An Efficient Algorithm for Computing Network Reliability in Small Treewidth

Amir Kafshdar Goharshady

IST Austria (Institute of Science and Technology Austria)
Am Campus 1, 3400 Klosterneuburg, Austria
amir.goharshady@ist.ac.at

Fatemeh Mohammadi

School of Mathematics, University of Bristol
University Walk, Bristol, BS8 1TW, United Kingdom
fatemeh.mohammadi@bristol.ac.uk

Abstract

We consider the classic problem of Network Reliability. A network is given together with a source vertex, one or more target vertices, and probabilities assigned to each of the edges. Each edge appears in the network with its associated probability and the problem is to determine the probability of having at least one source-to-target path. This problem is known to be NP-hard.

We present a linear-time fixed-parameter algorithm based on a parameter called treewidth, which is a measure of tree-likeness of graphs. Network Reliability was already known to be solvable in polynomial time for bounded treewidth, but there were no concrete algorithms and the known methods used complicated structures and were not easy to implement. We provide a significantly simpler and more intuitive algorithm that is much easier to implement.

We also report on an implementation of our algorithm and establish the applicability of our approach by providing experimental results on the graphs of subway and transit systems of several major cities, such as London and Tokyo. To the best of our knowledge, this is the first exact algorithm for Network Reliability that can scale to handle real-world instances of the problem.

Keywords: Network Reliability, Fixed-parameter Algorithms, Tree Decomposition, Treewidth, FPT

1. Introduction

Network Reliability. Consider a network modeled as a graph $G = (V, E)$, where each edge $e \in E$ has a known probability of failure. For example, the graph might be a model of communication links in a mobile network or railway lines between subway stations. Given a source vertex $\mathfrak{s}$ and a set $\mathfrak{T}$ of target vertices, the goal of the Network Reliability problem is to assess the reliability of connections between $\mathfrak{s}$ and $\mathfrak{T}$. Concretely, the Network Reliability problem asks for the probability of existence of at least one source-to-target path that does not pass through failed edges. In the examples mentioned above, this is equivalent to asking for the probability of being able to send a message from $\mathfrak{s}$ to $\mathfrak{T}$ through the mobile network or the probability of being able to travel from $\mathfrak{s}$ to $\mathfrak{T}$ in the subway network.

Short History. Network reliability is an important and well-studied problem with surveys appearing as early as 1983 [1]. Aside from the obvious applications such as the two mentioned above, the problem has many other surprising applications, including analysis and elimination of redundancy in electronic systems and electrical power networks [2]. Network Reliability was shown to be NP-hard for general graphs [3] and hence researchers turned to solving it in special cases [3, 1], such as series-parallel graphs [4] and graphs with limited number of cuts [5]. Genetic [6], randomized [7], approximate [8] and Monte Carlo [9] algorithms

are studied extensively as well. There are also several algebraic studies of the problem with the goal of obtaining bounds in series-parallel and other special families of graphs [10, 11, 12, 13, 14]. Several variants of the problem are defined [15, 16, 17], and approaches to modify the network for its optimization are also investigated [18]. In this paper, we consider a parameterization of Network Reliability and obtain a linear-time algorithm.

Parameterized Algorithms. An efficient *parameterized* algorithm solves an optimization problem in polynomial time with respect to the size of input, but possibly with non-polynomial dependence on a specific aspect of the input's structure, which is called a "parameter" [19]. A problem that can be solved by an efficient parameterized algorithm is called *fixed-parameter tractable* (FPT). For example, there is a polynomial-time algorithm for computing minimal cuts in graphs whose runtime is exponentially dependent on the size of the resulting cut [20]. Exploiting the additional benefit of having a parameter, parameterized complexity provides finer detail than classical complexity theory [21].

Treewidth. A well-studied parameter for graphs is the *treewidth*, which is a measure of tree-likeness of graphs [22]. Many hard problems are shown to have efficient solutions when restricted to graphs with small treewidth [23, 24, 25, 26, 27, 28, 29]. Notably, [30] introduces a general framework that shows several variants of the network reliability problem can be solved in polynomial time when parameterized by the treewidth. Many real-world graphs happen to have small treewidth [31, 32, 33, 34]. In this work, we show that subway and transit networks often have this property.

Our contribution. Our contribution is providing a new fixed-parameter algorithm for finding the exact value of Network Reliability, using treewidth as the parameter. Our algorithm, while being linear-time, is much shorter and simpler than the general framework utilized in [30]. We also provide an implementation of our algorithm and experimental results over the graphs of subway networks of several major cities. To the best of our knowledge, this is the first algorithm for finding the exact value of Network Reliability that can scale to handle real-world instances, i.e. subway and transit networks of major cities.

Graphs with constant treewidth are the most general family of networks for which exact algorithms for computing reliability are found. This family contains trees, series-parallel graphs and outerplanar graphs [31].

Structure of the Paper. The present paper is organized as follows: First, Section 2 provides formal definitions of the Network Reliability problem and Treewidth. Then, Section 3, which is the main part of the paper, presents our simple linear algorithm for solving Network Reliability in graphs with constant treewidth. Section 4 contains a report of our implementation, which is publicly available, and establishes the applicability of our approach by providing experimental results on real-world subway networks.

2. Preliminaries

In this section, we formalize our notation, and define the problem of Network Reliability and the notion of treewidth.

Multigraphs. A multigraph is a pair $G = (V, E)$ where V is a finite set of vertices and E is a finite multiset of edges, i.e. each $e \in E$ is of the form $\{u, v\}$ for $u, v \in V$. The vertices u and v are called the endpoints of e. Note that E might contain distinct edges that have the same endpoints. In the sequel, we only consider multigraphs and simply call them graphs for brevity.

Notation. Given a graph $G = (V, E)$, a *path* from $u \in V$ to $w \in V$ is a finite sequence $u = u_0, u_1, \ldots, u_l = w$ of distinct vertices such that for every $i < l$, there exists an edge $e = \{u_i, u_{i+1}\} \in E$. We write $u \rightsquigarrow_G w$ to denote the existence of a path from u to w in G. We simply write $u \rightsquigarrow w$ if G can be deduced from the context. For a set $E' \subseteq E$, we write $u \rightsquigarrow_{E'} w$ if there exists a path from u to w whose every edge is in E'. A *connected component* of a graph G is a maximal subset $C \subseteq V$ such that for every $c_1, c_2 \in C$, we have $c_1 \rightsquigarrow_G c_2$. The graph G is called *connected* if it has exactly one connected component. A *cycle* in the graph G is a sequence $w_0, w_1, \ldots, w_l$ of vertices with $l > 0$, such that for every $i < l$, there exists an edge $e = \{w_i, w_{i+1}\} \in E$ and all w_i are distinct except that $w_0 = w_l$. A graph is called a *forest* if it has no cycles. A forest is called a *tree* if it is connected. In other words, a tree is a connected graph with no cycles.

Network Reliability Problem. A Network Reliability problem *instance* is a tuple $I = (G, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr})$ where $G = (V, E)$ is a connected multigraph, $\mathfrak{s} \in V$ is a "source" vertex and $\mathfrak{T} \subseteq V$ a set of "target" vertices. $\mathtt{Pr}$ is a function of the form $\mathtt{Pr} : E \to [0, 1]$ which assigns a probability to every edge of the graph G. The reliability problem on instance I is then defined as follows: A new graph G^s is probabilistically constructed such that its vertex set is V and each edge $e \in E$ appears in it with probability $\mathtt{Pr}(e)$. Appearance of the edges are stochastic and independent of each other. The Network Reliability problem asks for the probability $\mathfrak{Rel}(I)$ of having at least one path from the source vertex $\mathfrak{s}$ to a target vertex $\mathfrak{t} \in \mathfrak{T}$ in G^s.

Remark. We are describing our approach on undirected graphs. However, it is straightforward to change all the steps of the algorithm to handle direcetd graphs as well.

We now provide a quick overview of the basics of tree decompositions and treewidth. A much more involved treatment can be found in [20, 31].

Tree Decompositions. Given a connected multigraph $G = (V, E)$, a tree $T = (\mathfrak{B}, E_T)$ with vertex set $\mathfrak{B}$ and edge set E_T is called a tree decomposition of G if the following four conditions hold:

- Each vertex $b \in \mathfrak{B}$ of the tree T has an assigned set of vertices $V(b) \subseteq V$. To distinguish vertices of T and G, we call each vertex of T a *bag*.
- Each vertex appears in some bag, i.e. $\bigcup_{b \in \mathfrak{B}} V(b) = V$.
- Each edge appears in some bag, i.e. $\forall e = \{u, v\} \in E \;\; \exists b \in \mathfrak{B}$ s.t. $\{u, v\} \subseteq V(b)$. We denote the set of edges that appear in a bag b with $E(b)$. Note that an edge appears in b if and only if both of its vertices do.
- Each vertex $v \in V$ appears in a connected subtree of T. More precisely, we let $\mathfrak{B}_v$ to be the set of bags whose vertex sets contain v, then $\mathfrak{B}_v$ must be a connected subtree of T.

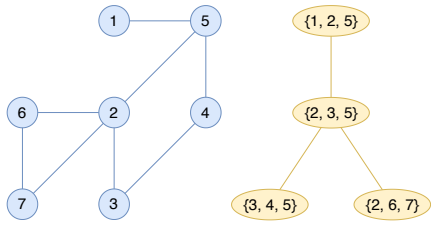

Figure 1: A graph (left) and a 2-decomposition of it (right)

See Figure 1 for an example. A rooted tree decomposition is a tree decomposition in which a unique bag is specified as "root". Given two bags b and b', we say that b is an *ancestor* of b' if b appears in the unique path from the root to b'. In this case, we say that b' is a *descendant* of b. Note that each bag is both an ancestor and a descendant of itself. The bag b is called the *parent* of b' if it is an ancestor of b' and has an edge to b', i.e. $\{b, b'\} \in E_T$. In this case, we say that b' is a *child* of b. A bag b with no children is called a *leaf*.

Treewidth. If a tree decomposition T has bags of size at most $k + 1$, then it is called a k-decomposition or a decomposition of *width* k. The *treewidth* of a graph G is defined as the smallest k for which a k-decomposition of G exists. Intuitively, the treewidth of a graph measures how tree-like it is and graphs with smaller treewidth are more similar to trees.

Cut Property. Tree decompositions are important for algorithm design because removing the vertices of each bag b from the original graph G cuts it into connected components corresponding to the subtrees formed in T by removing b [31]. We call this the "cut property" and it allows bottom-up dynamic programming algorithms to operate on tree decompositions almost the same way as in trees [27]. We use this property in our algorithm in Section 3. We now formalize this point:

Separators. Given a graph $G = (V, E)$ and two sets of vertices $A, B \subseteq V$, we call the pair (A, B) a separation of G if (i) $A \cup B = V$, and (ii) no edge connects a vertex in $A \setminus B$ to a vertex in $B \setminus A$. We call $A \cap B$ the separator corresponding to the separation (A, B).

Lemma 1 (Cut Property [20]). *Let $T = (\mathfrak{B}, E_T)$ be a tree decomposition of the graph G and let $e = \{a, b\} \in E_T$ be an edge of T. By removing e, T breaks into two connected components, T^a and T^b, respectively containing a and b. Let $A = \bigcup_{t \in T^a} V(t)$ and $B = \bigcup_{t \in T^b} V(t)$. Then (A, B) is a separation of G with separator $V(a) \cap V(b)$.*

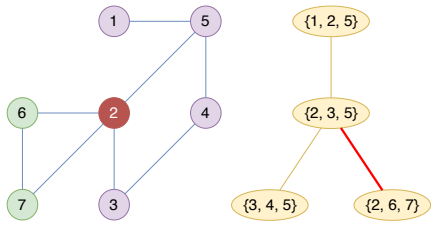

Figure 2: Cut Property

For example, consider the same graph as in Figure 1. By removing the edge from $\{2, 3, 5\}$ to $\{2, 6, 7\}$, the tree decomposition breaks into two connected components, containing the vertices $A = \{2, 6, 7\}$ and $B = \{1, 2, 3, 4, 5\}$, respectively. This is a separation of the graph with separator $\{2\}$. These points are illustrated in Figure 2.

Computing a Tree Decomposition. In our algorithm in Section 3, when we operate on a graph G with n vertices and constant treewidth, we assume that we are also given a tree decomposition of G as part of the input. This is justified by an algorithm of Bodlaender [35], that given a graph G and a constant k, decides in *linear time* whether G has treewidth at most k and if so, produces a k-decomposition of G with $O(n \cdot k)$ bags.

3. Algorithm for Network Reliability

In this section, we provide an algorithm for solving instances of the Network Reliability problem on graphs based on their tree decompositions.

Specification. The input to the algorithm is a Network Reliability instance $I = (G, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr})$ together with a k-decomposition $T = (\mathfrak{B}, E_T)$ of the graph G. The output is the reliability $\mathfrak{Rel}(I)$, i.e. the probability of existence of a path from $\mathfrak{s}$ to $\mathfrak{T}$. Given that the tree decomposition can be rooted at any bag, without loss of generality, we assume that the source vertex $\mathfrak{s}$ is in the root bag. We also assume that G has n vertices and $|\mathfrak{B}| \in O(n \cdot k)$. This can be obtained by an algorithm described in [35].

Methodology. Our algorithm is based on a technique called "kernelization" [20]: Using the tree-decomposition T, we repeatedly shrink the graph G to obtain smaller graphs that all have the same reliability as G. We continue our shrinking until we reach a graph that has very few vertices, i.e. at most $O(k)$ vertices. We then use brute force to compute the reliability of this graph.

Discrete Probability Distributions. Given a finite set X, a *probability distribution* over X is a function $\mathtt{Pr} : X \to [0, 1]$, that assigns a probability to each member of X, such that $\sum_{x \in X} \mathtt{Pr}(x) = 1$.

We first define an extension of the Network Reliability problem, in which the probabilities of appearance of the edges need not be independent anymore, i.e. some edges are *correlated*. Although this extension makes the problem more general, it helps in finding a solution. As we will later see, it allows us to apply a shrinking procedure as described above.

Extended Network Reliability. An Extended Network Reliability instance with r parts is a tuple $I = (G, E_1, \ldots, E_r, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr}_1, \ldots, \mathtt{Pr}_r)$ in which:

- $G = (V, E)$ is a connected graph;
- The E_i's are pairwise disjoint multisets of edges and $\bigcup_{i=1}^{r} E_i = E$;
- $\mathfrak{s} \in V$ is the source vertex;
- $\mathfrak{T} \subseteq V$ is the set of target vertices; and
- Each $\mathtt{Pr}_i : 2^{E_i} \to [0, 1]$ is a probability distribution over the subsets of E_i.

We now define the Extended Network Reliability problem on the instance I as follows: a new graph G^s is probabilistically constructed such that its vertex set is V and its edge set is a subset $E^s \subseteq E$ chosen probabilistically as follows:

- For every part E_i, a subset $E_i^s \subseteq E_i$ of edges is probabilistically chosen according to the distribution $\mathtt{Pr}_i$. The E_i^s's are chosen independently of each other.
- The set E^s is defined as $E^s = \bigcup_{i=1}^{r} E_i^s$.

The Extended Network Reliability problem asks for the probability $\mathfrak{Rel}(I)$ that the probabilistically-constructed graph G^s contains a path from $\mathfrak{s}$ to $\mathfrak{T}$. Intuitively, appearance of every edge in each part E_i is correlated to every other edge in E_i, but independent of all the edges outside of E_i. A Network Reliability instance is simply an Extended Network Reliability instance in which each E_i consists of a single edge, i.e. every edge is independent of every other edge.

We now provide a simple brute force algorithm for the Extended Network Reliability problem. This algorithm will later serve as a subprocedure in our main algorithm.

The Brute Force Algorithm. Consider an Extended Network Reliability instance I as above and a graph $G' = (V, E')$ where $E' \subseteq E$, i.e. a graph with the same set of vertices as G, but only a subset of its edges. We can easily compute $\mathbb{P}(G^s = G')$, i.e. the probability that the probabilistically-constructed graph G^s is equal to G'. We use each $\mathtt{Pr}_i$ to find the probability of the specific combination of correlated edges that are present in $E' \cap E_i$. Therefore, we have:

$$\mathbb{P}(G^s = G') = \prod_{i=1}^{r} \mathtt{Pr}_i(E' \cap E_i).$$

Now $\mathfrak{Rel}(I)$ is simply the sum of $\mathbb{P}(G^s = G')$ over those graphs G' in which there is a path from $\mathfrak{s}$ to $\mathfrak{T}$. Hence, we can use the brute force method as in Algorithm 1 for answering the Extended Network Reliability problem. The algorithm creates all possible subgraphs G' and checks if there is a path from $\mathfrak{s}$ to $\mathfrak{T}$ in G'. If so, it computes the probability $\mathbb{P}(G^s = G')$. Finally, it returns the sum of computed probabilities.

Algorithm 1: The brute force method

Input : An Extended Network Reliability instance $I = (G, E_1, \ldots, E_r, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr}_1, \ldots, \mathtt{Pr}_r)$
Output: $\mathfrak{Rel}(I)$

```
1 ans ← 0;
2 foreach E' ⊆ E do
3 |   G' ← (V, E');
4 |   if s ⇝_{G'} T then
5 |   |   p ← 1;
6 |   |   for i ∈ {1, ..., r} do
7 |   |   |   p ← p · Pr_i(E' ∩ E_i)
8 |   |   ans ← ans + p;
9 return ans;
```

Complexity of the Brute Force Algorithm. Assuming that the graph G has n vertices and m edges, Algorithm 1 considers at most 2^m different cases for E' (Line 2). In each case, checking reachability (Line 4)

can be done in $O(m)$ using standard algorithms such as DFS or BFS, and computing $\mathbb{P}(G^s = G')$, i.e. the variable p in the algorithm (Lines 5–7), also takes $O(m)$ time. Hence, Algorithm 1 has a total runtime of $O(m \cdot 2^m)$ which is exponential. Therefore, this algorithm is only applicable to very small graphs.

In order to use Algorithm 1 on larger graphs, we need to shrink them to smaller graphs with the same reliability. The following lemmas are our main tools in doing so.

Lemma 2. *Let $G_B = (B, E_B)$ be a graph, $\mathfrak{T}$ a set of target vertices, $B^* \subseteq B$ a subset of vertices that contains a target vertex $\mathtt{t}^* \in \mathfrak{T}$. Also, let E^* be the set of all possible edges over B^*, i.e. $E^* = \{\{u, v\} \mid u, v \in B^*\}$. Then, there exists a function $f : 2^{E_B} \to 2^{E^*}$ that maps every subset E' of edges of E_B to a subset $f(E')$ of edges of E^*, such that:*

- *For all $a \in B^*$, we have $a \rightsquigarrow_{f(E')} \mathfrak{T}$ if and only if $a \rightsquigarrow_{E'} \mathfrak{T}$.*
- *For all $a, b \in B^*$ such that $a \not\rightsquigarrow_{E'} \mathfrak{T}$ and $b \not\rightsquigarrow_{E'} \mathfrak{T}$, we have $a \rightsquigarrow_{f(E')} b$ if and only if $a \rightsquigarrow_{E'} b$.*
- *For all $a, b \in B^*$ such that $a \rightsquigarrow_{E'} \mathfrak{T}$ and $b \rightsquigarrow_{E'} \mathfrak{T}$, we have $a \rightsquigarrow_{f(E')} b$.*

Moreover, given E', one can compute $f(E')$ in linear time, i.e. $O(|B| + |E_B|)$.

Intuitively, the lemma above says that from a graph with vertex set B, one can create a smaller "digest" graph with vertex set $B^* \subseteq B$, in which (i) a vertex has a path to a target if and only if it used to have a path to a target in the first place, (ii) any two vertices that do not have a path to a target are in the same connected component if and only if they used to be in the same connected component in the first place, and (iii) all the vertices that can reach a target are put in the same connected component. Indeed, the construction below merges all the vertices that could originally reach a target into a single connected component, and keeps the other connected components intact.

Proof. We assume an arbitrary total order on the vertices so that given a set of vertices, we can talk of the vertex with the smallest index. We construct $f(E')$ as follows:

- For every vertex $a \in B^* \setminus \{\mathtt{t}^*\}$ such that $a \rightsquigarrow_{E'} \mathfrak{T}$, we add the edge $\{a, \mathtt{t}^*\}$ to $f(E')$.
- We consider the connected components $C_1, C_2, \ldots, C_s$ of the graph (B, E'). For every C_i, if $C_i \cap \mathfrak{T} = \emptyset$ and $C_i \cap B^* \neq \emptyset$, we let c_i be the vertex in $C_i \cap B^*$ with the smallest index. For every vertex $c'_i \in C_i \cap B^* \setminus \{c_i\}$, we add the edge $\{c_i, c'_i\}$ to $f(E')$.

Figure 3 shows an example application of f.

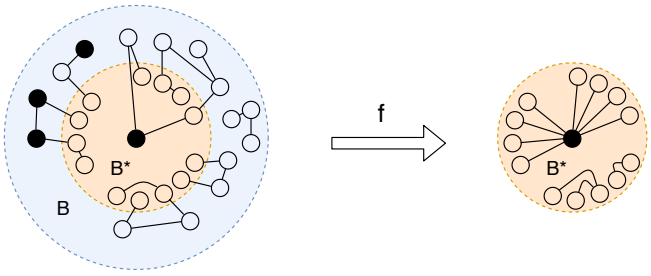

Figure 3: An example application of f. Target vertices are shown in black. The graph (B, E') is shown on the left and $(B^*, f(E'))$ is the graph on the right.

By the above construction, it is easy to verify that $f(E')$ has the desired properties. Moreover, $f(E')$ can be computed by a single use of a classical reachability algorithm, such as DFS, that finds the connected components C_i. Hence, it can be computed in $O(|B| + |E_B|)$. □

Lemma 3 (Shrinking Lemma)**.** *Let $I = (G, E_1, \ldots, E_r, \mathfrak{s}, \mathfrak{T}, \mathit{Pr}_1, \ldots, \mathit{Pr}_r)$ be an* Extended Network Reliability *instance such that:*

- $G = (V, E)$ *and* (A, B) *is a separation of* G*;*
- $\mathfrak{s} \in A$*;*
- *There exists a target vertex* $\mathtt{t}^* \in A \cap B \cap \mathfrak{T}$*;*
- *Every part* E_i *is either entirely in* A *or entirely in* B*. Without loss of generality, we assume that* $E_1, \ldots, E_t$ *are entirely in* A *and* $E_{t+1}, \ldots, E_r$ *are entirely in* B*. We define* $E_A := E_1 \cup \cdots \cup E_t$ *and* $E_B := E_{t+1} \cup \cdots \cup E_r$*. If an* E_i *is entirely in* $A \cap B$*, we consider it to be part of* E_B*;*

then, there exists a smaller instance $\overline{I} = (\overline{G}, E_1, \ldots, E_t, \overline{E_{t+1}}, \mathtt{Pr}_1, \ldots, \mathtt{Pr}_t, \overline{\mathtt{Pr}_{t+1}})$ *with vertex set* A*, i.e.* $\overline{G} = (A, \overline{E})$*, such that* $\mathfrak{Rel}(\overline{I}) = \mathfrak{Rel}(I)$*.*

Intuitively, given a few conditions, this lemma provides a way of shrinking an instance by means of decreasing the number of vertices in the instance, i.e. removing all the vertices in $B \setminus A$, without changing the reliability.

Proof. We use the function f as in Lemma 2, by considering $B^* = A \cap B$ and $G_B = (B, E_{t+1} \cup \cdots \cup E_r) = (B, E_B)$. We let the new part $\overline{E_{t+1}} = E^*$ consist of all the possible edges over B^*. We define the probability distribution $\overline{\mathtt{Pr}_{t+1}} : 2^{\overline{E_{t+1}}} \to [0, 1]$ as follows. For every $F \subseteq \overline{E_{t+1}}$, we let

$$\overline{\mathtt{Pr}_{t+1}}(F) := \sum_{E' \in f^{-1}(F)} \prod_{i=t+1}^{r} \mathtt{Pr}_i(E' \cap E_i). \tag{1}$$

Claim. Let $E'_A \subseteq E_A$ and $E'_B \subseteq E_B$, we claim that $\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T}$ if and only if $\mathfrak{s} \rightsquigarrow_{E'_A \cup f(E'_B)} \mathfrak{T}$.
Proof of Claim. Note that $B^* = A \cap B$ is a separator in G, so there is no edge between $A \setminus B$ and $B \setminus A$. Consider the graphs $G_1 = (A \cup B, E'_A \cup E'_B)$ and $G_2 = (A, E'_A \cup f(E'_B))$, we want to prove that $\mathfrak{s} \rightsquigarrow_{G_1} \mathfrak{T}$ if and only if $\mathfrak{s} \rightsquigarrow_{G_2} \mathfrak{T}$.

First, we assume $s \rightsquigarrow_{G_1} \mathfrak{T}$. Consider a path $\pi := u_0, u_1, \ldots, u_l$ in G_1 from $u_0 = \mathfrak{s}$ to some target vertex $u_l \in \mathfrak{T}$. We construct a path π' from $\mathfrak{s}$ to $\mathfrak{T}$ in G_2 by following π step-by-step. At each step, we assume that we have a prefix of π' from $\mathfrak{s}$ to u_i. We extend this prefix as follows:

1. If the edge $\{u_i, u_{i+1}\}$ is in E'_A, then it has appeared in both G_1 and G_2. So we simply extend π' by adding u_{i+1}. In particular, this case always happens if at least one of u_i and u_{i+1} are in $A \setminus B$.
2. Otherwise, if both u_i and u_{i+1} are in $A \cap B$, then by Lemma 2, we have $u_i \rightsquigarrow_{f(E'_B)} u_{i+1}$. So, we extend π' by a path from u_i to u_{i+1} in $f(E'_B)$.
3. Otherwise, if $u_i \in A \cap B$ and $u_{i+1} \in B \setminus A$, then we consider two cases:
   (a) if the path π ends in a target vertex in $B \setminus A$ without coming back to $A \cap B$, then by Lemma 2, we have $u_i \rightsquigarrow_{f(E'_B)} \mathtt{t}^*$. We extend π' by adding the path from u_i to $\mathtt{t}^*$.
   (b) otherwise, π re-enters $A \cap B$. Assume that this re-entry happens at u_j for some $j > i + 1$. Then, by Lemma 2, we have $u_i \rightsquigarrow_{f(E'_B)} u_j$. So, we extend π' with the path from u_i to u_j in $f(E'_B)$ and continue from u_j.

Note that no other case is possible, given that $A \cap B$ is a separator. Also, following the steps above, we either end up at $\mathtt{t}^*$ or u_l, both of whom are target vertices. Therefore $\mathfrak{s} \rightsquigarrow_{G_2} \mathfrak{T}$. The other side can be proven similarly, i.e. by taking a path in G_2 and replacing every contiguous sequence of edges in $f(E'_B)$ by edges in E'_B.

We now continue our proof of the shrinking lemma. We prove that $\mathfrak{Rel}(\overline{I}) = \mathfrak{Rel}(I)$. We have

$$\mathfrak{Rel}(I) = \sum_{E' \subseteq E} \mathbb{P}(G^s = (V, E')) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'} \mathfrak{T})$$

where δ is the indicator function that has a value of 1 if its parameter is true and 0 otherwise, i.e. $\delta(p) = \begin{cases} 1 & p \\ 0 & \neg p \end{cases}$. Therefore, if we consider $E'_A = E' \cap E_A$ and $E'_B = E' \cap E_B$, we have:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \sum_{E'_B \subseteq E_B} \mathbb{P}(G^s = (V, E'_A \cup E'_B)) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T})$$

We now divide the second summand based on the value of $f(E'_B)$, to get:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \sum_{F \subseteq \overline{E_{t+1}}} \sum_{E'_B \in f^{-1}(F)} \mathbb{P}(G^s = (V, E'_A \cup E'_B)) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T})$$

The appearance of edges in E_A and E_B are independent of each other, so we have $\mathbb{P}(G^s = (V, E'_A \cup E'_B)) = \mathbb{P}(G^s \cap E_A = E'_A) \cdot \mathbb{P}(G^s \cap E_B = E'_B)$, therefore:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \sum_{F \subseteq \overline{E_{t+1}}} \sum_{E'_B \in f^{-1}(F)} \mathbb{P}(G^s \cap E_A = E'_A) \cdot \mathbb{P}(G^s \cap E_B = E'_B) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T})$$

Given that $\mathbb{P}(G^s \cap E_A = E'_A)$ is independent of the two inner summations, we have:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \mathbb{P}(G^s \cap E_A = E'_A) \cdot \sum_{F \subseteq \overline{E_{t+1}}} \sum_{E'_B \in f^{-1}(F)} \mathbb{P}(G^s \cap E_B = E'_B) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T})$$

Moreover, appearance of edges in each E_i is independent of every other E_j, so $\mathbb{P}(G^s \cap E_A = E'_A) = \prod_{i=1}^{t} \mathtt{Pr}_i(E'_A \cap E_i)$ and similarly, $\mathbb{P}(G^s \cap E_B = E'_B) = \prod_{i=t+1}^{r} \mathtt{Pr}_i(E'_B \cap E_i)$. Therefore, we have:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \prod_{i=1}^{t} \mathtt{Pr}_i(E'_A \cap E_i) \cdot \sum_{F \subseteq \overline{E_{t+1}}} \sum_{E'_B \in f^{-1}(F)} \prod_{i=t+1}^{r} \mathtt{Pr}_i(E'_B \cap E_i) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T})$$

According to the Claim proven in Page 7, we have $\delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup E'_B} \mathfrak{T}) = \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup f(E'_B)} \mathfrak{T})$. Note that $f(E'_B)$ is simply F, given that $E'_B \in f^{-1}(F)$. So, we have:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \prod_{i=1}^{t} \mathtt{Pr}_i(E'_A \cap E_i) \cdot \sum_{F \subseteq \overline{E_{t+1}}} \sum_{E'_B \in f^{-1}(F)} \prod_{i=t+1}^{r} \mathtt{Pr}_i(E'_B \cap E_i) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup F} \mathfrak{T})$$

In Equation (1), we defined $\overline{\mathtt{Pr}_{t+1}}(F)$ as $\sum_{E' \in f^{-1}(F)} \prod_{i=t+1}^{r} \mathtt{Pr}_i(E' \cap E_i)$, therefore:

$$\mathfrak{Rel}(I) = \sum_{E'_A \subseteq E_A} \prod_{i=1}^{t} \mathtt{Pr}_i(E'_A \cap E_i) \cdot \sum_{F \subseteq \overline{E_{t+1}}} \overline{\mathtt{Pr}_{t+1}}(F) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup F} \mathfrak{T})$$

We now reverse all the steps above, using $\overline{G^s}$, i.e. the probabilistic graph obtained according to the Extended Network Reliability instance $\overline{I}$:

$$\begin{array}{rcl} \mathfrak{Rel}(I) & = & \sum_{E'_A \subseteq E_A} \mathbb{P}(\overline{G^s} \cap E_A = E'_A) \cdot \sum_{F \subseteq \overline{E_{t+1}}} \mathbb{P}(\overline{G^s} \cap \overline{E_{t+1}} = F) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup F} \mathfrak{T}) \\ & = & \sum_{E'_A \subseteq E_A} \sum_{F \subseteq \overline{E_{t+1}}} \mathbb{P}(\overline{G^s} \cap E_A = E'_A) \cdot \mathbb{P}(\overline{G^s} \cap \overline{E_{t+1}} = F) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup F} \mathfrak{T}) \\ & = & \sum_{E'_A \subseteq E_A} \sum_{F \subseteq \overline{E_{t+1}}} \mathbb{P}(\overline{G^s} = (V, E'_A \cup F)) \cdot \delta(\mathfrak{s} \rightsquigarrow_{E'_A \cup F} \mathfrak{T}) \\ & = & \sum_{\overline{E'} \subseteq E_A \cup \overline{E_{t+1}}} \mathbb{P}(\overline{G^s} = (V, \overline{E'})) \cdot \delta(\mathfrak{s} \rightsquigarrow_{\overline{E'}} \mathfrak{T}) \\ & = & \mathfrak{Rel}(\overline{I}). \end{array}$$

□

The Complexity of Shrinking Lemma. To apply the Shrinking Lemma, we compute $f(E'_B)$ for every $E'_B \subseteq E_B$. There are $2^{|E_B|}$ such subsets. Moreover, by Lemma 2, each computation of f takes $O(|B|+|E_B|)$, therefore the overall complexity of the Shrinking Lemma is $O(2^{|E_B|} \cdot (|B|+|E_B|))$, which is exponential in the size of E_B. Hence, the Shrinking Lemma should only be applied when the set B is small. In our algorithm below, whenever we use the Shrinking Lemma, the set B is a bag of the tree decomposition and therefore has size at most $O(k)$.

The following lemma provides the last ingredient for our main algorithm:

Lemma 4. *Let $I = (G, E_1, \ldots, E_r, \mathfrak{s}, \mathfrak{T}, \mathit{Pr}_1, \ldots, \mathit{Pr}_r)$ be an* Extended Network Reliability *instance with $r \geq 2$. There exists an instance $I_{r-1,r} = (G, E_1, \ldots, E_{r-2}, E_{r+1}, \mathfrak{s}, \mathfrak{T}, \mathit{Pr}_1, \ldots, \mathit{Pr}_{r-2}, \mathit{Pr}_{r+1})$ such that $\mathfrak{Rel}(I) = \mathfrak{Rel}(I_{r-1,r})$. We refer to $I_{r-1,r}$ as the instance obtained by* merging *E_{r-1} and E_r in I.*

Proof. Let $W \subseteq V$ be the subset of vertices consisting of all the endpoints of edges in E_{r-1} and E_r. We define E_{r+1} as a set containing all possible edges over W, i.e. $E_{r+1} = \{\{u, v\} \mid u, v \in W\}$. Let $F_{r-1} \subseteq E_{r-1}$ and $F_r \subseteq E_r$, we define $F_{r-1} \oplus F_r$ as the subset of E_{r+1} that contains an edge from u to v if at least one of F_r and F_{r+1} do. Intuitively, $F_{r-1} \oplus F_r$ is a special kind of union that ignores repeated edges with the same endpoints. For every $F_{r+1} \subseteq E_{r+1}$, we define

$$\mathtt{Pr}_{r+1}(F_{r+1}) := \sum_{F_{r-1} \oplus F_r = F_{r+1}} \mathtt{Pr}_{r-1}(F_{r-1}) \cdot \mathtt{Pr}_r(F_r).$$

Informally, we took two parts E_{r-1} and E_r which used to be independent and merged them into a single correlated part. It is straightforward to verify that this construction preserves the reliability. □

Note that the order of E_i's in I does not matter. Hence, we can define $I_{i,j}$ as the instance obtained by merging E_i and E_j in I and construct it in a similar manner.

We are now ready to present our main algorithm for computing Network Reliability when the underlying graph has a small treewidth k. Basically, our algorithm applies the Shrinking Lemma repeatedly until the number of vertices in the graph is reduced to $O(k)$. Then, it runs the brute force algorithm over it.

Input. As mentioned earlier, the input to the algorithm is a Network Reliability instance $I = (G, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr})$ together with a k-decomposition $T = (\mathfrak{B}, E_T)$ of the graph G rooted at a bag $r \in \mathfrak{B}$ that contains the source vertex, i.e. $\mathfrak{s} \in V(r)$.

Our Algorithm. We compute $\mathfrak{Rel}(I)$ as follows:

1. We take an arbitrary target vertex $\mathfrak{t}^* \in \mathfrak{T}$ and add it to the vertex set of every bag in $\mathfrak{B}$.
2. As long as $\mathfrak{B}$ contains more than one bag, we do the following:
   (a) We take an arbitrary leaf bag $b \in \mathfrak{B}$. Let p be the parent bag of b.
   (b) We apply the Shrinking Lemma with $(A, B) = (\bigcup_{a \in \mathfrak{B} \setminus \{b\}} V(a), V(b))$. This effectively removes all the vertices in $B \setminus A$ from G.
   (c) We remove b from T.
   (d) We take all the edge parts $E_{i_1}, E_{i_2}, \ldots, E_{i_j}$ that are entirely in $V(p)$ and merge them together using Lemma 4.
3. If $\mathfrak{B}$ contains a single bag r, we simply run the brute force algorithm on $V(r)$ for computing $\mathfrak{Rel}(I)$.

Consider the graph depicted in Figure 1 with arbitrary probabilities. We assume $\mathfrak{s} = 1$ and $\mathfrak{T} = \{7\}$. Therefore, in Step (1), we add 7 to every bag. Figure 4 shows the iterations of Step (2), i.e. each panel shows one application of shrinking and merging. The figure does not show the probabilities. In each iteration, a leaf bag b with parent p is chosen, it is removed from the tree decomposition and the vertices that only appeared in b are deleted from the graph. Moreover, a new edge part is added that covers all possible edges between the vertices in the intersection of $V(b)$ and $V(p)$. It is then merged with the already existing edge parts of $V(b)$. This process continues until only one bag r (the root) remains. At this point, the brute force algorithm is used to compute the reliability.

Lemma 5 (Correctness)**.** *The algorithm above correctly computes $\mathfrak{Rel}(I)$.*

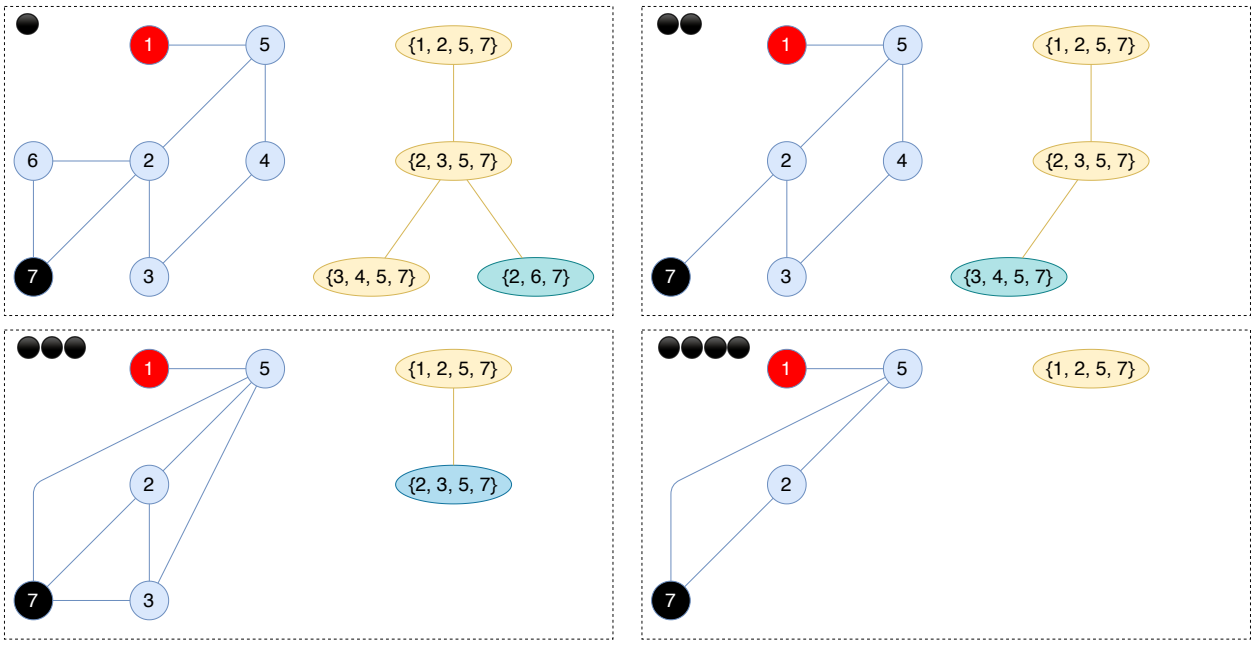

Figure 4: Changes in G and T in every iteration of Step (2)

Proof. Each iteration of Step 2 above reduces the number of bags by one, hence the algorithm terminates. We show that the reliability is preserved after each iteration of Step 2. We also show that before and after each iteration of Step 2, for every correlated edge part E_i, there exists a bag $b_{E_i} \in \mathfrak{B}$ such that every endpoint of every edge in E_i is in $V(b_{E_i})$. Note that initially, each E_i consists of a single edge and hence this property holds by definition of tree decompositions.

In Step 2(b), we can apply the Shrinking Lemma because (i) (A, B) is a separation of G by the Cut Property (Lemma 1); (ii) $\mathfrak{s} \in V(r) \subseteq A$; (iii) $\mathfrak{t}^* \in A \cap B \cap \mathfrak{T}$ because $\mathfrak{t}^*$ is in the vertex set of every bag; and (iv) for every edge part E_i, there exists a bag such that E_i is entirely in that bag. As shown in Page 6, Shrinking Lemma preserves the reliability. Moreover, by Lemma 1, we have $A \cap B = V(p) \cap V(b)$ and therefore the newly added edge part is entirely in $A \cap B \subseteq V(p)$. Finally, by Lemma 4, merging edge parts in Step 2(d) does not change the reliability.

It follows by an easy induction that the reliability is preserved when we reach Step 3. At this point, $\mathfrak{Rel}(I)$ is computed by the brute force algorithm. Hence, our algorithm computes $\mathfrak{Rel}(I)$ correctly. □

Complexity of Our Algorithm. The algorithm applies the shrinking lemma $O(n \cdot k)$ times and in each time we have $|B| \leq k + 2$ and $|E_B| \leq (k + 2) \cdot (k + 1)$. Hence, the overall runtime for the calls to shrinking lemma is $O(n \cdot k^3 \cdot 2^{(k+2)(k+1)})$. Similarly, the algorithm performs $O(n \cdot k)$ merge operations, each of which on two parts of size at most $2^{\binom{k+2}{2}}$. Hence each merge operation takes at most $O(k^2 \cdot 2^{(k+2)(k+1)})$ and the overall runtime for merging is also $O(n \cdot k^3 \cdot 2^{(k+2)(k+1)})$. Finally, the algorithm runs the brute force procedure on a graph with at most $\binom{k+2}{2}$ edges, which takes $O(k^2 \cdot 2^{\binom{k+2}{2}})$ time. All the other operations are performed in linear time. Hence the total runtime of our algorithm is $O(n \cdot k^3 \cdot 2^{(k+2)(k+1)})$, which depends linearly on n and exponentially on k. Hence, we have the following theorem:

Theorem. *There exists a linear-time fixed-parameter algorithm for solving* Network Reliability *when parameterized by the treewidth, i.e. when the treewidth is a small constant.*

Pseudocode. Our approach is summarized in Algorithm 2.

Algorithm 2: Computing Network Reliability using a Tree Decomposition

Input : A Network Reliability instance $I = (G, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr})$ or equivalently an Extended Network Reliability instance $I = (G, E_1, \ldots, E_m, \mathfrak{s}, \mathfrak{T}, \mathtt{Pr}_1, \ldots, \mathtt{Pr}_m)$ in which $|E_i| = 1$ for every i; and a k-decomposition $T = (\mathfrak{B}, E_T)$ of G rooted at r with $\mathfrak{s} \in V(r)$.

Output: $\mathfrak{Rel}(I)$

```
t* ← an arbitrary vertex in 𝔗;
foreach b ∈ 𝔅 do
    V(b) ← V(b) ∪ {t*};
while |𝔅| > 1 do
    b ← an arbitrary leaf bag in 𝔅;
    A ← ∪_{a∈𝔅\{b}} V(a);
    B ← V(b);
    Shrinking-Lemma(I, A, B);
    p ← b.parent;
    𝔅 ← 𝔅 \ {b};
    while ∃E_i, E_j s.t. ∀e = {u, v} ∈ E_i ∪ E_j   u, v ∈ V(p) do
        I ← I_{i,j}
return Brute-Force(I);
```

4. Implementation and Experimental Results

We implemented our approach in Java. Our code is available at `https://ist.ac.at/~akafshda/reliability`. We used a tool called FlowCutter [36] for computing the tree decompositions. FlowCutter applies a state-of-the-art heuristic algorithm to find tree decompositions of small width. However, it is not guaranteed to find an optimal tree decomposition. In each case, we limited FlowCutter to a maximum runtime of 10 minutes.

We experimented with the subway networks of several major cities, including Berlin, London, Tehran, Tokyo and Vienna. Table 1 provides a summary of the instances. Notably, it shows that all of these major subway networks have a small treewidth. In each case, we set the source and target vertices as the subway stations next to some of the major universities. Specifically:

- In Berlin, we used the Technical University of Berlin (Ernst-Reuter-Platz) as the source and the Freie University (Thielplatz) and the Humboldt University (Friedrichstrasse) as the targets.
- In London, we used the London School of Economics (Holborn) as the source and the King's College (Temple), Imperial College (South Kensington) and University College London (Euston Square) as the targets.
- In Tehran, we used Amirkabir University (Teatr-e Shahr) as the source and Sharif University (Daneshgah-e Sharif), University of Tehran (Meydan-e Enqelab) and the Iran University of Science and Technology (Daneshgah-e Elm o San'at) as the targets.
- In Tokyo, we used the University of Tokyo (Okachimachi) as the source and Keio University (Mita) and Waseda University (Waseda) as the targets.
- In Vienna, we used the University of Vienna (Schottenring) as the source and the Technical University of Vienna (Karlsplatz) as the target.

In our experiments, we assumed that every edge of the network appears with the same probability p. We provide experimental results for different values of p between 0 and 1 with step size 0.05. Note that this is not a requirement of our algorithm, which can handle different probability values for different edges.

Network	n	$\lvert\mathfrak{T}\rvert$	k	Runtime (s)
Berlin U-Bahn	173	2	3	1.4
London Tube	261	3	5	1.9
Tehran Metro	103	3	2	1.2
Tokyo Subway	200	2	6	12.4
Vienna U/S-Bahn	141	1	5	1.6

Table 1: A Summary of Our Benchmark Networks. In each case n is the number of vertices in the network, $|\mathfrak{T}|$ is the number of target vertices, and k is the width of the obtained tree decomposition. The runtimes are reported in seconds and are the average runtime over all p.

However, we did not have access to the failure probabilities of the connections in the subway networks, given that such information is classified in most countries.

As shown in Table 1, our algorithm is extremely efficient and, in all these real-world cases, answers the Network Reliability problem in just a few seconds. In contrast, previous exact approaches such as [14, 12] could only handle academic examples with less than 10 vertices. Figure 5 provides a summary of our experimental results.

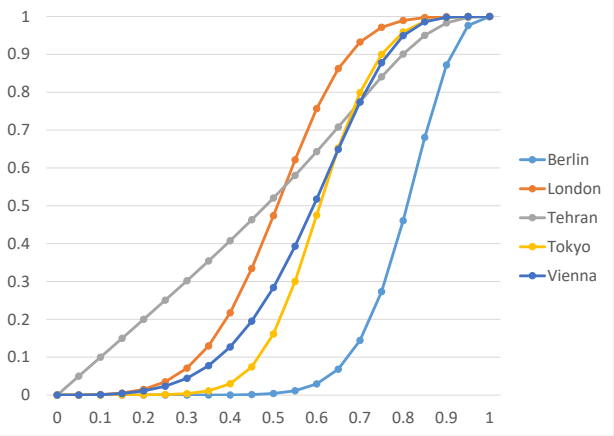

Figure 5: Our Experimental Results. The x-axis is the probability p of the appearance of each edge of the network, and the y-axis is the reliability $\mathfrak{Rel}(I)$.

5. Conclusion

In this paper, on the theoretical side, we presented a linear-time algorithm for computing Network Reliability on graphs with small treewidth. Our algorithm uses the concept of kernelization, i.e. it repeatedly transforms an instance into a smaller one with the same reliability. On the experimental side, we showed that subway networks of several major cities have small treewidth and hence our algorithm can be applied to them. We also demonstrated that our algorithm is extremely efficient and can handle these real-world instances in a few seconds, while previous exact methods could only handle academic examples with a handful of edges.

Acknowledgments. The research was partially supported by the EPSRC Early Career Fellowship EP/R023379/1, Grant № SC7-1718-01 of the London Mathematical Society, an IBM PhD Fellowship, and a DOC Fellowship of the Austrian Academy of Sciences (ÖAW).

References

[1] A. Agrawal, R. E. Barlow, A survey of network reliability and domination theory, Operations Research 32 (1984) 478–492.

[2] E. F. Moore, C. E. Shannon, Reliable circuits using less reliable relays, Journal of the Franklin Institute 262 (1956) 191–208.

[3] M. O. Ball, Computational complexity of network reliability analysis: An overview, IEEE Transactions on Reliability 35 (1986) 230–239.

[4] A. Satyanarayana, R. K. Wood, A linear-time algorithm for computing k-terminal reliability in series-parallel networks, SIAM Journal on Computing 14 (1985) 818–832.

[5] J. S. Provan, M. O. Ball, Computing network reliability in time polynomial in the number of cuts, Operations Research 32 (1984) 516–526.

[6] D. W. Coit, A. E. Smith, Reliability optimization of series-parallel systems using a genetic algorithm, IEEE Transactions on reliability 45 (1996) 254–260.

[7] D. R. Karger, A randomized fully polynomial time approximation scheme for the all-terminal network reliability problem, SIAM review 43 (2001) 499–522.

[8] C. Srivaree-ratana, A. Konak, A. E. Smith, Estimation of all-terminal network reliability using an artificial neural network, Computers & Operations Research 29 (2002) 849–868.

[9] I. B. Gertsbakh, Y. Shpungin, Models of network reliability: analysis, combinatorics, and Monte Carlo, CRC press, 2016.

[10] J. I. Brown, C. J. Colbourn, D. G. Wagner, Cohen–macaulay rings in network reliability, SIAM Journal on Discrete Mathematics 9 (1996) 377–392.

[11] F. Mohammadi, Combinatorial and geometric view of the system reliability theory, in: International Congress on Mathematical Software, Springer, 2016, pp. 148–153.

[12] E. Sáenz-de Cabezón, H. P. Wynn, Hilbert functions in design for reliability, IEEE transactions on reliability 64 (2015) 83–93.

[13] F. Mohammadi, E. Sáenz-de Cabezón, H. P. Wynn, The algebraic method in tree percolation, SIAM Journal on Discrete Mathematics 30 (2016) 1193–1212.

[14] F. Mohammadi, Divisors on graphs, orientations, syzygies, and system reliability, Journal of Algebraic Combinatorics 43 (2016) 465–483.

[15] R. Guidotti, P. Gardoni, Y. Chen, Network reliability analysis with link and nodal weights and auxiliary nodes, Structural Safety 65 (2017) 12–26.

[16] X. Zhang, S. Mahadevan, A game theoretic approach to network reliability assessment, IEEE Transactions on Reliability (2017).

[17] W.-C. Yeh, C. Bae, C.-L. Huang, A new cut-based algorithm for the multi-state flow network reliability problem, Reliability Engineering & System Safety 136 (2015) 1–7.

[18] W. J. Gutjahr, G. C. Pflug, A. Ruszczyński, Configurations of series-parallel networks with maximum reliability, Microelectronics Reliability 36 (1996) 247–253.

[19] R. Niedermeier, Invitation to fixed-parameter algorithms (2002).

[20] M. Cygan, F. V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized algorithms, Springer, 2015.

[21] R. G. Downey, M. R. Fellows, Parameterized complexity, Springer Science & Business Media, 2012.

[22] N. Robertson, P. D. Seymour, Graph minors. iv. tree-width and well-quasi-ordering, Journal of Combinatorial Theory, Series B 48 (1990) 227–254.

[23] B. Courcelle, The monadic second-order logic of graphs. i. recognizable sets of finite graphs, Information and computation 85 (1990) 12–75.

[24] K. Chatterjee, A. K. Goharshady, R. Ibsen-Jensen, A. Pavlogiannis, Algorithms for algebraic path properties in concurrent systems of constant treewidth components, in: ACM SIGPLAN Notices, volume 51, ACM, 2016, pp. 733–747.

[25] B. Courcelle, M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, Theoretical Computer Science 109 (1993) 49–82.

[26] K. Chatterjee, A. K. Goharshady, N. Okati, A. Pavlogiannis, Efficient parameterized algorithms for data packing, Proceedings of the ACM on Programming Languages 3 (2019) 53.

[27] H. L. Bodlaender, Dynamic programming on graphs with bounded treewidth, in: International Colloquium on Automata, Languages, and Programming, Springer, 1988, pp. 105–118.

[28] K. Chatterjee, R. Ibsen-Jensen, A. K. Goharshady, A. Pavlogiannis, Algorithms for algebraic path properties in concurrent systems of constant treewidth components, ACM Transactions on Programming Languages and Systems (TOPLAS) 40 (2018) 9.

[29] K. Chatterjee, A. K. Goharshady, R. Ibsen-Jensen, A. Pavlogiannis, JTDec: A tool for tree decompositions in soot, in: Fifteenth International Symposium on Automated Technology for Verification and Analysis, ATVA, Springer, 2017.

[30] T. Wolle, A framework for network reliability problems on graphs of bounded treewidth, Lecture notes in computer science (2002) 137–149.

[31] H. L. Bodlaender, A tourist guide through treewidth, Acta cybernetica 11 (1994).

[32] M. Thorup, All structured programs have small tree width and good register allocation, Information and Computation 142 (1998) 159–181.

[33] J. Gustedt, O. Mæhle, J. Telle, The treewidth of java programs, Algorithm Engineering and Experiments (2002) 57–59.

[34] K. Chatterjee, A. K. Goharshady, E. K. Goharshady, The treewidth of smart contracts, in: ACM Symposium on Applied Computing (SAC), 2019.

[35] H. L. Bodlaender, A linear-time algorithm for finding tree-decompositions of small treewidth, SIAM Journal on computing 25 (1996) 1305–1317.

[36] M. Hamann, B. Strasser, Graph bisection with pareto optimization, Journal of Experimental Algorithmics (JEA) 23 (2018).