 x 2 grid

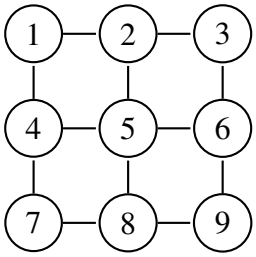

Figure 3.2: 3 x 3 grid

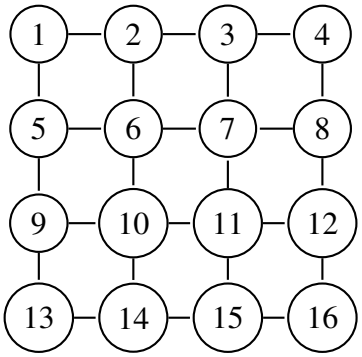

Figure 3.3: 4 x 4 grid

The reliability polynomials for square grid graphs have been computed in [SI98] for $n \leq 5$. For convenience, we provide them in Appendix A. We have no knowledge of other results on reliability of square grid graphs.

3.2.5 Known reliability polynomials for some family graphs

The study on network reliability resulted in exact expressions for the polynomial of some graphs. For testing the reliability system implemented in this paper, we used the known polynomials of the following graphs:

- The path graph, a non-square grid of width 1, with n nodes and $n-1$ edges, as in Figure 3.4 for $n = 4$.
- The pan graph, with n main nodes and 1 auxiliary node and n edges, as in Figure 3.5 for $n = 4$.
- The ladder graph, a non-square grid of width 2, with $2n$ nodes and $2(n-1)+n = 3n-2$ edges, as in Figure 3.6 for $n = 4$.
- The cycle graph, with n edges and n nodes, each connected with 2 other nodes, as in Figure 3.7 for $n = 4$.
- The star graph, with 1 central node connected with $n-1$ nodes, using a total of $n-1$ edges, as in Figure 3.8 for $n = 4$.

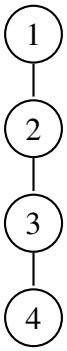

Figure 3.4: 4-Path

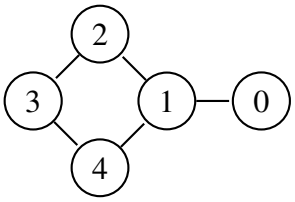

Figure 3.5: 4-Pan

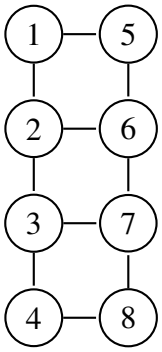

Figure 3.6: 4-Ladder

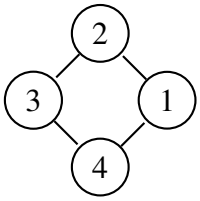

Figure 3.7: 4-Cycle

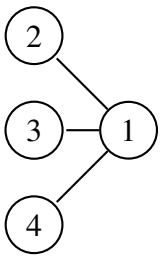

Figure 3.8: 4-Star

Their polynomials are presented in Table 3.1. Notice that the path and star graphs have the same polynomial. Moreover, $Z_{rel}(Pan_n, p) = p \cdot Z_{rel}(Cycle_n, p)$. Due to these equivalences, we used only the growth rate of the reliability of path, pan and star graphs as comparisons with the behaviour of the reliability of grids. The polynomials of three other type of graphs are presented in [Wei08].

graph	**polynomial**
path	$Z_{rel} = (1-p)^{n-1}$
pan	$Z_{rel} = (1-p)^n[1+(n-1)p]$
ladder	$Z_{rel} = \frac{(p-1)^{2n-1}}{2^n\sqrt{p(9p+2)+1}}[(3p-\sqrt{p(9p+2)+1}+1)^n-(3p+\sqrt{p(9p+2)+1}+1)^n]$
star	$Z_{rel} = (1-p)^{n-1}$
cycle	$Z_{rel} = (1-p)^{n-1}[1+(n-1)p]$

Table 3.1: Known reliability polynomials

In addition, a recurrence relation for the reliability of complete graphs has been established in [Tit99]. A complete graph has n nodes, with an edge between any two nodes, hence a total of $\frac{n(n-1)}{2}$ nodes. An example is provided in Figure 3.9, for $n = 4$. Since it is the only graph for which we have an exact relation that has the number of edges a square function of the number of nodes, as similar to the grid graphs, we also used the complete graph to test our system and compare the results. For a complete graph with n nodes, the reliability relation is defined below. In Appendix B, we provided the polynomial for $n \leq 6$, as computed in [SI98].

$$Z_{rel}(Complete_n, p) = 1 - \sum_{k=1}^{n-1} \binom{n-1}{k-1} p^{k(n-k)} Z_{rel}(Complete_k, p),$$

$$Z_{rel}(Complete_1, p) = 1.$$

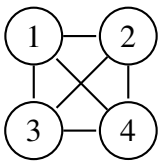

Figure 3.9: 4-Complete

3.3 Network reachability

Despite research in the area, there is still no efficient algorithm to find all connected spanning subgraphs of a graph, therefore, directly computing the reliability through the expressions previously presented cannot be done in polynomial time. Hence, we aim to approximate the reliability value through the *reachability* measure. Before we give the relation between reliability and reachability in Section 3.4, we define and explain an estimation for the reachability of a directed graph.

Definition 3.3.1. *A directed graph is **root-connected** if there exists a path from any vertex (excluding the root) to the root, where the root is a previously selected node of the graph.*

Definition 3.3.2. *The **reachability** $Z_{reach}(G,r,p)$ of directed graph G = (V, A) with a root r is the probability that, if each arc e fails with probability p_e independently, the remaining graph is still root-connected in r, where $p = \{p_e\}_{e\in A}$* [GP14].

If the failure probability of an arc e if p_e, we can define the **weight** of a subgraph S of G as:

$$wt(S) = \prod_{e\in S}(1-p_e)\prod_{e\notin S} p_e.$$

Then, the reachability of the directed graph can be found using the weights of the following root-connected subgraphs, with the following expression:

$$Z_{reach}(G,r,p) = \sum_{\substack{S\subseteq A \\ (V,S)root-connected}} wt(S),$$

where $p = \{p_e\}_{e\in A}$ if the vector of failure probabilities [GJ19].

However, it can be noticed that the above expression needs to find all root-connected subgraphs of the graph as well, making the problem again generally #P-hard [GP14]. However, as presented in [GJ19] and detailed in Section 3.5, there is a polynomial-time approximation scheme for the network reachability.

3.4 From network reachability to network reliability

By [Bal80], if G' is the digraph obtained from G, with the same failure probability in either direction, then:

$$Z_{rel}(G,p) = Z_{reach}(G',r,p),$$

where p is the set of failure probability for each arc, r is the root chosen arbitrarily and the value $Z_{reach}(G,r,p)$ can be approximated using the algorithm described in Section 3.5.

Definition 3.4.1. *The **digraph** G' of an n-grid G is obtained by adding to G' two arcs (u,v) and (v,u) for each edge $\{u,v\}$ of G.*

An example of a digraph obtained from a 3 x 3 grid is provided in Figure 3.11.

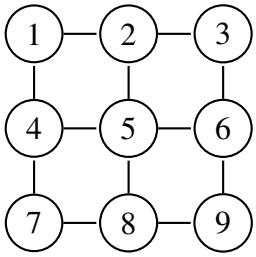

Figure 3.10: Undirected grid

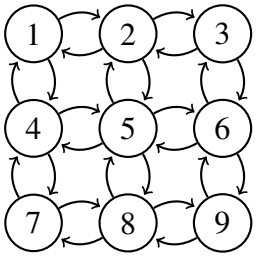

Figure 3.11: Digraph grid

3.5 Network reachability approximation

The reachability is a hard problem, but a sequence of steps can be applied to estimate the value in polynomial time, using an approximate counting algorithm presented in [GJ19] and based on Gorodezky and Pak's [GP14] observation on sampling root-connected subgraphs for bi-directed graphs. The result is generated such that it is in the $(1 \pm \varepsilon)$ interval deviation from the real value, based on confidence given as an input to the system.

The steps of the algorithm are presented below, adapted to our experiments, where we have assumed that edges fail independently with the same probability p.

1. Starting with an initial bi-directed graph $G = (V, A)$ and a randomly selected root $r \in V$, construct a sequence of graphs $G_0, .., G_{n-1}$, where $n = |V|$ and $G_0 = G$. Each $G_i = (V_i, A_i)$, $i > 0$, is obtained by choosing two arbitrary adjacent vertices $u, v \in V_{i-1}$, removing all arcs between them and adding a new contracted vertex. The approach keeps parallel arcs (multiple arcs), but not self-loops.

   Definition 3.5.1. *If (u, v) is an edge in a directed graph G=(V, E), then (u, v) is* ***incident*** *from or leaves vertex u and is incident to or enters vertex v* [RS02].

   Definition 3.5.2. ***Multiple edges*** *are two or more edges that are incident to the same two vertices.*

   Definition 3.5.3. *A* ***loop*** *is an edge that connects a vertex to it*

   The root r is updated if one of the selected nodes was the root. The last contraction will results in G_{n-1} having no arcs and the root as the only vertex. For a graph with $|V| = n$ vertices, then the algorithm does $n-1$ iterations. These steps are applied by the *Contraction Module*, explained in detail in Section 4.3.

2. For each contracted graph with $i > 0$, choose a number s root-connected samples drawn from the distribution $\pi_{Gi}(\cdot)$.

   Definition 3.5.4. *For the bi-directed graph G = (V, A),* $\pi_G(\cdot)$ *is the distribution resulting from choosing each arc e∈ A independently with probability 1-*p_e*, and conditioning on the resulting graph being root-connected..*

   Hence, $\pi_G(\cdot)$ represents the collection of all root-connected subgraphs, and the probability of each subgraph S is proportional to its weight $wt(S)$. The sam-

pling algorithm is described in Section 4.4. Its implementation is based on [GH18], which introduced the polynomial cluster-popping with Tarjan's algorithm to sample root-connected subgraph from the $\pi_G(\cdot)$ distribution.

3. For each sample, expand back the corresponding contracted nodes and add the arcs between them independently with probability $1-p$. This step is realised by the *Expansion Module*, described in Section 4.5.

4. For each expansion of each sample S_j of iteration i, define a random variable $R_{i,j}$, such that $R_{i,j} = 1$ if the expansion is root-connected. By [GJ19], for one iteration,
$$\mathbb{E}[R_i] = \frac{Z_{reach}(G_{i-1}, r, p)}{Z_{reach}(G_i, r, p)}.$$
However, we can estimate ER_i by the empirical mean R'_i of the s independent samples for each contracted graph:
$$R'_i = \sum_{j=1}^{s} \frac{R_{i,j}}{s}.$$

5. In the end,
$$Z_{reach}(G, r, p) = \prod_{i=1}^{n-1} \mathbb{E}[R'i]$$
can be estimated using the empirical means R'_i:
$$Z'_{reach}(G, r, p) = \prod_{i=1}^{n-1} R'_i = \prod_{i=1}^{n-1} \frac{\sum_{j=1}^{s} R_{i,j}}{s}.$$

In [GJ19], the suggested number of samples for a graph with $|V| = n$ vertices and failure probability p, such that the approximation is in the $(1 \pm \varepsilon)$ interval from the exact value, with $0 < \varepsilon < 1$ should be:
$$s = \lceil 5(1-p)^{-2}(n-1)\varepsilon^{-2} \rceil.$$

By, [GP14] this number s of samples gives a confidence interval of 75%, or a probability that the results is outside the $(1 \pm \varepsilon)$ interval is $\delta = 0.25$:
$$Pr(|Z_{reach}(G, r, p) - Z'_{reach}(G, r, p)| > \varepsilon Z_{reach}(G, r, p)) \leq \frac{1}{4}.$$

However, we were interested in having a system with a better accuracy, by increasing the constant in the number of samples. According to the analysis in [GP14], $\frac{1}{\delta}$ scales roughly linearly with the number of samples. Thus, for $\delta = 0.05$, we would need $\frac{0.25}{0.05}s = 5s$ samples. Due to the fact that the time complexity of the entire system highly depends on the performance of the Sampling Module, it is in our advantage to tighten the constant as much as possible.

To calculate the number of samples for $\delta = 0.25$, Gorodezky and Pak [GP14] used the following inequality:

$$e^{x/5} \leq 1 + \frac{1}{4},$$

for $x \in [0,1]$. To find the constant in the number of samples expression for $\delta = 0.05$, we are looking for the lowest possible positive integer c such that:

$$e^{x/c} \leq 1 + \frac{1}{20},$$

for $x \in [0,1]$. An analysis of this inequality, provided in Appendix C, reaches the conclusion that the minimum constant is $c = 21$. Since the an n x n grid has, in fact, n^2 vertices, the numbers of samples we need in Step 2 of the approximation algorithm for the reachability (and implicitly reliability) of a square grid, with a failure probability on edges p and a confidence interval of 95% for an error rate ε is:

$$s = \lceil 21(1-p)^{-2}(n^2-1)\varepsilon^{-2} \rceil.$$

In Appendix D, we provide a table with the exact values for s for different (n,p) configurations for estimating the reliability of a square grid. For example, if $n = 10$ and $p = 0,5$, then $s = 831600$.

3.5.1 Logarithmic reliability using the reachability approximations

We conclude the background chapter with an observation on the logarithmic reachability and its convergence, using the value of approximated value of reachability (and implicitly reliability). For a general graph, with n vertices, we know that:

$$Z_{rel}(G,r,p) = Z'_{reach}(G',r,p) = \prod_{i=1}^{n-1} R'_i.$$

By the inequality of geometric and arithmetic mean,

$$\prod_{i=1}^{n-1} R'_i \leq (\frac{\sum_{i=1}^{n-1} R'_i}{n-1})^{n-1} \Rightarrow \log_2 \prod_{i=1}^{n-1} R'_i \leq \log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1})^{n-1} \Rightarrow$$

$$\log_2 \prod_{i=1}^{n-1} R'_i \leq (n-1)\log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1}) \Rightarrow \frac{1}{n}\log_2 \prod_{i=1}^{n-1} R'_i \leq \frac{n-1}{n}\log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1}) \Rightarrow$$

$$\frac{1}{n}\log_2 Z_{rel}(G,r,p) \leq \frac{n-1}{n}\log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1}) \Rightarrow L_{rel}(G,r,p) \leq \frac{n-1}{n}\log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1}) \Rightarrow$$

$$\lim_{n\to\infty} L_{rel}(G,r,p) \leq \lim_{n\to\infty} \frac{n-1}{n}\log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1}) \Rightarrow \lim_{n\to\infty} L_{rel}(G,r,p) \leq \lim_{n\to\infty} \log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1}).$$

Thus, $(\frac{\sum_{i=1}^{n-1} R'_i}{n-1})^{n-1}$ is an upper bound for $Z_{rel}(G,r,p)$ and $\lim_{n\to\infty} \log_2 (\frac{\sum_{i=1}^{n-1} R'_i}{n-1})$ is an upper bound for $\lim_{n\to\infty} L_{rel}(G,r,p)$. These properties have been experimentally observed in the results of our experiments, presented in Section 6.

Chapter 4

Algorithms and Implementation Decisions

This chapter presents the tools, libraries, algorithms and implementation decisions involved in creating the system to run the experiments described in Chapter 6. Estimating the reliability was done by implementing the Contraction (Section 4.3), Sampling (Section 4.4) and Expansion (Section 4.5) Modules, following suggestions presented in [GJ19]. Of particular interest are the approaches taken to sample root-connected subgraphs, detailing in Section 4.4.3 an iterative solution. Each module description is extended with an example on a grid graph. This chapter highlights as well in Section 4.6.4 the attempts to improve the real running time of the system by generating the sequences of samples with a parallel implementation.

4.1 Programming language

Similarly to the implementation of the MInf Project Part I, the programming language used was Python 3.7.0 ([LLCa]). Python is a free and open-source programming language, whose portability enabled us to easily build and run the programs both on the Dice machines provided by the School of Informatics and on a personal device. Due to its object-oriented features, the code could be organised in data objects, especially for a better representation of the graph structures described in Section 4.2.

One Python library used in the implementation of the algorithms to sample and expand graphs, described in Section 4.4 and 4.5, is **random** [LLCc], a module that implements pseudo-random number generators. In particular, *random.choice* returns a random element from the non-empty sequence. In addition, we used Python's **time** library [LLCd] to save the running time of the Sampling and Reliability Modules, for efficiency comparison of different approaches and results analysis. Lastly, we used **matplotlib** [HJ07], a Python plotting library, for generating the plots presented in Results (Section 6).

4.2 Graph data structure

For the MInf Project Part I, the **NetworkX** Python package [HSSC08] was exclusively used to represent, work with and manipulate graphs. The package, *"for the creation, manipulation and study of the structure, dynamics and functions of complex networks"* [HSSC08], provides data structures for graphs and many standard graph algorithms, which were extensively helpful for the required operations. Although a *MultiDiGraph* object for directed graphs with multiple edges is available through the NetworkX package, the need to efficiently access the multiplicity of an arc or store its original source if involved in a contraction made it necessary to construct our own data structure. Hence, we also introduced a custom **MultiDiGraph** object.

4.2.1 Graph representation

The implemented **MultiDiGraph** object is in many aspects similar to the well-known implementations of graphs. It provides access to the list of *nodes* and *edges* (or *arcs*), stored in a set and, respectively, a dictionary, for efficient look-up operations. The arcs dictionary maps each directed arc with its multiplicity. In addition, we considered it essential to keep a *neighbours* dictionary associating each vertex to the nodes it connects to.

Moreover, a *previous_sources* is the dictionary that allows the retrieval of previous vertices of an arc and its respective multiplicity. An arc has previous sources if the graph is the result of a contraction. Take, for example, the graph in Figure 4.1, whose contraction on the nodes 1 and 2 results in the graph in Figure 4.2. For such examples, it is useful to record the original sources of the the two arcs. Hence, the corresponding MultiDiGraph would retrieve that the two arcs ("$1-2$" $\rightarrow 3$ (multiplicity 2) come from exactly one arc $1 \rightarrow 3$ and one arc $1 \rightarrow 2$. Lastly, the MultiDiGraph class provides access to standard operations such as adding and removing arcs.

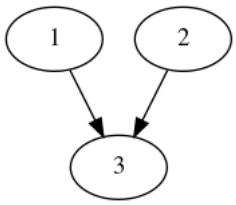

Figure 4.1: Initial graph G

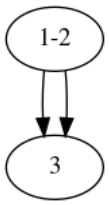

Figure 4.2: Contraction of G

4.2.2 Graph generation

To construct the required undirected and their bi-directed graphs of type *MultiDiGraph*, we introduced a **GraphsGenerator** helper class. The generator is used to return the required grids. A grid of order n has n^2 nodes, enumerated from 1 to n^2.

The helper class can also construct path, pan, star, cycle, ladder and complete graphs (exemplified in Section 3.2.5), whose known reliability polynomial was used for testing the system, as described in Section 4.6.2.

4.2.3 Graph manipulation

In addition to the methods to access elements of a graph provided by the *MultiDiGraph* structure, we added a class **GraphHelper** to support the following operations:

- Given a directed graph and a root node, check whether the graph is root-connected, by establishing the existence of a path from each other node to the root, using the breath-first search traversal algorithm. This method is used by the Reliability Module, the establish the value of the random variables associated to each contraction step.
- Check whether two graphs are equal: they have the same nodes and arcs, and eventually the same mapping from the current arcs to their previous sources. This method used for testing purposes, by comparing the output of the recursive and iterative sampling algorithms, as presented in Section 4.4.4.

4.2.4 Graph visualisation

NetworkX provides the function to plot directed graphs, but it does not allow visualisation of multiple arcs. Hence, we found the **graphviz** [gra] package useful for this task. This package facilitates the creation, rendering and drawing of graphs using the DOT language [DOT] in Python. The package enables the user to save the source code to a file and view its plot. While testing and using the system, all the figures of graphs were generated with *graphviz*. The graphs in these report, however, were drawn using LaTeX's **tikz** library [Hac17]. Although graphviz can display multiple directed arcs, it does not draw the arcs in a grid-like structure, making it difficult for the eye to visualise the examples.

4.3 Contraction Module

In Section 3.5, we explained that reachability (and the equivalent reliability) is approximated by sampling root-connected subgraphs from a sequence of contractions. The approximation counting algorithm does a number of iterations (contractions), starting with a digraph, until the last contraction result has only one node, namely the root. Hence, the first step of each iteration is computing the contraction of the current graph. Algorithm 4.3.1 summarises the steps taken in the implementation of the *Contraction Module*. The Algorithm was implemented following the high-level description of the faster algorithm for network reliability estimation in [GJ19]. This section continues with its complexity analysis and provides an example at the end.

4.3.1 Algorithm overview

The algorithm takes a digraph, named G for convenience, represented using the custom *MultiDiGraph* object, and returns its random contraction.

The algorithm starts by choosing two arbitrary adjacent vertices u and v. A new node is created for the contracted graph using the two randomly selected vertices. If one of u or v was the input graph's root, then the root must be updated with the new node. Next,

it is important to decide what happens to each arc in G. Since contraction does not preserve self-loops, the arcs between u and v, in any direction, are discarded. Otherwise, an arc can be in one of the three cases:

1. An arc leaves from u or v to another vertex b, ($b \neq u$ and $b \neq v$). For all such arcs, new arcs from the new node to b are added to the new graph, with the same multiplicity as the original arc. The original $(source, destination)$ pair is saved for the new arc (in *previous_sources*) with the corresponding multiplicity.
2. An arc leaves from a vertex a ($a \neq u$ and $a \neq v$) to u or v. For all such arcs, new arcs from a to the new node are added to the new graph with the same multiplicity as the original arc. The original $(source, destination)$ pair is saved for the new arc (in *previous_sources*) with the corresponding multiplicity.
3. An arc leaves from a vertex a ($a \neq u$ and $a \neq v$) to another vertex b, ($b \neq u$ and $b \neq v$). In this case, for all such arcs (considering the multiplicity of the original arc), the arcs from a to b are added to the new graph, with the same multiplicity.

```
def contract_graph(G, root):
  contraction = MultiDiGraph()
  pick adjacent nodes u, v from G.arcs;
  create new_node = u + '-' + v
  add new_node to contraction.nodes

  if u == root or v == root:
    root = new_node

  for arc (a,b) in G.arcs:
    # If arc would not give a self-loop a self-loop.
    if (a,b) is not (u,v) and (a,b) is not (v,u):
      m = multiplicity of (a,b) in G.arcs
      if a == u or a == v:
        add arc (new_node, b) with mult. m to contraction.arcs
        save (a,b) with mult. m for new arc in contraction.previous
      else if b == u or b == v:
        add arc (b, new_node) with mult. m to contraction.arcs
        save (a,b) with mult. m for new arc in contraction.previous
      else:
        add (a,b) with mult. m to contraction.arcs

  return contraction, root, (u, v)
```

Algorithm 4.3.1: Graph contraction

The correctness of the system depends on meeting two important requirements:

- Adding the arcs with the correct multiplicity. This ensures the sampling algorithm (Section 4.4) considers all possible arcs between two nodes. Through the many iterations of the complete system, the new arcs multiplicity might increase. For example, if a graph has 1 arc 1→2 and 2 arcs 1→3, with contracted nodes 2 and 3, the resulting graph has 3 arcs 1→"2-3".
- Storing the previous $(source, destination)$ pairs, with the correct multiplicity, for each new arc. Both the sampling and expansion (Section 4.5) algorithms needs

to differentiate between the arcs to and from the same vertices. For example, assume again that a graph has 1 arc 1→2 and 2 arcs 1→3. If the contracted nodes are 2 and 3, then the resulting graph has 3 arcs 1→"2-3", but they represent different instances, having different origins.

The algorithm returns the contracted graph, the updated root and the two vertices picked for contraction, to be used by the other modules.

4.3.2 Complexity analysis

Accessing any information from the input graph G, such as multiplicity, takes constant time, due to the use of dictionary data structures in the graph's implementation. Each update to the new graph, such as addition of arcs, takes also constant time. Hence, the running time of the contraction algorithm is mostly influenced by the need to iterate through all arcs of the input $G = (V,A)$. A square grid of order n has initially $2n(n-1)$ edges. Hence, its digraph has $4n(n-1)$ arcs, each with multiplicity 1, and each contraction of an iteration keeps almost all arcs, excluding loops. If A is the set of arcs for the n x n grid, then the expected running time is $O(|A|) = O(4n(n-1)) = O(n^2)$.

4.3.3 Example

We provide a complete example of the sequence $G_0, G_1, ..G_{n^2-1}$ of contracted graphs, starting with a $3x3$ digraph, with 9 vertices and 24 arcs, where G_0 is the first input graph. The root is initially the node 4. The algorithm does 8 iterations.

1. Iteration 1. The graph in Figure 4.4 is the result of contracting vertices 4 and 5. The root comes from the vertices resulting from the contraction.
2. Iteration 2. The graph in Figure 4.5 is the result of contracting vertices 2 and 3.
3. Iteration 3. The graph in Figure 4.6 is the result of contracting vertices 1 and "4-5". Notice the existence of multiple arcs and the update of the root.
4. Iteration 4. The graph in Figure 4.7 is the result of contracting vertices "3-2" and "1-4-5". The root is updated again.
5. Iteration 5. The graph in Figure 4.8 is the result of contracting vertices 8 and 7. The root does not change.
6. Iteration 6. The graph in Fig.4.9 is the result of contracting vertices 9 and "8-7".
7. Iteration 7. The graph in Figure 4.10 is the result of contracting vertices 6 and "8-7-9". Notice the existence of multiple arcs and the update of the root.
8. Iteration 8. The graph in Figure 4.11 is the result of contracting vertices "3-2-1-4-5" and "6-8-7-9". The root is updated again. The resulting graph has only the root-vertex and no arcs.

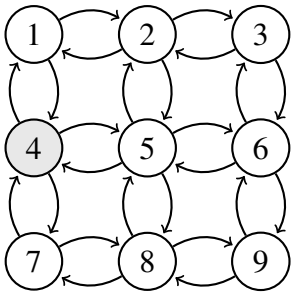

Figure 4.3: G_0

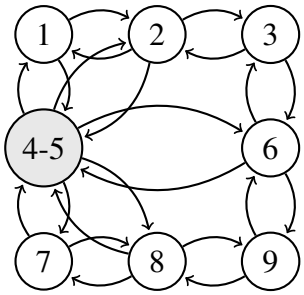

Figure 4.4: G_1

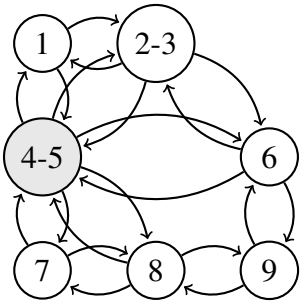

Figure 4.5: G_2

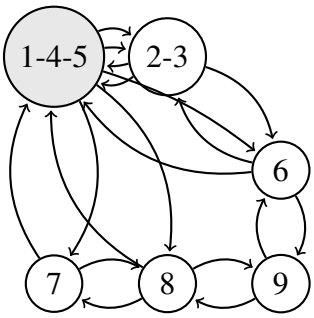

Figure 4.6: G_3

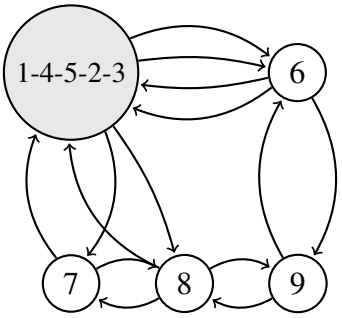

Figure 4.7: G_4

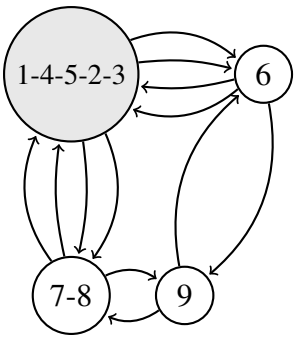

Figure 4.8: G_5

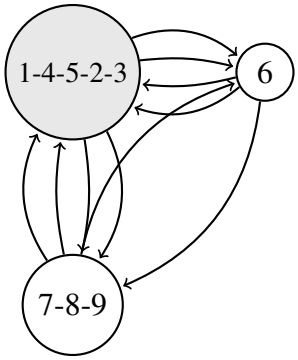

Figure 4.9: G_6

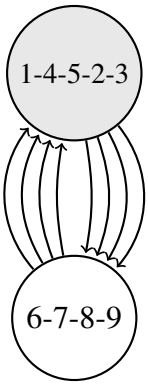

Figure 4.10: G_7

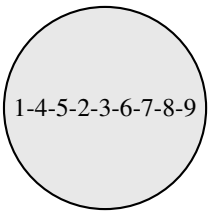

Figure 4.11: G_8

4.4 Sampling

The approximation approach presented in this paper is based on generating a sequence of independent root-connected subgraphs of the current contracted graph G_i of the iteration i, with each subgraph drawn from the $\pi_{Gi}(\cdot)$ distribution (defined in Section 3.5). [GH18] introduces the *Cluster-popping with Tarjan's algorithm*, a recursive depth-first strategy for retrieving root-connected subgraphs drawn from a distribution. The algorithm, under the partial rejection sampling framework, randomises all arcs independently and repeatedly resamples arcs from minimal clusters until no clusters exist (subset of vertices not including the root and without any out-going arc).

However, recursive algorithms often face the risk of reaching the stack limit of the system they are running on. Indeed, the implemented recursive sampling algorithm, reaches a stack limit of 1000 calls for an n as small as 38 (with 1444 vertices). While the stack limit can be increased, we considered it beneficial to address this issue with an iterative implementation, guaranteeing a more robust Reliability Module. Hence, in the following sections, after presenting the sampling set-up common to both approaches, we detail and compare the recursive and iterative implementations.

4.4.1 Sampling set-up

Before we introduce the recursive or iterative steps in the sampling algorithm, we present in Algorithm 4.4.1a the set-up which is common to both implementations. The set-up consists in assigning the (initially empty) sample a subset of arcs independently drawn from the input graph G with probability $1-p$, where p is the failure probability (lines 3-7). For correctness, the addition must keep track of the multiplicity and, if existing, source of the arc in the un-contracted graph. These checks are highlighted in Algorithm 4.4.1b. Hence, the sample is now a subgraph drawn from the $\pi_G(\cdot)$ distribution, not necessarily root-connected.

```
1  def set_up(G, root, p):
2      # Construct subgraph of G with arcs independently drawn.
3      sample = MultiDiGraph()
4      for node in G.nodes:
5          add node to sample.nodes
6      for arc in G.arcs:
7          add_arc(arc, G, sample,p)
8
9      # Set-up algorithm dictionaries.
10     index_dict = dict() with None initially for each node
11     root_dict = dict() with None initially for each node
12
13     index_dict[root] = 1
14     root_dict[root] = 1
15     index = 2
16     stack = [root]
```

Algorithm 4.4.1a: Sampling set-up

```
1  def add_arc(arc, from_G, to_G, p):
2      if arc has previous_sources:
3              previous_sources = from_G.get_previous_sources(arc)
4              for source in previous_sources:
5                  m = multiplicity of source
6                  for i from 1 to m:
7                      if should_accept_with_prob(1 - p):
8                          add arc to to_G.arcs
9                          save source of arc to to_G.previous_sources
10     else:
11         m = multiplicity of arc in from_G.arcs
12         for i from 1 to m:
13             if should_accept_with_prob(1 - p):
14                 add arc to to_G.arcs
```

Algorithm 4.4.1b: Drawing arcs from a graph

Following Tarjan's algorithm for finding the strongly connected components [Tar72], the graph traversal implementation of the sampling algorithm keeps track of the visited node, assigning to each v an unique integer *index* id (in *index_dict*), corresponding to a visiting level. The index id numbers the nodes in the increasing order in which they are discovered. It also maintains *root* id (in *root_dict*), representing the smallest index of any node known to be reachable from v in a depth-first search traversal (cluster's root). These values are initialised in the set-up part, lines 10-19. Lastly, nodes are added on a stack in the order in which they are visited. Initially, the stack contains only the graph's root, the first node to be traversed.

In terms of the complexity of the set-up, it is straightforward to see that the nodes and arcs iterations imply the algorithm runs in at most $O(|A|+|V|) = O(n^2)$ time, for an n x n grid.

4.4.2 Recursive Sampling

We continue with the main part of the recursive sampling algorithm described in [GH18], adapting the arcs resampling strategy to consider the multiplicity and previous sources of the arcs as well. We emphasize that these conditions are essential for correctly determining the root-connectivity in the expanded graph.

The algorithm, presented in Algorithm 4.4.2, is based on finding minimal clusters and, if found, resampling arcs, until all nodes have been traversed and no clusters are present. The sample computed in the set-up step, the source graph (called G), the probability failure p and the index_dict, root_dict and the stack variable must be known.

The algorithm updates the index and root dictionaries, used to determine the existence of minimal clusters, by traversing nodes in a depth-first-search style. Given the current node v, lines 11-16 visit its never-seen neighbours and update v's index and root ids.

The existence check of clusters with vertices that cannot reach the graph's root is done in line 18. Intuitively, all vertices w that have been added to the stack after the current v (call this subset of the stack W), by the Dynamic-DFS calls (in line 13 and their

resulting recursive calls), must be reached by v. hence $index\_dict[v] \leq index\_dict[w]$ and $root\_dict[v] \leq root\_dict[w]$. If $root\_dict[v] = index\_dict[v]$, then $root\_dict[w] \geq index\_dict[v]$ (otherwise $root\_dict[v] < index\_dict[v]$, which contradicts the fact that we considered them equal). Thus, $index\_dict[v] \leq root\_dict[w] \leq index\_dict[w]$. Assume that there is a $w' \in W$ that cannot reach v. If more such nodes exists, pick the w' with the minimum index. Because $index\_dict[v] \leq root\_dict[w'] \leq index\_dict[w']$, then v can reach the root r of w' (since it was indexed after v). But w was the node with minimum index that cannot reach v, hence r must reach v. By transitivity, w reaches v, which is a contradiction. Hence, all $w \in W$ reach v and $W \cup \{v\}$ forms a strongly connected subset of nodes. Furthermore, this cluster is minimal, since no arc can go out of it, with the index of its destination lower than v's index, without contradicting $index\_dict[v] \leq root\_dict[w] \leq index\_dict[w]$. Lines 20-29 remove the nodes in $W \cup \{v\}$ of the cluster from the visited stack and resample all the arcs leaving from these nodes.

When the algorithm returns, there is no other cluster and all nodes can reach the graph's root, hence the sample is root-connected.

```
1  def get_sample():
2      while there exists a node v with index None:
3          Dynamic_DFS(v)
4      return sample
5
6  def Dynamic_DFS(v):
7      index_dict[v] = index
8      root_dict[v] = index_dict[v]
9      index = index + 1
10     stack.append(v)
11     for w in sample.get_neighbours(v):
12         if index_dict[w] is undefinded:
13             Dynamic_DFS(w)
14             root_dict[v] = min(root_dict[v], root_dict[w])
15         else:
16             root_dict[v] = min(root_dict[v], index_dict[w])
17     # If minimal cluster is found.
18     if root_dict[v] == index_dict[v]:
19         w = None
20         while w != v:
21             w = stack.pop()
22             index_dict[w] = None
23             root_dict[w] = None
24             index = index - 1
25             remove all arcs leaving from w from sample
26             # Resampling arcs.
27             for neighbour in G.get_neighbours(w):
28                 arc = (w, neighbour)
29                 add_arc(arc, G, sample, p)
30         Dynamic_DFS(v)
```

Algorithm 4.4.2: Recursive Sampling

4.4.3 Iterative Sampling

The iterative approach is very similar to the recursive implementation. The logic behind the sampling does not change and minimal cluster check and resampling operations are the same for both implementations.

```
1  def get_sample():
2      v = node with index None
3      queue = []
4      queue.append((v, 0, None))
5      while queue is not empty
6          (v, step, w) = queue.pop()
7          if step == 0:
8              if w != None and index_dict[v] != None:
9                  continue
10             index_dict[v] = index
11             root_dict[v] = index_dict[v]
12             index = index + 1
13             stack.append(v)
14             for w in sample.get_neighbours(v):
15                 if index_dict[w] == None:
16                     if nothing was added to the queue in this loop:
17                         queue.append((v, 2, w))
18                     else:
19                         queue.append((v, 1, w))
20                     queue.append((w, 0, v))
21                 else:
22                     root_dict[v] = min(root_dict[v], index_dict[w])
23             if the queue was updated:
24                 jump to the loop in line 5
25         if step == 1 or step == 2:
26             root_dict[v] = min(root_dict[v], root_dict[w])
27         if step == 0 or step == 2:
28             # If minimal cluster is found.
29             if root_dict[v] == index_dict[v]:
30                 w = None
31                 while w != v:
32                     w = stack.pop()
33                     index_dict[w] = None
34                     root_dict[w] = None
35                     index = index - 1
36                     remove all arcs leaving from w from sample
37                     # Resampling arcs.
38                     for neighbour in G.get_neighbours(w):
39                         arc = (w, neighbour)
40                         add_arc(arc, G, sample, p)
41                 queue.append((v, 0, None))
42             if queue is empty:
43                 if exists node v with index None:
44                     queue.append((v, None))
45         return sample
```

Algorithm 4.4.3: Iterative Sampling

The algorithm keeps a list (queue) of vertices to be traversed. The challenge comes from correctly assigning the order in which vertices are traversed. In particular, we ensured that the Dynamic-DFS calls in lines 3, 13 and 30 in Algorithm 4.4.2 have a corresponding element in the traversal queue, in the right order:

- Line 3 of Algorithm 4.4.2 corresponds to the queue update in lines 4 and 46 of Algorithm 4.4.3.
- Line 13 of Algorithm 4.4.2 corresponds to the queue updates in lines 18-22 of Algorithm 4.4.3.
- Line 30 of Algorithm 4.4.2 corresponds to the queue update in line 43 of Algorithm 4.4.3.

An entry in the queue is a tuple $(v, step, w)$, where v is the vertex to be traversed and w one of its neighbours. The value of $step$ plays a decisive role in deciding the correct order of operations:

- $step = 0$ corresponds to either entering the Dynamic-DFS or checking for minimal clustering with resampling after visiting the neighbours of the current node. Note that by the time the vertex v of an entry in the queue is considered, its value might have already been updated by previous calls, which implies the need of the check in line 8 of Algorithm 4.4.3.
- $step = 1$ corresponds to the update in line 14 of Algorithm 4.4.2. after visiting the neighbours of a node.
- $step = 2$ signals that all neighbours of the vertex have been visited and the only operations left are the update in line 13 of Algorithm 4.4.2 and the the cluster check.

4.4.4 Sampling Module testing

The first test to apply to the output of both algorithms is to check for root-connectivity. Once we ensured the output of each module is root-connected, we wanted to check the equivalence between the two models. Comparison of the outputs of both approaches, however, was not a straightforward task since the algorithms are randomised. To check that the two implementations produce the same output, we used a testing strategy that forces deterministic sampling. Each arc in the input graph was attached a precomputed list of random variables (probabilities). For drawing arcs with probability p, instead of generating a random probability every time, the algorithms picked the same variable from the arc's computed list of probabilities to decide whether it is added to the sample or not. If the algorithms visit the nodes in the same order, the resulting samples have the same arcs with equal multiplicities. Thus, for testing the iterative sampling, we modified the independent arcs sampling of Algorithm 4.4.1b using the described deterministic strategy and asserted that the vertices are visited in the same order and the sample outputs are equal graphs (using the graph equality comparison method implemented in *GraphHelper*, described in Section 4.2.3).

In addition, by running the algorithms with the same input, we empirically established that there are not any differences in their actual running time. In support of this claim,

we conducted a two-tailed t-test on the real running time of the two implementations. The data was generated by averaging the time to sample 10 root-connected subgraphs for n x n grids, with $2 \leq n \leq 30$. The test value of 0.78, clearly greater then a p-value of 0.05, fails to reject the null hypothesis that the running times have statistically different (expected) average values. Appendix E provides the complete list of values used in this test.

4.4.5 Complexity analysis

The time complexity of both algorithms depends on the number of arcs resampled due to the occurrence of minimal clusters. Paper [GH18] proves that the expected running time of the recursive algorithm is $O(|A| + \frac{p|A||V|}{1-p})$. The resampling rules are the same for iterative algorithm, with identical orders of arcs resampling and identical final state. This claim implies that the time complexity of the iterative sampling is $O(|A| + \frac{p|A||V|}{1-p})$ as well. For an n x n grid, $|A| = 4n(n-1)$ and $|V| = n^2$, the complexity of a sampling from a complete graph becomes $O(4n(n-1) + \frac{4pn^3(n-1)}{1-p}) = O(n^4)$, disregarding the $\frac{p}{1-p}$ constant. In our implementation, the size of the graph decreases with each contraction, influencing the real running time. In Section 6.1, we present the real running time of the sampling and its decreasing rate when the graph contracts.

4.4.6 Example

Figure 4.13 and Figure 4.14 are two root-connected subgraphs of the graph in Figure 4.12, resulted in the fifth contraction step of an 3 x 3 grid, where each arc had probability $p = 0.5$ of failing. Notice that for each node in the sampled graph, there is a path to the root "3-2-1-4-5". For example, *Sample 1* has the path "$8-7$" $\rightarrow 9 \rightarrow 6 \rightarrow$ "$3-2-1-4-5$" and *Sample 2* has the path $6 \rightarrow 9 \rightarrow$ "$8-7$" $\rightarrow$ "$3-2-1-4-5$".

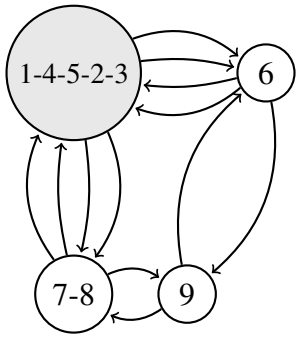

Figure 4.12: Sampling input

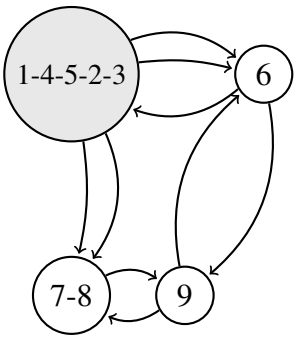

Figure 4.13: Sample 1

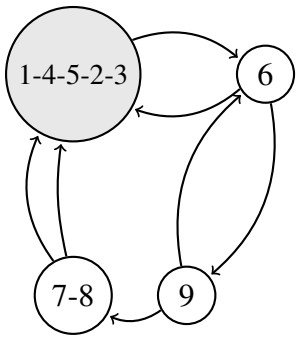

Figure 4.14: Sample 2

4.5 Expansion Module

For each root-connected subgraph of a contraction, the system needs to check for root-connectivity in the previous un-contracted graph, the input of the Contraction Module of the current iteration. We have called this step *expansion*, as it consists of identifying the contracted nodes and expanding them back, keeping the arcs returned from sampling. The algorithm is detailed in Algorithm 4.5.1 and its time complexity is analysed in Section 4.5.2.

4.5.1 Algorithm overview

The algorithm starts with the sampled graph S that needs expansion. It also gets as input the initial digraph G of the current iteration (the input of the contraction). The expanded graph output is a subgraph of G. In addition to the failure probability p, for convenience, the vertices $contracted\_u$ and $contracted\_v$ which have been selected for contraction in this iteration are provided.

```
def expand_graph(G, S, new_node, contracted_u, contracted,_v p):
    expansion = MultiDiGraph()
    for node in G.nodes:
        add node to expansion.nodes

    for arc (a,b) in S.arcs:
        if a != new_node and b != new_node:
            m = multiplicity of (a,b) in S.arcs
            add (a,b) with multiplicity m to expansion.arcs
        else:
            prev_sources = previous sources of (a,b) in S
            for (x,y) in prev_sources:
                m = mult. of (a,b) with source (x,y) in S.arcs
                add (x, y) with mult. m to expansion.arcs

    m = multiplicity of (u,v) in G.arcs
    for i from 1 to m:
        if should_accept_with_prob(1 - p):
            add arc (u,v) to expansion.arcs

    m = multiplicity of (v, u) in G.arcs
    for i from 1 to m:
        if should_accept_with_prob(1 - p):
            add arc (v,u) to expansion.arcs

    return expansion
```

Algorithm 4.5.1: Graph expansion

The expanded graph must have all the nodes of the graph G, although the output might not be a connected graph. Each arc from the sample S must be added to the output following one of the two rules:

1. The arc does not leave from or enter the contracted node. In this case, the sampling has preserved the arc from the initial graph G, with the same source and target vertices from G. The sampling might have reduced the multiplicity of the

arc, hence it is important to add it back to the expansion with the the multiplicity of the arc in the sample and *not* in the initial graph.

2. The arc (a,b) leaves from or enters the contracted vertex (resulting from the contraction of G on u and v), hence $a =$ "$u-v$" or $b =$ "$u-v$". This scenario highlights the importance of correctly storing the original *(source, target)* pairs during the contraction (Section 4.3) and sampling step (Section 4.4). Using these values, the algorithm can correctly associate the arcs in S with its original corresponding arcs in G, keeping the multiplicity of the sampling. Hence, the expanded graph will either have the arcs (a,u) or (a,v) or both (or respectively (u,b), (v,b)).

Lastly, the expansion module needs to add the arcs between the contracted vertices u and v with probability $1-p$. Given the failure probability p, then the acceptance probability is $1-p$. Therefore, we need to get existence and the multiplicity of these arcs from the original G.

4.5.2 Complexity analysis

The algorithm must ensure all nodes in $G=(V,A)$ are added to the expanded graph. Given a bi-directed grid with maximum n^2 nodes and the efficient use of dictionaries with amortised constant time, the first *for* loop takes at most $O(|V|) = 0(n^2)$ time. The next *for* loops deal with the addition of arcs from the sampling. In the worst case scenario, the sampling keeps all arcs in the contraction of G, with a worst case multiplicity of 1. Since addition takes (amortised) constant time and the contraction can have at most $0(|E|) = 0(4n(n-1)) = O(n^2)$ arcs, then adding these to the expanded graph takes at most $0(|E|) = 0(4n(n-1)) = 0(n^2)$ time. In total, the Expansion Module takes $0(|V|+|E|) = 0(n^2)$ time.

4.5.3 Example

We provide two examples for the Expansion Module, based on the contraction examples provided in Section 4.3.3.

- Example 1. The input graph G in given in Figure 4.15. Contraction chooses vertices 8 and 7, removing the arcs between them. As seen in Figure 4.16, sampling reduces the multiplicity of the rest of the arcs. The expanded graph in Figure 4.17 separated the contracted vertices 7 and 8. In this example, the module does not add any arcs between these two vertices. The arcs originating from or directed to the contracted vertex "8-7" are separated to the correct *(source, target)* pairs. For example, the arc 9→"8-7" becomes 9→8 (as it can be seen in the input graph of Figure 4.17, there were no arcs 9→7 originally in G). The expansion also suggests that the sampling has kept only the arcs between root (vertex "3-2-1-4-5") and 7 and none between the root and vertex 8. The rest of the arcs are preserved according to the sampling result.

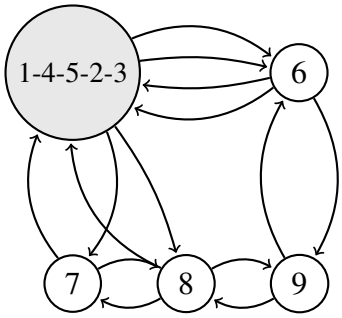

Figure 4.15: Iteration input example1

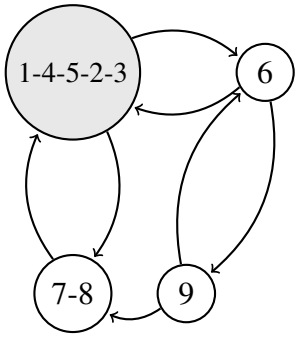

Figure 4.16: Sample example 1

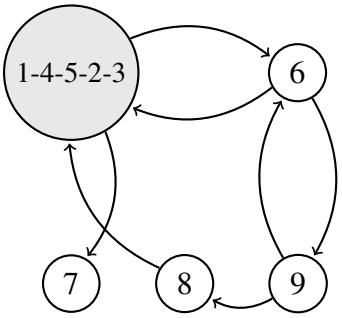

Figure 4.17: Expansion example 1

- Example 2. The input graph G in given in Figure 4.18. Contraction chooses vertices 6 and "8-7-9", removing the arcs between them. As seen in Figure 4.19, sampling reduces the multiplicity of the rest of the arcs. The expanded graph in Figure 4.20 separated the contracted vertices 6 and "8-7-9". In this example, the module adds the arc "8-7-9"→6 between them. The arcs originating from or directed to the contracted vertex are separated to the correct *(source, target)* pairs, according to the distribution decided by the sampling algorithm. For example, notice the the expanded graph of Figure 4.20 shows us that the 2 arcs "3-2-1-4-5"→ "6-8-7-9" in Figure 4.18 were a result of sampling 1 out of the 2 arcs "3-2-1-4-5"→6 and 1 out of the 2 arcs "3-2-1-4-5"→"8-7-9" in Figure 4.18.

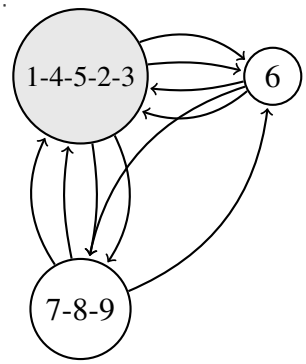

Figure 4.18: Iteration input example 2

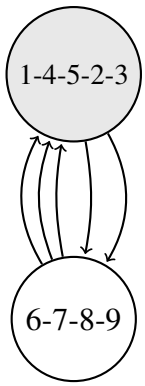

Figure 4.19: Sample example 2

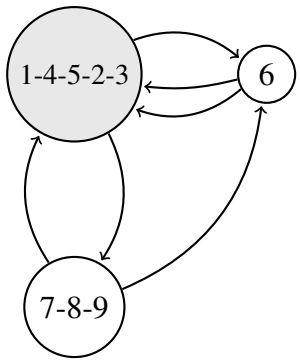

Figure 4.20: Expansion example 2

4.6 Reliability Module

In Section 3.5, we explained that the reliability of an n-grid is computed by approximating the reachability of its digraph. The approximate counting algorithm uses the product of the empirical mean of s random variables depending on the samples of contractions of the original graph. Therefore, we built a main system that connects all the individual modules explained in the previous sections.

4.6.1 Algorithm overview

In Algorithm 4.6.1 we enumerate the sequence of operations applied to compute the desired reliability given a failure probability p for each edge of the n x n grid graph. The algorithm constructs the square graph and its digraph using the *MultiDiGraph* object (Section 4.2.1) and the *GraphsGenerator* class (Section 4.2.2). It proceeds with $n^2 - 1$ iterations, each consisting of:

- Contraction phase, described in Section 4.3.
- Sampling, described in Section 4.4.
- Expansion phase, described in Section 4.5.

The number s of samples is computed using the expression in Section 3.5, given the desired error ε. The expanded graph of each iteration is checked for root-connectivity, given the initial root of the current iteration and the result is updated by following the equation in Section 3.5. At the end of each iteration, it is essential to ensure that the last contracted graph becomes the input for the next iteration. This update also keeps track of the corresponding root.

```
def get_reliability(n, p, eps):
    construct the n-grid graph G
    construct the bi-directd graph DG
    get number of samples s given n*n nodes, prob. p, error eps
    choose a random root r from DG.nodes
    initialize estimated_reachability = 1
    contracted_graph = first contraction of DG

    while contracted_graph is not the root:
        number_root_connected_samples = 0
        for i from 1 to s:
            get sample_graph of contracted_graph
            get extension_graph of sample_graph
            if(extension_graph is root-connected):
                number_root_connected_samples++
        estimated_reachability *= number_root_connected_samples / s
        contract contracted_graph again
    return estimated_reachability
```

Algorithm 4.6.1: Reliability Module

4.6.2 Reachability Module testing

We could not test the correctness of the implementation by comparing the results with the real reliability value of n-grids for $n > 4$, due to the difficulty in computing the exact value imposed by the exponential generation of subgraphs. For $n \in \{2,3,4\}$ and $p\{0.1,0.25,0.5,0.75,0.9\}$, we generated the reliability of the grids with a naive implementation and compared these values with the results of the our Reliability Module. These values were also compared with the polynomials found in [SI98] and presented in Appendix A. The naive approach generated all subgraphs of the undirected grid and followed the reliability expression in 3.2.1.1. All the results were in the (1 ± 0.1) interval of the real value. Table 4.1 provides an examples for $p = 0.5$.

n	exact Z_{rel}	approximated Z_{rel}
2	0.31250	0.31224
3	0.10523	0.10526
4	0.03309	0.03299

Table 4.1: Approximated and exact reliability for an n x n grid, $p = 0.5$

In addition, we extended *GraphsGenerator* (Section 4.2.2) with methods to generate 6 graphs whose reliability polynomial is already known: *path, pan, cycle, star, ladder and complete*. Their structure and reliability polynomials were discussed in Section 3.2.5. We applied the reliability estimations to these graphs for $2 \leq n \leq 8$ and $p = 0.5$ and concluded that our system output is not significantly different from the real value given by corresponding polynomial evaluation. We provided exact and approximated reliability values for this test in Appendix F.

4.6.3 Complexity analysis

According to [GJ19], the expected running time of the bi-directed reachability estimation algorithm is $O(\varepsilon^{-2}(1-p)^{-1}|A||V|^2 \log(|V|)\log(\frac{1}{1-p}))$ for an $(1 \pm \varepsilon)$ estimations.

With $|A| = 4n(n-1)$ and $|V| = n^2$, we have an algorithm that for an n x n grid runs in $O(\varepsilon^{-2}(1-p)^{-1}(4n(n-1))(n^2)^2 \log(n^2)\log(\frac{1}{1-p})) = O(8\varepsilon^{-2}(1-p)^{-1}n^6 \log(n)\log(\frac{1}{1-p}))$ time complexity.

In conclusion, we presented an algorithm that can estimate the *NP*-hard problem of reliability in polynomial time. Compared to the exponential naive implementation that generates $O(2^{2n(n-1)})$ subgraphs of the undirected grid, our algorithm is faster for $n \geq 5$, but slightly slower for $n \in \{2,3,4\}$ due to the sampling overhead. For example, for $p = 0.5$, the naive approach for $n = 3$ takes 2 seconds, compared to the 72 seconds of the approximation implementation. For $n = 4$ with $p = 0.5$, the naive algorithm takes around 4 minutes to complete, while the approximation algorithm runs for only 7.5 minutes. However, for $n = 5$, the naive approach did not finish running in 12 hours, while the approximation system takes around 25 minutes for $p = 0.5$.

4.6.4 Parallel Reliability

With a polynomial time complexity of order 6, the real running time of the reliability module grows fast as we increase the size of the grid. The complexity is highly influenced by the number of samples required for a contraction (provided in Appendix D). The number of samples itself could not be decreased, as we already picked the most optimal value for a 95% confidence level that the result lies in a (1 ± 0.1) accuracy interval (Section 3.5). However, root-connected subgraphs are independently generated. Hence, we tried to optimise the polynomial constant by updating our system to generate the s samples in parallel runs. Line 10 to 17 of the sequential Algorithm 4.6.1 were updated with a parallel version detailed in Algorithm 4.6.4, using Python's **multiprocessing** library [LLCb].

```
1  def sequential_sampling(contracted_graph, p, samples_per_process):
2      current_number_root_connected = 0
3
4      for i from 1 to samples_per_process:
5          get sample_graph of contracted_graph
6          get extension_graph of sample_graph
7          if(extension_graph is root_connected):
8              current_number_root_connected++
9
10     return current_number_root_connected
11
12 def get_reliability(n, p, samples_per_process):
13     ...
14     s = total_number_samples
15     while contracted_graph is not empty:
16         number_root_connected = 0
17         number_processes = s/samples_per_process
18         create multiprocessing pools on 4 CPUs for number_processes
       processes.
19         map pools with sequential_sampling function
20         wait for pools to finish
21         results = list of pools results
22         number_root_connected = sum elements of results
23         estimated_result *= number_root_connected / number_samples
24     ...
```

Algorithm 4.6.4: Parallel Reachability

For $p = 0.5$, for each grid, $\frac{s}{8400}$ processes were created and spread over 4 (for $n \in \{9,10,11,12\}$ or 8 (for $n \in \{13,14,15\}$) CPU cores. Each processes generated 8400 samples and their expansion, computing the root-connectivity Booleans. The results were collected at the end to generated the required random variables for estimating the reachability.

The choice of sequential computations for a process played an important role in dealing with the overhead resulted from the setup and scheduling of processes. Given the expression for s in 3.5, the value 8400 is the least common divisor of the number of samples for each n, for $p = 0.5$ and $\varepsilon = 0.1$. Similarly, for the rest of the probabilities, we assigned $s_p = \lceil 21(1-p)^{-2}\varepsilon^{-2} \rceil = \lceil 2100(1-p)^{-2} \rceil$ samples to $\frac{s}{s_p}$ processes, with $\varepsilon = 0.1$.

The efficiency of the parallel implementation was tested by comparing its running time for $2 \leq n \leq 7$ and $p = 0.5$ with the running time of the sequential algorithm. Table 4.2 suggests that the time (in seconds) has considerably decreased, by 44% to 80%, measured on 4 CPUs. This test also helps us assert the correctness of the implementation, by ensuring that the returned results for the two implementations are similar.

n	time_sequential (s)	time_parallel (s)
2	4.24	1.81
3	72.33	33.71
4	450.17	212.39
5	1465.84	812.65
6	15085.5983	3352.55
7	50151.9478	9059.7858

Table 4.2: Time in seconds to compute the reliability for an n x n grid

4.6.5 Example

For a better understanding of the sequence of steps explained in the previous section, we provide a complete example with one sample for each iteration.

Iteration 1.

- We start with the bi-directed graph in Figure 4.21, with the random root 2.
- The first contraction gives in the graph in Figure 4.22, with the nodes 1 and 3 chosen. Notice how the arcs between 1 and 3 in the initial graph have been removed. The root does not change.
- The sampling algorithm returns the graph in Figure 4.23, which is indeed root-connected for the root 2.
- The contracted nodes are expanded in Figure 4.24, keeping the arcs selected by the sampling algorithm. Both arcs between 1 and 3 have been added. The expanded graph is root-connected for the root 2. The approximated result after this iteration is 1.

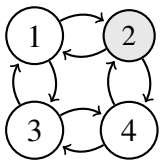

Figure 4.21: G_0

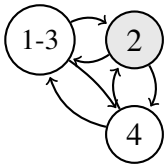

Figure 4.22: G_1

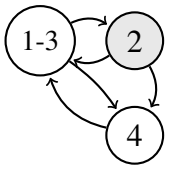

Figure 4.23: Sample 1

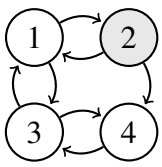

Figure 4.24: Expansion 1

Iteration 2.

- We start with the graph in Figure 4.25, with the root 2. Notice that for the second iteration, the input graph is exactly the contraction result of the first iteration.

- The second contraction gives is the graph in Figure 4.26, with the nodes 2 and 4 chosen. The arcs between 2 and 4 in the initial graph have been removed and the root updated. The examples highlights the existence of multiple arcs.
- The sampling algorithm returns the graph in Figure 4.27, which is indeed root-connected for the root 2.
- The contracted nodes are expanded in Figure 4.28, keeping the arcs selected by the sampling algorithm. No arcs between 2 and 4 have been added. The expanded graph is not root-connected for the root 2. The approximated results after this iteration is $1 \cdot 0 = 0$.

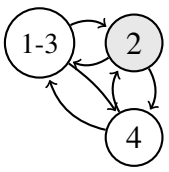

Figure 4.25: G_1

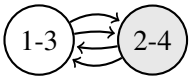

Figure 4.26: G_2

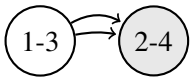

Figure 4.27: Sample 2

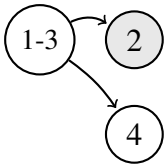

Figure 4.28: Expansion 2

Iteration 3.

- We start with the graph in Figure 4.29, with the root "4-2". Notice that for the third iteration, the input graph is exactly the contraction result of the second iteration.
- The third and last contraction gives is the graph in Figure 4.30, with the root as the only node.
- The sampling algorithm does not have any arcs to sample from. Hence, the sampled graph in Figure 4.31 is the same as the contracted graph.
- The contracted nodes are expanded in Figure 4.32, adding some of the arcs between the two contracted vertices 4-2 and "1-3". The expanded graph is root-connected for the "4-2" root. The approximated results after this iteration is $0 \cdot 1 = 0$.

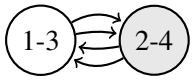

Figure 4.29: G_2

Figure 4.30: G_3

Figure 4.31: Sample 3

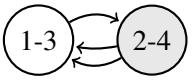

Figure 4.32: Expansion 3

4.7 Coupling Module

A question that arises from the algorithm we described is how to represent the equivalent of a directed subgraph in the initial undirected graph. In [GJ19], the coupling between reliability and bi-directed reachability approach is presented, explaining how to construct a directed graph given a subset of the arcs of an undirected graph and vice-versa. We are only interested in the directed to undirected direction, detailed in

the next section. While the Coupling Module is not essential for estimating the reliability, we considered it as a good extra feature of the complete system, mainly as a way to visualise the generated subgraphs.

4.7.1 Algorithm overview

The algorithm presented in Algorithm 4.7.1 explores all vertices that can reach the root in a breadth-first search order, with the difference that traversal does not go from a parent to a child node, but rather in the opposite direction. Arcs from the subgraph that belong to a directed path to the root are added to the undirected equivalent. The algorithm keeps a set of active and explored nodes. The active list corresponds to the nodes that have not been traversed yet, starting with the root. A vertex is added to the active list when there is an arc from it to the node that is currently explored. When the exploration of the current node is finished (all its incoming neighbours have been considered), the vertex is moved to the explored list. Since we construct a subgraph of the undirected graph, we are not interested in the multiplicity of an arc. If it belongs to a path to the root, it must be added only once. Similarly, if arcs in both directions exist for two nodes u and v ($u \to v$ and $v \to u$), it is clear that only one corresponding edge is added to the directed graph, due to the check in line 10. Once the arc $u \to v$ was added and v moved to the expanded list, v will not be considered as the incoming neighbour of u. No arc may be added if one of the vertices does not have a path to the root. If there is no path from a node to the root, than it is impossible to reach it during the graph traversal. Thus, the undirected graph is connected if and only if the directed graph is root-connected.

```
1  def get_coupling_graph(directed_graph, root):
2      active_nodes = stack()
3      active_nodes.push(root)
4      expanded_nodes = []
5      undirected_graph = new undirected graph
6      add the nodes of directed_graph to undirected_graph.nodes
7      while there are active nodes:
8          v = active_nodes.pop()
9          for each node u such that (u,v) is an arc:
10             if u not in expanded_nodes:
11                 add edge (u,v) to undirected_graph.edges
12                 if u not in active_nodes:
13                     active_nodes.add(u)
14         expanded_nodes.push(v)
15     return undirected_graph
```

Algorithm 4.7.2: Coupling between directed subgraph and undirected graph

4.7.2 Example

In the example we provide, the graph in Figure 4.38 is the equivalent of the expansion result in Figure 4.37 of 3 x 3 grid. Since the expanded graph is root-connected (with the node 4 as the root), the equivalent undirected subgraph is connected as well.

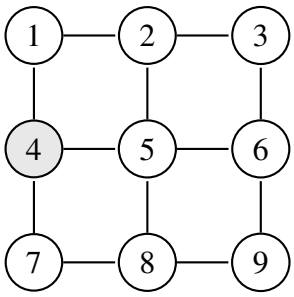

Figure 4.33: Undirected grid

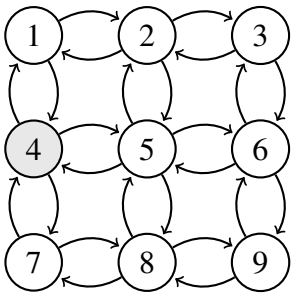

Figure 4.34: Bi-directed grid

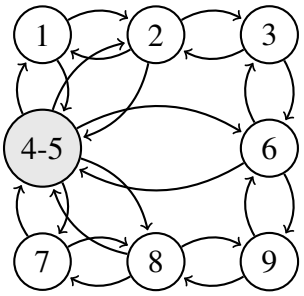

Figure 4.35: Contraction

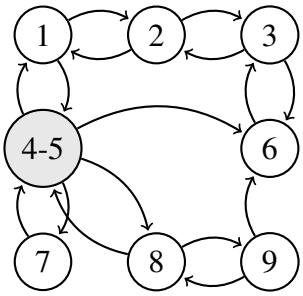

Figure 4.36: Root-connected subgraph

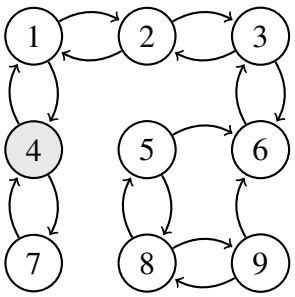

Figure 4.37: Expansion

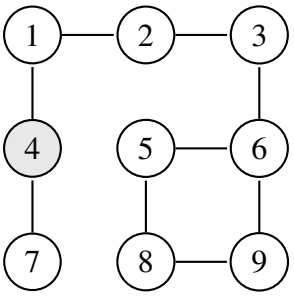

Figure 4.38: Coupling of undirected grid

Chapter 5

Experiments Running Environment

To generate the results presented in Section 6, we run the Reliability Module, sometimes referred to as *the system*, with different parameters:

- **The type of the graph**. We were mainly interested in the reliability of grid graphs, but we have also tested the system with path, pan, star, cycle ladder and complete graphs.
- **The size *n* of the graph**, with the number of nodes and edges a function of n.
- **The failure probability *p***. Although the experiments were aimed at $p = 0.5$, we also run the system for $p \in \{0.1, 0.25, 0.75, 0.9\}$ for some values of n, as described later in Section 6.

The three values, together with the choice between the Serial and the Parallel Reliability Module (discussed in Section 4.6.4), highly influenced the running time of the system. With the objective of computing as much data as possible, we also varied the platforms the system run on. This approach allowed us to generate different data at the same time, maximising the set of results. The platforms are presented below:

- **Personal device** with processor Intel Core i5-6200U CPU 2.30GHz x 4, used for developing the system and comparing the performance of the Parallel Reliability Module with the serial implementation. The reliability of grids of size $n \leq 8$ with probability $p = 0.5$ was generated using this platform. However, this resource was not sustainable for larger graphs, due to the increasing computational demands. The platform was also used for the number of arcs resampling of the Sampling Module on grids with size $n \leq 100$ and for the reliability of the six auxiliary graphs (path, pan, star, cycle, ladder and complete), for system testing. Results for $n = 11$ and $p \in \{0.1, 0.25\}$, $n \in \{9, 10\}$ and $p = 0.75$ and $n = 8$ and $p = 0.9$ were also generated using this platform with the parallel version of the system. Taking into account that this platform is used for various other tasks unrelated to the project, we decided to also use other environments that would allow continuous runs of the system.
- **student.compute servers of the School of Informatics**, with processor Intel(R) Xeon(R) CPU E5-2690 CPU 3.00GHz x 10. The School's clusters are available

for extensive computational jobs. They were used for the computing the reliability of grids with $9 \leq n \leq 15$ with $p = 0.5$, using the parallel implementation on 4 cores (for $n \in \{9,10,11,12\}$ and 8 cores (for $n \in \{13,14,15\}$). The running time of a process on these clusters depends on the scheduling time and number of requests. Unfortunately, the platform became a less reliable environment when its demand increased, with different computational jobs unrelated to this project, resulting in a substantial increase in the running time of the experiments. In particular, the process of generating the reliability for $n = 14$ was on the scheduling queue in the middle of the 15th contraction for 5 days, making no computational progress. These unexpected circumstances, together with the undeniable complexity of the system, affected the possibility of generating results for $n > 15$. Hence, we decided to rely on a different environment for $p \neq 0.5$.

- **Google Cloud Platform (GCP)**. Although not initially part of the goals of this project, the time resources allowed us to compute the reliability of grid graphs for different failure probabilities. However, we needed a reliable platform for continuously running the experiments. The Google Cloud Platform ([gcp] for the official page) provides cloud computing services on reliable infrastructure. A virtual machine with 2 instances were set. Their first option of 1 virtual CPU and 614 MB memory RAM proved to be powerful enough only for $n \leq 7$ with maximum failure probability $p = 0.25$. Thus, we increased it to 1 virtual CPU with 1.7GB memory RAM for $8 \leq n \leq 10$ and $p = 0.25$. However, due to the limited RAM memory, the first instance was slow and for $p = 0.75$ and $p = 0.9$, we invested in a second instance with 1 vCPU with 3.65GB memory RAM. The costs of running on the Google Cloud Platform allowed us to buy a virtual instance with 4 CPUs only for a limited amount of time. Thus, we could use the parallel implementation on this platform to improve the running time only for a small subset of the experiments: for $n \geq 12$ and $p \in \{0.1, 0.25\}$.

Table 5.1 summarises the parameters configurations that were provided to the system on each of the presented environments.

platform	***n***	***p***	**system version**
personal device	$2 \leq n \leq 8$	$p = 0.5$	serial and parallel
student.compute clusters	$9 \leq n \leq 15$	$p = 0.5$	parallel
GCP	$2 \leq n \leq 10$	$p \in \{0.1, 0.25\}$	serial
personal device	$n = 11$	$p \in \{0.1, 0.25\}$	parallel
GCP	$n \in \{12,13,14\}$	$p \in \{0.1, 0.25\}$	parallel
GCP	$2 \leq n \leq 8$	$p = 0.75$	serial
personal device	$n \in \{9,10\}$	$p = 0.75$	parallel
GCP	$2 \leq n \leq 7$	$p = 0.9$	serial
personal device	$n = 8$	$p = 0.9$	parallel

Table 5.1: Parameters variations on different running platforms

Chapter 6

Results

This chapter presents the results of the reliability system described in Section 4.6. The main aim of this project was to analyse the reliability of grid graphs for failure probability of $p = 0.5$. We defined auxiliary goals related to the performance of the Sampling Module, presented in Section 4.4. The following list summarises the accomplishments for each task.

- **Sampling running time**. We did not have any previous knowledge of an implementation of the *Cluster popping with Tarjan's algorithm* (Section 4.4) and we were aware of the theoretical bounds of the running complexity of the algorithm. As part of this project, we analysed the real running time of the Sampling Module for each contraction of the n x n grids, for each (n, p) configuration of parameters of the Reliability Module. In addition, we computed the running time for sampling root-connected subgraphs from complete grid graphs (*not* contractions) with $2 \leq n \leq 100$ for $p \in \{0.1, 0.25, 0.5, 0.75\}$ and $2 \leq n \leq 82$ for $p = 0.9$. The results are presented in Section 6.1.

- **Resampling count**. We saved the number of arcs resampled and minimal clusters found by the Sampling Module, which influence the running time of the algorithm. Section 6.2 provides the results for each run of the Reliability Module and for sampling root-connected subgraphs from complete grid graphs with $2 \leq n \leq 100$ for $p \in \{0.1, 0.25, 0.5, 0.75\}$ and with $2 \leq n \leq 82$ for $p = 0.9$.

- **Network reliability and logarithmic growth rate**. Running the Reliability Module on different platforms, as presented in Section 5, we computed the network reliability and the logarithmic growth rate for grids of size up to 15 x 15 for 5 failure probabilities, with 95% confidence of being in the (1 ± 0.1) interval:

  - $2 \leq n \leq 14$ and $p = 0.1$;
  - $2 \leq n \leq 14$ and $p = 0.25$;
  - $2 \leq n \leq 15$ and $p = 0.5$;
  - $2 \leq n \leq 10$ and $p = 0.75$, results for $n \geq 8$ with 75% confidence;
  - $2 \leq n \leq 8$ and $p = 0.9$, results for $n \geq 6$ with 75% confidence.

In addition, we compared the reliability behaviour with the values for path, pan, ladder and complete graphs of size $n \leq 100$ with known polynomials. The results are discussed in Section 6.3. We also compared the estimated reliability of grids with the exact value for $n \leq 5$.

6.1 Sampling running time

One of the goals of this project was to analyse the practical efficiency of the sampling algorithm with *Cluster popping with Tarjan's algorithm*. We remind the reader that the time complexity of the sampling algorithm is $O(|A| + \frac{p|A||V|}{1-p})$. For the n x n grid graphs we worked with, $O(|A|) = O(|V|) = n^2$. Hence, the theoretical analysis suggests that the time complexity is bounded by a function of n^4.

6.1.1 Sampling running time for grids

We run the algorithm to sample root-connected subgraphs on grids of size up to 100 x 100 using the iterative implementation. The iterative approach allowed us to test with graphs of larger size, as the recursive algorithm provided in [GH18] reached the system's stack limit for $n = 38$ (tested on the personal device, as described in Section 5). The results were computed by averaging the real running time of 1000 runs for each grid-size, using the personal device.

As suggested in Figure 6.1, the average sampling time (in milliseconds) generally increases with the size of the grid, with the exception of a few outliers that results in spikes of a few milliseconds in the running time, that may be caused by the performance and demand of the platform used. In addition, the plot highlights that an increasing failure probabilities on the edges increases the sampling time. This observation is expected, as a higher failure probability results in more arcs resampled (Section 6.2). Results for the sampling time on grids are also provided in Appendix G.

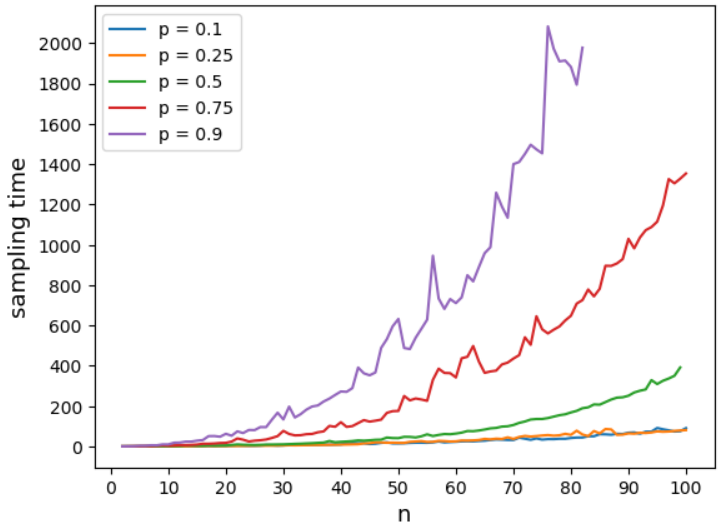

Figure 6.1: Average sampling time (ms)

In addition, we tried to analyse the growth rate of the sampling time by comparing it with a function of n. While it is clear from Figure 6.1 that the time (in milliseconds) is less than n^2, for $n \leq 100$, it would be incorrect to generalize this statement, as we do not have data for n approaching infinity. However, we notice that $\frac{sampling\_time}{n^3}$ is a (generally) decreasing and converging functions, as seen in Figure 6.3. Our analysis has shown that this fact is not true if the time is divided by smaller polynomial functions, such as n, n^2 or $n^2 \log(n)$ (Figure 6.2).

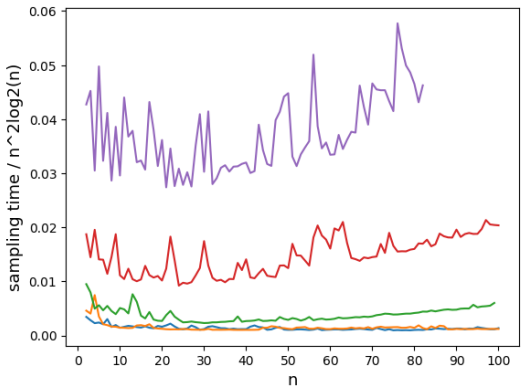

Figure 6.2: Sampling time / $n^2 \log(n)$

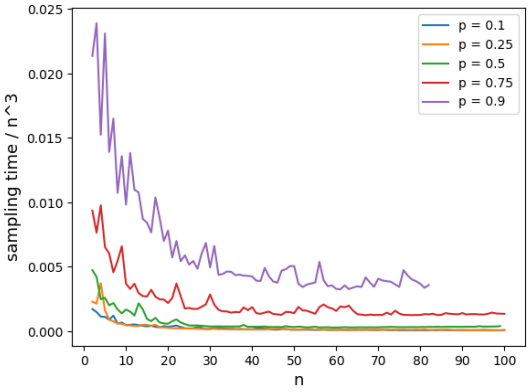

Figure 6.3: Sampling time / n^3

Lastly, we notice that $\frac{\log_2(sampling\_time)}{n}$ is rapidly converging towards 0 for all probabilities, as displayed in Figure 6.5, which implies that the same convergence is true for the logarithmic growth rate given n^2, the number nodes: $\frac{\log_2(sampling\_time)}{n^2}$. For completeness, the behaviour of $\log_2(sampling\_time)$ is provided in Figure 6.4.

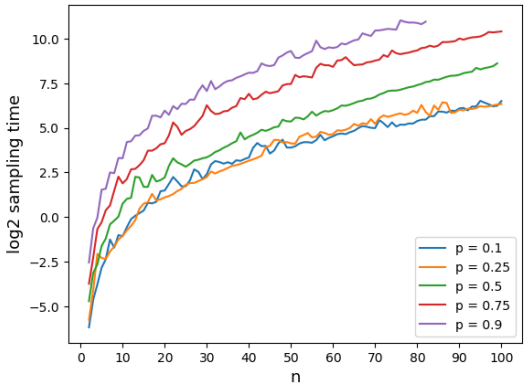

Figure 6.4: $\log_2$(sampling time)

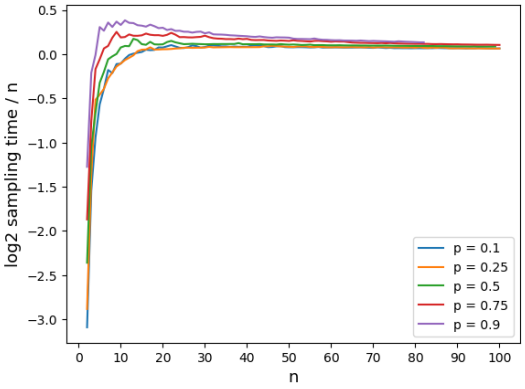

Figure 6.5: $\log_2$(sampling time)/n

6.1.2 Sampling running time for contractions

The reliability approximation algorithm samples root-connected subgraphs from sequential contractions of the initial grid. Intuitively, it was expected to see a decrease in the sampling time for consecutive iterations (contractions), as each time the number of edges in the graph decreases. This assumption was validated experimentally, with a few exceptions on iterations whose running was influenced by the platform being unavailable for the current task. For example, the behaviour of the sampling algorithm on consecutive contractions can be seen in Figure 6.6 and Figure 6.7, for the failure probability on edges $p = 0.5$ and $p = 0.75$. The time was measured in milliseconds.

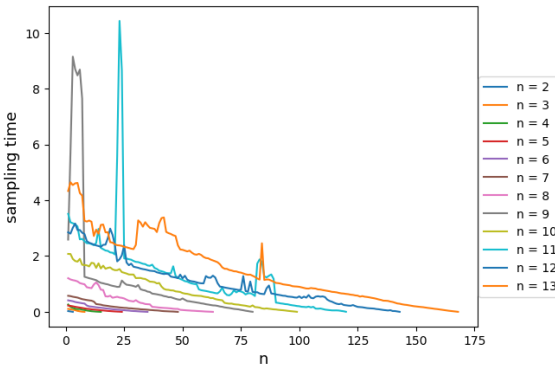

Figure 6.6: Sampling time for $p = 0.5$

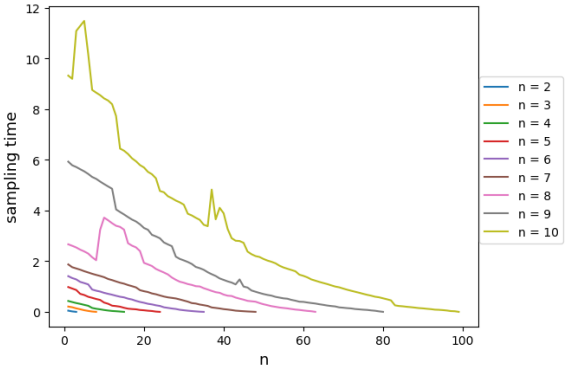

Figure 6.7: Sampling time for $p = 0.75$

6.2 Resampling count

The algorithm for sampling root-connected subgraphs, described in Section 4.4, is based on finding minimal clusters and resampling their arcs: removing them from the current graph and independently adding them back with $1 - p$ probability, where p is the failure probability on edges. It is clear that the time complexity of this approach depends on the number of arcs resampled before the algorithm returns. Hence, to gain a better understanding of the cluster-popping algorithm, we saved the total number of arcs resampled and minimal clusters found for returning one root-connected subgraph.

6.2.1 Arcs resampled and minimal clusters for grids

We run the sampling algorithm on grids of size up to 100 x 100 for failure probabilities in $\{0.1, 0.25, 0.5, 0.75, 0.9\}$, 1000 times for each (n, p) configuration. Some of the data obtained is provided in Appendix H.

We noticed that, for the same failure probability p, the increase of the two values follow similar behaviours, as it can be seen in Figure 6.8 and Figure 6.9. Naturally, there are always more arcs resampled and minimal clusters found for the same graphs with a higher failure probability. In addition, these numbers increase substantially for $p \in \{0.75, 0.9\}$, while the differences for $p \in \{0.1, 0.25\}$ are harder to observe. For a better visualisation of the differences for various probabilities, we also plotted the $\log_2$ values of the data, as it can be seen in Figure 6.10 and Figure 6.11.

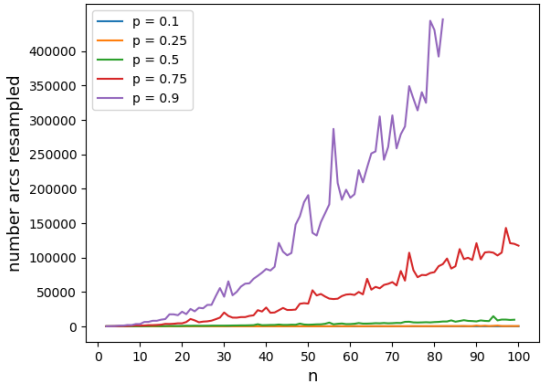

Figure 6.8: Avg. number arcs resampled

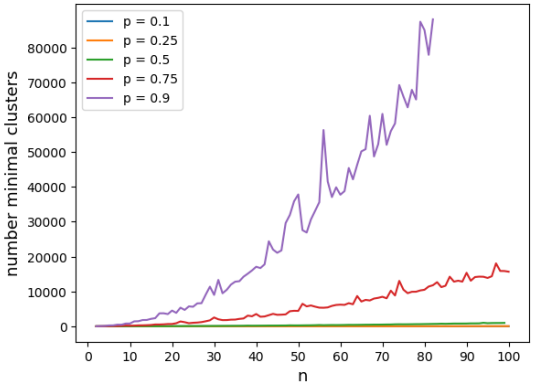

Figure 6.9: Avg. number minimal clusters

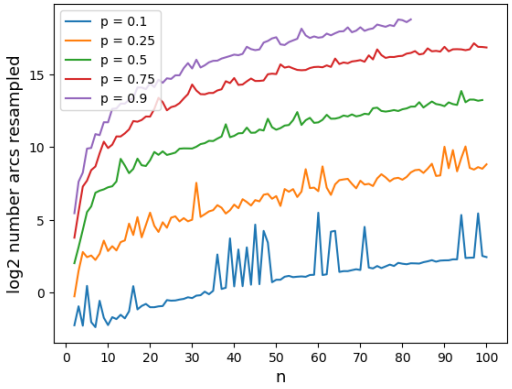

Figure 6.10: $\log_2(avg\_arcs\_resampled)$

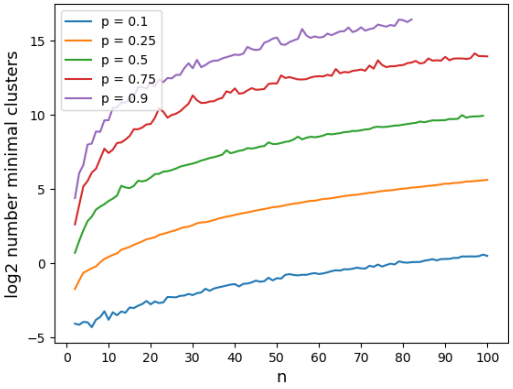

Figure 6.11: $\log_2(avg\_minimal\_clusters)$

Figures 6.12 and 6.13 suggest that $\frac{number\_arcs\_resampled}{n^2 \log_2(n)}$ and $\frac{number\_minimal\_clusters}{n^2 \log_2(n)}$ tend to be in an interval around a constant, implying that that $number\_arcs\_resampled$ and $number\_minimal\_clusters$ are in an interval around $constant \cdot n^2 \log_2(n)$. This observation suggests a $O(n^2 \log_2(n))$ complexity of the arcs resampling step. In the analysis of the sampling time, we noticed a clearer convergence for a division over n^3. Indeed, the data is less noisy and the values are more obviously decreasing for $\frac{number\_arcs\_resampled}{n^3}$ and $\frac{number\_arcs\_resampled}{n^3}$. Figure 6.14 and Figure 6.15 clearly show that the last two values are decreasing with the increase in the size of the grid, approaching 0.

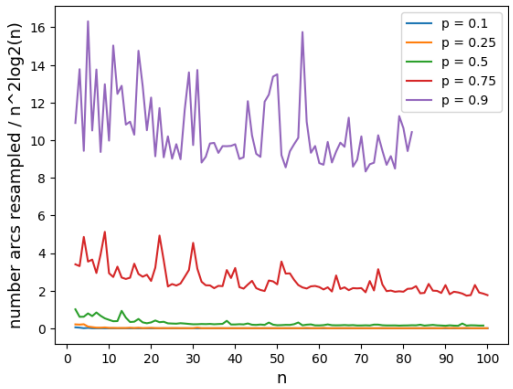

Figure 6.12: $avg\_arcs\_resampled/(n^2\log_2(n))$

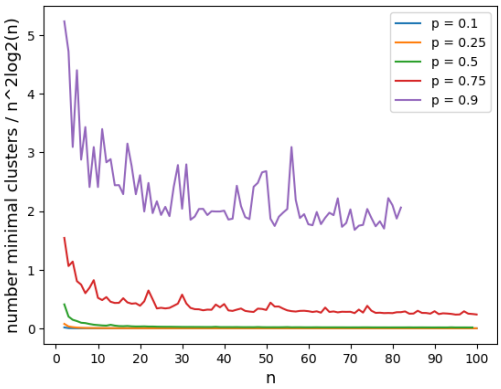

Figure 6.13: $avg\_minimal\_clusters/(n^2\log_2(n))$

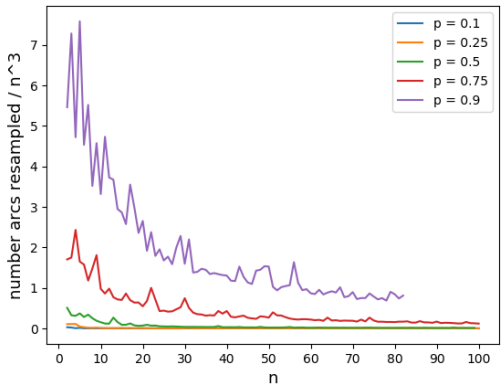

Figure 6.14: $avg\_arcs\_resampled/n^3$

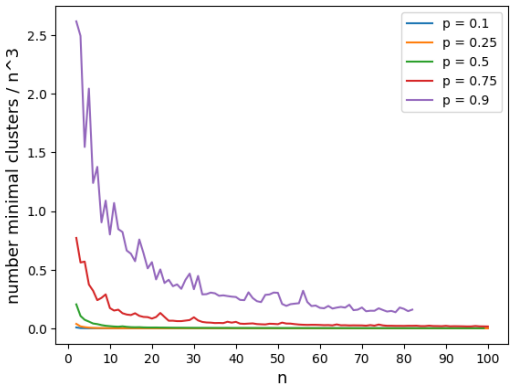

Figure 6.15: $avg\_minimal\_clusters/n^3$

The last observation we make is that, similarly to the behaviour of the sampling time, $\frac{\log_2(avg\_arcs\_resampled)}{n}$ and $\frac{\log_2(avg\_minimal\_cluster)}{n}$ converge to 0 as the size of the grid increases, as shown in Figure 6.16 and Figure 6.17.

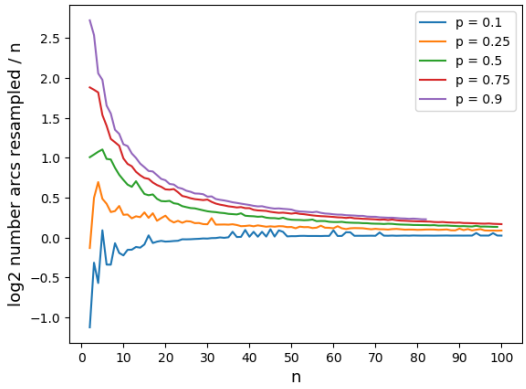

Figure 6.16: $\log_2(avg\_arc\_resampled)/n$

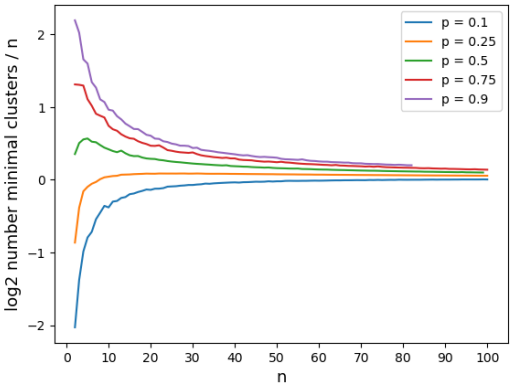

Figure 6.17: $\log_2(avg\_min\_clusters)/n$

6.2.2 Arcs resampled and minimal clusters for contractions

Analysis of the number of arcs resampled and minimal clusters found for sampling root-connected subgraphs from each contraction, averaged for the s runs, has shown that, for the same failure probability p, these two values decrease with every contraction. This observation was expected, as the graph decreases in size for each iteration. In fact, we noticed that the number of arcs decreases by an average of 2 per contraction, for at least $\frac{2}{3}n^2$ iterations. Figure 6.18 and 6.19 exemplify for $p = 0.5$ how the two values decrease for each contraction, for $2 \leq n \leq 15$. Moreover, the number of arcs resampled or minimal clusters found at the same iteration are usually bigger for a larger graph. In addition, Figure 6.20 and Figure 6.21show that increasing the failure probability on edges results in a higher number of arcs resampled and minimal clusters found, with $p = 0.9$ having the biggest impact.

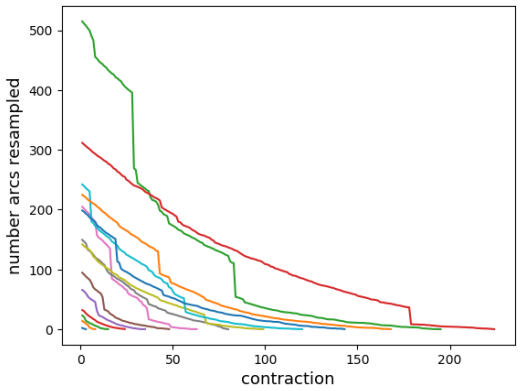

Figure 6.18: Arcs resampled for $p = 0.5$

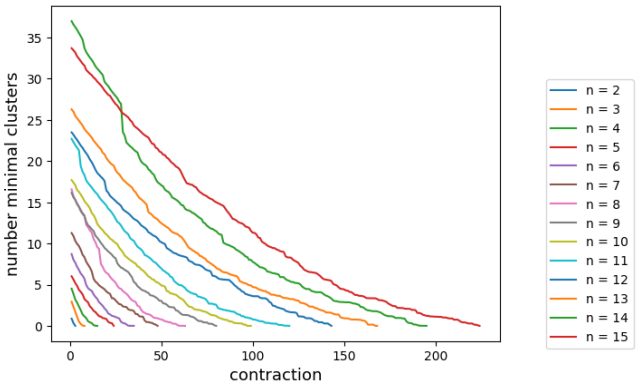

Figure 6.19: Minimal clusters for $p = 0.5$

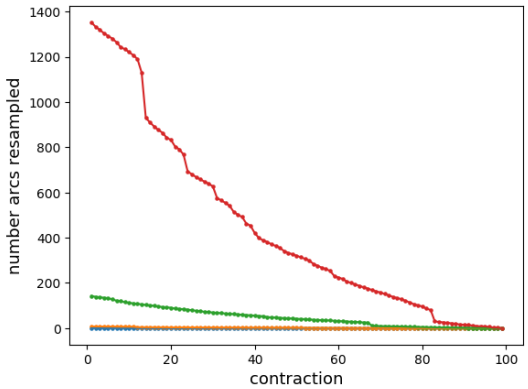

Figure 6.20: Arcs resampled for $n = 10$

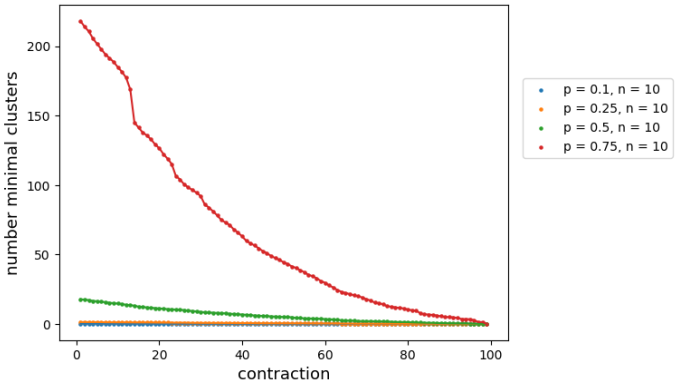

Figure 6.21: Minimal clusters for $n = 10$

6.3 Reliability results

6.3.1 Reliability of graphs with known polynomials

For a better understanding of the behaviour of the reliability value, we will firstly look at the all-terminal reliability and logarithmic growth rate of some graphs with known polynomials: paths, pans, ladder and complete. While the number of nodes and edges of the path, pan and ladder are $O(n)$, for the complete graph they are $O(n^2)$, as for the grids. However, we will notice in the next section and the behaviour of the reliability and logarithmic reliability of the grids is more similar to those of the ladder (which is also a lattice graph). Figures 3.4, 3.5, 3.6 and 3.9 show that the reliability decreases and approaches 0 with the increase in the size of the graph, for any failure probability p. This is also true for the ladder graph with $p = 0.9$, however the convergence is more obvious for $n > 200$. However, the reliability of the complete graph follows a different pattern, increasing with the size of the graph (but still decreases with the increase in the failure probability).

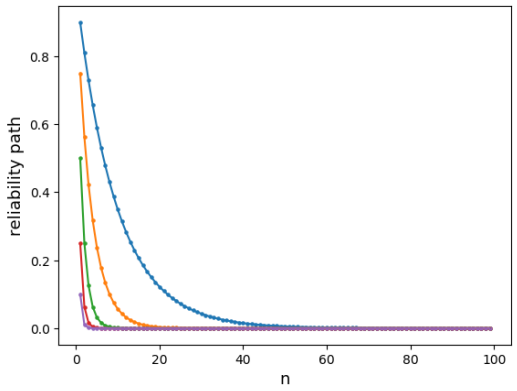

Figure 6.22: Reliability of path graphs

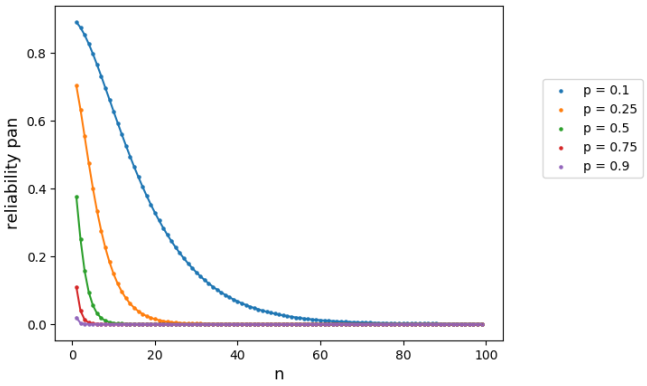

Figure 6.23: Reliability of pan graphs

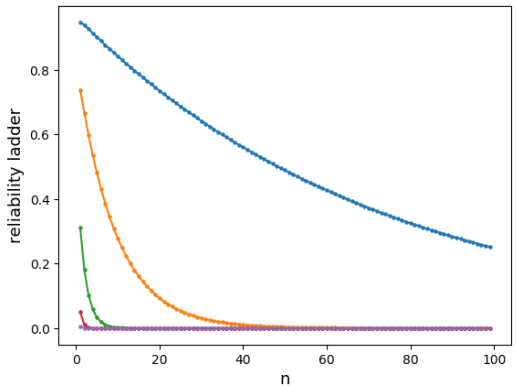

Figure 6.24: Reliability of ladder graphs

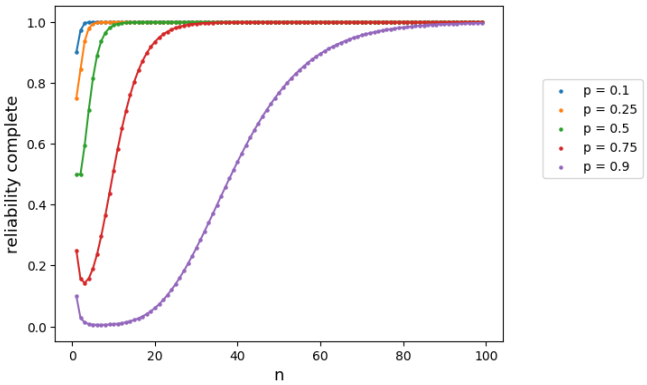

Figure 6.25: Reliability of complete graphs

Figures 6.26 to 6.29 show the logarithmic reliability ($\frac{log_2(reliability)}{number\_nodes}$) of the four graphs. For the ladder graph, the logarithmic reliability is also increasing for $p \leq 0.5$. What

these plots prove is that the value is indeed converging to a constant, as we had expected. In particular, given the known polynomial presented in Section 3.2.5, it can be trivially derived that the constant for the path graphs is $\log_2(1-p)$. The computed reliability and growth rate for $n = 100$ for these graphs are provided in Appendix I.

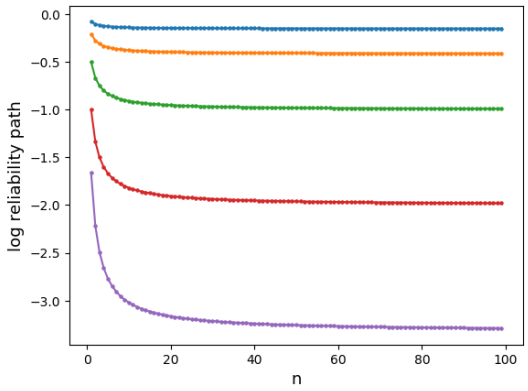

Figure 6.26: Log reliability of path graphs

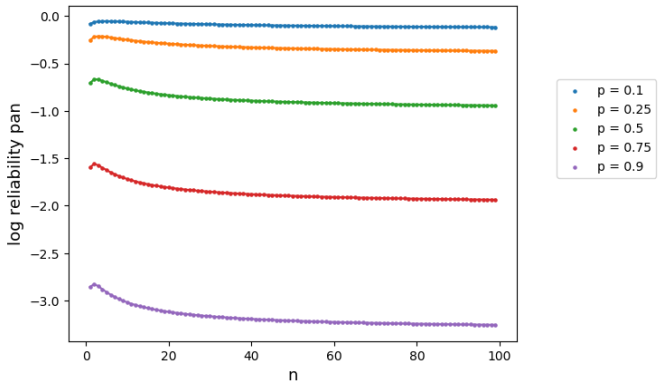

Figure 6.27: Log reliability of pan graphs

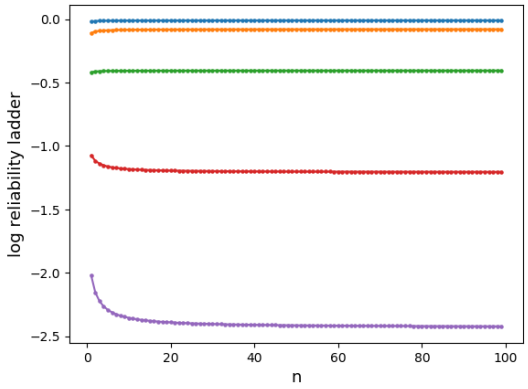

Figure 6.28: Log reliability of ladder graphs

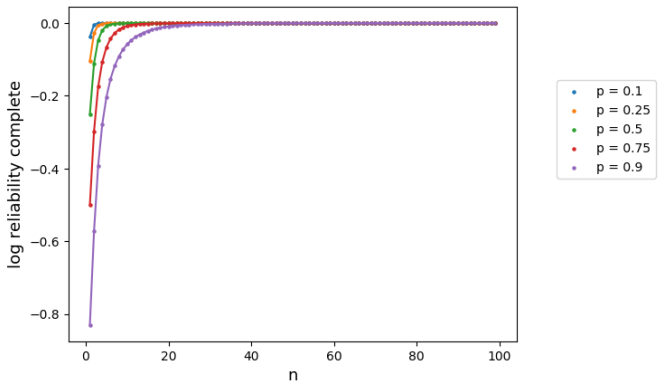

Figure 6.29: Log rel. of complete graphs

6.3.2 Reliability of grids

We computed the reliability of grids for different failure probabilities and sizes of the graph. Due to the complexity of the naive implementation, we could compare the results with the exact values only for $n \leq 4$. Indeed, our approximated reliability values were in the $(0.9, 1.1)$ interval around the result of exact computations. In fact, we noticed that the interval could be tighten to $(0.99, 1.008)$. The values of the approximated reliability and logarithmic reliability of grids are provided in Appendix J. Based on our results, we make the following observations:

- The reliability decreases with the increase of the size of the grid.
- The reliability decreases with the increase of the failure probability.
- The logarithmic reliability increases with the increase of the size of the grid.
- The logarithmic reliability decreases with the increase of the failure probability.

Figure 6.30 and Figure 6.31 show the growth rate of the two values for the 5 failure probabilities we worked with. For a better visualisation, we also provide in Figures 6.32 and 6.33 the individual plots for $p = 0.5$, the main objective of this project. Appendix K provides the individual plots for the rest of the failure probabilities.

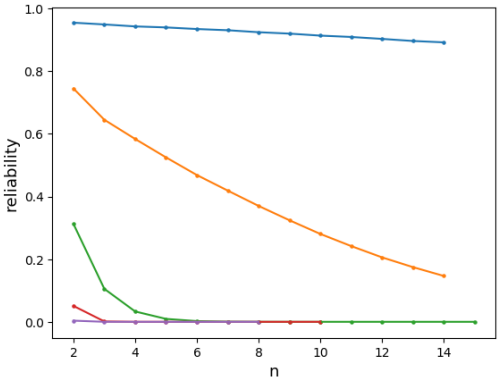

Figure 6.30: Reliability of grid graphs

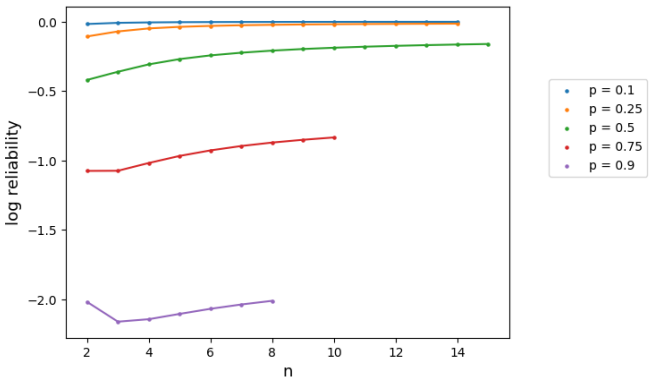

Figure 6.31: Log rel. of grid graphs

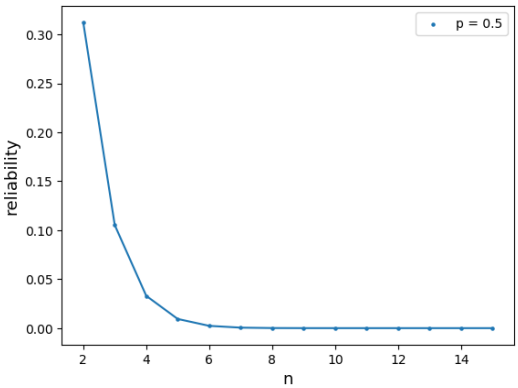

Figure 6.32: Reliability of grids, $p = 0.5$

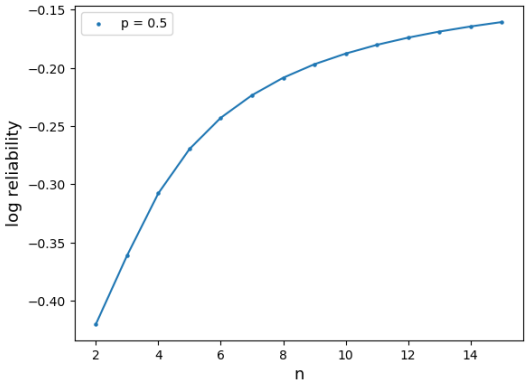

Figure 6.33: Log rel. of grids, $p = 0.5$

The data collected suggests that, while the reliability is approaching 0, the logarithmic reliability will eventually converge to a constant. The converge is faster for smaller failure probabilities. While the complexity of the approximation algorithm makes it difficult to obtain results for large graphs, our results allow us to establish upper bounds for the reliability (which is a decreasing function of n given constant p) and lower bounds for the logarithmic reliability (which is an increasing function of n given constant p). These bounds are provided in Table 6.1.

6.3.2.1 Reliability running time

Figures 6.34 to 6.37 highlight the differences in the running time of the Reliability Module for different probabilities. However, it is important to remember that the time to estimate the reliability depends both on the performance of the platform used and on the version of the system (iterative or parallel). For example, although it it expected to compute the reliability for $p = 0.25$ faster than for $p = 0.5$, the GCP platform we used

p	Upper bound Z_{rel}	Lower bound L_{rel}
0.1	0.8920	-0.0008
0.25	0.1465	-0.0142
0.5	1.3027e-11	-0.1607
0.75	7.7700e-26	-0.8342
0.9	1.7698e-39	-2.0114

Table 6.1: Bounds for reliability and logarithmic reliability

for these computation was worse performance-wise than the *student.compute* clusters. For $p = 0.5$, for which we used the *student.compute* clusters, the algorithm took between 4 seconds (for $n = 2$) and 4.25 days (for $n = 13$). For $n = 14$, the servers were unreliable, resulting in a computation of more than 12 days. The consequences of varying the platform could also be noticed for $p = 0.1$, with a slower performance of the personal device for $n = 11$. Lastly, Figure 6.35 for $p = 0.25$ shows the effect of updating the performance of the GCP virtual machine at $n = 8$. Thus, we cannot accurately compare the running time of the complete Reliability Module for sets of experiments on different platforms.

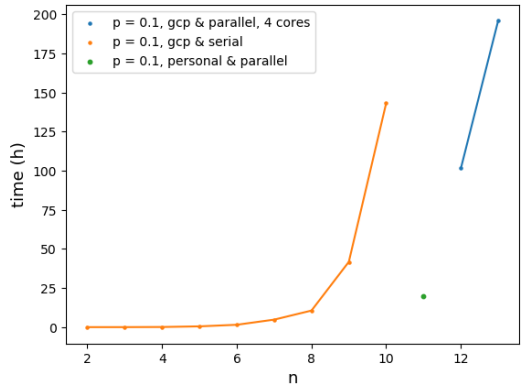

Figure 6.34: Running time, $p = 0.1$

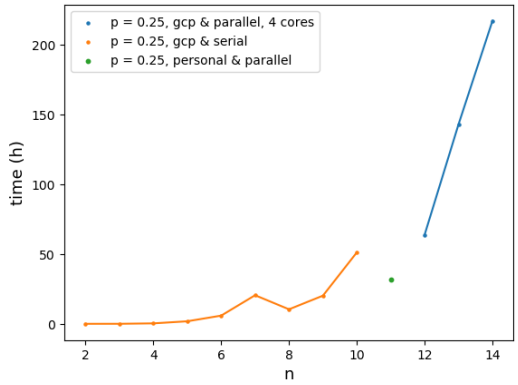

Figure 6.35: Running time, $p = 0.25$

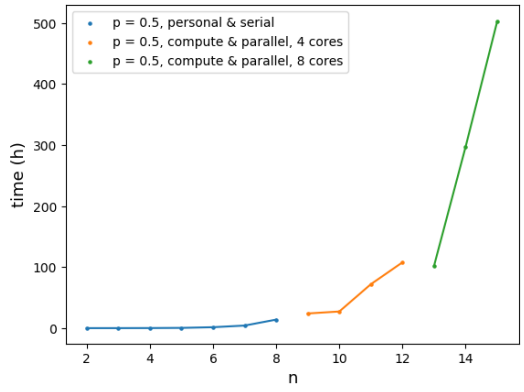

Figure 6.36: Running time, $p = 0.5$

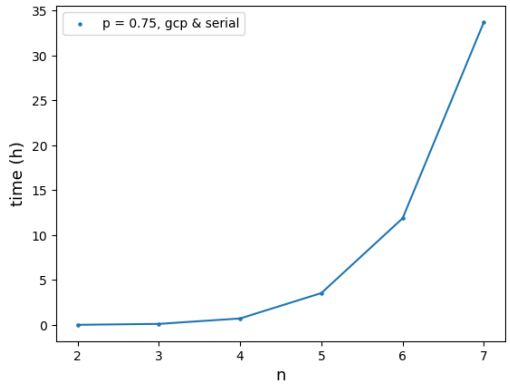

Figure 6.37: Running time, $p = 0.75$

6.3.2.2 Number connected spanning subgraphs

The reliability of grids for $p = 0.5$ allows us to also approximate the total number of connected spanning subgraphs of the input un-directed graph. For $2 \leq n \leq 4$, we were able to run the naive algorithm that gives the exact values. In particular, we found that:

- A 2 x 2 grid has 5 connected spanning subgraphs.
- A 3 x 3 grid has 431 connected spanning subgraphs.
- A 4 x 4 grid has 555195 connected spanning subgraphs.

For $n = 5$, the naive implementation did not complete the computation for the connected subgraph of minimal size $(n^2 - 1)$ in 12 hours. The reliability results allow us to approximate this value for $n \geq 5$, with 95% confidence of (1 ± 0.1) deviation. Table 6.2 presents these estimations (called K_n), rounded to the nearest integer, and their logarithmic value.

n	K_n	$\log_2(K_n)/n^2$
2	5	0.5801
3	431	0.9724
4	553497	1.1923
5	10289687302	1.3304
6	2692756361079525	1.4238
7	9.7767e+21	1.4908
8	4.9990e+29	1.5415
9	3.5254e+38	1.5808
10	3.4099e+48	1.6122
11	4.5738e+59	1.6379
12	8.4249e+71	1.6592
13	2.1381e+85	1.6772
14	7.4015e+99	1.6926
15	3.52712e+115	1.7059

Table 6.2: Approximation of total connected spanning subgraphs of n x n grids.

6.3.2.3 Observation on the number of root-connected expansions

As presented in Section 3.5, the reliability is estimated with a product of the means of s (Appendix D) random variables for the consecutive iterations. In fact, the sum of random variables for a contraction is equal to the total number of root-connected expansions retrieved during the current iteration. Thus, we can write the reliability of an n x n grid G with edges failure probability as:

$$Z_{rel}(G,p) = Z'_{reach}(G',r,p) = \prod_{i=1}^{n^2-1} R'_i = \prod_{i=1}^{n^2-1} \frac{\sum_{j=1}^{s} R_{i,j}}{s} = \prod_{i=1}^{n^2-1} \frac{C_i}{s},$$

where G' is the bi-directed graph of G, r the randomly selected root and C_i the total number of root-connected expansions at iteration i. In section 3.5.1, we showed that:

$$Z_{rel}(G,p) \leq (\frac{\sum_{i=1}^{n^2-1} R_i'}{n^2-1})^{n^2-1}.$$

Since $R_i' = \frac{C_i}{s}$, then:

$$Z_{rel}(G,p) \leq (\frac{\sum_{i=1}^{n^2-1} \frac{C_i}{s}}{n^2-1})^{n^2-1} = (\frac{1}{s})^{n^2-1}(\frac{\sum_{i=1}^{n^2-1} C_i}{n^2-1})^{n^2-1}.$$

The bond on the reliability allows us to define a new variable. We let $avg\_C = \frac{\sum_{i=1}^{n^2-1} C_i}{n^2-1}$ be the average number of root-connected expansions over the n^2-1 contractions of an n x n grid. We make some observations on the values of C_i and avg_C for grids. Firstly, each $C_i \leq s$, the total number of root-connected samples drawn, with s being $O(n^2)$. Figure 6.38 shows us how C_i increases with the size of the grid. Moreover, for a better approximation of the reliability, s increases with the failure probability. Figure 6.39 highlights the effect of the increase in the number of samples with the probability on the average number of root-connected expansion, given constant grid size (n).

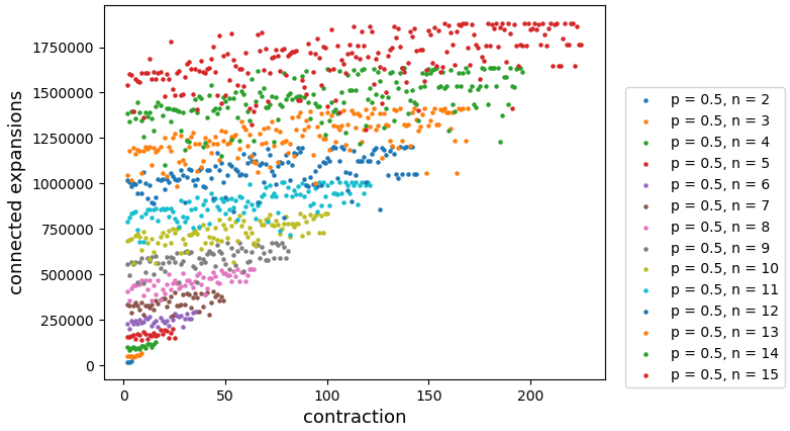

Figure 6.38: Root-connected expansions for each iteration for $p = 0.5$.

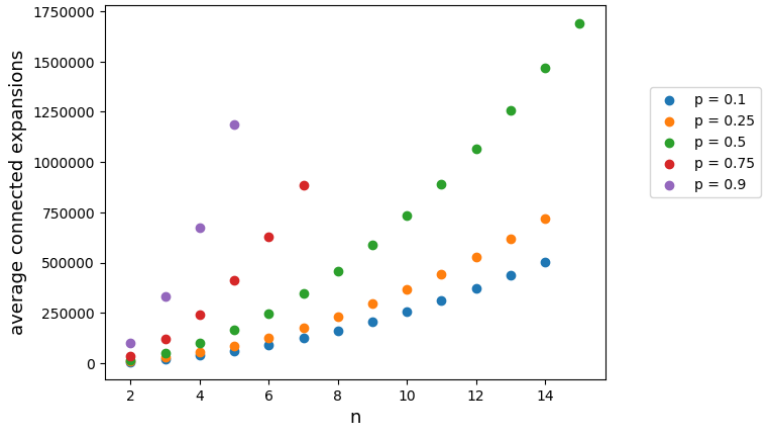

Figure 6.39: Average root-connected expansions

Thus, a question that was raised was how far is the reliability from this bound. We define the value $\Delta_{rel}(p,G)$ as the difference between the upper bound on the reliability and the approximate result:

$$\Delta_{rel}(p,G) = (\frac{1}{s})^{n^2-1}(\frac{\sum_{i=1}^{n^2-1} C_i}{n^2-1})^{n^2-1} - Z_{rel}(G,p).$$

The values of $\Delta_{rel}(p,G)$ for each p are provided in Appendix L. In Table 6.3, we present the average of the distances to the upper bound for each failure probability p, over all the grids for which reliability has been estimated for p.

$$avg\_\Delta_p = \frac{\sum_{i=2}^{n_p} \Delta_{rel}(p,G_i)}{n_p - 1},$$

where n_p is the maximum grid-size for which reliability has been computed for p and $n_p - 1$ is the total number of grids with reliability for p.

The low values of avg_Δ_p and of the standard deviation suggest that the distance from the reliability to the upper bound is relatively small. However, for $p < 0.5$, this distance is not a monotone function as the size of the grid increases. For $p \geq 0.5$, our analysis shown that the average distance is a decreasing function.

p	avg_Δ_p	*standard deviation*
0.1	0.0001	5.5042e-05
0.25	0.0033	0.0009
0.5	0.0010	0.0018
0.75	0.0001	0.0004
0.9	2.0145e-05	4.8946e-05

Table 6.3: Average distance from estimated reliability to upper bound.

From section 3.5.1, we also know an upper bound on the logarithmic reliability for the n x n grid:

$$L_{rel}(G,r,p) \leq \frac{n^2-1}{n}\log_2(\frac{\sum_{i=1}^{n^2-1} R'_i}{n^2-1}).$$

If we use the fact that fact $R'_i = \frac{C_i}{s}$:

$$L_{rel}(G,r,p) \leq \frac{n^2-1}{n^2}\log_2(\frac{\sum_{i=1}^{n^2-1} C_i}{s(n^2-1)}) = \frac{n^2-1}{n^2}\log_2(\frac{\sum_{i=1}^{n^2-1} C_i}{n^2-1}) - \frac{n^2-1}{n^2}\log_2(s).$$

Looking again at the difference $\Delta_{log_p}(p,G)$ between upper bound and the logarithmic reliability, we noticed that they are more considerable for larger p, as suggested in Table 6.4, where $avg\_\Delta_{log_p}$ is computed by averaging $\Delta_{log_p}(p,G)$ for all grids with estimated reliability for p.

To exemplify these differences, in Figure 6.40 and Figure 6.41 we plotted the reliability and logarithmic reliability of grids for $p = 0.5$ and their corresponding upper bounds.

p	$avg\_\Delta_{log_p}$	*standard deviation*
0.1	7.5414e-06	5.8951e-06
0.25	0.0003	0.0001
0.5	0.0064	0.0018
0.75	0.0363	0.0109
0.9	0.0827	0.0292

Table 6.4: Average distance from estimated logarithmic reliability to upper bound.

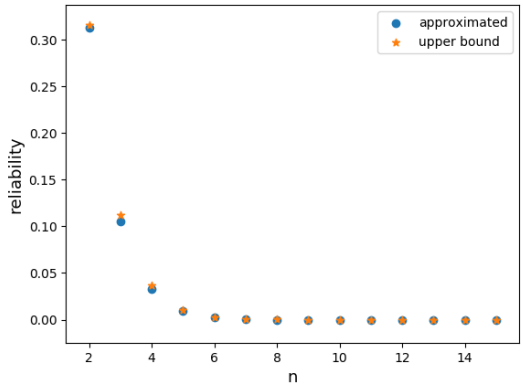

Figure 6.40: Reliability of grids and its upper bound for $p = 0.5$

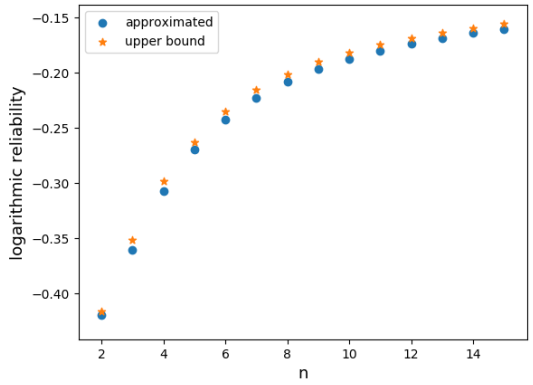

Figure 6.41: Logarithmic reliability of grids and its upper bound for $p = 0.5$

This analysis suggest that a better understanding of the average number of root-connected expansions could enable us to also approximate the behaviour of the reliability and, for failure probabilities not too large, of the logarithmic reliability. Computing such values with the reliability algorithm presented in this paper still requires the same computational complexity. We can notice, however, the behaviour of the average number of connected expansions for all iterations over the total number of samples, $\frac{avg\_C}{s}$, for each failure probability, as the size of the grid increases. What this analysis shows us it that this proportion of s, $\frac{avg\_C}{s} = \frac{\sum_{i=1}^{n^2-1} C_i}{s(n^2-1)}$, and its logarithmic growth rate, $\frac{1}{n^2}\log_2(\frac{avg\_C}{s})$, are increasing functions of n. Assuming the functions keeps their monotonically increasing behaviour, our results for the highest n for each p give a lower bound for these values, presented in Table 6.5. The data for each (n, p) is provided in Appendix M.

p	avg_connected / s	log. growth of (avg_connected / s)
0.1	0.9994	-4.3070e-06
0.25	0.9902	-7.1815e-05
0.5	0.8971	-0.0006
0.75	0.5746	-0.0079
0.9	0.2599	-0.0303

Table 6.5: Lower bounds for percentage of average connected expansions out of number of samples and its logarithmic growth

Chapter 7

Conclusion

In the first part of the MInf project, we focused on establishing the existence of polynomial approximation algorithms for #P-hard counting constraint satisfaction problems, by looking at the distribution of the roots around the unit circle of the associated graph polynomials. In the second part of the MInf project, we shifted our focus to the implementation of an already existing approximation scheme for a problem that elicits much research interest. This paper presented experimental results on the study of the all-terminal reliability, a measure with substantial importance in many areas where the relationship between objects can be represented with graphs, such as computer networking. While computing the reliability is generally a #P-hard problem, we implemented the first fully polynomial-time approximation scheme, that estimates the reliability of an un-directed graph by using its equivalence with the reachability value in the corresponding bi-directed graph with a randomly selected root. The reachability approximation algorithm involves a sequence of iterations, with each iteration reducing the size of bi-directed graph by contracting two adjacent vertices. From each contraction, the algorithm draws a number of root-connected subgraphs, using a sampling algorithm that recursively finds minimal clusters and independently resamples the arcs with $1-p$ probability, where p is the arcs failure probability. The reachability is approximated using the means of random variables that check whether the expansion of each root-connected subgraph is still root-connected, where an expansion un-contracts the two previously chosen nodes and independently adds the arcs between them with $1-p$ probability.

Another contribution of this paper is the introduction of an equivalent iterative implementation for the sampling algorithm, which was only recursively defined before this project. We also presented an approach in which the root-connected subgraphs are sampled in parallel, in order to decrease the running time of the reliability approximation algorithm.

We analysed the performance of the sampling algorithm, the main ingredient of the approximation scheme, on bi-directed grids of size up to 100 x 100 for the 5 different failure probabilities, presenting the average number of arcs resampled and minimal clusters found (over 1000 runs for each grid size). We showed how these values are influenced by the increase in the failure probability and grid size. By running the

complete system on three different platforms, we extended our goal of computing the reliability and logarithmic reliability of n x n grids for $p = 0.5$ to a total 5 different failure probabilities. The number of samples was chosen such that the approximation results have 95% confidence of being in the (1 ± 0.1) interval. In addition, we used the known reliability polynomials of some family of graphs to test the system and compare the behaviour of the reliability function. Furthermore, a bonus feature of our system is that the reliability for $p = 0.5$ allowed us to estimate the total number of connected spanning subgraphs of a grid and we also looked at the growth rate of these values. In the end, we noticed the upper bounds of the reliability and logarithmic growth, given the average number of root-connected expansions over all iterations.

We acknowledge that our reliability system is limited by the computational performance of the platforms it run on. Using various environments, we traded consistency for an extended results sets, which influenced our ability of accurately comparing the running running time to estimate the reliability for different edges failure probabilities. Moreover, the complexity of the approximation algorithm made it unsustainable to obtain reliability results for graphs with more than 225 node and 840 arcs (of the 15 x 15 bi-direct grid). Thus, we empirically showed that the logarithmic reliability growth rate tends to approach a constant. Although we cannot precisely compute the limits, our results allowed us to determine certain bounds. Since the reliability of square grids is decreasing with the size of the grid, we could establish the upper bounds of this value for larger n, given the 5 failure probabilities. Similarly, we presented the lower bounds of the logarithmic reliability growth rate, which is increasing with the size of the grid. For example, for a failure probabilities of 0.5, the reliability for square grids with $n \geq 15$ is at most $1.3026 \cdot 10^{-11}$ and the logarithmic reliability at least -0.1607.

The experiments in this paper represent a only a small subset of the work that could be done in the area of reliability approximation and there is space for plenty of further research. More computational power than our resources would be necessary for an advancement in estimating the reliability of square grids of larger size. Furthermore, similar experiments to those presented here could be carried out for other types of graphs, to check how the behaviour we noticed is affected. Examples of graphs interesting to analyse and compare with the square grids include non-square lattices with arbitrary length and width. Similarly to the MInf Project Part I, we could focus our attention to random connected regular graphs. For any chosen family of graphs to experiment with, the reliability could be estimated for different failure probabilities, eventually expanding the set of 5 probabilities of our research. Lastly, we could compare our approximated values with the results of reducing the confidence level from 95% to, for example, 75%, which would imply a number of necessary samples per iteration $\frac{21}{5} = 4.2$ times smaller than for our current implementation. A more permissive confidence interval would also allow a less expensive computation of the reliability, eventually leading to an increase of the grid size limit.

Bibliography

[Bal80] Michael O Ball. Complexity of network reliability computations. *Networks*, 10(2):153–165, 1980. Available at https://doi.org/10.1002/net.3230100206.

[BC92] Jason I Brown and Charles J Colbourn. Roots of the reliability polynomials. *SIAM Journal on Discrete Mathematics*, 5(4):571–585, 1992. Available at https://epubs.siam.org/doi/pdf/10.1137/0405047.

[BD19] Jason I Brown and Corey DC DeGagné. All terminal reliability roots of smallest modulus. *arXiv preprint arXiv:1906.02359*, 2019. Available at https://arxiv.org/abs/1906.02359.

[BM17] Jason Brown and Lucas Mol. On the roots of all-terminal reliability polynomials. *Discrete Mathematics*, 340(6):1287–1299, 2017. Available at https://doi.org/10.1016/j.disc.2017.01.024.

[BN79] Michael O Ball and George L Nemhauser. Matroids and a reliability analysis problem. *Mathematics of Operations Research*, 4(2):132–143, 1979. Available at https://doi.org/10.1287/moor.4.2.132.

[Bol01] Béla Bollobás. *Random graphs*. Cambridge University Press, 2nd edition, 2001.

[BP83] Michael O Ball and J Scott Provan. Calculating bounds on reachability and connectedness in stochastic networks. *Networks*, 13(2):253–278, 1983. Available at https://doi.org/10.1002/net.3230130210.

[Buz80] John A Buzacott. A recursive algorithm for finding reliability measures related to the connection of nodes in a graph. *Networks*, 10(4):311–327, 1980. Available at https://doi.org/10.1002/net.3230100404.

[CGW16] J. Cai, H. Guo, and T. Williams. A complete dichotomy rises from the capture of vanishing signatures. *SIAM Journal on Computing*, 45(5):1671–1728, 2016. Available at https://doi.org/10.1137/15M1049798.

[CLX09] Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holant problems and counting CSP. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 715–724. ACM, 2009. Available at https://dl.acm.org/citation.cfm?id=1536511.

[CS03] Shu-Chiuan Chang and Robert Shrock. Reliability polynomials and their asymptotic limits for families of graphs. *Journal of statistical physics*, 112(5-6):1019–1077, 2003. Available at https://doi.org/10.1023/A:1024663508526.

[CS04] Shu-Chiuan Chang and Robert Shrock. Tutte polynomials and related asymptotic limiting functions for recursive families of graphs. *Advances in Applied Mathematics*, 32(1-2):44–87, 2004. Available at https://doi.org/10.1016/S0196-8858(03)00077-0.

[DOT] The DOT language. Documentation at https://www.graphviz.org/doc/info/lang.html (last accessed on 23/01/2020).

[EMM11] Joanna A Ellis-Monaghan and Criel Merino. Graph polynomials and their applications I: The Tutte polynomial. In *Structural analysis of complex networks*, pages 219–255. Springer, 2011. Available at https://arxiv.org/pdf/0803.3079.pdf.

[Fis86] George S Fishman. A Monte Carlo sampling plan for estimating network reliability. *Operations Research*, 34(4):581–594, 1986. Available at https://doi.org/10.1287/opre.34.4.581.

[gcp] Google Cloud Platform. https://cloud.google.com (last accessed on 10/04/2020).

[GH18] Heng Guo and Kun He. Tight bounds for popping algorithms. *arXiv preprint arXiv:1807.01680*, 2018. Available at https://arxiv.org/abs/1807.01680.

[GJ08] Leslie Ann Goldberg and Mark Jerrum. Inapproximability of the Tutte polynomial. *Information and Computation*, 206(7):908–929, 2008. Available at https://doi.org/10.1016/j.ic.2008.04.003.

[GJ19] Heng Guo and Mark Jerrum. A polynomial-time approximation algorithm for all-terminal network reliability. *SIAM Journal on Computing*, 48(3):964–978, 2019. Available at https://doi.org/10.1137/18M1201846.

[GJL19] Heng Guo, Mark Jerrum, and Jingcheng Liu. Uniform sampling through the Lovász local lemma. *Journal of the ACM (JACM)*, 66(3):1–31, 2019. Available at https://doi.org/10.1145/3310131.

[GLLZ] Heng Guo, Chao Liao, Pinyan Lu, and Chihao Zhang. Zeros of Holant problems: locations and algorithms. *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2262–2278. Available at https://doi.org/10.1137/1.9781611975482.137.

[GP14] Igor Gorodezky and Igor Pak. Generalized loop-erased random walks and approximate reachability. *Random Structures & Algorithms*, 44(2):201–223, 2014. Available at https://doi.org/10.1002/rsa.20460.

[gra] Graphviz - graph visualization software. https://www.graphviz.org/ (last accessed on 26/02/2020).

[GS16] Ilya B Gertsbakh and Yoseph Shpungin. *Models of network reliability: analysis, combinatorics, and Monte Carlo*. CRC press, 2016.

[Hac17] Jürgen Hackl. Tikz-network manual. *arXiv preprint arXiv:1709.06005*, 2017.

[Hag91] Jane Nichols Hagstrom. Computing rooted communication reliability in an almost acyclic digraph. *Networks*, 21(5):581–593, 1991. Available at https://doi.org/10.1002/net.3230210507.

[HJ07] D Hunter John. Matplotlib: a 2d graphics environment comput. *Sci. Eng*, 9:90–5, 2007.

[HMW74] Eberhard Hänsler, GK McAuliffe, and RS Wilkov. Exact calculation of computer network reliability. *Networks*, 4(2):95–112, 1974. Available at https://doi.org/10.1002/net.3230040202.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. 2008. Available at https://www.osti.gov/biblio/960616.

[Jae88] François Jaeger. Tutte polynomials and link polynomials. *Proceedings of the American Mathematical Society*, 103(2):647–654, 1988. Available at https://doi.org/10.1090/S0002-9939-1988-0943099-0.

[Kel65] AK Kelmans. Some problems of network reliability analysis. *Automation and Remote Control*, 26(3):564, 1965.

[KL85] Richard M Karp and Michael Luby. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *Journal of Complexity*, 1(1):45–64, 1985. Available at https://doi.org/10.1016/0885-064X(85)90021-4.

[Kur05] James F Kurose. *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[LGW03] Alberto Leon-Garcia and Indra Widjaja. *Communication Networks*. McGraw-Hill, Inc., USA, 2 edition, 2003.

[LLCa] Python Software Foundation LLC. Python language reference, version 3.7. https://docs.python.org/3/ (last accessed on 18/03/2020).

[LLCb] Python Software Foundation LLC. Python multiprocessing. https://docs.python.org/2/library/multiprocessing.html (last accessed on 11/02/2020).

[LLCc] Python Software Foundation LLC. Python random library. https://docs.python.org/3/library/random.html (last accessed on 25/02/2020).

[LLCd] Python Software Foundation LLC. Time access and conversions. https://docs.python.org/3/library/time.html (last accessed on 25/02/2020).

[LPRS16] JL Lebowitz, Boris Pittel, D Ruelle, and ER Speer. Central limit theorems, Lee–Yang zeros, and graph-counting polynomials. *Journal of Combinatorial Theory, Series A*, 141:147–183, 2016. Available at https://doi.org/10.1016/j.jcta.2016.02.009.

[MCOD06] Richard C Murphy, Scott M Carter, Mario G Ornelas, and Shrikant Deshpande. *System and method for dynamic resource configuration using a dependency graph*. Google Patents, December 19 2006.

[Min59] Hisashi Mine. Reliability of a physical system. *IRE Transactions on Circuit Theory*, 6(5):138–151, 1959. Available at https://ieeexplore.ieee.org/abstract/document/1086604.

[Mos58] Fred Moskowitz. The analysis of redundancy networks. *Transactions of the American institute of electrical engineers, part i: communication and electronics*, 77(5):627–632, 1958. Available at https://ieeexplore.ieee.org/abstract/document/6372698.

[MS56] Edward F Moore and Claude E Shannon. Reliable circuits using less reliable relays. *Journal of the Franklin Institute*, 262(3):191–208, 1956. Available at https://doi.org/10.1016/0016-0032(56)90559-2.

[Pan94] Victor Y Pan. Simple multivariate polynomial multiplication. *Journal of Symbolic Computation*, 18(3):183–186, 1994. Available at https://doi.org/10.1006/jsco.1994.1042.

[PB83] J Scott Provan and Michael O Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983. Available at https://doi.org/10.1137/0212053.

[PP94] Lavon B Page and Jo Ellen Perry. Reliability polynomials and link importance in networks. *IEEE Transactions on Reliability*, 43(1):51–58, 1994. Available at https://ieeexplore.ieee.org/abstract/document/285108.

[PR17] Viresh Patel and Guus Regts. Deterministic polynomial-time approximation algorithms for partition functions and graph polynomials. *SIAM Journal on Computing*, 46(6):1893–1919, 2017. Available at https://doi.org/10.1137/16M1101003.

[PR18] Hebert Pérez-Rosés. Sixty years of network reliability. *Mathematics in Computer Science*, 12(3):275–293, 2018. Available at https://link.springer.com/article/10.1007/s11786-018-0345-5.

[PS86] Themistocles Politof and A Satyanarayana. Efficient algorithms for reliability analysis of planar networks-a survey. *IEEE Transactions on Relia-*

bility, 35(3):252–259, 1986. Available at https://ieeexplore.ieee.org/abstract/document/4335427.

[RKP86] Suresh Rai, Arun Kumar, and EV Prasad. Computing terminal reliability of computer network. *Reliability Engineering*, 16(2):109–119, 1986. Available at https://doi.org/10.1016/0143-8174(86)90079-X.

[RS02] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2002.

[RS04] Gordon Royle and Alan D Sokal. The Brown–Colbourn conjecture on zeros of reliability polynomials is false. *Journal of Combinatorial Theory, Series B*, 91(2):345–360, 2004. Available at https://doi.org/10.1016/j.jctb.2004.03.008.

[SF71] Richard Van Slyke and Howard Frank. Reliability of computer-communication networks. In *Proceedings of the 5th conference on Winter simulation*, pages 71–82, 1971. Available at https://doi.org/10.1145/800294.811426.

[SH81] A Satyanarayana and Jane N Hagstrom. Combinatorial properties of directed graphs useful in computing network reliability. *Networks*, 11(4):357–366, 1981. Available at https://doi.org/10.1002/net.3230110405.

[SI98] Kyoko Sekine and Hiroshi Imai. Computation of the network reliability. *University of Tokyo, Department of Information Science*, 1998. Available at https://www.researchgate.net/publication/240158906_Computation_of_the_Network_Reliability_Extended_Abstract.

[SW99] Angelika Steger and Nicholas C Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4):377–396, 1999. Available at https://pdfs.semanticscholar.org/a11c/daa94e777b0d9752a326224f742aa3f71c3b.pdf.

[Tan06] Christian Tanguy. Exact solutions for the two-and all-terminal reliabilities of a simple ladder network. *arXiv preprint cs/0612143*, 2006. Available at https://arxiv.org/abs/cs/0612143.

[Tan19] Diana Tanase. Experimental study of the roots of graph polynomials. School of Informatics, The University of Edinburgh, 2019. Available at https://project-archive.inf.ed.ac.uk/ug4/20191505/ug4_proj.pdf.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972. Available at https://doi.org/10.1137/0201010.

[Tit99] Peter Tittmann. Partitions and network reliability. *Discrete applied mathematics*, 95(1-3):445–453, 1999. Available at https://doi.org/10.1016/S0166-218X(99)00092-X.

[Val79] Leslie G Valiant. The complexity of computing the permanent. *Theoretical computer science*, 8(2):189–201, 1979. Available at https://doi.org/10.1016/0304-3975(79)90044-6.

[Ver05] Dirk Vertigan. The computational complexity of Tutte invariants for planar graphs. *SIAM Journal on Computing*, 35(3):690–712, 2005. Available at https://doi.org/10.1137/S0097539704446797.

[VSF71] R Van Slyke and Howard Frank. Network reliability analysis: Part I. *Networks*, 1(3):279–290, 1971. Available at https://doi.org/10.1002/net.3230010307.

[Wei] Eric Weisstein. Grid graph. MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/GridGraph.html (last accessed on 11/02/2020).

[Wei08] Eric Weisstein. Reliability polynomial. 2008. Available at https://mathworld.wolfram.com/ReliabilityPolynomial.html (last accessed on 05/03/2020).

[Wel01] Barry Wellman. Computer networks as social networks. *Science*, 293(5537):2031–2034, 2001. Available at https://science.sciencemag.org/content/293/5537/2031.

[WW93] Dominic Welsh and David J Welsh. *Complexity: knots, colourings and countings*, volume 186. Cambridge University Press, 1993.

[YKE13] Mina Youssef, Yasamin Khorramzadeh, and Stephen Eubank. Network reliability: The effect of local network structure on diffusive processes. *Physical Review E*, 88(5):052810, 2013. Available at https://doi.org/10.1103/PhysRevE.88.052810.

Appendices

Appendix A

Reliability polynomials for grids with $n \leq 5$

n	**Reliability polynomial**
2	$-3p^4+8p^3-6p^2+1$
3	$79p^{12}-560p^{11}+1668p^{10}-2656p^9+2331p^8-960p^7+96p^5+21p^4-16p^3-4p^2+1$
4	$-17493p^{24}+232144p^{23}-1409764p^{22}+5168576p^{21}-12693232p^{20}+21854512p^{19}-$ $-26726036p^{18}+22824576p^{17}-12739373p^{16}+3710880p^{15}+139672p^{14}-370176p^{13}-$ $-35464p^{12}+63968p^{11}+5912p^{10}-7808p^9-1791p^8+656p^7+204p^6+64p^5-8p^4-$ $-16p^3-4p^2+1$
5	$32126211p^{40}-681809240p^{39}+6852471548p^{38}-43322118652p^{37}+192968405711p^{36}-$ $-642590690400p^{35}+1655933457966p^{34}-3370276114636p^{33}+5476061558391p^{32}-$ $-7122774813980p^{31}+7375859530466p^{30}-5981426876044p^{29}+3667377815630p^{28}-$ $-1573096624396p^{27}+375423772810p^{26}+9584416484p^{25}-26112103320p^{24}-$ $-6268146140p^{23}+8011274210p^{22}-1051500660p^{21}-575028980p^{20}-$ $-53196700p^{19}+139031550p^{18}-2265380p^{17}-10705120p^{16}-3593556p^{15}+$ $+1357510p^{14}+394172p^{13}+35042p^{12}-49636p^{11}-10290p^{10}-2036p^9+1021p^8+$ $+164p^7+250p^6+64p^5-11p^4-20p^3-4p^2+1$

Table A.1: Reliability polynomials for n x n grids [SI98]

Appendix B

Reliability polynomials for complete graphs with $n \leq 6$

n	**Reliability polynomial**
2	$-p+1$
3	$2p^3-3p^2+1$
4	$-6p^6+12p^5-3p^4-4p^3+1$
5	$24p^{10}-60p^9+30p^8+20p^7-10p^6-5p^4+1$
6	$-120p^{15}+360p^{14}-270p^{13}-90p^{12}+120p^{11}+20p^9-15p^8-6p^5+1$

Table B.1: Reliability polynomials for n-complete graphs [SI98]

Appendix C

Proof of minimum constant in the number of samples expression

We want to find the minimum positive integer c such that

$$e^{x/c} \leq 1 + \frac{x}{20},$$

for $x \in [0,1]$. Let's consider the function $f_c : [0,1] \to R$, $f_c(x) = e^{x/c} - 1 - \frac{x}{20}$. We need to find minimum positive integer c such that $f_c(x) \leq 0$ for any $x \in [0,1]$.

Firstly, it must be true that $f_c(0) \leq 0$ and $f_c(1) \leq 0$.

$$f_c(0) = 1 - 1 - 0 = 0 \leq 0.$$

$$f_c(1) = e^{1/c} - \frac{21}{20}.$$

$$f_c(1) \leq 0 \Rightarrow e^{1/c} \geq \frac{21}{20} \Rightarrow \frac{1}{c} \leq \ln\frac{21}{20} \Rightarrow c \geq \frac{1}{\ln\frac{21}{20}} \approx 20.5$$

Thus, since c must be an integer, $c \geq 21$. We will prove now that $c = 21$ does satisfy our requirements.

Define $f : [0,1] \to R, f(x) = e^{1/21} - 1 - \frac{x}{20}$. It is obvious that f is a continuous function. We analyse the monotony of f using the first derivative.

$$f'(x) = \frac{1}{21}e^{x/21} - \frac{1}{20}.$$

We find the zeros of f' to prove that f is strictly decreasing.

$$f'(x) = 0 \Rightarrow \frac{1}{21}e^{x/21} = \frac{1}{20} \Rightarrow e^{x/21} = \frac{21}{20} \Rightarrow \frac{x}{21} = \ln\frac{21}{20} \Rightarrow x = 21\ln\frac{21}{20} > 1.$$

Thus, for $x \in [0,1]$, $f'(x) = 0$ does not have a solution and hence, f' does not change its (positive/negative) sign. Moreover, f' is the a strictly increasing function (being composed of an exponential and a constant function), which implies that for $x \leq 1 < 21\ln\frac{21}{20}$, $f'(x) < 0$. Hence, f is strictly decreasing and for $x \in [0,1]$:

$$f(1) \leq f(x) \leq f(0) \Rightarrow f(x) \leq 0 \Rightarrow e^{x/21} \leq 1 + \frac{x}{20}.$$

Appendix D

Number of samples for a grid

n	$p = 0.1$ $s = \frac{70000}{27}(n^2-1)$	$p = 0.25$ $s = \frac{11200}{3}(n^2-1)$	$p = 0.5$ $s = 8400(n^2-1)$	$p = 0.75$ $s = 33600(n^2-1)$	$p = 0.9$ $s = 210000(n^2-1)$
2	7778	11200	25200	100800	630000
3	20741	29867	67200	268800	1680000
4	38889	56000	126000	504000	3150000
5	62223	89600	201600	806400	5040000
6	90741	130667	294000	1176000	7350000
7	124445	179200	403200	1612800	10080000
8	163334	235200	529200	2116800	13230000
9	207408	298667	672000	2688000	16800000
10	256667	369600	831600	3326400	20790000
11	311112	448000	1008000	4032000	25200000
12	370741	533867	1201200	4804800	30030000
13	435556	627200	1411200	5644800	35280000
14	505556	728000	1638000	6552000	40950000
15	580741	836267	1881600	7526400	47040000

Table D.1: Number of samples for an n x n grid for estimating reliability with $\varepsilon = 0.1$ and 95% confidence.

Note: For running the Reliability Module using parallelism of the consecutive sampling runs, we used slightly more samples for $p = 0.1$ and $p = 0.25$, to be able to assign an equal number of tasks to processes:

- $2593(n^2-1)$ for $p = 0.1$.
- $3734(n^2-1)$ for $p = 0.25$.

Appendix E

Recursive and Iterative Sampling time

Tables E.1 present the running time to generate a root-connected subgraph using the recursive and iterative implementations, averaged over 10 runs for each grid.

n	recursive	iterative
2	0.03	0.04
3	0.14	0.12
4	0.30	0.45
5	0.53	0.67
6	0.82	1.13
7	1.76	3.42
8	1.47	1.56
9	2.07	3.91
10	2.76	3.22
11	4.47	4.10
12	5.24	4.65
13	5.05	5.73
14	4.00	7.56
15	9.31	6.15

n	recursive	iterative
16	7.88	7.89
17	10.75	13.23
18	15.41	11.10
19	11.87	9.47
20	9.33	8.54
21	11.85	12.34
22	13.92	13.73
23	34.18	31.55
24	39.05	12.31
25	23.56	19.80
26	15.01	15.95
27	15.12	18.24
28	23.33	19.63
29	22.27	27.84
30	23.30	26.54

Table E.1: Average time in ms to generate 10 root-connected subgraphs for each n-grid

Appendix F

Exact and approximated reliability for path, pan, star, cycle and ladder and complete graphs

Tables F.1 to F.6 present for different types of graphs, for size $2 \leq n \leq 8$, the exact reliability values computed using the corresponding reliability polynomial and the approximated results of our Reliability Module, for failure probability $p = 0.5$. All estimated values are within the (1 ± 0.1) accuracy interval.

n	exact Z_{rel}	approx. Z_{rel}
2	0.5000	0.4951
3	0.2500	0.2468
4	0.1250	0.1236
5	0.0625	0.0637
6	0.0312	0.0305
7	0.0156	0.0157
8	0.0078	0.0078

Table F.1: Reliability of path graph

n	exact Z_{rel}	approx. Z_{rel}
2	0.3750	0.3777
3	0.2500	0.2535
4	0.1562	0.1564
5	0.0938	0.0931
6	0.0547	0.0544
7	0.0312	0.0315
8	0.0176	0.0176

Table F.2: Reliability of pan graph

n	exact Z_{rel}	approx. Z_{rel}
2	0.5000	0.4975
3	0.2500	0.2526
4	0.1250	0.1217
5	0.0625	0.0623
6	0.0312	0.0312
7	0.0156	0.0160
8	0.0078	0.0077

Table F.3: Reliability of star graph

n	exact Z_{rel}	approx. Z_{rel}
2	0.7500	0.7501
3	0.5000	0.5031
4	0.3125	0.3112
5	0.1875	0.1892
6	0.1094	0.1090
7	0.0625	0.0620
8	0.0352	0.0357

Table F.4: Reliability of cycle graph

n	exact Z_{rel}	approx. Z_{rel}
2	0.3125	0.3096
3	0.1797	0.1784
4	0.1025	0.1039
5	0.0585	0.0581
6	0.0333	0.0329
7	0.0190	0.0190
8	0.0108	0.0108

Table F.5: Reliability of ladder graph

n	exact Z_{rel}	approx. Z_{rel}
2	0.5000	0.4924
3	0.5000	0.4988
4	0.5938	0.5930
5	0.7109	0.7094
6	0.8149	0.8136
7	0.8899	0.8901
8	0.9371	0.9367

Table F.6: Reliability of complete graph

Appendix G

Sampling running time on grids

.

n	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.5$	$p = 0.75$
5	0.1396	0.2043	0.3253	0.817	2.8885
10	0.4747	0.4806	1.687	3.6984	9.8295
15	1.2891	1.6819	3.2282	9.1157	28.4469
20	2.8081	2.1498	4.6596	17.5798	62.4887
25	3.3146	3.481	7.0673	28.333	80.8582
30	5.2845	4.7295	10.1825	77.0745	133.7406
35	8.4184	6.7324	15.9345	61.8941	197.8669
40	10.0981	8.9303	22.7832	119.9797	272.332
45	11.7738	15.8979	30.2179	122.9136	352.739
50	14.8631	17.5263	40.8679	175.9184	632.1395
55	17.9069	22.1209	59.8267	226.0938	628.7326
60	23.0718	25.1502	63.598	341.9351	710.7768
65	28.4645	37.2717	83.7792	364.8171	958.4511
70	31.653	37.3352	106.1449	435.6148	1400.0201
75	33.7771	53.9339	135.9878	580.8345	1453.6386
80	41.6784	57.4063	168.4629	647.905	1882.0843
85	60.3112	64.4347	207.0318	781.7487	-
90	67.9382	64.5729	251.3356	1029.5263	-
95	91.3844	74.6063	309.3275	1114.6956	-
99	75.409	78.9209	391.5794	1327.9111	-

Table G.1: Running time of the iterative sampling algorithm (ms), averaged over 1000 runs for each (n, p))

Appendix H

Number arcs resampled on grids

.

n	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.9$
5	1.371; 0.064	5.388; 0.716	45.965; 7.18	205.821; 46.699	947.192; 255.579
10	0.212; 0.072	7.208; 1.35	150.055; 18.375	973.113; 172.769	3314.376; 800.673
15	0.408; 0.128	26.61; 2.156	294.213; 33.545	2365.762; 382.072	9654.624; 2147.673
20	0.496; 0.148	44.871; 3.215	544.568; 54.93	4352.789; 666.406	21204.055; 4514.632
25	0.678; 0.207	35.267; 4.475	738.511; 77.728	6830.664; 1015.608	26173.482; 5613.835
30	0.767; 0.228	32.028; 5.943	951.135; 105.409	20055.73; 2537.648	43020.469; 9011.312
35	1.084; 0.308	50.579; 7.537	1358.174; 141.775	13410.039; 1932.493	61934.721; 12805.714
40	1.338; 0.38	65.767; 9.623	1740.488; 179.981	27297.057; 3524.505	83309.886; 17088.454
45	25.363; 0.447	81.503; 11.757	2027.55; 219.688	23750.946; 3303.441	103160.007; 21101.145
50	1.835; 0.501	98.736; 13.966	2330.472; 263.344	32994.549; 4423.833	190503.068; 37832.41
55	2.122; 0.571	94.6; 16.562	5398.234; 369.61	40246.862; 5360.041	177148.289; 35604.289
60	44.694; 0.609	124.846; 19.353	3301.052; 370.598	46639.726; 6216.363	186735.555; 37757.098

Table H.1: Average arcs resampled (first value) and minimal clusters (second value) found for 1000 runs of the sampling algorithm

n	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.9$
65	2.666; 0.714	210.787; 22.304	4142.571; 436.033	53235.367; 7110.675	251074.101; 50191.062
70	2.924; 0.794	205.207; 25.399	4584.519; 496.898	64214.228; 8478.652	306402.338; 60943.101
75	3.197; 0.861	279.523; 29.226	5696.192; 578.644	81567.517; 10511.982	330770.209; 65920.58
80	3.94; 1.05	215.381; 32.578	6168.17; 648.849	78710.162; 10487.729	430438.485; 84996.908
85	4.236; 1.13	297.712; 36.268	6707.124; 724.511	87246.122; 11650.08	-
90	4.623; 1.229	1037.56; 41.309	7151.051; 800.715	120810.621; 15358.169	-
95	5.176; 1.376	1049.414; 45.699	8614.663; 900.203	102947.263; 13880.075	-
99	5.708; 1.505	362.66; 48.291	9583.923; 980.664	119809.253; 15864.103	-

Table H.2: Average arcs resampled (first value) and minimal clusters (second value) found for 1000 runs of the sampling algorithm

Appendix I

Reliability and growth rate of path, pan, ladder and complete graphs

graph	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.9$
path	3.2791e-05; -0.1504	5.7016e-13; -0.4108	3.1554e-30; -0.9898	9.9568e-60; -1.9797	9.9999e-99; -3.2883
pan	0.0003; -0.1173	1.0904e-11; -0.3678	7.8886e-29; -0.9429	1.8544e-58; -1.9371	8.9199e-98; -3.2564
ladder	0.2545; -0.0099	1.7829e-05; -0.0796	6.8090e-25; -0.4054	1.5482e-72; -1.2047	3.9383e-145; -2.4227
complete	1; 0	1; 0	1; 0	0.9999; -8.3088e-15	0.9967; -4.7788e-07

Table I.1: Reliability (first value) and logarithmic reliability (second value) for $n = 100$

Appendix J

Reliability results on grids

n	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.9$
2	0.9449; -0.0204	0.7445; -0.1064	0.3122; -0.4198	0.0507; -1.0752	0.0037; -2.0191
3	0.9490; -0.0083	0.6448; -0.0703	0.1052; -0.3608	0.0012; -1.0743	1.3904e-06; -2.1617
4	0.9427; -0.0053	0.5834; -0.0485	0.0329; -0.3076	1.2449e-05; -1.01834	4.7148e-11; -2.1439
5	0.9395; -0.0035	0.5247; -0.0372	0.0093; -0.2695	5.2457e-08; -0.9673	1.4196e-16; -2.1058
6	0.9344; -0.0027	0.4684; -0.0303	0.0023; -0.2428	8.8857e-11; -0.9274	3.7862e-23; -2.0689
7	0.9306; -0.0021	0.4188; -0.0256	0.0005; -0.2234	6.1150e-14; -0.8958	8.7014e-31; -2.0379
8	0.9242; -0.0017	0.3699; -0.0224	9.6278e-05; -0.2084	1.6571e-17; -0.8710	1.7698e-39; -2.0114
9	0.9197; -0.0014	0.3241; -0.0200	1.5808e-05; -0.1969	1.7747e-21; -0.8510	-
10	0.9134; -0.0013	0.2805; -0.0183	2.2251e-06; -0.1877	7.7690e-26; -0.8341	-
11	0.90910; -0.0011	0.2418; -0.0169	2.7144e-07; -0.1802	-	-
12	0.9029; -0.0010	0.2057; -0.0158	2.8421e-08; -0.1740	-	-
13	0.8963; -0.0009	0.1746; -0.0149	2.5626e-09; -0.1688	-	-
14	0.8919; -0.0008	0.1465; -0.0141	1.9697e-10; -0.1644	-	-
15	-	-	1.3026e-11; -0.1607	-	-

Table J.1: Reliability (first value) and logarithmic reliability (second value) of n x n grids

Appendix K

Plots for reliability and logarithmic growth rate

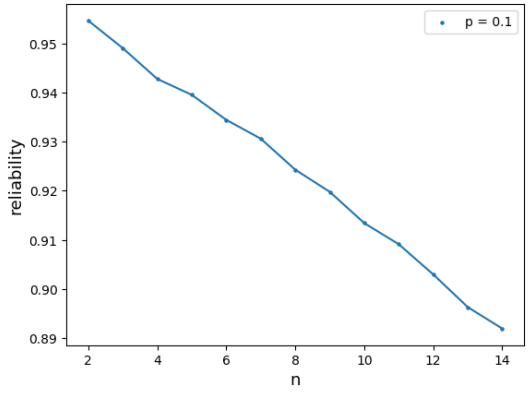

Figure K.1: Reliability of grids, $p = 0.1$

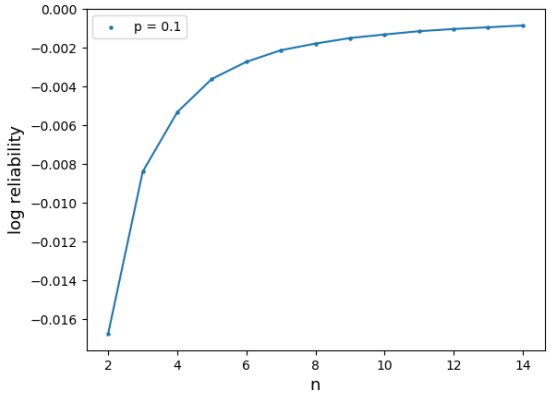

Figure K.2: Log. reliability of grids, $p = 0.1$

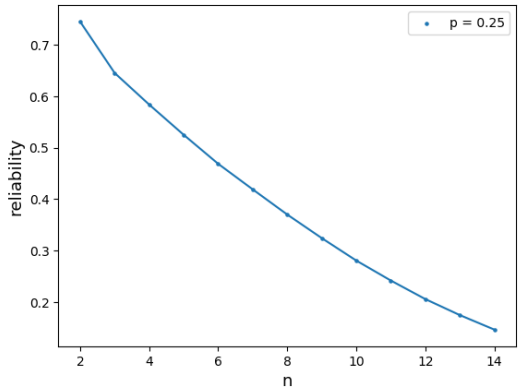

Figure K.3: Reliability of grids, $p = 0.25$

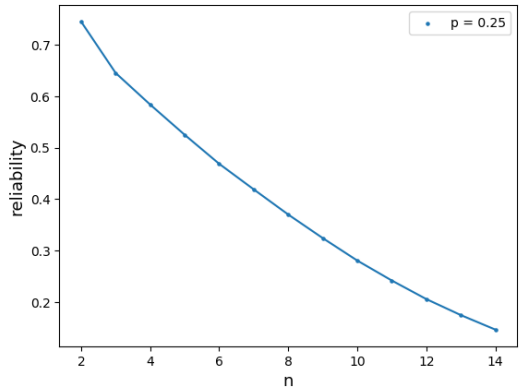

Figure K.4: Log.reliability of grids,$p = 0.25$

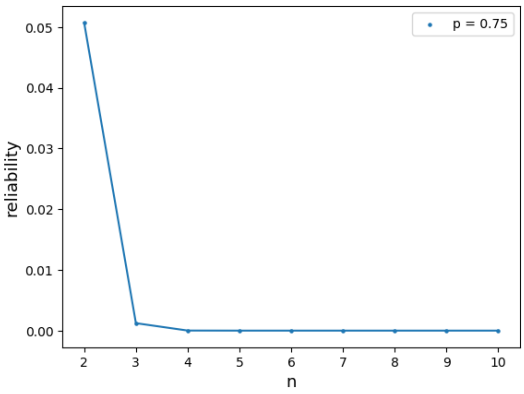

Figure K.5: Reliability of grids, $p = 0.75$

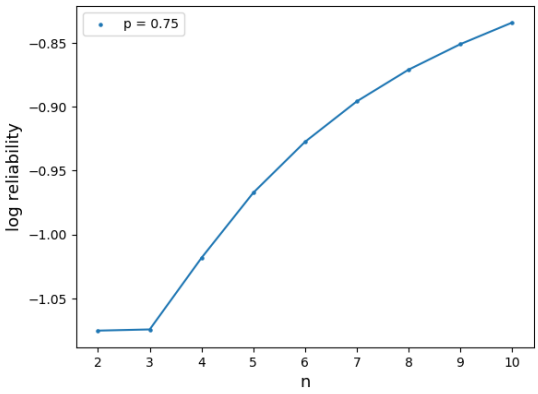

Figure K.6: Log.reliability of grids,$p = 0.75$

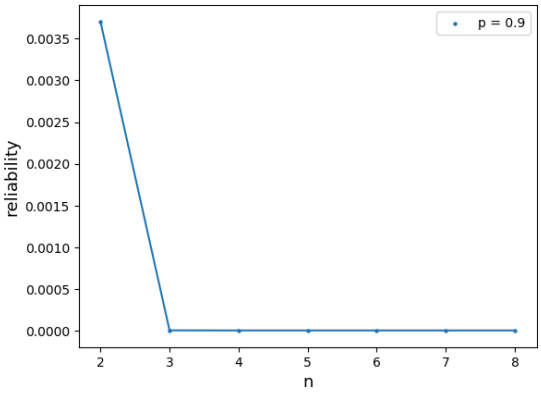

Figure K.7: Reliability of grids, $p = 0.9$

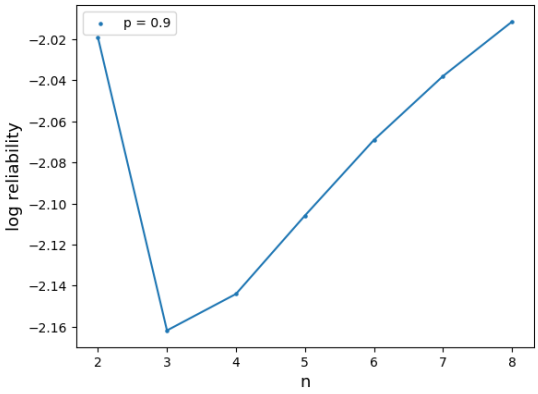

Figure K.8: Log. reliability of grids, $p = 0.9$

Appendix L

Distance from reliability to upper bound

n	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.9$
2	6.6226e-05	0.0008	0.0027	0.0012	0.0001
3	0.00010	0.0029	0.0063	0.0002	9.8319e-07
4	0.00012	0.0025	0.0035	4.1255e-06	4.7667e-11
5	0.00017	0.0031	0.0011	4.2837e-08	6.7890e-16
6	0.00019	0.0049	0.0004	1.9416e-10	4.6010e-22
7	0.00017	0.0040	0.0001	2.0739e-13	1.4567e-29
8	0.00019	0.0037	3.3425e-05	9.6873e-17	1.3466e-37
9	0.00022	0.0041	6.5760e-06	1.6373e-20	-
10	0.00020	0.0037	1.0469e-06	1.4488e-24	-
11	0.00020	0.0042	1.6189e-07	-	-
12	0.00022	0.0035	1.8048e-08	-	-
13	0.00021	0.0027	2.0354e-09	-	-
14	0.00023	0.0026	1.8166e-10	-	-
15	-	-	1.4674e-11	-	-

Table L.1: Distance from reliability to upper bound on n x n grids

Appendix M

Percentage of average connected expansions out of number of samples

n	$p = 0.1$	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 0.9$
2	0.9845; -0.0056	0.9066; -0.03532	0.6803; -0.1389	0.3731; -0.3555	0.1566; -0.6685
3	0.9934; -0.0010	0.9471; -0.0087	0.7602; -0.0439	0.4418; -0.1309	0.1981; -0.2595
4	0.9960; -0.0003	0.9649; -0.0032	0.8020; -0.0198	0.4800; -0.0661	0.2146; -0.1387
5	0.9974; -0.0001	0.9737; -0.0015	0.8271; -0.0109	0.5098; -0.0388	0.2351; -0.0835
6	0.9980; -7.7399e-05	0.9788; -0.0008	0.8452; -0.0067	0.5335; -0.0251	0.2462; -0.0561
7	0.9985; -4.3999e-05	0.9822; -0.0005	0.8581; -0.0045	0.5471; -0.0177	0.2510; -0.0406
8	0.9987; -2.8097e-05	0.9844; -0.0003	0.8675; -0.0032	0.5583; -0.0131	0.2599; -0.0303
9	0.9989; -1.8562e-05	0.9861; -0.0002	0.8747; -0.0023	0.5665; -0.0101	-
10	0.9990; -1.3165e-05	0.9873; -0.0001	0.8802; -0.0018	0.5746; -0.0079	-
11	0.9992; -9.4456e-06	0.9883; -0.0001	0.8850; -0.0014	-	-
12	0.9992; -7.1352e-06	0.9891; -0.0001	0.8886; -0.001	-	-
13	0.9993; -5.5504e-06	0.9897; -8.7863e-05	0.8920; -0.0009	-	-
14	0.9994; -4.3070e-06	0.9902; -7.1815e-05	0.8947; -0.0008	-	-

Table M.1: Percentage and logarithmic growth of connected expansions out of s