Studies on Network Graph Analysis with Decision Diagram Structures

Kengo Nakamura

Abstract

This dissertation is devoted to network analysis problems, which fall into theoretically difficult computational complexity classes such as NP-complete, NP-hard, and #P-complete. The difficulty often comes from the fact that a network infrastructure provides its service with a combination of its components, resulting in a combinatorial explosion of computational time. With the help of binary decision diagrams (BDDs) and similar structures that can represent the set of combinatorial objects compactly, we develop practically efficient algorithms for various network analysis problems.

First, we address the problem of computing equilibrium of combinatorial congestion games. Combinatorial congestion games can model the selfish behavior of users of network infrastructures, and computing their equilibrium is essential for congestion analysis of network infrastructures where users act selfishly. There are many scenarios in which the computation of equilibrium falls in difficult complexity classes, e.g., budgeted routing and multi-location communication. We address the equilibrium computation of general combinatorial congestion games by making use of zero-suppressed binary decision diagrams (ZDDs) along with Frank–Wolfe-style iterative algorithms. We theoretically prove the convergence rate for obtaining ϵ-approximate equilibrium and empirically validated that the proposed method can efficiently compute equilibrium on the computationally tough scenarios.

Next, we delve into network reliability analysis, which reveals the robustness of network infrastructures against the failures of network components. Traditionally, network reliability is defined as the probability that the specified vertices are connected, and existing approaches using BDDs provide relatively fast algorithms although computing it is #P-complete. We first address the network reliability evaluation for client-server model, which is equivalent to evaluate network reliability repetitively for every node. Although the exiting approach requires separate BDD for every node, we have only one BDD-like structure to evaluate all the reliability values, yielding extreme efficiency. Then, we propose faster exact computation algorithms for more enhanced reliability measures. First, we consider the expected number of connected nodes or node pairs, which is suitable for the reliability measures of the whole network. Although this requires an evaluation of reliability for every pair of nodes, we again have only one BDD-like structure to evaluate them. We provide theoretical and empirical computational time analysis as well as applications for critical link identification and server placement. Next, we address the problem of computing variance of network reliability when each network component also has a variance (an uncertainty) in its working probability. We develop an efficient computation algorithm using a BDD and reveal the behavior of the uncertainty in network reliability for the first time. Finally, we propose an efficient algorithm for computing unreliability per outage scale, i.e., the number of disconnected nodes, which is useful in designing network infrastructures. The proposed algorithm builds a multi-terminal variant of BDD, and its practical efficiency is verified by real-world network topologies.

Finally, we consider subgraph counting problems that counts the number of subgraphs of an input graph satisfying some graph constraints. This can be used for evaluating the importance of every component (vertex) in a network, but this requires solving this computationally difficult problem repetitively when such an importance measure is computed for every vertex. By extending the network reliability evaluation algorithm for client-server model described above, we propose an efficient algorithm that does not require repetitive construction of decision diagrams for each vertex. The runtime of the proposed method is theoretically and empirically compared with the existing approach.

Acknowledgements

First, I would like to express my deepest gratitude to my supervisor, Shin-ichi Minato, for his continuous support to my research works. Although he had much busy schedule, he has always paid attention to my works as well as my Ph.D. student days by discussing with me and the other members in our laboratory. Especially, he gave invaluable comments on the problem settings and the algorithms on the works that are the building blocks of this dissertation. Also, he understood my circumstance that I am also an employee of a company and gave much efforts to make my Ph.D. student days delightful.

Next, I would like to express my appreciation to my collaborators: Shinsaku Sakaue, Norihito Yasuda, Takeru Inoue, Masaaki Nishino, Kunihiko Sadakane, Kotaro Matsuda, Shuhei Denzumi, Naoki Kobayashi, Tsutomu Hirao, Hidetaka Kamigaito, Manabu Okumura, Masaaki Nagata, and Ryoma Onaka. Some of the joint works with them become the building blocks of this dissertation. Especially, I am grateful to Masaaki Nishino, Norihito Yasuda, Takeru Inoue, and Shinsaku Sakaue for their continuous support to my research and working life. Also, I am thankful to my previous supervisors, Takayasu Matsuo and Kunihiko Sadakane. Without their supervision in my bachelor and master student days, I might not enter into the world of scientific research.

Next, I would like to sincerely thank current and former members of our laboratory. Especially, I am grateful to Jun Kawahara, Yuni Iwamasa, and Ryosuke Matsuo for giving me insightful comments on my works and supporting my everyday life in our laboratory. I am also grateful to Hiromi Emoto and Shou Ooba for their valuable suggestions on my research.

Next, I would like to express my thanks to my family and friends for their continuous support and encouragement.

Finally, I thank JSPS KAKENHI Grant Number JP20H05963 for supporting the works in Chapters 5–9 and JST CREST Grant Number JP-MJCR22D3 for supporting the works in Chapters 6, 8, and 9. I am also grateful to Grant-in-Aid for Transformative Research Areas (A) Algorithmic

Foundations for Social Advancement (AFSA project) for holding invaluable research events many times.

List of Publications

Publications Included in This Dissertation

This dissertation includes contents of the following six publications.

Refereed Conference Proceedings

1. (**Chapter 3**) Kengo Nakamura, Shinsaku Sakaue, and Norihito Yasuda.
   Practical Frank–Wolfe Method with Decision Diagrams for Computing Wardrop Equilibrium of Combinatorial Congestion Games.
   In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI 2020)*, pp. 2200–2209, 2020.
   `doi:10.1609/aaai.v34i02.5596`
   © 2020, Association for the Advancement of Artificial Intelligence (`www.aaai.org`)

2. (**Chapter 5**) Kengo Nakamura, Takeru Inoue, Masaaki Nishino, and Norihito Yasuda.
   Efficient Network Reliability Evaluation for Client-Server Model.
   In *Proceedings of the 2021 IEEE Global Communications Conference (IEEE GLOBECOM 2021)*, pp. 1–6, 2021.
   `doi:10.1109/GLOBECOM46510.2021.9685283`
   © 2021 IEEE
   Figure 4.1a in Chapter 4 is also extracted from this publication.

3. (**Chapter 6**) Kengo Nakamura, Takeru Inoue, Masaaki Nishino, Norihito Yasuda, and Shin-ichi Minato.
   A Fast and Exact Evaluation Algorithm for the Expected Number of Connected Nodes: an Enhanced Network Reliability Measure.
   In *Proceedings of the 2023 IEEE International Conference on Computer Communications (IEEE INFOCOM 2023)*, pp. 1–10, 2023.

doi:10.1109/INFOCOM53939.2023.10228897
© 2023 IEEE

4. (**Chapter 7**) Kengo Nakamura, Takeru Inoue, Masaaki Nishino, and Norihito Yasuda.
Impact of Link Availability Uncertainty on Network Reliability: Analyses with Variances.
In *Proceedings of the 2022 IEEE International Conference on Communications (IEEE ICC 2022)*, pp 2713–2719, 2022.
doi:10.1109/ICC45855.2022.9838781
© 2022 IEEE

5. (**Chapter 8**) Kengo Nakamura, Takeru Inoue, Masaaki Nishino, Norihito Yasuda, and Shin-ichi Minato.
Exact and Efficient Network Reliability Evaluation per Outage Scale.
In *Proceedings of the 2023 IEEE International Conference on Communications (IEEE ICC 2023)*, pp. 4564–4570, 2023.
doi:10.1109/ICC45041.2023.10279779
© 2023 IEEE

6. (**Chapter 9**) Kengo Nakamura, Masaaki Nishino, Norihito Yasuda, and Shin-ichi Minato.
CompDP: A Framework for Simultaneous Subgraph Counting Under Connectivity Constraints.
In *Proceedings of the 21st International Symposium on Experimental Algorithms (SEA 2023)*, LIPIcs Vol. 265, pp 11:1–11:20, 2023.
doi:10.4230/LIPIcs.SEA.2023.11
© 2023 Kengo Nakamura, Masaaki Nishino, Norihito Yasuda, and Shin-ichi Minato
Licensed under Creative Commons Attribution 4.0 International license

Other Refereed Publications

The author also published the following papers, which are not included in this dissertation. The contents of papers 1.–3. were included in the author's master thesis. Although the papers 8. and 9. are related to the contents dealt in this dissertation, they are not included in this dissertation because the first authors of these papers are not the author of this dissertation.

1. Kengo Nakamura and Kunihiko Sadakane.
Space-Efficient Fully Dynamic DFS in Undirected Graphs.

Algorithms, Vol. 12(3), No. 52, 2019.
`doi:10.3390/a12030052`

2. Kengo Nakamura and Kunihiko Sadakane.
A Space-efficient Algorithm for the Dynamic DFS Problem in Undirected Graphs.
In *Proceedings of the 11th International Conference and Workshops on Algorithms and Computation (WALCOM 2017)*, LNCS Vol. 10167, pp. 295–307, 2017.
`doi:10.1007/978-3-319-53925-6_23`

3. Kengo Nakamura.
Fully Dynamic Connectivity Oracles under General Vertex Updates.
In *Proceedings of the 28th International Symposium on Algorithms and Computation (ISAAC 2017)*, LIPIcs Vol. 92, pp. 59:1–59:12, 2017.
`doi:10.4230/LIPIcs.ISAAC.2017.59`

4. Kotaro Matsuda, Shuhei Denzumi, Kengo Nakamura, Masaaki Nishino, and Norihito Yasuda.
Approximated ZDD Construction Considering Inclusion Relations of Models.
In *Proceedings of the Special Event on Analysis of Experimental Algorithms (SEA2 2019)*, LNCS Vol. 11544, pp. 265–282, 2019.
`doi:10.1007/978-3-030-34029-2_18`

5. Naoki Kobayashi, Tsutomu Hirao, Kengo Nakamura, Hidetaka Kamigaito, Manabu Okumura, and Masaaki Nagata.
Split or Merge: Which is Better for Unsupervised RST Parsing?
In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP 2019)*, pp. 5797–5802, 2019.
`doi:10.18653/v1/D19-1587`

6. Kengo Nakamura, Shuhei Denzumi, and Masaaki Nishino.
Variable Shift SDD: A More Succinct Sentential Decision Diagram.
In *Proceedings of the 18th International Symposium on Experimental Algorithms (SEA 2020)*, LIPIcs Vol. 160, pp. 22:1–22:13, 2020.
`doi:10.4230/LIPIcs.SEA.2020.22`

7. Masaaki Nishino, Norihito Yasuda, and Kengo Nakamura.
Compressing Exact Cover Problems with Zero-Suppressed Binary Decision Diagrams.
In *Proceedings of the 30th International Joint Conference on Artificial*

Intelligence (IJCAI 2021), pp. 1996–2004, 2021.
`doi:10.24963/ijcai.2021/275`

8. Shinsaku Sakaue and Kengo Nakamura.
Differentiable Equilibrium Computation with Decision Diagrams for Stackelberg Models of Combinatorial Congestion Games.
In *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, pp. 9416–9428, 2021.
`Available at NeurIPS proceedings`

9. Ryoma Onaka, Kengo Nakamura, Takeru Inoue, Masaaki Nishino, Norihito Yasuda, and Shinsaku Sakaue.
Exact and Scalable Network Reliability Evaluation for Probabilistic Correlated Failures.
In *Proceedings of the 2022 IEEE Global Communications Conference (IEEE GLOBECOM 2022)*, pp. 5547–5552, 2022.
`doi:10.1109/GLOBECOM48099.2022.10001640`

10. Masaaki Nishino, Kengo Nakamura, and Norihito Yasuda.
Generalization Analysis on Learning with a Concurrent Verifier.
In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022)*, 2022.
`Available at NeurIPS proceedings`

Contents

Chapter 1

Introduction

Contemporary society greatly depends on several network infrastructures, such as telecommunication, transportation (road and train networks), electric power, and cloud computing. Analysis of such network infrastructures is frequently performed both in construction time and operating time. For example, in designing networks, congestion or reliability analysis is used for evaluating the performance of designed network topologies in order to choose the better one, and in operating networks, criticality analysis is used for detecting the weak network components that should be reinforced. Mathematically, most network infrastructures can be modeled as a *graph* with *vertices* and *edges*, where each edge connects two vertices.

The feature of a network infrastructure is that it provides its service with a combination of its components. For example, in a telecommunication network, two nodes can be communicated when there is a route between them that consists of multiple links, i.e., wires and fibers. Another example is an electric power distribution network where power is distributed from a substation through some switches. This suggests that when analyzing network infrastructures, it is often indispensable to consider the combinations of network components. In other words, when a network infrastructure is modeled as a graph, the combinations of vertices and edges should be involved in formulating the analysis of networks. More formally, many network analysis problems can be formulated with a set of combinations of vertices and edges. A typical example is the shortest-path problem; it can be formulated as the problem of choosing the path with smallest cost from the set of paths from source to destination, where a path can be seen as a combination of edges.

We here observe that the set of combinations may contain an exponentially many objects. Fortunately, this does not cause difficulty for some network analysis problems. For example, although there are exponentially many possible paths from source to destination, the shortest-path problem can be solved in polynomial time with respect to the number of vertices and edges, meaning that it can be solved efficiently. However, unfortunately, some other network analysis problems are known to be computationally difficult. That is, these problems fall in NP-complete, NP-hard, or #P-complete, all of which are computational complexity classes known to be difficult. For these problems, polynomial time algorithms must not exist unless P = NP.

To alleviate such difficulty, there exist approaches to represent the set of combinatorial objects in a compressed and tractable form. The most prominent approach among them is to use *binary decision diagram* (*BDD*) [Bryant, 1986] or its variant *zero-suppressed binary decision diagram* (*ZDD*) [Minato, 1993] to represent the set of combinatorial objects. Although their representation size generally remains exponential, it is verified on both theoretical and practical sides that their size will be much smaller when they represent the set of combinatorial objects stemming from a real-world sparse graph topologies. Moreover, BDDs and ZDDs are not only compact representations of the set of combinatorial objects, but also efficient tools for solving queries about the representing set. That is, BDDs and ZDDs can answer some queries about the representing set in polynomial time with respect to their size. In other words, once the set of combinatorial objects is compiled into a BDD or a ZDD, it can be used as a succinct index of the representing set. With such features, BDDs and ZDDs have succeeded in various network analysis problems including network reliability analysis [Hardy *et al.*, 2007; Herrmann, 2010], network reliability optimization [Nishino *et al.*, 2018], influence spread computation [Maehara *et al.*, 2017], power distribution network verification [Inoue *et al.*, 2015], and finding and enumerating balanced electoral district division [Kawahara *et al.*, 2017a].

These problems have similarity in that each of them is not a problem to extract only one combination from the set. Since a BDD or a ZDD retains all the combinatorial objects in the set, it exhibits its strength when we have to repetitively query the set or even enumerate or count it, and all the problems described above fall in such problems. By expanding this line of research while keeping in mind the above observation, we address the following problems in this dissertation.

1.1 Combinatorial Congestion Games

Congestion games are game-theoretic non-cooperative games that model the selfish behavior of players in resource allocation scenarios. Among them, the combinatorial congestion game is a variant where each player chooses a combination of resources, which is called *strategy*, selfishly. It can model various network infrastructures; for example, in a road network, each driver travels from source to destination selfishly, i.e., by choosing the fastest path, and in a telecommunication network, each pair of users communicates by using the fastest route according to routing protocol. Here, each resource in congestion games correspond to the link in the network, or equivalently the edge in the graph.

In congestion games, the cost of a resource increases with the rise of the number of players using it, which corresponds to the situation that each link becomes congested and incurs more cost when many people use it. If some resources become more costly and another strategy becomes more cost-effective, players have incentive to change their own strategies. Eventually, the situation will be achieved that all the players experience the same cost and no other strategies have smaller cost. This situation is called (*Wardrop*) *equilibrium*, and it can be regarded as the ultimate congestion state induced by the selfish players. Therefore, computing equilibrium of combinatorial congestion games is important for the congestion analysis of network infrastructures where users act selfishly.

One famous example of combinatorial congestion games is *selfish routing* [Roughgarden, 2005], where the each player chooses a path between source and destination. This models the user's selfish behavior to move between source and destination as fast as possible. For selfish routing, efficient methods for computing equilibrium were developed [Fabrikant *et al.*, 2004; Thai, 2017] by a reduction to polynomial-time-solvable problems. However, when the set of strategies becomes more complex, equilibrium computation becomes suddenly difficult such as NP-hard and APX-hard. Such examples include budgeted selfish routing [Jahn *et al.*, 2005], where each strategy is a path satisfying a budget (knapsack) constraint, and multi-location communication [Imase and Waxman, 1991], where each strategy is a Steiner tree connecting the vertices to communicate.

In Chapter 3, we practically address the problem of computing equilibrium of general combinatorial congestion games by utilizing ZDDs and variants of Frank–Wolfe algorithm [Frank and Wolfe, 1956]. As in [Sandholm, 2001], the equilibrium computation can be seen as a convex minimization problem constrained by a complex polytope defined from the set of strategies. We address this problem with repetitive linear minimization on a ZDD representing the set of strategies, which is used as a subroutine of Frank–Wolfe-style iterative algorithm [Lacoste-Julien and Jaggi, 2015]. We theoretically prove the convergence rate for the proposed algorithm to achieve ϵ-approximate Wardrop equilibrium, meaning that all the players experience cost no more than the minimum cost plus ϵ. We also validated the usefulness of the proposed method with budgeted selfish routing and multi-location communication settings.

1.2 Network Reliability Analysis

Network reliability [Moskowitz, 1958] is an indicator for the robustness of a network against failures of network components. Network reliability analysis is fundamental in designing and evaluating network infrastructures, especially for communication networks. Since emerging standards such as sixth-generation mobile communication system (6G) requires extremely high reliability up to 99.99999% (seven 9s), it is crucial for network design and analysis to rigorously evaluate network reliability without approximation.

When a network is modeled as a graph, the failures of network components can be considered as the stochastic presence (working) or absence (failing) of edges [Moskowitz, 1958]. For network infrastructures, the most basic requirement for providing service between two nodes in the network is that these nodes are connected with only the correctly working links [Nojo and Watanabe, 1987, 1993; Tollar and Bennett, 1995]. Therefore, the basic definition of network reliability, called *K-terminal network reliability* (*K-NR*) given a subset K of vertices, is defined as the probability that the specified vertices K called *terminals* are connected with only the working edges under stochastic failures of edges. Despite of its simple definition, computing K-NR is known to fall in #P-complete [Valiant, 1979], a computationally difficult class. A practically fast K-NR computation algorithm using BDDs was proposed by Hardy *et al.* [2007], which can evaluate K-NR of networks with around 200 links.

This dissertation deals with the following four topics of network reliability analysis. The first one accelerates K-NR evaluations for modern network infrastructures, while the latter three propose new reliability measures and faster exact computation algorithms for them.

Reliability Evaluation under Client-Server Model

Some modern network infrastructures follow a client-server model. Typical example is a cloud service where services are provided from servers to browsers (clients). Since each client works independently, the reliability should be evaluated separately for every client to meet service level agreement. That is, we should compute the probability that a client is connected to servers for every client. We call this problem *CSNR problem.* Since even single computation of K-NR is a computationally tough task, the CSNR problem incurs much more computational cost as it needs K-NR evaluation

$O(n)$ times where n is the number of vertices.

In Chapter 5, we propose an efficient algorithm for solving CSNR problem. The proposed algorithm relies on the construction of BDD-like structure. However, unlike the existing approach where a BDD is built by [Hardy *et al.*, 2007] for every client, we need only one data structure to compute all the reliability values. As a result, the proposed algorithm solves the CSNR problem $O(n)$ times faster than the existing approach. Moreover, as a by-product, we can derive a faster algorithm for computing single K-NR compared to the existing method [Hardy *et al.*, 2007] when the number of terminals is small. The efficiency of the proposed method was empirically verified on both synthetic and real-world network topologies; the proposed method is two-orders-of-magnitude faster than the existing approach in solving CSNR problem. In addition, the proposed method succeeded in solving CSNR problem within a reasonable time on the real-world topologies with 400–850 links, on which even single K-NR computations had not been conducted yet.

Expected Number of Connected Nodes and Node Pairs

Let us consider an evaluation of the whole network infrastructure. Traditionally, the *all-terminal network reliability* (*All-NR*), the probability that all the vertices are connected each other, has been used as a reliability measure of the whole network [Hardy *et al.*, 2005; Chaturvedi, 2016; Gaur *et al.*, 2021]. However, the All-NR sometimes exhibits a counter-intuitive value, and so enhanced reliability measures for an entire network are in demand. When a network follows client-server model, which let users (clients) to access shared resources (servers), the number of clients that are connected to servers is a reasonable performance measure. When a network falls in point-to-point (P2P) infrastructures, which let users to access each other, the number of connected node pairs becomes a reasonable performance measure. Under the stochastic failure of links, their expected numbers should be good reliability measures for the whole network.

In Chapter 6, *the expected number of connected nodes* (*ECN*) and *the expected number of connected node pairs* (*ECP*) are defined based on the above observation. Although computing ECP involves $O(n^2)$ times of K-NR computation, it needs running the method of Hardy *et al.* [2007] $O(n^2)$ times or running the method proposed in Chapter 5 $O(n)$ times, where n is the number of vertices. Unlike this, the proposed method in Chapter 6 builds only one BDD-like data structure and perform elaborated dynamic programming to obtain ECP value. As a result, regarding the ECP computation, the proposed method runs in $O(n)$ times faster compared to the method in

Chapter 5. Moreover, the proposed method can also obtain the ECN values for every vertex; the precise problem definition is in Chapter 6. The efficiency of the proposed method is again verified on both synthetic and real-world network topologies; the ECP of an 821-link real-world network topology was computed in only 10 seconds with the proposed method, while the other methods did not finish the computation within an hour.

Moreover, two applications of the proposed method are exhibited in Chapter 6. First, we consider a problem of finding critical links with respect to ECP value in a similar way as the other criticality problems [Kuo *et al.*, 2007; Inoue, 2019], and develop an algorithm for solving this using automatic differentiation [Griewank and Walther, 2008]. We empirically verified that the proposed algorithm can find critical links of P2P infrastructures. Second, we consider a server placement problem, that is, the problem of placing a server to maximize ECN. We demonstrated that the proposed method can solve this problem in 70–10000 times faster.

Variance of Network Reliability

Traditionally, the K-NR is computed under the assumption that the probability that each link works is precisely known. However, this does not hold in real situations; for example, the optical fiber's lifespan is estimated as a distribution [Aso *et al.*, 2012]. This arises a natural question: how the uncertainty in each link's probability affects the uncertainty in network reliability measures?

Chapter 7 addresses this issue by focusing on *variances*. We define the problem of computing the variance of K-NR when the variance of each edge's working probability is given. Although this seems more difficult than just computing K-NR that itself is #P-complete, we develop an efficient algorithm for solving this problem by using BDDs. More specifically, we use the BDD built by the method of Hardy *et al.* [2007], but we perform more complicated dynamic programming to obtain the value of variance. Empirically, the proposed method can compute the variance of K-NR in less than 0.1 seconds for real network topologies with 100–200 links.

The magnitude and behavior of the variance of K-NR is also empirically examined in Chapter 7. It reveals us that for most cases, the variance of K-NR remains at most in the same order as the variance of edge's probability. Moreover, even if one link's variance of probability becomes significant, its effect on the variance of K-NR is marginal. These properties are desirable for network design, and they constitute an evidence for rigorously computing the network reliability measures.

Scale-Wise Unreliability

When an outage of communication network occurs, its *scale*, i.e., significance, is usually measured by the number of affected users or nodes [Tollar and Bennett, 1995; Matsukawa and Funakoshi, 2010]. To avoid serious outages, network operators often specify admissible outage probability for each outage scale. Although there are some works for approximately computing such *scale-wise* reliability [Taka and Abe, 1994; Watanabe *et al.*, 2003] by restricting the stochastic failure scenarios of edges, rigorous evaluation is needed for designing networks satisfying the above requirements, as described in Section 1.2.

In Chapter 8, we formally define the problem of computing unreliability for each outage scale, called *scale-wise unreliability.* We propose an algorithm for precisely computing scale-wise unreliability. This algorithm constructs an extension of BDD called *multi-terminal BDD*, and probability computation is performed by dynamic programming on the built structure. As a result, the probability of every scale can be computed simultaneously. Empirically, the proposed method can evaluate scale-wise unreliability of real-world network topologies with nearly 200 links in at most 1.3 hours.

Note that the work in Chapter 6 is similar to this work in that the expected outage scale can easily evaluated by ECN computation of Chapter 6. However, we empirically confirmed in Chapter 8 that just computing the expected outage scale does not reveal the scale-wise unreliability since the scale-wise unreliability decays bumpy with respect to the outage scale, meaning that an interpolation of scale-wise unreliability from the expected outage scale is difficult.

1.3 Subgraph Counting

Subgraph counting problem computes the (possibly weighted) number of subgraphs of an input graph that satisfy some constraints. The traditional K-NR computation [Moskowitz, 1958] falls in this problem where the constraint is of the form “the terminal vertices are connected each other”. Other than network reliability analysis, the subgraph counting is regarded as a fundamental problem in computer science [Bax, 1994; Alon *et al.*, 1997; Flum and Grohe, 2004; Curticapean, 2018]. Although there are a few exceptions such as spanning trees (countable in polynomial time by matrix-tree theorem; see [Tutte, 2001]), subgraph counting problem becomes #P-complete for many graph constraints.

In network infrastructures, subgraph counting problem can be used for evaluating importance of every network component. For example, in a communication network where two vertices s and t are communicating, the communication route forms s, t-simple path. Thus, the number of s, t-simple paths that passes through another vertex v can be regarded as the importance of vertex v because it equals the number of lost communication routes when v fails. Other examples include multi-location communication [Imase and Waxman, 1991] where Steiner trees are counted and power distribution network [Inoue *et al.*, 2015] where rooted spanning forests are counted.

However, to evaluate importance measure for every vertex, we have to solve multiple subgraph counting problems corresponding to every vertex. Since even computing single count value is a cumbersome task, solving it multiple times may incur prohibitively large computation time.

In Chapter 9, we first formally define the problem of computing multiple subgraph count values for every vertex by focusing on *connectivity constraints*, which require that some pairs of vertices are connected for each and other pairs are disconnected for each. Then, by extending the work in Chapter 5, we propose an efficient algorithm for solving the above problem. Like the algorithm in Chapter 5, only one ZDD is built to obtain multiple subgraph count values. The resulting algorithm can deal with various graph constraints such as simple paths, cycles, Steiner trees, and rooted spanning forests. Theoretically, the proposed method runs in $O(n)$ times faster than the existing approach where a ZDD is built separately for every vertex by [Kawahara *et al.*, 2017b]. Empirically, the proposed method solves the above problem about 10–20 times faster than the existing approach for many graph constraints.

1.4 Notation and Terminology

First, we introduce notations used in this dissertation. In all of the subsequent chapters in this dissertation, the network topology is modeled as an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges[1]. The objects related to BDDs, ZDDs, and similar structures, are denoted by typewriter letters, e.g., $\mathtt{B}, \mathtt{Z}, \mathtt{N}, \mathtt{A}, \mathtt{n}, \mathtt{r}, \mathtt{R}$. For example, a BDD and a ZDD are typically denoted as $\mathtt{B}$ and $\mathtt{Z}$. Real-valued vectors are denoted by bold italic letters, such as $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s}, \boldsymbol{d}$; they are mainly used in Chapter 3. Random variables are denoted by bold capital roman letters, such as $\mathbf{P}, \mathbf{Q}, \mathbf{R}$; they are mainly used in Chapter 7. The objects that can be regarded as a set of combinatorial objects are often denoted by calligraphic letters. For example,

[1]In Chapters 2 and 3, m represents the cardinality of item set. This is because in the subsequent chapters, the item set in Chapter 2 equals the edge set E that has m edges.

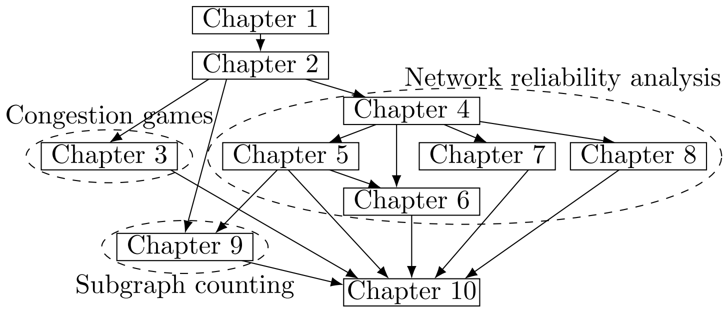


Figure 1.1: Dependence structure of this dissertation.

a family of subsets of general item set is often denoted by $\mathcal{I}$ and a set of subsets of edges by $\mathcal{E}$.

Next, we introduce some terminologies to avoid confusion. Since the structures of BDDs and ZDDs are directed graphs, it should not be confused with the graph $G = (V, E)$ that is an abstracted network topology. Thus, we call the vertices and edges in BDDs, ZDDs, and similar structures as *dnode* and *arc*. Moreover, the terms *node* and *link* are used when our focus is on applications for networked infrastructures, and *vertex* and *edge* are used when our focus is on the abstracted network model or the algorithms working on the network model.

1.5 Organization

The rest of this dissertation is organized as follows. Chapter 2 describes the background of BDDs and ZDDs, which will be used in the subsequent chapters. Chapter 3 addresses the equilibrium computation of combinatorial congestion games. Chapter 4 presents the background of network reliability analysis, which will be used in Chapters 5–8. Chapter 5 is about the network reliability evaluation under client-server model, and Chapter 6 is devoted to the efficient computation of the expected number of connected nodes or connected pairs. Chapter 7 proposes an algorithm for computing the variance of network reliability, and Chapter 8 addresses the computation of scale-wise unreliability. Chapter 9 presents an efficient method for solving multiple subgraph counting problems under similar connectivity constraints. Finally, Chapter 10 concludes this dissertation. The dependence structure of this dissertation is depicted in Figure 1.1.

Chapter 2

Background of Decision Diagrams

This chapter describes an overview on BDDs, binary decision diagrams, and ZDDs, zero-suppressed binary decision diagrams. These are data structures that are repetitively used in this dissertation.

As described in Chapter 1, in the area of network analysis, when the problems are formulated mathematically, families of subgraphs, i.e., sets of subsets of edges, appear frequently. For example, in the congestion analysis of telecommunication networks, we should choose a path or a tree connecting communication sites whose cost is the smallest. Here, a path or a tree can be seen as a subgraph of the original network graph, and this problem can be seen as choosing the most cost-effective one from the set of all subgraphs connecting communication sites. Since there are at most 2^m subgraphs for a graph with m edges, explicitly storing them is usually prohibitive even for graphs of moderate size. Decision diagrams explained in this chapter can sometimes remedy this issue; they are data structures designed to compactly store a family of subgraphs, or more generally a family of subsets. Below we first explain the relation between families of sets and Boolean functions, since BDDs were originally designed to efficiently represent Boolean functions. Then, we explain the structures and semantics of BDDs and ZDDs. After that, we focus on operations and efficient construction methods for BDDs and ZDDs.

2.1 Boolean Functions and Families of Sets

Given binary variables $x_1, \ldots, x_m$, *Boolean function* f is a function that maps every assignment of $x_1, \ldots, x_m$ into a binary value *true* or *false*. Boolean functions are basically described by Boolean expressions composed of binary variables and Boolean operators such as logical AND ($\wedge$), logical OR ($\vee$), and logical NOT ($\neg$).

Given item set $I = \{a_1, \dots, a_m\}$, *family of sets* $\mathcal{I} \subseteq 2^I$ is a set of subsets of I. Family of sets can be described in extensional notation, i.e., described as an explicit list of all the subsets in $\mathcal{I}$, or as an expression in cooperation with some set operators such as set union ($\cup$), set intersection ($\cap$), and set join ($\sqcup$). Here the set join is an operation for two sets I and J defined as: $I \sqcup J \coloneqq \{a \cup b \mid a \in I, b \in J\}$.

We can relate these two concepts. For Boolean functions, since there are two assignments, *true* or *false*, for each variable x_i, there are 2^m assignments for the set of variables $x_1, \dots, x_m$. For families of sets, there are 2^m subsets of I since we can choose whether a_i is included in or excluded from the subset for each item a_i. These observations lead to the following correspondence with Boolean functions and families of subsets: Given Boolean function f, we consider a family of sets $\mathcal{I}_f$ satisfying the condition that for any $I' = \{a_{i_1}, \dots, a_{i_k}\} \subseteq I$, $I' \in \mathcal{I}_f$ if and only if f evaluates to *true* by assigning $x_{i_1}, \dots, x_{i_k}$ *true* and the other binary variables *false*. In other words, $\mathcal{I}_f$ is the family of subsets collecting the assignments of binary variables where f evaluates to *true*. This constitutes one-to-one correspondence with Boolean functions (with m binary variables) and families of subsets (of the set with m items), and we can now identify them. Although BDDs were originally designed for representing Boolean functions and ZDDs were originally designed for representing families of subsets, we can now regard them as representing the objects of the same class.

2.2 Binary Decision Diagrams (BDDs)

A *Binary Decision Diagram* [Bryant, 1986], abbreviated as *BDD*, is a rooted directed acyclic graph (DAG)-shaped data structure to compactly represent a Boolean function. BDD $\mathtt{B} = (\mathtt{N}, \mathtt{A})$ has dnode set $\mathtt{N}$ and arc set $\mathtt{A}$; here, as described in Section 1.4, we call the nodes of BDD as *dnode*. The root dnode is denoted by $\mathtt{r} \in \mathtt{N}$. Dnode set $\mathtt{N}$ has two *terminal* dnodes $\top$ and $\bot$ that have no outgoing arcs and other internal dnodes. Each internal dnode has exactly two outgoing arcs, *lo-arc* and *hi-arc*, and the dnodes pointed by them are called *lo-child* and *hi-child*, respectively. For internal dnode $\mathtt{n} \in \mathtt{N}$, the lo-child and hi-child of $\mathtt{n}$ are denoted by $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$, respectively. Each internal dnode also has an integer ranging $[1, m]$ called *label*, and is denoted by $\mathsf{lb}(\mathtt{n})$. For convenience, we let the labels of terminals as $m + 1$. The size $|\mathtt{B}|$ of BDD $\mathtt{B}$ is defined by the number of dnodes $|\mathtt{N}|$. We impose the following *ordered* property: for any internal dnode $\mathtt{n}$, its label must be smaller than the children's one, i.e., $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathsf{lo}(\mathtt{n}))$ and $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathsf{hi}(\mathtt{n}))$ must hold. Thanks to the ordered property, the BDD structure can be seen as a DAG layered by dnodes' labels; dnodes with smaller label are placed on a higher layer.

Given BDD $\mathtt{B}$, we can associate for each dnode of $\mathtt{B}$ a Boolean function in a recursive manner, and the Boolean function $f_{\mathtt{B}}$ represented by $\mathtt{B}$ is defined as follows.

Definition 2.1. Given BDD $\mathtt{B} = (\mathtt{N}, \mathtt{A})$, Boolean function $f_{\mathtt{n}}$ represented by dnode $\mathtt{n} \in \mathtt{N}$ is defined as follows:

- If $\mathtt{n} = \top$ or $\bot$, $f_{\mathtt{n}} = \mathit{true}$ or false, respectively. Here, *true* and *false* are identity functions that evaluate *true* and *false*.
- Otherwise, $f_{\mathtt{n}} = (\neg x_{\mathsf{lb}(\mathtt{n})} \wedge f_{\mathsf{lo}(\mathtt{n})}) \vee (x_{\mathsf{lb}(\mathtt{n})} \wedge f_{\mathsf{hi}(\mathtt{n})})$, where $f_{\mathsf{lo}(\mathtt{n})}$ and $f_{\mathsf{hi}(\mathtt{n})}$ are the Boolean functions represented by dnodes $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$.

The Boolean function $f_{\mathtt{B}}$ represented by $\mathtt{B}$ is defined as the Boolean function $f_{\mathtt{r}}$ represented by root dnode $\mathtt{r} \in \mathtt{N}$.

The above definition stands on Boolean functions. We can also consider the associated *family of subsets* by considering the set of paths from root to terminal. Given path $\mathtt{R}$ from root $\mathtt{r}$ to terminal $\top$ in BDD $\mathtt{B}$, the subsets satisfying the following conditions are associated: (i) If $\mathtt{R}$ traverses hi-arc outgoing from dnode with label i, $a_i \in I$ must be included. (ii) If $\mathtt{R}$ traverses lo-arc outgoing from dnode with label i, $a_i \in I$ must not be included. (iii) If $\mathtt{R}$ does not traverse any dnode with label i, we can freely choose the inclusion of $a_i \in I$. Then, the family of subsets represented by BDD $\mathtt{B}$ is the union of associated subsets of all the paths from $\mathtt{r}$ to $\top$.

For example, Figure 2.1a depicts the BDD representing the family of subsets of $\{a_1, \ldots, a_5\}$ such that the cardinality is less than or equal to 2. In other words, the BDD in Figure 2.1a represents a Boolean function that evaluates to *true* if and only if the number of binary variables assigned to *true* is less than or equal to 2. Each internal dnode is drawn as a circle with a label inside it, and solid and dashed lines indicate hi- and lo-arcs. In this BDD, the path 1-(lo)-2-(hi)-3-(lo)-4-(lo)-$\top$ corresponds to two subsets $\{a_2\}$ and $\{a_2, a_5\}$; since it does not pass through any dnode with label 5, it does not care whether a_5 is included.

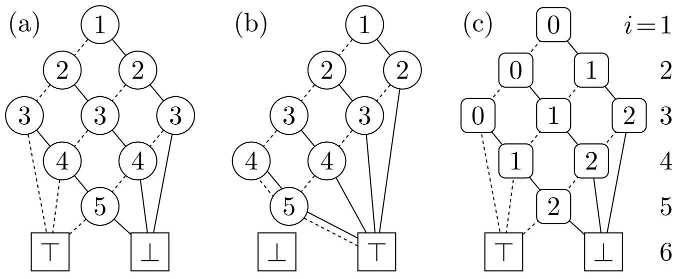


Figure 2.1: (a) BDD of the family of subsets of $\{a_1, \ldots, a_5\}$ such that the cardinality is less than or equal to 2. The integer inside an circle is a label, and solid and dashed lines indicate hi- and lo-arcs, respectively. (b) ZDD of the same family of subsets. (c) Result of Construct(*Cardinality*$_2$) when $m = 5$. The integer inside a dnode is a configure, i.e., the k' value.

2.3 Zero-suppressed Binary Decision Diagrams (ZDDs)

A *Zero-suppressed Binary Decision Diagram* [Minato, 1993], abbreviated as ZDD, is also a rooted DAG-shaped data structure. It was originally intended to compactly represent a sparse family of subsets, that frequently appears in real applications. ZDDs can be considered as a variant of BDDs, and the structure of a ZDD is identical to that of a BDD. That is, ZDD $\mathtt{Z} = (\mathtt{N}, \mathtt{A})$ has dnode set $\mathtt{N}$ and arc set $\mathtt{A}$ where the dnode set consists of two terminal dnodes $\top, \bot$ and the other internal dnodes that have two outgoing arcs. Here we use the term "dnode" since the structural property is identical. We also borrow the following structural terms of BDDs for ZDDs: lo-arc, hi-arc, lo-child, hi-child, and label. The ordered property is also imposed for ZDDs, i.e., for any internal dnode $\mathtt{n} \in \mathtt{N}$, $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathsf{lo}(\mathtt{n}))$ and $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathsf{hi}(\mathtt{n}))$ must hold. The size $|\mathtt{Z}|$ of ZDD $\mathtt{Z}$ is again defined by the number of dnodes $|\mathtt{N}|$.

The difference of ZDDs and BDDs comes from the semantics. Given ZDD $\mathtt{Z}$, we can associate for each dnode of $\mathtt{Z}$ a family of subsets in a recursive manner, and the family $\mathcal{I}_{\mathtt{Z}}$ of subsets represented by $\mathtt{Z}$ is defined as follows.

Definition 2.2. Given ZDD $\mathtt{Z} = (\mathtt{N}, \mathtt{A})$, family $\mathcal{I}_{\mathtt{n}}$ of subsets represented by dnode $\mathtt{n} \in \mathtt{N}$ is defined as follows:

- If $\mathtt{n} = \top$ or $\bot$, $\mathcal{I}_{\mathtt{n}} = \{\emptyset\}$ or $\emptyset$, respectively. Here, $\{\emptyset\}$ is the set containing only an empty set, while $\emptyset$ is simply an empty set.

- Otherwise, $\mathcal{I}_{\mathtt{n}} = \mathcal{I}_{\mathsf{lo}(\mathtt{n})} \cup [\{a_{\mathsf{lb}(\mathtt{n})}\} \sqcup \mathcal{I}_{\mathsf{hi}(\mathtt{n})}]$, where $\mathcal{I}_{\mathsf{lo}(\mathtt{n})}$ and $\mathcal{I}_{\mathsf{hi}(\mathtt{n})}$ are the families of subsets represented by dnodes $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$.

The family $\mathcal{I}_{\mathtt{Z}}$ of subsets represented by $\mathtt{Z}$ is defined as the family $\mathcal{I}_{\mathtt{r}}$ of subsets represented by root dnode $\mathtt{r} \in \mathtt{N}$.

The difference can be more clearly stated by viewing from the set of paths from root to terminal as in Section 2.2. Given path $\mathtt{R}$ from root $\mathtt{r}$ to terminal $\top$ in ZDD $\mathtt{Z}$, a subset satisfying the following conditions is associated: Item $a_i \in I$ is included if and only if $\mathtt{R}$ traverses hi-arc outgoing from dnode with label i. The difference of ZDDs and BDDs appears in the treatment of skipped labels; in BDD, for skipped label i, we can freely choose the inclusion of a_i, while in ZDD, we must exclude a_i.

For example, Figure 2.1b depicts the ZDD representing the family of subsets of $\{a_1, \ldots, a_5\}$ such that the cardinality is less than or equal to 2. In this ZDD, the path 1-(lo)-2-(hi)-3-(lo)-4-(hi)-$\top$ corresponds to subset $\{a_2, a_4\}$. Unlike BDD, only single subset is associated with a path from root to $\top$ in a ZDD.

2.4 Operations on DDs

One of the features BDDs and ZDDs have is that they can solve various problems related to Boolean functions or families of subsets in polynomial time with respect to the size of them. Here, we focus on two problems, linear optimization problem and weighted model counting problem. We describe these problems with families of subsets.

We consider the situation that family $\mathcal{I}$ of subsets of I is given in some form, e.g., in the form of a BDD or a ZDD. Then, these problems are defined as follows.

Problem 2.3. Given weight $w_a \in \mathbb{R}$ for every item $a \in I$, *linear optimization problem* is to compute subset $I' \in \mathcal{I}$ minimizing the sum of weights. In other words, this problem computes

$$\operatorname*{argmin}_{I' \in \mathcal{I}} \sum_{a \in I'} w_a. \tag{2.1}$$

If there are multiple subsets that minimize (2.1), we can choose an arbitrary one from them.

Problem 2.4. Given weights $w_a^+, w_a^- \in \mathbb{R}$ for every item $a \in I$, *weighted model counting problem* is to compute

$$\sum_{I' \in \mathcal{I}} \left[\left(\prod_{a \in I'} w_a^+ \right) \cdot \left(\prod_{a \in I \setminus I'} w_a^- \right) \right]. \tag{2.2}$$

As regards the first problem, when we set all weights w_a to 1, this problem computes the subset in $\mathcal{I}$ with the smallest number of items. As regards the second problem, when we set all weights w_a^+, w_a^- to 1, the value (2.2) equals the number of subsets in $\mathcal{I}$.

If $\mathcal{I}$ is given explicitly, i.e., $\mathcal{I}$ is given in an extensional notation, these problems can be solved in $O(|I||\mathcal{I}|)$ time by scanning all the subsets in $\mathcal{I}$. Although it is linear in $|\mathcal{I}|$, since $\mathcal{I}$ has generally an exponential number of subsets, its bound is prohibitively large. Meanwhile, if $\mathcal{I}$ is given implicitly, these problems typically fall in computationally tough complexity classes. For example, if I is the set of edges in an undirected graph and $\mathcal{I}$ is defined as the family of subgraphs of T-Steiner trees where T is the subset of vertices, Problem 2.1 is exactly the minimum Steiner tree problem, which is known to be NP-complete.

However, if a BDD or a ZDD representing $\mathcal{I}$ is given, we can solve these problems in linear time with respect to the size of the decision diagram; e.g., see [Knuth, 2011].

Theorem 2.5. *Suppose that BDD* $\mathtt{B}$ *representing* $\mathcal{I}$ *is given. Then, Problems 2.3 and 2.4 can be solved in* $O(|\mathtt{B}|)$ *time.*

Theorem 2.6. *Suppose that ZDD* $\mathtt{Z}$ *representing* $\mathcal{I}$ *is given. Then, Problems 2.3 and 2.4 can be solved in* $O(|\mathtt{Z}|)$ *time.*

These can be solved by a dynamic programming (DP) on the decision diagram structures; for BDDs, the algorithms are much similar to Algorithms C and B in [Knuth, 2011], and those are also much similar for ZDDs. For readers who are not familiar with decision diagrams, we briefly describe the algorithm for solving these problems with a ZDD.

For the linear optimization problem (Problem 2.3), as described in Section 2.3, each subset in $\mathcal{I}$ corresponds to a path from root $\mathtt{r}$ to terminal $\top$ in ZDD. Therefore, we try to solve the linear optimization by finding the shortest path from $\mathtt{r}$ to $\top$. Here, we associate arcs' length as follows: First, we set the length of each lo-arc to 0. Second, we set the length of each hi-arc outgoing from the dnode with label i to w_{a_i}, since traversing this hi-arc means the inclusion of item a_i. Then, the desired answer is the subset corresponding to the shortest path from $\mathtt{r}$ to $\top$. Since ZDD is a DAG, the shortest path can be computed in linear time with respect to its size by computing, for each dnode $\mathtt{n}$, the shortest path length from $\mathtt{r}$ in a top-down manner.

For the weighted model counting problem (Problem 2.4), we consider computing partial sums associated with dnodes in a bottom-up manner.

More specifically, for dnode $\mathtt{n} \in \mathtt{Z}$, we consider the following sum:

$$\mathsf{DP}[\mathtt{n}] := \sum_{I' \in \mathcal{I}_{\mathtt{n}}} \left[\left(\prod_{a \in I'} w_a^+ \right) \cdot \left(\prod_{a \in \{a_{\mathsf{lb}(\mathtt{n})}, \ldots, a_m\} \setminus I'} w_a^- \right) \right], \tag{2.3}$$

where $\mathcal{I}_{\mathtt{n}}$ is the family of subsets associated with dnode $\mathtt{n}$ defined in Definition 2.2. Although (2.3) is the sum of potentially exponentially many products, DP has the following recursive formula:

$$\mathsf{DP}[\mathtt{n}] = w_{a_{\mathsf{lb}(\mathtt{n})}}^- \cdot \left(\prod_{i=\mathsf{lb}(\mathtt{n})+1}^{\mathsf{lb}(\mathsf{lo}(\mathtt{n}))} w_{a_i}^- \right) \cdot \mathsf{DP}[\mathsf{lo}(\mathtt{n})] + w_{a_{\mathsf{lb}(\mathtt{n})}}^+ \cdot \left(\prod_{i=\mathsf{lb}(\mathtt{n})+1}^{\mathsf{lb}(\mathsf{hi}(\mathtt{n}))} w_{a_i}^- \right) \cdot \mathsf{DP}[\mathsf{hi}(\mathtt{n})]. \tag{2.4}$$

Starting from $\mathsf{DP}[\top] = 1$ and $\mathsf{DP}[\bot] = 0$ and computing DP in a bottom-up manner, we can obtain the value of (2.2) by $(\prod_{i=1}^{\mathsf{lb}(\mathtt{r})-1} w_{a_i}^-) \cdot \mathsf{DP}[\mathtt{r}]$, where $\mathtt{r}$ is the root dnode.

2.5 Top-Down Construction Methods

In Section 2.4, we observe that once the BDD or ZDD is built, we can solve some difficult problems regarding the family of subsets. The remaining is how to build BDDs and ZDDs. One choice is *bottom-up construction* where a decision diagram is built from a recursive composition of small decision diagrams. It may be a nice choice when the family of subsets (or the Boolean function) is represented as a mathematical expression (or a Boolean expression).

However, for network analysis applications, such a situation is rare. For example, in the congestion analysis of telecommunication networks, the family $\mathcal{I}$ will be defined as the set of all paths or all Steiner trees. In such a case, *top-down construction* of decision diagrams may be a good choice.

As far as we know, the idea of top-down construction originates in the paper of Sekine *et al.* [1995], where their focus was on the computation of Tutte polynomial of the graph. Now their method can be seen as constructing a BDD representing the family of all the spanning trees. Since then, top-down construction algorithms for various families of subsets have been developed. Among them, the most famous one is Knuth's Simpath algorithm [Knuth, 2011], which builds a ZDD representing the family of all the simple s,t-paths. By expanding such research, Kawahara *et al.* [2017b] proposed frontier-based search, which can build a ZDD of various graph structures such as connected component, Steiner tree, and rooted spanning forest. Hereafter, we explain the framework of top-down construction in accordance with the paper of

Iwashita and Minato [2013]. Note that since we can build ZDDs by only minor modifications, we first explain the framework for constructing BDDs, and then we explain the difference when ZDDs are constructed.

We start with a naive construction of BDD. We order the items $a_1, \ldots, a_m$ and we consider the process of determining one by one whether a_i is excluded from or included in the subset one by one. By repetitively trying this process with different choices, a binary decision tree is generated. At the i-th level of this tree, all the subsets of $I_{<i} \coloneqq \{a_1, \ldots, a_{i-1}\}$ are enumerated, which we call *i-th subsets*. After all, the leaves corresponding to the subsets included in the desired family are marked $\top$, while the other leaves are marked $\bot$. Finally, BDD can be built by merging identical subtrees of it.

The top-down construction [Iwashita and Minato, 2013] refines this idea by introducing *configures* of i-th subsets. Configures define equivalence classes among i-th subsets. More specifically, we design specific configures for every i-th subset with respect to the desired family $\mathcal{I}$ of subsets satisfying the following conditions:

(i) For any two i-th subsets $X, Y \subseteq I_{<i}$ whose configures are identical, for any $Z \subseteq I_{\geq i} \coloneqq \{a_i, \ldots, a_m\}$, $X \cup Z$ and $Y \cup Z$ are equivalent in whether it is included in $\mathcal{I}$.

(ii) For any two i-th subsets $X, Y \subseteq I_{<i}$ whose configures are identical, the configures of $(i+1)$-st subsets X and Y, and those of $X \cup \{a_i\}$ and $Y \cup \{a_i\}$, must be identical for each.

Regarding condition (ii), for configure s of i-th subset, the configures of $(i+1)$-st subsets made by excluding or including a_i are called *lo-* and *hi-child configures*, respectively.

If such a configure can be designed for the desired family $\mathcal{I}$ of subsets, the procedure of top-down construction can be described as Algorithm 2.1. Here $\mathtt{L}_i$ is the set of all the dnodes with label i. $S.$Root and $S.$Child procedures are designed for every class S of desired families. Root() returns the level and the configure of the root of decision tree, and Child($\langle i, s\rangle, f \in \{\text{lo}, \text{hi}\}$) receives configure s of i-th subset and returns the f-child configure (with its level) of s. Note that Child can return $\langle m+1, 0\rangle$ only if all the descendants are $\bot$, and $\langle m+1, 1\rangle$ only if all the descendants are $\top$. First, in Lines 2–3, the root dnode `r` is created by the Root procedure. After that, all the created dnodes are scanned in a top-down manner (Lines 4–5). In Line 8, the f-child configure of `n`'s configure is generated. If Child returns $\langle m+1, 0\rangle$ or $\langle m+1, 1\rangle$, we set the appropriate terminal to the f-child of `n` (Lines 9–10). Otherwise, we search a dnode of label i whose configure is the generated one (Line 12). If such a dnode exists, we set it to the f-child of `n` (Line 13).

Algorithm 2.1: Top-down construction framework.

```
procedure Construct($S$):
    $\langle i_0, s_0 \rangle \leftarrow S$.Root()
    Create root dnode $\mathtt{r} \in \mathtt{L}_{i_0}$ whose configure is $s_0$
    for $i = i_0$ to $m$ do
        foreach $\mathtt{n} \in \mathtt{L}_i$ do
            foreach $f \in \{\mathrm{lo}, \mathrm{hi}\}$ do
                $s \leftarrow$ (n's configure)
                $\langle i', s' \rangle \leftarrow S$.Child($\langle i, s \rangle, f$)
                if $\langle i', s' \rangle = \langle m+1, 0 \rangle$ then Set $\perp$ to the $f$-child of n
                else if $\langle i', s' \rangle = \langle m+1, 1 \rangle$ then Set $\top$ to the $f$-child of n
                else
                    if $\exists \mathtt{n}' \in \mathtt{L}_{i'}$ s.t. configure is $s'$ then
                        Set $\mathtt{n}'$ to the $f$-child of n
                    else
                        Create dnode $\mathtt{n}'' \in \mathtt{L}_{i'}$ whose configure is $s'$
                        Set $\mathtt{n}''$ to the $f$-child of n
```

Algorithm 2.2: Procedures for the family of subsets whose cardinality is less than or equal to k.

```
procedure $Cardinality_k$.Root():
    return $\langle 1, 0 \rangle$
procedure $Cardinality_k$.Child($\langle i, k' \rangle, f$):
    if $f = \mathrm{hi}$ then $k' \leftarrow k' + 1$
    if $k' > k$ then return $\langle m+1, 0 \rangle$
    else if $k' + (m - i) \leq k$ then return $\langle m+1, 1 \rangle$
    else return $\langle i+1, k' \rangle$
```

Otherwise, a new dnode $\mathtt{n}''$ with label i' and configure s' is generated (Line 15) and we set it to the f-child of $\mathtt{n}$ (Line 16). Compared to the decision tree where $(i+1)$-st subsets are generated from i-th subsets directly, Algorithm 2.1 generates the configures of $(i+1)$-st subsets from those of i-th subsets.

As a brief example, we design configures for a family of subsets whose cardinality is less than or equal to k. For such a class of families, we can take an integer indicating the cardinality of i-th subset as a configure. At root, the configure of an empty set is simply 0. For a dnode whose configure is k', the lo-child configure is k' and the hi-child configure is $k'+1$. If the configure exceeds k, we can return $\langle m+1, 0 \rangle$ since the cardinality must exceed k. If the label is i and the configure is smaller than or equal to $k - (m - i + 1)$, we can return $\langle m+1, 1 \rangle$ since the cardinality never exceeds k even if all the remaining items in $I_{\geq i} = \{a_i, \ldots, a_m\}$ are included in the subset. Now the Root and Child procedures can be described as Algorithm 2.2.

Figure 2.1c depicts the resultant BDD of CONSTRUCT(*Cardinality*$_2$) with $m = 5$. In other words, the construction of a BDD representing family of subsets whose cardinality is less than or equal to 2 is drawn in Figure 2.1c. The integer inside each dnode (rounded squares) indicates the k' value.

Finally, we mention the difference when ZDDs are constructed. Indeed, even for ZDDs, the framework of top-down construction described in Algorithm 2.1 remains unchanged. The difference appears in the specification of CHILD procedure when $\langle m+1, 1\rangle$ is returned. In constructing ZDD, CHILD returns $\langle m+1, 1\rangle$ only if the descendant reached by excluding all the remaining items in $I_{\geq i+1}$ is $\top$ and all the other descendants are $\perp$. This comes from the difference of BDDs and ZDDs described in the last paragraph of Section 2.3; in BDD, for skipped label i, we can freely choose the inclusion of a_i, while in ZDD, we must exclude a_i.

Chapter 3

Equilibrium Computation of Combinatorial Congestion Games

Computation of equilibria for congestion games has been an important research subject. In many realistic scenarios including network congestion analysis, each strategy of congestion games is given by a combination of elements that satisfies certain constraints; such games are called combinatorial congestion games. For example, given a road network with some toll roads, each strategy of routing games is a path (a combination of edges) whose total toll satisfies a certain budget constraint. Generally, given an item set of n elements, the set of all such strategies, called the strategy set, can be large exponentially in n, and it often has complicated structures; these issues make equilibrium computation very hard. In this chapter, we propose a practical algorithm for such hard equilibrium computation problems. We use ZDDs to compactly represent strategy sets, and we develop a Frank–Wolfe-style iterative equilibrium computation algorithm whose per-iteration complexity is linear in the size of the ZDD representation. We prove that an ϵ-approximate Wardrop equilibrium can be computed in $O(\mathrm{poly}(m)/\epsilon)$ iterations, and we improve the result to $O(\mathrm{poly}(m)\log\epsilon^{-1})$ for some special cases. Experiments confirm the practical utility of the proposed method.

3.1 Introduction

Congestion games form an important subclass of non-cooperative games since they can model various resource allocation scenarios where every user acts selfishly. Motivated by a wide variety of applications, computation of equilibria for congestion games, which enable us to examine structures of equilibria (e.g., the price of anarchy), has been an attracting research subject. In many practical congestion games including network congestion analysis, each strategy forms a subset of a finite ground set, $I \coloneqq \{a_1, \dots, a_m\}$, and the set of all strategies, or the strategy set, can be large exponentially in m. We call such games *combinatorial congestion games*. A typical example is selfish routing [Roughgarden, 2005]: Given a graph with edge set $I = E$ and origin-destination pair (s, t), each player chooses an s–t path selfishly to go from s to t as fast as possible. In this case, the strategy set is the collection of all s–t paths that is a family of subsets of E, whose size is generally exponential in m. For the standard selfish routing, efficient equilibrium computation methods have been developed [Fabrikant *et al.*, 2004; Thai, 2017]. These methods handle the huge strategy sets by utilizing efficient polynomial-time algorithms for the min-cost flow or shortest path problems. In more general settings, however, strategy sets can have more complicated structures, for which such polynomial-time algorithms are unavailable. Below we list two such examples:

Budgeted Selfish Routing.

We consider a variant of selfish routing based on [Jahn *et al.*, 2005]: The graph representing a road network has some edges corresponding to toll roads, and players do not choose s–t paths whose total toll is too expensive. Namely, each strategy must satisfy a certain budget constraint. Given a cost value for each edge, which indicates the degree of congestion, to find the minimum-cost s–t path is at least as hard as the NP-hard knapsack problem.

Multi-location Communication.

We consider the multi-location communication problem on a network based on [Imase and Waxman, 1991]. Let I be an edge set E of a graph representing the communication network. Each edge connects two cities, and there are some special cities called *terminals*. In this game, a player is a group of people who are in the terminals and have a multi-person conference. Each strategy is a Steiner tree that includes all the terminals, and each player chooses a Steiner tree selfishly to minimize communication delay. Given the degree of delay for each edge, the problem of finding the Steiner tree with the smallest delay reduces to the APX-hard minimum Steiner tree problem.

Congestion games have many other applications related to resource allocation on networks [Shakkottai *et al.*, 2007; Han *et al.*, 2012], and their combinatorial variants can naturally appear in practice. Therefore, to develop practical equilibrium computation methods that can handle various combinatorial strategy sets is important for advancing studies into real-world congestion games. However, we are currently missing such general methods due to the difficulty of dealing with a wide variety of huge and complicated strategy sets.

3.1.1 Contribution

We develop a practical equilibrium computation algorithm for combinatorial congestion games. Motivated by the fact that many real-world games have a huge number of players, we employ the continuous-player setting [Sandholm, 2001], where there are infinitely many players with an infinitesimal mass. As elucidated later, equilibrium computation for such games can be reduced to constrained potential function minimization problems, which are generally NP-hard due to the complicated structures of strategy sets. We address this hardness by using ZDDs [Minato, 1993] explained in Section 2.3, which can represent various combinatorial strategy sets compactly. We minimize potential functions with a Frank–Wolfe-style iterative algorithm [Lacoste-Julien and Jaggi, 2015] that utilizes the compact ZDD representation. In practice, since the algorithm is performed with finite precision, output solutions generally have objective errors. Fortunately, however, we can prove that the solutions achieve an ϵ-approximate Wardrop equilibrium. Below we detail our contributions:

- We propose a Frank–Wolfe-style iterative equilibrium computation algorithm for continuous-player combinatorial congestion games, whose per-iteration complexity is linear in the total size of ZDDs representing strategy sets.
- We prove that our algorithm outputs an ϵ-approximate Wardrop equilibrium in $O(\text{poly}(m)/\epsilon)$ iterations if potential functions have differentiability, convexity, and Lipschitz-continuous gradient. We also improve the result to $O(\text{poly}(m)\log\epsilon^{-1})$ with some additional assumptions.
- We validate the proposed method via experiments. The results demonstrate that our method is useful for practical equilibrium computation problems.

While ZDDs can generally be large exponentially in m, their empirical sizes are often small enough for practical use. Moreover, ZDD sizes can sometimes be polynomial in m. In such cases, our method can compute an

ϵ-approximate Wardrop equilibrium in $O(\text{poly}(m)/\epsilon)$ (or $O(\text{poly}(m)\log\epsilon^{-1})$) time.

Our method can also compute the social optimum as in Section 3.2. Therefore, our method can be used for examining the price of anarchy, which is defined by the ratio of the total cost value of the equilibrium to that of the social optimum.

3.1.2 Related Work

Continuous-player games and their connection to continuous optimization have been widely studied. Beckmann *et al.* [1956] first used a convex optimization formulation to study traffic equilibria. Sandholm [2001] established a general framework of continuous-player games characterized as potential function minimization, called potential games, but combinatorial strategy sets were not considered. As regards the combinatorial setting, many existing works are devoted to selfish routing [Roughgarden, 2005]: Roughgarden and Tardos [2002] studied the inefficiency of equilibria, or the price of anarchy. Convergence of various dynamics to equilibria were also studied in [Fischer and Vöcking, 2004; Blum *et al.*, 2006; Fischer *et al.*, 2010]. Fabrikant *et al.* [2004] developed a polynomial-time algorithm for computing an approximate equilibrium. Optimization-based algorithms, including the Frank–Wolfe algorithm, for equilibrium computation have also been widely studied [Vliet, 1987; Bar-Gera, 2002; Correa and Stier-Moses, 2011; Thai, 2017]; note that the work in this chapter is the first that shows the relationship between the precision of a Frank–Wolfe-style algorithm and approximate Wardrop equilibria. Congestion games other than selfish routing have also been studied [Altman *et al.*, 2006; Shakkottai *et al.*, 2007; Han *et al.*, 2012], but those did not consider addressing the computational hardness caused by the huge and complicated strategy sets. Jahn *et al.* [2005] addressed a computationally hard problem of finding system-optimal flow for the budgeted routing setting. Their approach is, however, based on a linear integer programming formulation specialized for the budgeted routing setting, while our method can work with various combinatorial strategy sets that can be represented with ZDDs, e.g., see [Kawahara *et al.*, 2017b].

The Frank–Wolfe algorithm [Frank and Wolfe, 1956], a.k.a. conditional gradient algorithm, is a well-established algorithm for constrained convex minimization, and it has recently been attracting much attention thanks to the useful convergence analysis based on the duality gap [Jaggi, 2013]. Lacoste-Julien and Jaggi [2015] studied several variants of the Frank–Wolfe algorithm and proved their global linear convergence under some assumptions; our method uses one of the variants called the fully-corrective Frank–

Wolfe algorithm. Abernethy and Wang [2017] studied the Frank–Wolfe algorithm from a perspective of zero-sum games; note that their research direction is different from ours. As explained in Section 3.2.1, a typical bottleneck of the Frank–Wolfe algorithm is a linear optimization step. While this bottleneck can be resolved in some cases [Lacoste-Julien *et al.*, 2013; Kerdreux *et al.*, 2018; Braun *et al.*, 2019], these techniques do not work in our case with huge and complicated strategy sets.

3.2 Problem Settings

Let $[r] \coloneqq \{1, \dots, r\}$ be the set of populations. We consider the combinatorial setting: Given a finite ground set $I \coloneqq \{a_1, \dots, a_m\}$, each *strategy* is given by $S \subseteq I$, i.e., a subset of I. For each $p \in [r]$, we define strategy set $\mathcal{S}^p \coloneqq \{S_1, \dots, S_{|\mathcal{S}^p|}\} \subseteq 2^I$. Given any $S \subseteq I$, we let $\mathbf{1}_S \in \{0,1\}^m$ denote an indicator vector whose i-th entry is 1 if $a_i \in S$ and 0 otherwise. We sometimes abuse the notation and regard $\mathbf{1}_S$ in the same light as S.

We then explain the continuous-player setting. Each $p \in [r]$ has infinitely many players with an infinitesimal mass; let $w^p \in \mathbb{R}$ be the total mass. We assume $\sum_{p\in[r]} w^p = 1$ without loss of generality. Let $z^p_S \in [0,1]$ denote the proportion of players who choose strategy $S \in \mathcal{S}^p$; we have $\sum_{S\in\mathcal{S}^p} z^p_S = 1$. Consequently, $w^p z^p_S$ represents the mass of players in p who choose $S \in \mathcal{S}^p$. We call $\boldsymbol{z}^p = (z^p_{S_1}, \dots, z^p_{S_{|\mathcal{S}^p|}})$ a *strategy profile of* p and $\boldsymbol{z} = (\boldsymbol{z}^1, \dots, \boldsymbol{z}^r)$ a *strategy profile.*

We finally explain the congestion game setting. Given any strategy profile $\boldsymbol{z}$, we let $\boldsymbol{x}^p = \sum_{S\in\mathcal{S}^p} z^p_S \mathbf{1}_S \in [0,1]^n$ for each $p \in [r]$ and $\boldsymbol{x} = (\boldsymbol{x}^1, \dots, \boldsymbol{x}^r)$. Note that the i-th entry of $\boldsymbol{y} \coloneqq \sum_{p\in[r]} w^p \boldsymbol{x}^p$, denoted by y_i, represents the mass of players who choose strategies including $a_i \in I$. For any $p \in [r]$, the cost of strategy $S \in \mathcal{S}^p$ is given by $c_S(\boldsymbol{y}) \coloneqq \sum_{i:a_i\in S} c_i(y_i)$, where $c_i(\theta)$ ($\theta \in [0,1]$) is a function that indicates the cost of using $a_i \in I$ when the mass of players using $a_i \in I$ is θ. For convenience, we also define cost functions on $\mathbb{R}^{rn}$ as $C_S(\boldsymbol{x}) \coloneqq c_S(\sum_{p\in[r]} w^p \boldsymbol{x}^p)$. Each player chooses a strategy selfishly to minimize the cost. We define potential function $\Phi : \mathbb{R}^n \to \mathbb{R}$ as $\Phi(\boldsymbol{y}) \coloneqq \sum_{i:a_i\in I} \int_0^{y_i} c_i(\theta)\mathrm{d}\theta$. Note that we have $c_i(\boldsymbol{y}) = \nabla\Phi(\boldsymbol{y})_i$ and $c_S(\boldsymbol{y}) = \nabla\Phi(\boldsymbol{y})^\top \mathbf{1}_S$. We impose some assumptions on the cost and potential functions as summarized below; particularly, we assume $\Phi(\cdot)$ to be convex. As in [Sandholm, 2001], strategy profile $\boldsymbol{z}$ achieves an equilibrium iff it satisfies the KKT condition; thanks to the convexity of $\Phi(\cdot)$, it is characterized as a minimizer of

$$\underset{\boldsymbol{z}\geq\mathbf{0}}{\text{minimize}} \quad \Phi\left(\sum_{p\in[r]} w^p \sum_{S\in\mathcal{S}^p} z_S^p \mathbf{1}_S\right) \tag{3.1}$$

$$\text{subject to} \quad \sum_{S\in\mathcal{S}^p} z_S^p = 1 \quad (\forall p \in [r]). \tag{3.2}$$

Since $\boldsymbol{z}$ generally consists of exponentially many variables, to directly solve this problem is too expensive; hence we consider reformulating it. We define $\mathcal{V} \coloneqq \mathcal{S}^1 \times \cdots \times \mathcal{S}^r$. Since the Cartesian products "$\times$" and convex hulls "conv$(\cdot)$" are commutative, we have $\mathrm{conv}(\mathcal{S}^1) \times \cdots \times \mathrm{conv}(\mathcal{S}^r) = \mathrm{conv}(\mathcal{V})$, and so the above problem can be equivalently rewritten as

$$\begin{aligned} \underset{\boldsymbol{x}=(\boldsymbol{x}^1,\dots,\boldsymbol{x}^r)\in\mathbb{R}^{rm}}{\text{minimize}} \quad & F(\boldsymbol{x}) \coloneqq \Phi\left(\sum_{p\in[r]} w^p \boldsymbol{x}^p\right) \\ \text{subject to} \quad & \boldsymbol{x} \in \mathrm{conv}(\mathcal{V}), \end{aligned} \tag{3.3}$$

which has rm variables. We aim to solve this problem in what follows. Note that, for any $p \in [r]$, the gradient of $F(\boldsymbol{x})$ w.r.t. $\boldsymbol{x}^p$, denoted by $\nabla_p F(\boldsymbol{x})$, is given by $w^p \nabla\Phi(\boldsymbol{y}) \in \mathbb{R}^n$, where $\boldsymbol{y} = \sum_{p\in[r]} w^p \boldsymbol{x}^p$.

We need some remarks. Although the minimizer, $\boldsymbol{x}$, of (3.3) indicates the proportion of players choosing $a_i \in I$ for each $p \in [r]$, it does not give us strategy profile $\boldsymbol{z}$. Fortunately, however, the Frank–Wolfe-style algorithms output a solution as a convex combination of vertices, which enable us to obtain a strategy profile (see, Section 3.3.2). As in [Roughgarden, 2005], we can confirm that any minimizer of (3.3) has the following uniqueness of cost values: Given any two minimizers, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, of (3.3), we have $c_i(\boldsymbol{y}_1) = c_i(\boldsymbol{y}_2)$ $(\forall i : a_i \in I)$, where $\boldsymbol{y}_j = \sum_{p\in[r]} \boldsymbol{x}_j^p$ $(j = 1, 2)$. Therefore, the total cost of an equilibrium, $\sum_{i:a_i\in I} y_i \cdot c_i(y_i)$, can be uniquely computed by solving (3.3). Furthermore, computation of the social optimum can be formulated in almost the same manner as (3.3): We minimize $\Psi(\boldsymbol{y}) \coloneqq \sum_{i:a_i\in I} y_i \cdot c_i(y_i)$ instead of $\Phi(\boldsymbol{y})$, which our method can (approximately) solve under some assumptions detailed below. The price of anarchy can be computed as the ratio of those two total costs.

Assumptions on Cost and Potential Functions. We assume $c_i(\cdot)$ $(\forall i : a_i \in I)$ to be non-decreasing on $[0, 1]$, which implies that $\Phi(\cdot)$ and $F(\cdot)$ are convex on $[0, 1]^n$ and $[0, 1]^{rm}$, respectively. We also make the following assumptions to guarantee the convergence the Frank–Wolfe-style algorithm.

Let $\overline{w} := \max_{p\in[r]} w^p$ and $\underline{w} := \min_{p\in[r]} w^p$. We assume $\Phi(\cdot)$ to have L-Lipschitz-continuous gradient ($L > 0$) on $[0,1]^n$, which is equivalent to L-Lipschitz-continuity of $c_i : [0,1] \to \mathbb{R}$ ($\forall i : a_i \in I$); i.e., $|c_i(y_1) - c_i(y_2)| \leq L|y_1 - y_2|$ for any $y_1, y_2 \in [0,1]$. This is a common assumption (see, e.g., [Fabrikant *et al.*, 2004]). Note that, if $\Phi(\cdot)$ has L-Lipschitz-continuous gradient, $F(\cdot)$ also has Lipschitz-continuous gradient with constant $L\overline{w} \leq L$. In Theorem 3.2, we obtain an improved result by assuming $F(\cdot)$ to be strongly convex with some constant $\mu > 0$: $F(\boldsymbol{x} + \boldsymbol{d}) \geq F(\boldsymbol{x}) + \langle \nabla F(\boldsymbol{x}), \boldsymbol{d}\rangle + \frac{\mu}{2}\|\boldsymbol{d}\|_2^2$ for any $\boldsymbol{x}, \boldsymbol{d} \in \mathbb{R}^{rm}$. Note that, if $\Phi(\cdot)$ is μ-strongly convex, then $F(\cdot)$ is also strongly convex with constant $\mu\underline{w}$. Below we present a standard example setting satisfying the above assumptions.

When computing the social optimum, we need additional assumptions as in [Roughgarden, 2005]. We assume $c_i(\cdot)$ to be differentiable and semi-convex (i.e., $y \cdot c_i(y)$ is convex w.r.t. y) on $[0,1]$, which makes $\Psi(\cdot)$ differentiable and convex. We also assume $\Psi(\cdot)$ to have Lipschitz-continuous gradient.

Example: Budgeted Selfish Routing. Given a graph with edge set $I = E$, edge weights $T_1, \dots, T_m$ (where T_i can be considered as a toll of e_i), origin-destination pairs $(s_1, t_1), \dots, (s_r, t_r)$, and budget value W, each player at s_p selfishly chooses an s_p–t_p path whose total weight is at most W. In this case, each strategy $S \in \mathcal{S}^p$ is an edge subset that forms an s_p–t_p path and satisfies $\sum_{i:a_i\in S} T_i \leq W$. If the congestion degree of each edge $i \in I$ increases linearly in the amount of traffic, the cost function of $a_i \in I$ is given by $c_i(y_i) = A_i y_i + B_i$ with some $A_i, B_i \geq 0$. The potential function is $\Phi(\boldsymbol{y}) = \sum_{i:a_i\in I} \int_0^{y_i} c_i(\theta)\mathrm{d}\theta = \sum_{i\in I}(\frac{1}{2}A_i y_i^2 + B_i y_i)$, which is differentiable and convex. Note that each $c_i(\cdot)$ is A_i-Lipschitz continuous and that $\Phi(\cdot)$ is strongly convex with constant $\min_{i:a_i\in I} A_i$. Furthermore, $\Psi(\boldsymbol{y}) = \sum_{i:a_i\in I} y_i \cdot c_i(y_i) = \sum_{i:a_i\in I}(A_i y_i^2 + B_i y_i)$ is differentiable, convex, and has Lipschitz-continuous gradient with constant $2 \times \max_{i:a_i\in I} A_i$.

3.2.1 Frank–Wolfe Algorithm

In this section, we review Frank–Wolfe algorithm [Frank and Wolfe, 1956]. The Frank–Wolfe algorithm is an effective approach to constrained convex minimization problem $\min_{\boldsymbol{x}\in\mathcal{D}} f(\boldsymbol{x})$, where $\mathcal{D} \subseteq \mathbb{R}^m$ is a convex feasible region and $f(\cdot)$ is a convex objective function. Specifically, given an initial point $\boldsymbol{x}_0 \in \mathcal{D}$, we update it for $k = 0, \dots, K$ as follows:

1. $\boldsymbol{s}_k \in \mathrm{argmin}_{\boldsymbol{v}\in\mathcal{D}} \langle \nabla f(\boldsymbol{x}_k), \boldsymbol{v}\rangle$,

2. $\boldsymbol{x}_{k+1} = (1 - \gamma_k)\boldsymbol{x}_k + \gamma_k \boldsymbol{s}_k$,

where γ_k is usually set to $\frac{2}{k+2}$. It is known that $\boldsymbol{x}_k$ converges to an optimal solution, $\boldsymbol{x}^*$, at a rate of $O(\mathrm{poly}(m)/k)$ if $f(\cdot)$ has Lipschitz-continuous gradient [Frank and Wolfe, 1956; Jaggi, 2013]. Therefore, if the linear optimization problem on $\mathcal{D}$ can be solved, we can obtain a sequence of solutions that converges to $\boldsymbol{x}^*$.

Frank–Wolfe-style algorithms require to solve linear optimization problems time and again as in Step 1, and thus how to solve it heavily affects the efficiency of the algorithms. For the case of standard selfish routing, the linear optimization can be solved efficiently with algorithms for finding the shortest path, but this is not always the case with other settings. For example, if $\mathcal{S}^p$ ($p \in [r]$) is a collection of Steiner trees on a given graph with terminals, to solve the linear optimization problem on $\mathcal{D} = \mathrm{conv}(\mathcal{V})$ is at least as difficult as the APX-hard minimum Steiner tree problem. Note that, while the Frank–Wolfe algorithm accepts additive errors when solving the linear optimization [Jaggi, 2013], the error value must converge to 0 as k increases. To achieve this is still difficult particularly when the linear optimization is APX-hard, which is true if $\mathcal{S}^p$ is the collection of Steiner trees. Jaggi [2013] also mentioned the difficulty of performing the Frank–Wolfe algorithm for the case where the linear optimization step is NP-hard. In Section 3.3.1, we address this hardness by using ZDDs.

3.3 Proposed Method

A high-level sketch of our method is as follows: We first construct ZDD $\mathtt{Z}_{\mathcal{S}^p}$ that represent $\mathcal{S}^p$ for each $p \in [r]$, and then we perform the fully corrective Frank–Wolfe algorithm (FCFW) by utilizing the ZDD representation of the strategy sets. Our algorithm, described in Algorithm 3.1, consists of two main building blocks: Linear optimization with ZDDs (Step 8) and update of solutions (Step 12), called full correction, which we detail in Sections 3.3.1 and 3.3.2, respectively. Section 3.3.2 also explains how to obtain a strategy profile, and we prove that the obtained solution achieves an ϵ-approximate Wardrop equilibrium in Section 3.3.3.

3.3.1 Linear Optimization with ZDDs

In Step 8, we need to solve $\boldsymbol{s}_k^p \in \mathrm{argmin}_{\boldsymbol{u}\in\mathcal{S}^p} \langle \nabla\Phi(\boldsymbol{y}_k), \boldsymbol{u}\rangle$ for each $p \in [r]$. Note that, since $\langle \nabla\Phi(\boldsymbol{y}_k), \boldsymbol{u}\rangle$ is linear with respect to $\boldsymbol{u}$ and $\nabla_p F(\boldsymbol{x}_k) = w^p \nabla\Phi(\boldsymbol{y}_k)$ holds, we can rewrite the linear optimization as the optimization problem on the polytope: $\boldsymbol{s}_k^p \in \mathrm{argmin}_{\boldsymbol{u}\in\mathrm{conv}(\mathcal{S}^p)} \langle \nabla_p F(\boldsymbol{x}_k), \boldsymbol{u}\rangle$. Hence $\boldsymbol{s}_k \coloneqq (\boldsymbol{s}_k^1, \dots, \boldsymbol{s}_k^r) \in \mathrm{argmin}_{\boldsymbol{v}\in\mathrm{conv}(\mathcal{V})} \langle \nabla F(\boldsymbol{x}_k), \boldsymbol{v}\rangle$ holds, which implies that Step 8

Algorithm 3.1: FCFW with ZDDs for equilibrium computation

```
1  Construct $\mathsf{Z}_{\mathcal{S}^1}, \dots, \mathsf{Z}_{\mathcal{S}^r}$
2  Choose $S^p \in \mathcal{S}^p$ for $p \in [r]$
3  Let $\boldsymbol{x}_0^p = \mathbf{1}_{S^p}$, $\mathcal{A}_0^p = \{S^p\}$, and $z_{S^p}^p = 1$ for $p \in [r]$
4  $\boldsymbol{x}_0 = (\boldsymbol{x}_0^1, \dots, \boldsymbol{x}_0^r)$
5  for $k = 0$ to $K$ do
6      $\boldsymbol{y}_k = \sum_{p\in[r]} w^p \boldsymbol{x}_k^p$
7      foreach $p \in [r]$ do
8          Compute $\boldsymbol{s}_k^p \in \mathrm{argmin}_{\boldsymbol{u}\in\mathcal{S}^p} \langle \nabla\Phi(\boldsymbol{y}_k), \boldsymbol{u}\rangle$ by DP on $\mathsf{Z}_{\mathcal{S}^p}$
9          Let $\boldsymbol{d}_k^p = \boldsymbol{s}_k^p - \boldsymbol{x}_k^p$ and $g_k^p = \langle -\nabla\Phi(\boldsymbol{y}_k), \boldsymbol{d}_k^p\rangle$
10     if $\max_{p\in[r]} g_k^p \le \epsilon$ then
11         return $\boldsymbol{x}_k$, $\{\mathcal{A}_k^p\}_{p\in[r]}$, and $\{\{z_S^p\}_{S\in\mathcal{A}_k^p}\}_{p\in[r]}$
12     Execute Correction($\boldsymbol{x}_k, \{\mathcal{A}_k^p\}_{p\in[r]}, \{\boldsymbol{s}_k^p\}_{p\in[r]}, \epsilon$) to get
         $\boldsymbol{x}_{k+1} = (\boldsymbol{x}_{k+1}^1, \dots, \boldsymbol{x}_{k+1}^r)$, $\{\mathcal{A}_{k+1}^p\}_{p\in[r]}$, and $\{\{z_S^p\}_{S\in\mathcal{A}_{k+1}^p}\}_{p\in[r]}$
```

Algorithm 3.2: Correction($\boldsymbol{x}_k, \{\mathcal{A}_k^p\}_{p\in[r]}, \{\boldsymbol{s}_k^p\}_{p\in[r]}, \epsilon$)

```
1  Find $\boldsymbol{x}_{k+1} = (\boldsymbol{x}_{k+1}^1, \dots, \boldsymbol{x}_{k+1}^r)$ , $\{\mathcal{A}_{k+1}^p\}_{p\in[r]}$, and $\{\{z_S^p\}_{S\in\mathcal{A}_{k+1}^p}\}_{p\in[r]}$ satisfying
   the following conditions:
     1. $\boldsymbol{x}_{k+1}^p = \sum_{S\in\mathcal{A}_{k+1}^p} z_S^p \mathbf{1}_S$ for $p \in [r]$.
     2. $F(\boldsymbol{x}_{k+1}) \le \min_{\gamma\in[0,1]} F(\boldsymbol{x}_k + \gamma(\boldsymbol{s}_k - \boldsymbol{x}_k))$, where $\boldsymbol{s}_k = (\boldsymbol{s}_k^1, \dots, \boldsymbol{s}_k^r)$.
     3. $\max_{p\in[r]} \max_{\boldsymbol{u}\in\mathcal{A}_{k+1}^p} \langle -\nabla\Phi(\boldsymbol{y}_{k+1}), \boldsymbol{x}_{k+1}^p - \boldsymbol{u}\rangle \le \epsilon$, where $\boldsymbol{y}_{k+1} = \sum_{p\in[r]} w^p \boldsymbol{x}_{k+1}^p$.
```

can be seen as the linear optimization step of the Frank–Wolfe algorithm in Section 3.2.1. Since this linear optimization problem is generally hard to solve as mentioned in Section 3.2.1, we consider performing this step efficiently by utilizing ZDDs.

Indeed, given $\mathsf{Z}_{\mathcal{S}^p}$, $\boldsymbol{s}_k^p \in \mathrm{argmin}_{\boldsymbol{u}\in\mathcal{S}^p} \langle \nabla\Phi(\boldsymbol{y}_k), \boldsymbol{u}\rangle$ can be solved in linear time with respect to $|\mathsf{Z}_{\mathcal{S}^p}|$ since it is equivalent to solve linear optimization problem (Problem 2.3) with family $\mathcal{S}^p$ and weights $\nabla\Phi(\boldsymbol{y}_k)$.

This ZDD-based linear optimization is suitable as a subroutine of Frank–Wolfe-style algorithms since, once ZDDs are constructed, we can reuse them for $k = 0, \dots, K$. To the best of our knowledge, this is the first Frank–Wolfe-style algorithm that takes advantages of ZDDs. The complexity of Step 8, which is the most expensive step, is linear in the total size of ZDDs: $\sum_{p\in[r]} |\mathsf{Z}^p|$. We mention here the ZDD size. In general, the ZDD size, and the complexity of its construction, are exponential in n, but its empirical size is often small enough for practical use. Moreover, if $\mathcal{S}$ is defined on a graph with a constant path-width, the resulting ZDD size and construction complexity can be poly(m) [Inoue and Minato, 2016; Kawahara *et al.*, 2017b].

3.3.2 Full Correction

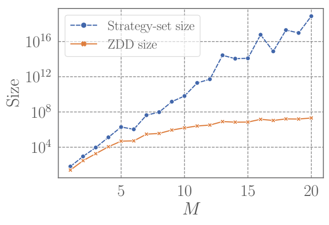
We here detail Step 12, which computes $\boldsymbol{x}_{k+1}$ via full correction (Algorithm 3.2). For each $p \in [r]$, $\mathcal{A}_k^p \subseteq \mathcal{S}^p$ represents a subset of strategies such that $\boldsymbol{x}_k^p = \sum_{S \in \mathcal{A}_k^p} z_S^p \mathbf{1}_S$ with some $z_S^p \in (0, 1]$; we call $\mathcal{A}_k^p$ the *active set.* Note that, if $\boldsymbol{x}_k$ is the output solution, the strategy profile of each $p \in [r]$ (or non-zero entries of $\boldsymbol{z}^p \in [0, 1]^{|\mathcal{S}^p|}$) is obtained as $\{z_S^p\}_{S \in \mathcal{A}_k^p}$.

Roughly speaking, the full-correction step minimizes $F(\cdot)$ on $\mathrm{conv}(\{\mathcal{A}_k^1 \cup \{\boldsymbol{s}_k^1\}\} \times \cdots \times \{\mathcal{A}_k^r \cup \{\boldsymbol{s}_k^r\}\})$, which can be formulated as convex minimization on a probabilistic simplex. An implementation of a similar full-correction step is presented in [Krishnan *et al.*, 2015], which uses a Frank–Wolfe-style algorithm called the away-steps Frank–Wolfe, and our full-correction step can be implemented by modifying it as detailed in Appendix 3.6. With the implementation of the full-correction step, we have $|\mathcal{A}_k^p| \leq k$. Therefore, while the size of a strategy profile is generally exponential in m, the strategy profile obtained by Algorithm 3.1 for each $p \in [r]$ is at most as large as the number of iterations, which can be bounded as in Theorem 3.2.

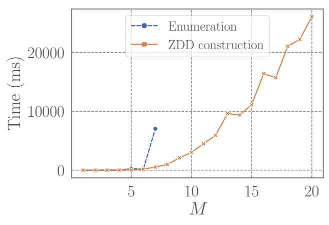
The fully corrective update of the current solution is more expensive than that of the standard Frank–Wolfe algorithm (Step 2 in Section 3.2.1). However, thanks to this full-correction step, FCFW achieves a better convergence result for strongly convex objective functions, which we will use in the proof of Theorem 3.2. Empirically, although the full-correction step increases the per-iteration computation time, it decreases the number of iterations by a great margin, which often reduces the total computation time. In our case, FCFW is a suitable choice because the full-correction step is typically far cheaper than the linear optimization with ZDDs, which means it is effective to reduce the number of iterations via full correction. Furthermore, thanks to the third condition of Algorithm 3.2, we can prove that the output solution attains an ϵ-approximate Wardrop equilibrium as shown below.

3.3.3 Approximate Wardrop Equilibrium

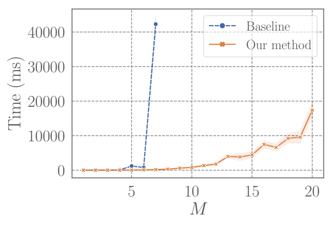
Let $\boldsymbol{x} = (\boldsymbol{x}^1, \dots, \boldsymbol{x}^p)$ be the obtained solution. For each $p \in [r]$, we let $\mathcal{A}^p$ be the active set and $\{z_S^p\}_{S \in \mathcal{A}^p}$ be the strategy profile; we have $\boldsymbol{x}^p = \sum_{S \in \mathcal{A}^p} z_S^p \mathbf{1}_S$. If our algorithm is performed with infinite precision ($\epsilon = 0$), the output attains the following Wardrop equilibrium for every $p \in [r]$: $C_S(\boldsymbol{x}) \leq C_{S'}(\boldsymbol{x})$ for any $S \in \mathcal{A}^p$ and $S' \in \mathcal{S}^p$. Namely, no player has an incentive to change his/her strategy. Unfortunately, since $\epsilon = 0$ is impossible in practice, the obtained solution does not always satisfy the above condition. Fortunately, however, we can show that the output of Algorithm 3.1 achieves an ϵ-approximate Wardrop equilibrium as follows:



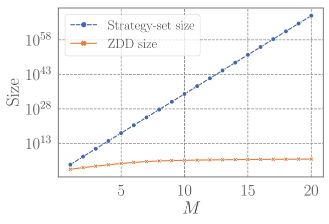
(a) BSR, Sizes (semi-log)



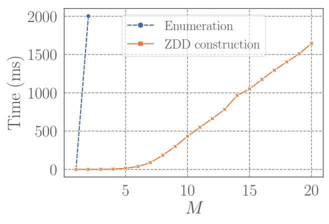
(b) BSR, Construction times



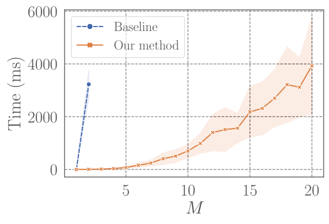
(c) BSR, FCFW times



(d) MC, Sizes (semi-log)



(e) MC, Construction times



(f) MC, FCFW times

Figure 3.1: Figures 3.1a–3.1c show the results of BSR instances, and Figures 3.1d–3.1f show those of MC instances. Figures 3.1a and 3.1d indicate the sizes of strategy sets and ZDDs; the strategy-set sizes are computed by DP on ZDDs. Figures 3.1b and 3.1e show the times required for constructing $\mathsf{Z}_{\mathcal{S}}$ and enumerating all $S \in \mathcal{S}$, where the enumerated strategy sets are used by the baseline method. Figures 3.1c and 3.1f present the running times of FCFW (Algorithm 3.1) performed with the baseline and our methods, where each curve and error band indicate the mean and standard deviation, respectively, over 50 random instances.

Theorem 3.1. *Fix any $p \in [r]$. Algorithm 3.1 outputs $\boldsymbol{x}$ satisfying $C_S(\boldsymbol{x}) - \epsilon \le C_{S'}(\boldsymbol{x}) + \epsilon$ for any $S \in \mathcal{A}^p$ and $S' \in \mathcal{S}^p$.*

Proof. Let $\boldsymbol{y} = \sum_{p \in [r]} w^p \boldsymbol{x}^p$. Thanks to Step 10, we have

$$\langle \nabla \Phi(\boldsymbol{y}), \boldsymbol{x}^p \rangle \le \min_{\boldsymbol{u} \in \mathcal{S}^p} \langle \nabla \Phi(\boldsymbol{y}), \boldsymbol{u} \rangle + \epsilon = \min_{S' \in \mathcal{S}^p} C_{S'}(\boldsymbol{x}) + \epsilon.$$

From the third condition in Algorithm 3.2, we obtain

$$\langle \nabla \Phi(\boldsymbol{y}), \boldsymbol{x}^p \rangle \ge \max_{\boldsymbol{u} \in \mathcal{A}^p} \langle \nabla \Phi(\boldsymbol{y}), \boldsymbol{u} \rangle - \epsilon = \max_{S \in \mathcal{A}^p} C_S(\boldsymbol{x}) - \epsilon.$$

Combining these inequalities, we obtain the claim. □

The number of iterations for computing the ϵ-approximate Wardrop equilibrium can be bounded as follows:

Theorem 3.2. *Algorithm 3.1 stops after $O(\mathrm{poly}(m)/\epsilon)$ iterations. If $F(\cdot)$ is strongly convex and the reciprocal of the* pyramidal width [Lacoste-Julien and Jaggi, 2015] *of* $\mathrm{conv}(\mathcal{V})$ *is $O(\mathrm{poly}(m))$, it stops after $O(\mathrm{poly}(m)\log\epsilon^{-1})$ iterations.*

Proof. If $F(\cdot)$ has differentiability, convexity, and Lipschitz-continuous gradient, the $\max_{\boldsymbol{v}\in\mathrm{conv}(\mathcal{V})}\langle\nabla F(\boldsymbol{x}_k),(\boldsymbol{x}_k-\boldsymbol{v})\rangle$ value, so-called the duality gap, is bounded by $O(\mathrm{poly}(m)/k)$ as in [Jaggi, 2013]. This fact implies that $\sum_{p\in[r]} w^p g_k^p \le \epsilon$ holds after $k = O(\mathrm{poly}(m)/\epsilon)$ iterations. Since each g_k^p is non-negative, we see that the algorithm stops after at most $\lceil k/\min_{p\in[r]} w^p\rceil = O(\mathrm{poly}(m)/\epsilon)$ iterations. We then obtain the improved result under the assumptions. Note that since $\sum_{p\in[r]} w^p = 1$, the third condition in Algorithm 3.2 implies $\max_{\boldsymbol{v}\in\mathrm{conv}(\mathcal{A}^1_{k+1}\times\cdots\times\mathcal{A}^r_{k+1})}\langle-\nabla F(\boldsymbol{x}_{k+1}),\boldsymbol{x}_{k+1}-\boldsymbol{v}\rangle\le\epsilon$, which means that the so-called away gap is less than or equal to ϵ (see, [Lacoste-Julien and Jaggi, 2015]). Consequently, the procedures of Algorithms 3.1 and 3.2 completely recover those of FCFW, and so, if the reciprocal of the pyramidal width is $O(\mathrm{poly}(m))$, the duality gap can be bounded by $O(\exp(-\Theta(k/\mathrm{poly}(m))))$ as in [Lacoste-Julien and Jaggi, 2015]. Therefore, the algorithm stops after $O(\mathrm{poly}(m)\log\epsilon^{-1})$ iterations. □

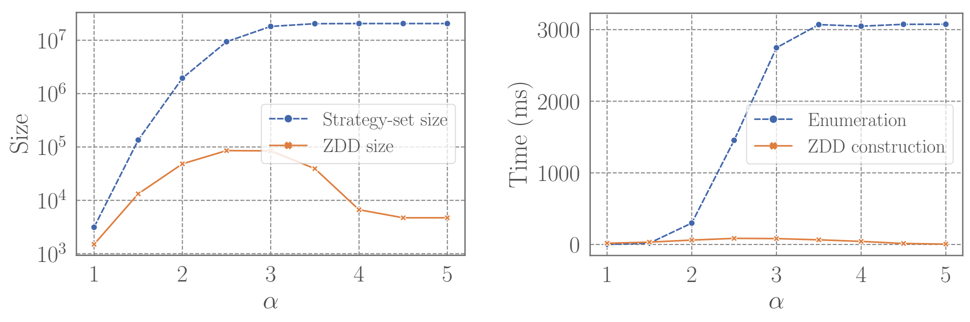
Lacoste-Julien and Jaggi [2015] conjectured that the pyramidal width is lower bounded by $\Omega(1/\sqrt{m})$ in general. Under the conjecture, its reciprocal is $O(\sqrt{m})$, and so the corresponding assumption in Theorem 3.2 is satisfied.

3.4 Experiments

We evaluate the proposed method via experiments. In Section 3.4.1, we show the empirical efficiency of our method with synthetic instances. In Section 3.4.2, we demonstrate that our method can work with real-world instances. For constructing ZDDs, we use Graphillion [Inoue *et al.*, 2016]. All the algorithms are implemented in C++, and the codes are complied with clang++ (Apple LLVM v10.0.0). We set the ϵ value of our algorithm to 10^{-10}. All the experiments are conducted on a 64-bit macOS (High Sierra) machine with 2.5 GHz Intel Core i7 CPU (single thread) and 16 GB RAM.

3.4.1 Synthetic Instances

We consider budgeted selfish routing (BSR) and multi-location communication (MC) instances on undirected grid graphs with $6\times h$ edges ($h = 1,2,\dots,20$); I is the edge set with $m = 13h+6$ edges. For simplicity, we consider the symmetric case ($r = 1$), and we omit index $p\in[r]$; e.g., $\mathcal{S}^1$

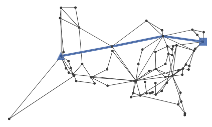


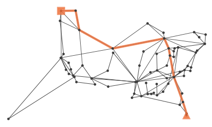
(a) Sizes (b) Construction times

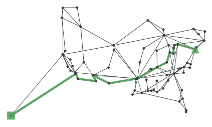
Figure 3.2: Figure 3.2a shows the strategy-set and ZDD sizes for the BSR instances with $h = 5$ and $\alpha = 1.0, 1.5, \ldots, 5.0$. Figure 3.2b indicates the times required for constructing $\mathsf{Z}_{\mathcal{S}}$ and enumerating all $S \in \mathcal{S}$.

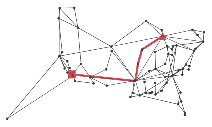
is denoted by $\mathcal{S}$. In BSR instances, each $S \in \mathcal{S}$ is an s–t path satisfying a budget constraint, $\sum_{i:a_i \in S} T_i \le W$, where s and t are placed on the diagonal corners of the grid. Edge toll T_i $(i \in I)$ is chosen uniformly at random from $\{0, 1, \ldots, 10\}$, and we let $W = 5\alpha \times (6 + h)$; note that, if $\alpha = 1$, W corresponds to the total weight of the shortest s–t path on average. The choice of α affects the sizes of $\mathcal{S}$ and $\mathsf{Z}_{\mathcal{S}}$. Figure 3.2a presents their sizes for the case with $h = 5$ and $\alpha = 1.0, 1.5, \ldots, 5.0$, where $\mathcal{S}$ includes all s–t paths when $\alpha \ge 4.5$. In Figure 3.2b, we also present the times required for constructing $\mathsf{Z}_{\mathcal{S}}$ and enumerating all $S \in \mathcal{S}$. Note that, while $|\mathcal{S}|$ increases with α, this is not always the case with $|\mathsf{Z}_{\mathcal{S}}|$, implying that our ZDD-based method can be applied to BSR instances with large W values. In the following experiments, we let $\alpha = 2$. In MC instances, $\mathcal{S}$ is the collection of all Steiner trees connecting four terminals placed on the corners of the grid. With both BSR and MC instances, we use cost functions defined as $c_i(\theta) = A_i\theta + 1$ $(i \in I)$, where A_i is drawn uniformly at random from $[0, 10]$. We make 50 random copies of cost functions, and all results that can vary with the objective function (i.e., those related to FCFW) are shown by the mean and standard deviation calculated over the 50 random instances. As a baseline method, we employ the following enumeration-based algorithm: We first enumerate all $S \in \mathcal{S}$ and then execute FCFW (Algorithm 3.1) whose linear optimization step (Step 8) is performed by choosing the best one from the enumerated $\mathcal{S}$. By comparing our ZDD-based method with the baseline, we examine how much the use of ZDD makes the algorithm efficient.

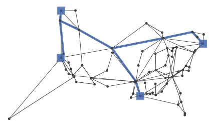
Figure 3.1 presents the results. The baseline method does not work with BSR and MC instances with $h \ge 8$ and $h \ge 3$, respectively, due to memory

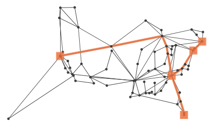

(a) BSR, $p = 1$

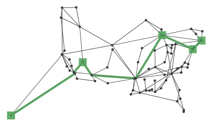

(b) BSR, $p = 2$


(c) BSR, $p = 3$


(d) BSR, $p = 4$


(e) MC, $p = 1$


(f) MC, $p = 2$


(g) MC, $p = 3$

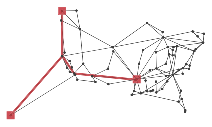

(h) MC, $p = 4$

Figure 3.3: Each figure illustrates the topology of TW Telecom network. Figures 3.3a–3.3d: For each $p = 1, \ldots, 4$, the square and triangular vertices indicate s_p and t_p, respectively, and the colored (bold) s_p–t_p path is the most-used strategy. Figures 3.3e–3.3h: For each $p = 1, \ldots, 4$, the square vertices indicate the terminals, and the colored (bold) Steiner tree is the most-used strategy.

shortage; note that the strategy-set sizes of those instances are calculated by using ZDDs. We see that ZDDs are far smaller than strategy sets, and ZDD construction can be performed far more efficiently than the enumeration of strategies. Thanks to the compactness of ZDDs, our method runs far faster than the baseline method. Our method is about 236 and 984 times faster than the baseline in the BSR instance with $h = 7$ and MC instance with $h = 2$, respectively.

3.4.2 Real-world Instances

We consider BSR and MC instances on a real-world network. We use TW Telecom dataset of Internet Topology Zoo [Knight *et al.*, 2011], which is a U.S. communication network. The original network has some isolated vertices and multiple edges; we remove them and obtain a graph with 71 vertices and 115 edges. Figure 3.3 presents the topology of the graph. In both BSR and MC instances, we let $r = 4$ and $(w^1, w^2, w^3, w^4) = (0.4, 0.3, 0.2, 0.1)$, and we use cost function $c_i(\theta) = A_i\theta^2 + B_i$; i.e., the cost increases quadratically in the mass of players. We let B_i be the Euclid distance of edge i and $A_i = B_iU_i$, where U_i is drawn uniformly at random from $[0, 100]$. In the BSR instance, $(s_1, t_1), \ldots, (s_4, t_4)$ are placed as in Figures 3.3a–3.3d. We set edge weight w_i at $\lfloor B_i(100 - U_i)\rceil$. For each $p = 1, \ldots, 4$, budget value W^p is set at $100L^p$, where L^p is the length of the shortest s_p–t_p path with respect to edge length

Table 3.1: Strategy-set sizes, ZDD sizes, and ZDD construction times for real-world instances.

	p	Strategy-set size	ZDD size	Time (ms)
BSR	1	9.099×10^5	3.925×10^4	2340
	2	2.725×10^6	8.382×10^4	4474
	3	6.219×10^5	3.504×10^4	2565
	4	9.426×10^4	9.941×10^3	485.9
MC	1	9.796×10^{28}	9.315×10^4	121.5
	2	4.286×10^{28}	7.930×10^4	81.04
	3	8.462×10^{28}	8.315×10^4	102.5
	4	9.281×10^{28}	8.798×10^4	152.0

B_i. In the MC instances, the terminals are placed as in Figures 3.3e–3.3h.

We apply our method to the instances and study the obtained results. Table 3.1 presents the strategy-set size, ZDD size, and ZDD construction time for each $p = 1, \ldots, 4$; notably, with the MC instance, ZDDs are about 10^{24} times smaller than the strategy sets. Additionally, FCFW takes 2391 ms and 3316 ms for BSR and MC instances, respectively. In total, our method computes approximate equilibria in 1.226×10^4 ms and 3773 ms for BSR and MC instances, respectively. Since our method can output a strategy profile, we can obtain the most-used strategy for each $p = 1, \ldots, 4$ as in Figure 3.3. We see that the strategy of each p tends to avoid using the same edges (resource) to each other. For example, in Figures 3.3g and 3.3h, the leftmost vertex is chosen as a terminal, which has two edges, and each strategy in $p = 3$ and 4 use one of the two edges that is different from each other. This result implies that the players successfully avoid congestion at the equilibria, and so the price-of-anarchy (PoA) values are expected to be close to 1; i.e., the equilibria are almost as efficient as the social optima. In fact, the PoA values of the BSR and MC instances are both about 1.01, which are obtained by computing the social optima with our method. Figure 3.4 presents the decrease in $\max_{S \in \mathcal{A}_k^p} C_S(\boldsymbol{x}_k) - \min_{S' \in \mathcal{S}^p} C_{S'}(\boldsymbol{x}_k)$ values over the iterations for $p = 1, \ldots, 4$, which converge to 0 at a rate of $O(\mathrm{poly}(m)/k)$ (or $O(\exp(-\Theta(k/\mathrm{poly}(m)))))$ as in Theorems 3.1 and 3.2. Consistent with the theoretical results, the values converge to 0 very quickly as k increases.

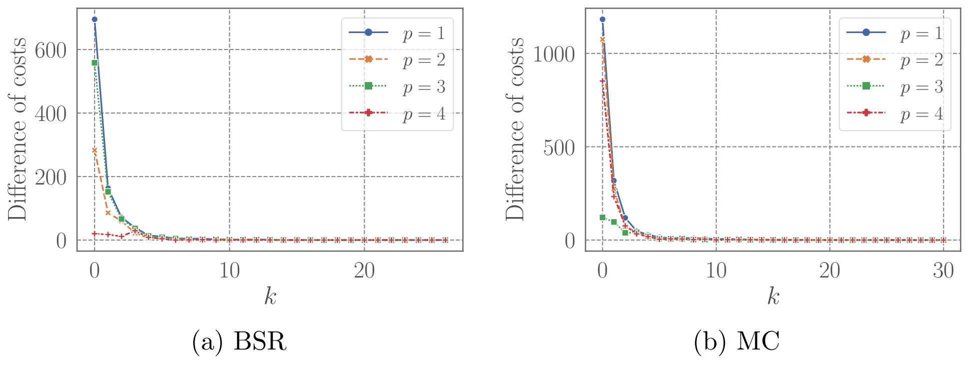


Figure 3.4: $\max_{S\in\mathcal{A}_k^p} C_S(\boldsymbol{x}_k) - \min_{S'\in\mathcal{S}^p} C_{S'}(\boldsymbol{x}_k)$ values ($p = 1,\dots,4$) for real-world instances.

3.5 Conclusion

We developed a practical Frank–Wolfe-style equilibrium computation method for continuous-player combinatorial congestion games, which utilizes the empirical compactness of ZDDs. We proved that the algorithm computes an ϵ-approximate Wardrop equilibrium in $O(\mathrm{poly}(m)/\epsilon)$ (or $O(\mathrm{poly}(m)\log\epsilon^{-1})$) iterations. Experiments demonstrated that our algorithm is useful for computing and studying equilibria of realistic continuous-player combinatorial congestion games, for which alternative methods are prohibitively costly.

3.6 Appendix: Detailed Implementation of Full-correction

We here detail the implementation of the full-correction step (Algorithm 3.2). As mentioned before, full correction $\textsc{Correction}(\boldsymbol{x}, \{\mathcal{A}^p\}_{p\in[r]}, \{\boldsymbol{s}^p\}_{p\in[r]}, \epsilon)$ is, roughly speaking, performed by minimizing $F(\cdot)$ on $\mathrm{conv}((\mathcal{A}^1 \cup \{\boldsymbol{s}^1\}) \times \cdots \times (\mathcal{A}^r \cup \{\boldsymbol{s}^r\}))$. Since this feasible region is again a convex hull of some points in $\mathbb{R}^{rm}$, the full correction can also be performed with Frank–Wolfe-style algorithms. For example, Krishnan *et al.* [2015] implemented the full-correction step with the away-steps Frank–Wolfe (AFW) [Lacoste-Julien and Jaggi, 2015], a variant of the Frank–Wolfe algorithm that achieves linear convergence for strongly convex functions. More precisely, they performed the full correction with AFW whose stopping criterion requires the away gap, as well as the Frank–Wolfe gap (or the duality gap), to be small enough; thus they guaranteed that the conditions required in the full-correction step of [Lacoste-Julien and Jaggi, 2015, Algorithm 4] is satisfied. The conditions

Algorithm 3.3: AFWCorrection($\boldsymbol{x}, \{\mathcal{A}^p\}_{p\in[r]}, \{\boldsymbol{s}^p\}_{p\in[r]}, \epsilon$): Modified AFW-based full correction.

```
1  $\boldsymbol{x}_0 = \boldsymbol{x} = (\boldsymbol{x}^1, \dots, \boldsymbol{x}^r)$
2  foreach $p \in [r]$ do
3      $\mathcal{B}^p = \mathcal{A}^p \cup \{\boldsymbol{s}^p\}$ and $\mathcal{A}^p_0 = \mathcal{A}^p$
4  for $l = 0$ to $L$ do
5      $\boldsymbol{y}_l = \sum_{p\in[r]} w^p \boldsymbol{x}^p_l$
6      foreach $p \in [r]$ do
7          Let $\boldsymbol{s}^p_l \in \mathrm{argmin}_{\boldsymbol{u}\in\mathcal{B}^p} \langle \nabla\Phi(\boldsymbol{y}_l), \boldsymbol{u}\rangle$ and $\boldsymbol{d}^{p,\mathrm{FW}}_l = \boldsymbol{s}^p_l - \boldsymbol{x}^p_l$
8          Let $\boldsymbol{v}^p_l \in \mathrm{argmax}_{\boldsymbol{u}\in\mathcal{A}^p_l} \langle \nabla\Phi(\boldsymbol{y}_l), \boldsymbol{u}\rangle$ and $\boldsymbol{d}^{p,\mathrm{A}}_l = \boldsymbol{x}^p_l - \boldsymbol{v}^p_l$
9      if $\max_{p\in[r]} \left\langle -\nabla\Phi(\boldsymbol{y}_l), \boldsymbol{d}^{p,\mathrm{FW}}_l + \boldsymbol{d}^{p,\mathrm{A}}_l \right\rangle \le \epsilon$ then
10         return $\boldsymbol{x}_l$ as $\boldsymbol{x}_{k+1}$, $\{\mathcal{A}^p_l\}_{p\in[r]}$ as $\{\mathcal{A}^p_{k+1}\}_{p\in[r]}$, and $\{\{z^p_S\}_{S\in\mathcal{B}^p}\}_{p\in[r]}$
11     $g^{\mathrm{FW}}_l = \sum_{p\in[r]} w^p \left\langle -\nabla\Phi(\boldsymbol{y}_l), \boldsymbol{d}^{p,\mathrm{FW}}_l \right\rangle$
12     $g^{\mathrm{A}}_l = \sum_{p\in[r]} w^p \left\langle -\nabla\Phi(\boldsymbol{y}_l), \boldsymbol{d}^{p,\mathrm{A}}_l \right\rangle$
13     if $g^{\mathrm{FW}}_l \ge g^{\mathrm{A}}_l$ then
14         $\boldsymbol{d}_l = (\boldsymbol{d}^{1,\mathrm{FW}}_l, \dots, \boldsymbol{d}^{r,\mathrm{FW}}_l)$ and $\gamma_{\max} = 1$
15     else
16         $\boldsymbol{d}_l = (\boldsymbol{d}^{1,\mathrm{A}}_l, \dots, \boldsymbol{d}^{r,\mathrm{A}}_l)$ and $\gamma_{\max} = \min_{p\in[r]} z^p_{\boldsymbol{v}^p_l}/(1 - z^p_{\boldsymbol{v}^p_l})$
17     $\gamma_l \in \mathrm{argmin}_{\gamma\in[0,\gamma_{\max}]} F(\boldsymbol{x}_l + \gamma \boldsymbol{d}_l)$
18     Update $\boldsymbol{x}_{l+1} = \boldsymbol{x}_l + \gamma_l \boldsymbol{d}_l$
19     Update $\{z^p_S\}_{S\in\mathcal{B}^p}$ accordingly for $p \in [r]$ (see [Lacoste-Julien and Jaggi, 2015])
20     Update $\mathcal{A}^p_{l+1} = \{S \in \mathcal{B}^p \mid z^p_S > 0\}$ for $p \in [r]$
```

required by our full correction (Algorithm 3.2) are analogous to those of [Lacoste-Julien and Jaggi, 2015, Algorithm 4], but they have a slight difference. Below we detail the difference and explain how to implement the full-correction step that works for our case.

In the original full-correction step of [Lacoste-Julien and Jaggi, 2015, Algorithm 4], the away gap is defined on the full dimension, which corresponds to $\mathbb{R}^{rm}$ in our case, and thus the aforementioned AFW-based full correction performed on $\mathbb{R}^{rm}$ works. In our case, however, the third condition of Algorithm 3.2 requires the away gap $\max_{\boldsymbol{u}\in\mathcal{A}^p_{k+1}} \left\langle -\nabla\Phi(\boldsymbol{y}_{k+1}), \boldsymbol{x}^p_{k+1} - \boldsymbol{u} \right\rangle$ to be small enough for every $p \in [r]$; this is the difference to be considered. Note that, since the "max" is taken for each $p \in [r]$, the away gap of each $p \in [r]$ is not guaranteed to be small no matter how small the away gap on $\mathbb{R}^{rm}$ is; this is the reason why the original full correction does not work. Thus, we consider modifying the AFW algorithm to make it work for our use. The pseudocode of our AFW-based full correction is presented in Algorithm 3.3. The differences from the original AFW-based full correction [Krishnan *et al.*, 2015] is as follows: We maintain the active set $\mathcal{A}^p_l$ for each $p \in [r]$, instead of the one defined on $\mathbb{R}^{rm}$, and the away direction, $\boldsymbol{d}_l^{p,\mathrm{A}}$, is computed for each $p \in [r]$ in Step 8. Note that, as with the original full correction of [Krishnan *et al.*, 2015], we employ the stopping criterion that requires the away and Frank–Wolfe gaps to be small (Step 9). The other parts are almost the same as those of AFW [Lacoste-Julien and Jaggi, 2015]. In Steps 15 and 16, we use $\nabla_p F(\boldsymbol{x}) = w^p \nabla\Phi(\boldsymbol{y})$ $(\forall p \in [r])$ to compute the Frank–Wolfe gap, g_l^{FW}, and away gap, g_l^{A}, defined on $\mathbb{R}^{rn}$. The resulting modified AFW exhibits the same convergence behavior as the standard AFW as follows:

Theorem 3.3. *After l iterations of Algorithm 3.3, the Frank–Wolfe gap (FW gap) satisfies $g_l^{\mathrm{FW}} \le O(\mathrm{poly}(m)/l)$. Let $\mathcal{B} := \mathcal{B}^1 \times \cdots \times \mathcal{B}^r$. If $F(\cdot)$ is strongly convex and the reciprocal of the pyramidal width of $\mathrm{conv}(\mathcal{B})$ is $O(\mathrm{poly}(m))$, then we have $g_l^{\mathrm{FW}} \le O(\exp(-\Theta(l/\mathrm{poly}(m))))$ after l iterations. Moreover, the FW gap of each $p \in [r]$, defined by $g_l^{p,\mathrm{FW}} := \left\langle -\nabla\Phi(\boldsymbol{y}_l), \boldsymbol{d}_l^{p,\mathrm{FW}} \right\rangle$, also converges at the same rates.*

Note that, once the convergence of the FW gap of each $p \in [r]$ is obtained, the away gap of each $p \in [r]$, defined by $g_l^{p,\mathrm{A}} := \left\langle \nabla\Phi(\boldsymbol{y}_l), \boldsymbol{d}_l^{p,\mathrm{A}} \right\rangle$, also asymptotically converges to 0 as follows: Since $\boldsymbol{x}_l^p$ is can be written as $\sum_{S \in \mathcal{A}_l^p} z_S^p \mathbf{1}_S$ in each iteration, $\boldsymbol{x}_l^p + \frac{z_S^p}{1-z_S^p}(\boldsymbol{x}_l^p - \mathbf{1}_S) = \frac{1}{1-z_S^p}\boldsymbol{x}_l^p - \frac{z_S^p}{1-z_S^p}\mathbf{1}_S \in \mathrm{conv}(\mathcal{B}^p)$ holds for any $S \in \mathcal{B}^p$. Therefore, the definition of the FW gap implies $\left\langle -\nabla\Phi(\boldsymbol{y}_l), \frac{z_S^p}{1-z_S^p}(\boldsymbol{x}_l^p - \mathbf{1}_S) \right\rangle = \frac{z_S^p}{1-z_S^p} \left\langle -\nabla\Phi(\boldsymbol{y}_l), (\boldsymbol{x}_l^p - \mathbf{1}_S) \right\rangle \le g_l^{p,\mathrm{FW}}$ for any $S \in \mathcal{B}^p$. From the definition of the away gap and $\mathcal{A}_l^p \subseteq \mathcal{B}^p$, if we let $\zeta_l^p := \min_{S \in \mathcal{A}_l^p} z_S^p > 0$, we have $\zeta_l^p g_l^{p,\mathrm{A}} \le g_l^{p,\mathrm{FW}}$. Hence the away gap converges to 0 if the FW gap does. To conclude, Algorithm 3.3 can update the input so as to satisfy the third condition in Algorithm 3.2. We can easily confirm that the first and second conditions are also satisfied due to the procedure of AFW (Algorithm 3.3).

Proof of Theorem 3.3. We first see that the convergence of g_l^{FW} implies that of the individual FW gaps ($g_l^{p,\mathrm{FW}}$ $(p \in [r])$ in the statement of Theorem 3.3). Since $\boldsymbol{s}_l^p \in \mathrm{argmin}_{\boldsymbol{u} \in \mathrm{conv}(\mathcal{B}^p)} \langle \nabla\Phi(\boldsymbol{y}_l), \boldsymbol{u} \rangle$ and $\boldsymbol{x}_l^p \in \mathrm{conv}(\mathcal{B}^p)$, we have $g_l^{p,\mathrm{FW}} = \langle -\nabla\Phi(\boldsymbol{y}_l), \boldsymbol{s}_l^p - \boldsymbol{x}_l^p \rangle \ge 0$ for all $p \in [r]$. Since $g_l^{\mathrm{FW}} = \sum_{p \in [r]} w^p g_l^{p,\mathrm{FW}}$, we have $g_l^{p,\mathrm{FW}} \le g_l^{\mathrm{FW}}/m^p$. Therefore, individual gaps $g_l^{p,\mathrm{FW}}$ $(p \in [r])$ converge to 0 if g_l^{FW} does.

Next, we show the $O(\mathrm{poly}(m)/l)$ convergence by following the argument of [Lacoste-Julien and Jaggi, 2015]. Let $\boldsymbol{x}^*$ be the minimizer of $F(\cdot)$ on $\mathrm{conv}(\mathcal{B})$ and $h_l = F(\boldsymbol{x}_l) - F(\boldsymbol{x}^*)$ be the suboptimality gap. Since $F(\cdot)$ has

L-Lipschitz-continuous gradient and convexity, for any $\gamma \in [0, \gamma_{\max}]$,

$$\begin{aligned} F(\boldsymbol{x}_{l+1}) &\le F(\boldsymbol{x}_l + \gamma \boldsymbol{d}_l) \\ &\le F(\boldsymbol{x}_l) + \gamma \langle \nabla F(\boldsymbol{x}_l), \boldsymbol{d}_l \rangle + \gamma^2 L \|\boldsymbol{d}_l\|^2/2 \\ &\le F(\boldsymbol{x}_l) - \gamma g_l^{\mathrm{FW}} + \gamma^2 L D^2/2 \end{aligned} \tag{3.4}$$

holds, where D is the diameter of $\mathrm{conv}(\mathcal{B})$; note that $D \le O(\mathrm{poly}(m))$ holds. The third inequality is due to $\langle \nabla F(\boldsymbol{x}_l), \boldsymbol{d}_l \rangle = \min\{-g_l^{\mathrm{FW}}, -g_l^{\mathrm{A}}\} \le -g_l^{\mathrm{FW}}$ (see, Step 13) and $\|\boldsymbol{d}_l\| \le D$. By subtracting $F(\boldsymbol{x}^*)$ from both sides, we obtain $h_{l+1} \le h_l - \gamma g_l^{\mathrm{FW}} + \gamma^2 L D^2/2$. Below we discuss the convergence by using the number of steps where the line search (Step 17) does not result in $\gamma = \gamma_{\max}$; such steps are called good steps, and the number of good steps among the first l steps, denoted by $G(l)$, can be lower bounded as follows: In a step with $\gamma = \gamma_{\max}$ (called a drop step), we set at least one of $z_S^p > 0$ to zero, which means the sum of the sizes of active sets decreases by at least one. Thus, the number of drop steps among the first l steps is upper bounded by $\sum_{p \in [r]} |\mathcal{A}^p| + (l - \sum_{p \in [r]} |\mathcal{A}^p|)/2$, which implies $G(l) \ge (l - \sum_{p \in [r]} |\mathcal{A}^p|)/2 = \Theta(l)$. Following the proof of [Jaggi, 2013, Theorems 1 and 2], we can show that $g_l^{\mathrm{FW}} = O(\mathrm{poly}(m)/G(l)) = O(\mathrm{poly}(m)/l)$ holds; hence the first clam is obtained.

Finally, we prove the linear convergence under the assumptions of the strong convexity and bounded pyramidal width. We define error vector $\boldsymbol{e}_l \coloneqq \boldsymbol{x}^* - \boldsymbol{x}_l$ and its normalized version $\hat{\boldsymbol{e}}_l \coloneqq \boldsymbol{e}_l / \|\boldsymbol{e}_l\|$. Thanks to the argument of [Lacoste-Julien and Jaggi, 2015, Section 2.1], we have

$$h_l - h_{l+1} \ge \frac{\mu}{L\|\boldsymbol{d}_l\|^2} \frac{\langle -\nabla F(\boldsymbol{x}_l), \boldsymbol{d}_l \rangle^2}{\langle -\nabla F(\boldsymbol{x}_l), \hat{\boldsymbol{e}}_l \rangle^2} h_l \tag{3.5}$$

if the l-th iteration is a good step. Since $\langle -\nabla F(\boldsymbol{x}_l), \boldsymbol{d}_l \rangle \ge \max\{g_l^{\mathrm{FW}}, g_l^{\mathrm{A}}\} \ge (g_l^{\mathrm{FW}} + g_l^{\mathrm{A}})/2 = \langle -\nabla F(\boldsymbol{x}_l), \boldsymbol{s}_l - \boldsymbol{v}_l \rangle /2$, where $\boldsymbol{s}_l \coloneqq (\boldsymbol{s}_l^1, \dots, \boldsymbol{s}_l^r)$ and $\boldsymbol{v}_l \coloneqq (\boldsymbol{v}_l^1, \dots, \boldsymbol{v}_l^r)$, we have

$$h_l - h_{l+1} \ge \frac{\mu}{4LD^2} \frac{\langle -\nabla F(\boldsymbol{x}_l), \boldsymbol{s}_l - \boldsymbol{v}_l \rangle^2}{\langle -\nabla F(\boldsymbol{x}_l), \hat{\boldsymbol{e}}_l \rangle^2} h_l. \tag{3.6}$$

Here, $\boldsymbol{v}_l \in \mathbb{R}^{rn}$ is selected from $\mathcal{A}_l^1 \times \cdots \times \mathcal{A}_l^r$ to maximize $\langle \nabla F(\boldsymbol{x}_l), \boldsymbol{v}_l \rangle$. Moreover, we can regard $\mathcal{A}_l^1 \times \cdots \times \mathcal{A}_l^r$ as the active set of $\boldsymbol{x}_l \in \mathbb{R}^{rn}$ since

$$\sum_{\boldsymbol{u}^1 \in \mathcal{A}_l^1} \cdots \sum_{\boldsymbol{u}^r \in \mathcal{A}_l^r} z_{\boldsymbol{u}^1}^1 \cdots z_{\boldsymbol{u}^r}^r (\boldsymbol{u}^1, \dots, \boldsymbol{u}^r) = \boldsymbol{x}_l, \tag{3.7}$$

$$\sum_{\boldsymbol{u}^1 \in \mathcal{A}_l^1} \cdots \sum_{\boldsymbol{u}^r \in \mathcal{A}_l^r} z_{\boldsymbol{u}^1}^1 \cdots z_{\boldsymbol{u}^r}^r = 1. \tag{3.8}$$

Therefore, by using [Lacoste-Julien and Jaggi, 2015, Theorem 3], we can show that $\langle -\nabla F(\boldsymbol{x}_l), \boldsymbol{s}_l - \boldsymbol{v}_l \rangle \,/\, \langle -\nabla F(\boldsymbol{x}_l), \hat{\boldsymbol{e}}_l \rangle$ is lower bounded by pyramidal width $\delta > 0$ of $\mathcal{B}$. Consequently, we obtain the geometric decrease of the suboptimality gap: $h_{l+1} \leq (1 - \frac{\mu}{4L}(\frac{\delta}{D})^2)h_l$. Therefore, if $1/\delta \leq O(\mathrm{poly}(m))$, we have $h_l = O(\exp(-\Theta(G(l)/\mathrm{poly}(m)))) = O(\exp(-\Theta(l/\mathrm{poly}(m))))$. The linear convergence of the FW gap g_l^{FW} can be obtained from [Lacoste-Julien and Jaggi, 2015, Theorem 2], which shows $g_l^{\mathrm{FW}} \leq O(D\sqrt{h_l})$. $\square$

As mentioned in Section 3.3.3, δ is conjectured to be lower bounded by $\Omega(1/\sqrt{m})$ [Lacoste-Julien and Jaggi, 2015]. More precisely, the pyramidal width of the rm-dimensional unit cube $\{0,1\}^{rm}$ is $1/\sqrt{rm}$, and it is conjectured in [Lacoste-Julien and Jaggi, 2015] that the pyramidal width does not increase when an another vertex is added. Following the conjecture, the pyramidal width of any $\mathcal{B}$ is at least $1/\sqrt{rm}$.

Chapter 4

Background of Network Reliability Analysis

As described in Chapter 1, network reliability analysis reveals the robustness of network service against the failure of network components. Although the network reliability can be defined in various way on various assumptions, we here describe the basic network model where the failure is occurred on the links of the network stochastically independently. Chapters 5–8 are based on this stochastic model. We also describe a traditional network reliability measure, named *K-terminal network reliability* (given subset K of vertices), defined on the above model. Then, we overview the literature on computing K-terminal network reliability. Finally, we describe a modern method for computing K-terminal network reliability with BDDs in detail.

4.1 Network Model

First, we define the network model used in the subsequent chapters in the same way as [Moskowitz, 1958]. A network is modeled as an undirected network $G = (V, E)$ with vertex set V and edge set E. The graph G has n vertices and m edges, i.e., $n = |V|$ and $m = |E|$. For example, for telecommunication networks, vertices V correspond to network nodes including, e.g., packet gateways, servers, base stations, and switches, and edges E correspond to network links each connecting two network nodes. Additionally, we are given probability $p_e \in [0, 1]$ for each edge $e \in E$, which is called an *availability* of edge e.

We consider two states for network links: working or failing. We represent these states by the presence of edge $e \in E$: e is present if e is correctly working, and e is absent otherwise. The following assumption is the key property of the network model.

Assumption 4.1. *Each edge $e \in E$ is present with probability p_e and absent with probability $1 - p_e$, and the state of e (present or absent) is stochastically independent of the other edges' states.*

From the viewpoint of networks, this is equivalent to the situation where each network link e fails with probability $1 - p_e$ independent of the other network links. We call this model an *independent failure model*.

Hereafter we represent by $E' \subseteq E$ the state of the graph that the edges in $E' \subseteq E$ are present and the other edges $(E \setminus E')$ are absent. Under Assumption 4.1, we can express the probability that the state is $E' \subseteq E$ as follows:

$$\Pr(E') = \prod_{e \in E'} p_e \cdot \prod_{e \in E \setminus E'} (1 - p_e). \tag{4.1}$$

4.1.1 Comparison with Other Models

We here mention some other network models. Since the proposal of independent failure model in [Moskowitz, 1958], it has been widely assumed for most studies of network reliability analysis. From the viewpoint of networks, this assumption can be considered as focusing on the behavior of ordinary times. That is, in ordinary times, each network component fails accidentally regardless of the other network components, which can be modeled with the independent failure model. Also, Assumption 4.1 corresponds to observing the long-term behavior of networks. That is, in long-term view, each network component's failure can be seen as an independent stochastic process.

Conversely, when a huge disaster such as earthquake or flood occurs and the short-term behavior is concerned, we should consider *correlated failure model* where the failure of some network components are stochastically correlated. For example, when an earthquake occurs, the network components near the epicenter will be likely to fail simultaneously. Such a model has emerged around 2010, e.g., [Xiao *et al.*, 2009; Xing, 2008; Shrestha *et al.*, 2012; Neumayer and Modiano, 2016]. However, as shown later, the analysis on independent failure model itself is a computationally hard task, and the correlated failure model seems much harder. Therefore, almost all the models in these works assume some additional assumptions. One of the most prominent assumptions is that the network components in the disaster area *must* fail. Although such an assumption can remove the stochastic nature, the network model becomes fairly unrealistic.

To sum up, the network model with Assumption 4.1 focuses on the long-term behavior of ordinary times, and may not be applicable for short-term disaster scenarios. However, even for the latter scenarios, we believe that assuming independent failures with averaged availabilities p_e is a reasonable choice since even the existing correlated failure models have some unrealistic assumptions.

At the end, we mention that the network model in this dissertation assumes that the vertices, i.e., the network nodes, work perfectly. This is justified by the fact that for some network infrastructures, the node failure probability is far less than the link failure probability. For example, it is known that in optical networks, the former is less than the latter by order of magnitude [Segovia *et al.*, 2008]. However, there are also network models where the network nodes also fail independently, e.g., [Kuo *et al.*, 2007; Kawahara *et al.*, 2019]. The support for vertex failures is an important future work of this dissertation, as described in Chapter 10.

4.2 K-Terminal Network Reliability

Assuming the independent (edge) failure model in Section 4.1, we define the traditional network reliability measure. In network infrastructures, the most fundamental requirement for network users is that the network nodes related to these users are connected with only the correctly working links [Nojo and Watanabe, 1987, 1993; Tollar and Bennett, 1995]. From the viewpoint of the network model in Section 4.1, it is equivalent to that the specified vertices are connected with only the present edges. Given the set K of specified vertices called *terminals*, K-terminal network reliability is defined as follows.

Definition 4.2. A *K-terminal network reliability* $R(K)$ is the probability that terminals, i.e., the vertices in K, are interconnected with the present edges under the independent failure model. Let $\mathcal{E}_K$ be the family of subgraphs of G such that the vertices in K are interconnected. Then, the K-terminal network reliability can be represented as

$$R(K) = \sum_{E' \in \mathcal{E}_K} \Pr(E') = \sum_{E' \in \mathcal{E}_K} \left[\prod_{e \in E'} p_e \cdot \prod_{e \in E \setminus E'} (1 - p_e) \right]. \tag{4.2}$$

We abbreviate the K-terminal network reliability as K-NR. Here we observe that (4.2) is much similar to the weighted model counting value (2.2). Indeed, computing (4.2) is identical to solving the weighted model counting problem with the family $\mathcal{I} = \mathcal{E}_K$ and the weights $w_e^+ = p_e$ and $w_e^- = 1 - p_e$.

Note that some special cases of K-NR problems have particular names. First, the K-NR with $|K| = 2$ is called *two-terminal network reliability*, which we abbreviated as *2-NR*. Second, the K-NR with $K = V$ is called *all-terminal network reliability*, which we abbreviated as *All-NR*.

4.2.1 Literature Overview on Computing K-Terminal Network Reliability

We here overview the complexity results and existing methods for computing the K-NR until the method using top-down construction of BDDs (explained in Section 4.3) emerged.

First, we mention the complexity of computing K-NR. We can easily verify that the K-NR computation is in #P, the complexity class of problems that are equivalent to count the number of accepted paths of a non-deterministic polynomial-time Turing machine. In 1979, Valiant [1979] proved that the K-NR computation is #P-complete, the most difficult problem among #P problems. Since #P-complete problems are at least as hard as NP-complete problems, and thus they are not solved in polynomial time unless at least P=NP, computing K-NR is much harder problem.

Next, we review some traditional existing methods. The most naive solution is to enumerate all the subgraphs in $\mathcal{E}_K$ and compute $R(K)$ directly with (4.2) like [Wing and Demetriou, 1964]. However, since there are 2^m subgraphs for a graph with m edges, such a method becomes suddenly intractable with the increase of graph size.

Since then, there are attempts to represent (4.2) as a sum of fewer and disjoint products. This approach is called *sum-of-disjoint products.* Here we focus on 2-NR problems. As seen in Section 2.1, for the family $\mathcal{E}_K$ of subgraphs, we can consider the corresponding Boolean function f_K. Since simplification methods for Boolean functions have been extensively studied in circuit community, applying similar approaches to f_K can be considered. In [Fratta and Montanari, 1973], by enumerating all simple paths between the two terminals in a graph, f_K is decomposed into the disjoint disjunction of the (relatively) small number of conjuncts. This admits the computation of $R(K)$ by the sum of the (relatively) small number of products of p_e and $(1-p_e)$. Later, similar decomposition methods were proposed by enumerating all tie-sets [Rosenthal, 1977] or all cut-sets [Tsukiyama *et al.*, 1980]. Although originally they can compute only 2-NRs, they can also compute K-NR in combination with some logic synthesis techniques.

Another line of research [Wood, 1986] tried to decompose $\mathcal{E}_K$ (or f_K) *one by one* by the case analysis on whether each edge is present or absent. This approach is called *factoring.* Let us choose an edge $e \in E$ and we conduct a case analysis described above. Edge e is present with probability p_e, and if e is present, we can consider the reduced graph $G * e$ made by contracting edge e of G. Edge e is absent with probability $(1 - p_e)$, and if e is absent, we can consider the reduced graph $G - e$ made by removing edge e from G.

Then, the reliability R^G on graph G can be decomposed as

$$R^G = p_e \cdot R^{G*e} + (1 - p_e) \cdot R^{G-e}, \tag{4.3}$$

where R^{G*e} is the reliability on graph $G * e$ and R^{G-e} is that on graph $G-e$. After contracting or removing an edge, a number of *graph simplification* techniques have employed. By the repetitive application of decomposition (4.3) and graph simplification, we can compute $R(K)$ with a smaller number of arithmetic operations.

We here mention a representing two graph simplification techniques from [Wood, 1986]: *degree 1 vertex elimination* and *degree 2 vertex elimination.* Regarding the former one, let $v \in V$ be a vertex with degree 1 and let e_v be the only edge incident to v. Then, $R^G(K)$, the K-terminal reliability on graph G, can be represented as

$$R^G(K) = \begin{cases} R^{G-v}(K) & (v \notin K) \\ p_{e_v} \cdot R^{G-v}(K \setminus \{v\}) & (v \in K) \end{cases}, \tag{4.4}$$

where $G - v$ is the graph obtained by removing vertex v (and edge e_v) from G. Regarding the second one, let $v \in V \setminus K$ be a vertex with degree 2 and let e_v^1, e_v^2 be the two edges incident to v. Then, let us consider a graph G' made by concatenating edges e_v^1, e_v^2 into one edge e' and remove the vertex v. We set $p_{e'} = p_{e_v^1} \cdot p_{e_v^2}$. Then,

$$R^G(K) = R^{G'}(K). \tag{4.5}$$

Note that these graph simplification techniques can also be employed as a preprocessing for the original graph G.

Indeed, the method of Hardy *et al.* [2007] using top-down construction of BDDs is basically based on the factoring approach. Now it is known that this method is far faster than the sum-of-disjoint product and factoring approaches explained in this section.

4.3 Computing K-Terminal Network Reliability with BDDs

In the factoring approach, many identical graphs emerge by repetitively removing or contracting edges. If the reliability is computed for a graph and an identical graph has emerged in the subsequent computation, we can reuse the result. Hardy *et al.* [2007] tried to capture the identicalness by focusing on the connectivity among only a limited number of vertices. Later, Herrmann

[2010] improved their algorithm in terms of memory efficiency, but the core procedure is almost identical. Therefore, we hereafter call their method *HH method*. In our view, HH method can be explained as the top-down construction method of a BDD representing $\mathcal{E}_K$. Since HH method forms a basis for some algorithms in the following chapters, we explain HH method in detail following the framework in Section 2.5.

As in Section 2.5, we start with a naive method. We order the edges in graph G: $e_1, \ldots, e_m$. Then, we consider the process of determining whether e_i is present or absent one by one. This generates a binary decision tree where all the i-th subsets of $E_{<i} \coloneqq \{e_1, \ldots, e_{i-1}\}$ are enumerated at the i-th level. Here we observe that determining the state of e_i corresponds to the factoring approach where we decide whether e_i is contracted or removed.

To design configures for i-th subsets, we focus on the connectivity among vertices on the subgraph induced by the present edges. This is because in the factoring approach, the vertices connected with the present edges are merged into one vertex since the present edges are contracted. If the connectivity among vertices are identical for two i-th subsets $X, Y \in E_{<i}$, i.e., for any two vertices u, v, whether u and v are connected is identical on the subgraph induced by X and that induced by Y, they produce the same resultant graph and thus we can identify them. This means that the connectivity among vertices can be used as a configure for building the BDD representing $\mathcal{E}_K$.

Hardy *et al.* [2007] further refines this idea by focusing on the connectivity among only a limited subset of vertices called *frontiers*[1]. Regarding i-th subsets, $E_{<i}$ is the set of processed edges, i.e., the edges the states (present or absent) are already determined, and $E_{\geq i} \coloneqq E \setminus E_{<i} = \{e_i, \ldots, e_m\}$ is the set of unprocessed edges. We define the subsets of vertices regarding the i-th subsets as follows.

Definition 4.3. The *i-th frontiers* F_i is the subset of vertices appearing in both the processed edges $E_{<i}$ and the unprocessed edges $E_{\geq i}$. Similarly, the *i-th processed vertices* A_i is the subset of vertices appearing in only the processed edges $E_{<i}$, and the *i-th unprocessed vertices* B_i is the subset of vertices appearing in only the unprocessed edges $E_{\geq i}$.

Intuitively, the reason why we can focus on the connectivity among only the frontiers can be explained as follows. Let $v \in A_i$ be a processed vertex. Since v is not incident to the unprocessed edges $E_{\geq i}$, if v is not connected to another vertex u with the present edges $X \subseteq E_{\geq i}$ and eventually connected

[1]In their paper [Hardy *et al.*, 2007], it is called *boundary set*. In this dissertation, we call this frontiers according to the frontier-based search [Kawahara *et al.*, 2017b], but their definitions are identical.

to u by determining the remaining edges' states, its connection must go through a frontier vertex $w \in F_i$. This means that (I) if v is not connected to any frontiers with the present edges $X \subseteq E_{\geq i}$, v never be connected to any further vertex by adding the edges in $E_{\geq i}$, and (II) if v is connected to a frontier vertex $w \in F_i$, v can be identified with w in terms of connectivity.

We formally define some notions used for the configure as follows.

Definition 4.4. The *i-th partition* of an i-th subset $X \subseteq E_{<i}$ is defined as follows: If all the vertices in K are interconnected on the subgraph induced by X, the partition is $\top$ (called *$\top$-pruned partition*). Otherwise, if there is a vertex in $K \cap A_i$ that is not connected to any frontier in F_i on the subgraph induced by X, the partition is $\bot$ (called *$\bot$-pruned partition*). Otherwise, the partition is defined as the set of *blocks* that partitions F_i. Here two vertices $u, v \in F_i$ are in the same block if and only if they are connected on the subgraph induced by X.

The i-th partition determines the connectivity among frontiers F_i. Since we should also keep track of the connectivity to terminals K, we prepare the following additional notion.

Definition 4.5. For i-th subset $X \subseteq E_{<i}$ whose i-th partition is $\mathcal{P}$, suppose that $\mathcal{P}$ is not a pruned partition. Let mark be a mapping from the blocks in P to either *true* or *false*. For block $B \in \mathcal{P}$, $\mathsf{mark}(B) =$ *true* if and only if vertices in B are connected with at least one vertex in K on the subgraph induced by X. We call mark as a *mark*.

Now we can verify that the pair of partition and mark constitutes a configure for building the BDD representing $\mathcal{E}_K$.

Theorem 4.6. *For any two i-th subsets $X, Y \subseteq E_{<i}$ whose i-th partitions and marks are identical for each, for any $Z \subseteq E_{<i}$, $X \cup Z$ and $Y \cup Z$ are equivalent in whether the terminals K are interconnected on the induced subgraph.*

Theorem 4.7. *For any two i-th subsets $X, Y \subseteq E_{<i}$ whose i-th partitions and marks are identical for each, the pairs of $(i+1)$-st partition and mark of $X, Y \subseteq E_{<i+1}$, and those of $X \cup \{e_i\}, Y \cup \{e_i\} \subseteq E_{<i+1}$, are also pairwise identical for each.*

Theorem 4.6 corresponds to the condition (i) in Section 2.5 and Theorem 4.7 corresponds to the condition (ii).

4.3.1 Pseudocode

Algorithm 4.1 is the Root and Child procedures for constructing the BDD of $\mathcal{E}_K$; calling Construct($\mathit{Connected}_K$) constructs the BDD representing $\mathcal{E}_K$ (also see Algorithm 2.1). Here, $\mathsf{B}_v^{\mathcal{P}}$ stands for the block of $\mathcal{P}$ containing vertex v. At root, since the only 1st subset is $\emptyset \subseteq E_{<1} = \emptyset$, the partition is empty and the mark is also empty (Line 2). Given configure $(\mathcal{P}, \mathsf{mark})$, we modify it to make a child configure as follows. First, for each vertex u newly entering into the frontier, we insert the block consisting of only u to $\mathcal{P}$ (Line 5). The newly added block's mark is *true* if and only if u is terminal (Line 6). If $f = \text{hi}$, which means that the edge e_i is determined to be present, and the blocks containing v and v' are different where v and v' are the endpoints of e_i, we merge these blocks (Lines 7–8). The merged block's mark is the disjunction of the marks of these blocks (Line 9). Here, if $\mathcal{P}$ contains only one marked block and all the vertices in K appear in $E_{<i}$, i.e., $K \subseteq A_i \cup F_i$, then we can go to $\top$-pruned partition since all the vertices in K are interconnected (Lines 11–12). After that, for each vertex u leaving from the frontier, we remove u from the block containing u (Lines 14–15). If this block is empty and it is marked *true*, then we can go to $\perp$-pruned partition since it means that the vertices in K connected to this block are left isolated. (Line 17–18). The empty and *false*-marked blocks are simply removed (Line 20). The resultant $(\mathcal{P}, \mathsf{mark})$ is the f-child configure (Line 21).

After the BDD $\mathtt{B}_K$ representing $\mathcal{E}_K$ is built, we can compute $R(K)$ value by a DP on $\mathtt{B}_K$ since it is equivalent to solve the weighted model counting problem on $\mathcal{E}_K$ as mentioned in Section 4.2.

Figure 4.1 depicts a running example of HH method, where the input graph is given in the top dnode of Figure 4.1a. Here, each block in a partition is represented as a squared parentheses [ ] with vertex ids inside it. If a block's mark is *true*, $*$ is associated on upper-right of this block. Although we draw multiple terminals, $\perp$ and $\top$, in Figure 4.1b in order to avoid complication, all the $\top$ ($\perp$) dnodes are identical. We observe that three 3rd subsets, $\{\}$, $\{e_1\}$, and $\{e_2\}$, are merged into one dnode in Figure 4.1a since they have the same partition.

4.3.2 Complexity

Here we derive time complexity bounds for computing K-NR by HH method. Let $W_F = \max_i |F_i|$ be the maximum size of frontier, which we called *frontier width.* First of all, the time complexity can be bounded by the size of resultant BDD.

Algorithm 4.1: Procedures for the family of subgraphs that connects the vertices in K.

```
procedure $Connected_K$.ROOT():
    return $\langle 1, (\emptyset, \emptyset) \rangle$
procedure $Connected_K$.CHILD($\langle i, (\mathcal{P}, \mathsf{mark}) \rangle, f$):
    foreach $u \in \{v, v'\} \setminus F_i$ do                    // $e_i = \{v, v'\}$
        Insert $[u]$ as a new block of $\mathcal{P}$        // $u$:vertex entering into frontier
        $\mathsf{mark}([u]) \leftarrow$ true if $u \in K$; false otherwise
    if $f =$ hi and $\mathsf{B}_v^{\mathcal{P}} \neq \mathsf{B}_{v'}^{\mathcal{P}}$ then
        Make new block $B$ by merging $\mathsf{B}_v^{\mathcal{P}}$ and $\mathsf{B}_{v'}^{\mathcal{P}}$
        $\mathsf{mark}(B) \leftarrow \mathsf{mark}(\mathsf{B}_v^{\mathcal{P}}) \vee \mathsf{mark}(\mathsf{B}_{v'}^{\mathcal{P}})$
        Remove blocks $\mathsf{B}_v^{\mathcal{P}}$, $\mathsf{B}_{v'}^{\mathcal{P}}$ from $\mathcal{P}$ and corresponding entries $\mathsf{mark}(\mathsf{B}_v^{\mathcal{P}})$,
          $\mathsf{mark}(\mathsf{B}_{v'}^{\mathcal{P}})$
    if $\mathcal{P}$ contains only one marked block and $K \subseteq A_i \cup F_i$ then
        return $\langle m+1, 1 \rangle$                 // All vertices in $K$ are connected
    foreach $u \in \{v, v'\} \setminus F_{i+1}$ do          // $u$:vertex leaving from frontier
        $B \leftarrow \mathsf{B}_u^{\mathcal{P}}$
        Remove $u$ from $B$
        if $B$ is empty then
            if $\mathsf{mark}(B) =$ true then
                return $\langle m+1, 0 \rangle$       // some vertices in $K$ are left isolated
            else
                Remove block $B$ from $\mathcal{P}$ and corresponding entry $\mathsf{mark}(B)$
    return $\langle i+1, (\mathcal{P}, \mathsf{mark}) \rangle$
```

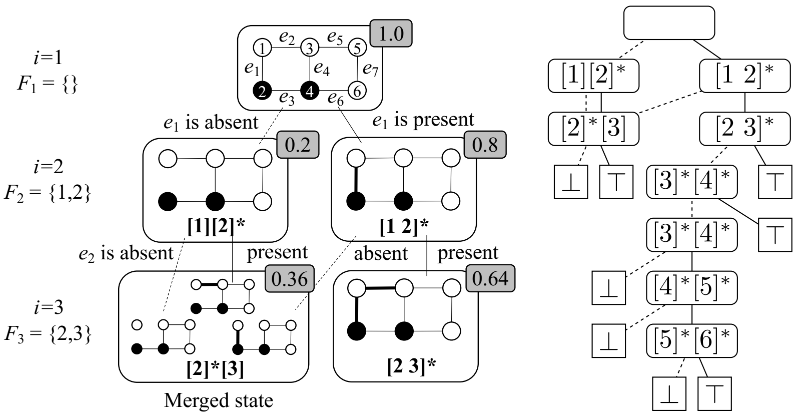
Lemma 4.8. *Let B_K be the BDD built by* CONSTRUCT(*$Connected_K$*). *Then, HH method can compute $R(K)$ in $O(W_F|\mathsf{B}_K|)$ time.*

Proof. Since the weighted model counting problem can be solved in $O(|\mathsf{B}_K|)$ time according to Theorem 2.5, we move our focus on the construction of B_K, i.e., CONSTRUCT($Connected_K$). For each dnode $\mathtt{n}$ in B_K, lo- and hi-child configures are made through CHILD procedure. Each child configure can be made in $O(|F_i|) = O(W_F)$ time by maintaining the partition as an integer sequence of length $|F_i|$ (see [Hardy *et al.*, 2007]). Also, by maintaining dnode list $\mathtt{L}_i$ with a hash map whose key is the configure, we can find the dnode with the desired configure (Line 12 of Algorithm 2.1) in $O(W_F)$ time. Thus, the overall cost for CONSTRUCT($Connected_K$) is bounded by $O(W_F|\mathsf{B}_K|)$. □

We can also bound the size of B_K by using W_F.

Lemma 4.9. $|\mathsf{B}_K| \leq m \cdot E_{W_F} \cdot 2^{W_F}$, *where E_k is the k-th* Bell number, *the number of possible partitions of a set with cardinality k.*

Proof. For i-th subsets, we consider the number of possible configures. The number of possible partitions is bounded by $E_{|F_i|} = O(E_{W_F})$. For each block



(a) Part of built diagram (b) Full diagram

Figure 4.1: (a) First three levels of the constructed diagram of $\textsc{Construct}(\mathit{Connected}_K)$, where K consists of the filled vertices (© 2021 IEEE). (b) Full constructed diagram of $\textsc{Construct}(\mathit{Connected}_K)$.

of each partition, we can choose the mark value, meaning that there are at most $O(2^{|F_i|}) = O(2^{W_F})$ patterns of mark for every partition. Thus, the number of possible configures is bounded by $O(E_{W_F} \cdot 2^{W_F})$. By accumulating it with $i = 1, \ldots, m$, the lemma holds. □

Combining Lemmas 4.8 and 4.9, we immediately obtain the time complexity bound.

Theorem 4.10. *HH method can compute $R(K)$ in $O(m \cdot W_F \cdot E_{W_F} \cdot 2^{W_F})$ time.*

For special cases, 2-NR and All-NR, we can further refine the time complexity.

Corollary 4.11. *When $|K| = 2$, HH method can compute $R(K)$ in $O(m \cdot W_F^3 \cdot E_{W_F})$ time.*

Proof. If $|K| = 2$, the number of blocks marked *true* is at most 2. Thus, the number of mark patterns for every partition is at most $O(|F_i|^2) = O(W_F^2)$. The remaining proof is the same as Theorem 4.10. □

Corollary 4.12. *When $K = V$, HH method can compute $R(K)$ in $O(m \cdot W_F \cdot E_{W_F})$ time.*

Proof. If $K = V$, all the blocks in a partition are marked *true*, and thus the number of mark patterns for every partition is exactly 1. The remaining proof is the same as Theorem 4.10. □

Since the Bell number E_k grows exponentially with respect to k and W_F becomes $\Omega(n)$ in the worst case (e.g., cliques), the aforementioned bounds are generally exponential. However, for typical network topologies, we can sometimes obtain W_F significantly smaller than n and m by carefully manipulating the order of edges $e_1, \ldots, e_m$. In such cases, the above bounds become fairly small.

Indeed, the frontier width W_F is closely related to *path-width* W_p [Robertson and Seymour, 1983] of the graph. In short, the path-width indicates how dissimilar the graph is to a path. Its value is 1 if the graph is just a path, 2 for cycles, and n for a clique with n nodes. We can compute an edge order satisfying $W_F = W_p$ from the optimal *path-decomposition* of graph G [Inoue and Minato, 2016]. Although obtaining optimal path-decomposition is generally NP-hard, we can use a beam-search-based path-width optimization heuristics [Inoue and Minato, 2016] to obtain a good edge order. In the experiments of the following chapters, we used [Inoue and Minato, 2016] to obtain an edge order.

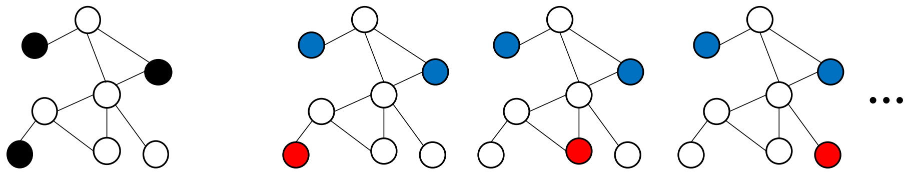
Chapter 5

Network Reliability Evaluation for Client-Server Model

As described in Chapter 4, since network reliability evaluation is a computationally heavy task, several methods have been proposed to efficiently perform it. However, modern network infrastructures follow the client-server model, where many clients are served independently, so we have to evaluate the network reliability for *every* set consisting of the servers and each client. This evaluation process involves repetitive evaluations while changing the set, which imposes a heavy burden on network operators. This chapter proposes a method that efficiently performs network reliability evaluation for the client-server model. Since our method is designed to evaluate reliability for multiple clients without explicit repetition, the computational complexity does not increase compared to the case where existing methods evaluate reliability for a single client. Numerical experiments using datasets of various topologies, including real communication networks, reveal great efficiency. Our method is more than 100 times faster than an existing method that requires repeated evaluation, e.g., it takes only 27 seconds to compute the reliability for 670 clients on a large network with 821 links.

5.1 Introduction

Today's network infrastructures usually follow the client-server model. For example, cloud services are provided from web servers to browsers (clients) distributed in a network, and electric power is delivered from a substation (server) to consumers (clients) through a power distribution network. Since clients are served independently, their reliability should also be evaluated independently to assess their service level agreement. However, this evalua-



(a) K-NR. (b) CSNR.

Figure 5.1: K-NR and CSNR problems ($k = 3$). (a) K-NR computes the probability of connecting the set of terminals (black). (b) CSNR computes the probability of connecting each client (red) with the fixed servers (blue).

tion task is very challenging for network operators, because computationally heavy network reliability (K-NR) evaluations have to be solved for many clients. Here, a terminal set of K-NR (Figure 5.1a) corresponds to each set consisting of servers and variable clients (Figure 5.1b). Since this new problem generalizes K-NR for variable clients with fixed servers, we call it K-*terminal network reliability for fixed servers and variable clients* (abbreviated as CSNR problem, where CS stands for client-server) in this dissertation. To the best of our knowledge, no method has been proposed to efficiently solve CSNR problem.

Although the method is efficient for K-NR, we have to repeat it for every client in CSNR problem.

This chapter proposes a dynamic programming (DP) method to solve CSNR problem. Surprisingly, the computational complexity of the proposed method is the same as that of the HH method (explained in Section 4.3), even though our method solves CSNR problem that computes $O(n)$ K-NRs while the HH method computes only one K-NR. The contributions in this chapter are summarized as follows:

- We design an efficient method for solving CSNR problem that first constructs a decision diagram-like structure downward by only considering servers and then computes reliability for every client while scanning the diagram upward. This process involves no repetition for clients, and it imposes no overhead in terms of computational complexity. In addition, the search process allows our method to solve K-NR more efficiently than does the HH method when the number of terminals is a small constant.

- The proposed method is evaluated numerically with synthetic graphs and real communication networks. As predicted by the computational complexity, our method runs much faster than the HH method in solving CSNR problems. Our method outperforms the HH method by more than 100 times; e.g., for a topology with 821 links and 670 clients, our method takes less than 27 seconds to solve CSNR problem, while the repeated use of the HH method finishes only 2% of the clients in an hour.

5.2 Problem Statement

We consider the independent failure network model described in Section 4.1: an undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges and an availability $p_e \in [0, 1]$ for every edge $e \in E$ are given. In this chapter, we also use the standard definition of K-terminal network reliability $R(K)$ defined in Section 4.2.

For the CSNR problem, we are additionally given the set $T \subseteq V$ of *source vertices*.

Problem 5.1 (CSNR problem)**.** The CSNR problem is to compute $R(T \cup \{v\})$ for every destination $v \in V \setminus T$[1]. That is, for every vertex $v \in V \setminus T$, we compute the probability $R(T \cup \{v\})$ that v is connected to all the vertices in T.

Note that this chapter assumes all vertices except sources are considered as destinations, since this situation maximizes the computational load; the proposed method works even when only a subset is destinations.

For simplicity, we impose some assumptions on G: G has no self-loops, and all vertices in G have a degree not less than 2. Note that these assumptions can be easily removed by preprocessing. First, the existence of self-loop edges does not affect the reliability measure. Second, the degree 1 vertices can be removed by degree 1 vertex elimination mentioned in Section 4.2.1. By recursively removing degree 1 vertices, we obtain a graph with only degree ≥ 2 vertices, or a singleton graph. The network reliability of the original graph can be recovered by applying (4.4) recursively.

5.3 Method

Indeed, the proposed method is based on HH method [Hardy *et al.*, 2007; Herrmann, 2010] explained in Section 4.3. To compute $R(K)$, in HH method, a BDD $\mathtt{B}_K$ representing $\mathcal{E}_K$ is built by using the top-down construction framework, and then the value $R(K)$ is computed by DP on the built BDD $\mathtt{B}_K$.

[1]To be agnostic on infrastructure types, we introduce the neutral terms "sources" and "destinations" instead of servers and clients, respectively.

Similarly, the proposed method also builds a diagram that is similar to BDD and perform DP on it. However, unlike HH method, all the reliability values, $R(T \cup \{v\})$ for all $v \in V \setminus T$, can be computed by single diagram and the subsequent DP. We here describe the intuition behind the proposed method.

In solving CSNR problem with HH method, for every $v \in V \setminus T$ we build BDD $\mathtt{B}_{T\cup\{v\}}$ representing the family $\mathcal{E}_{T\cup\{v\}}$ of subgraphs, the subgraphs such that the vertices in $T \cup \{v\}$ are interconnected, and perform DP on $\mathtt{B}_{T\cup\{v\}}$. However, since the constraints imposed on $\mathcal{E}_{T\cup\{v\}}$ have similarity in that at least the vertices in T must be interconnected, the built BDDs $\mathtt{B}_{T\cup\{v\}}$ also have similarity in their structures. Therefore, we first try to extract the common part of them by building a decision diagram-like structure only considering the sources T. Then, we try to perform the computation of each v by a more sophisticated DP on the built diagram.

To enable the latter part, we consider using the configures utilized in the top-down construction phase also in the DP phase. In HH method, the configures are simply discarded after the top-down construction phase since each state of the subsequent DP is just a dnode of the resultant BDD as described in Section 2.4. Meanwhile, in the proposed method, we consider a DP such that each state is a block of the configure within each dnode.

5.3.1 Construction of Diagram

We first describe the construction of diagram. As described above, we consider only the sources T in building the diagram. We use almost the same definition for partition as HH method (Definition 4.4), but we omit the $\top$-pruned partition. Precisely, the partition in the proposed method is defined as follows.

Definition 5.2. The *i-th partition* of an i-th subset $X \subseteq E_{<i}$ is defined as follows: If there is a vertex in $T \cap A_i$ that is not connected to any frontier in F_i on the subgraph induced by X, the partition is $\perp$ (called *$\perp$-pruned partition*). Otherwise, the partition is defined as the set of *blocks* that partitions F_i. Here two vertices $u, v \in F_i$ are in the same block if and only if they are connected on the subgraph induced by X.

The mark is defined in precisely the same way as Definition 4.5, where K is substituted for T. For the sake of completeness, we again state the definition of marks.

Definition 5.3. For i-th subset $X \subseteq E_{<i}$ whose i-th partition is $\mathcal{P}$, suppose that $\mathcal{P}$ is not a pruned partition. Let mark be a mapping from the blocks in P to either *true* or *false*. For block $B \in \mathcal{P}$, $\mathsf{mark}(B) = \mathit{true}$ if and only

if vertices in B are connected with at least one vertex in T on the subgraph induced by X. We call mark as a *mark*.

We here explain the reason why we omit the $\top$-pruned partition. In CSNR problem, we compute the probability that the vertices in $T \cup \{v\}$ are interconnected. To compute this probability, it is not a sufficient condition that only the vertices in T are interconnected. Therefore even after the vertices in T are interconnected, we should keep track of the connectivity among the other vertices. To realize this, we omit the $\top$-pruned partition. In contrast, if some vertices in T are left isolated, i.e., some portion of the vertices in T are determined to be disconnected from the other vertices in T, there is no chance that the vertices in $T \cup \{v\}$ are interconnected for any v. Thus, $\perp$-pruned partition is kept considered.

Indeed, since the built diagram does not contain $\top$, the resultant diagram only represents an empty set when seeing it as a BDD since there are no paths between the root dnode and $\top$. However, since the resultant diagram has much similar structure to a BDD, we again follow the top-down construction framework in Section 2.5 to describe the procedure.

We describe the procedures for building diagram in Algorithm 5.1. Note that this procedure is almost identical to Algorithm 4.1: the only difference is that we do not perform $\top$-pruning when all the vertices in T are connected.

For example, given the graph presented in Figure 5.2a and $T = \{2, 4\}$, the resultant diagram is drawn in Figure 5.2b. Here, we again use the notation for blocks and marks as in Section 4.3.

5.3.2 Level-wise Reliability Computation

We next describe how to compute $R(T \cup \{v\})$ using the built diagram. As described in Section 2.4, in computing K-NR with HH method, the answer is obtained from the root dnode of the diagram after bottom-up DP is performed. However, since we have to compute $R(T \cup \{v\})$ for every v, the DP computation must be performed $O(n)$ times if we decide to retrieve an answer from the root dnode. To avoid this, we consider retrieving answers from intermediate levels of the diagram.

More specifically, let us focus on the i-th level of the diagram, i.e., the dnodes with label i, $\mathtt{L}_i$ such that $v \in F_i$. Note that such i always exists for any $v \in V$ if all vertices in G have degree ≥ 2; e.g., $v \in F_{i'}$ such that $e_{i'-1}$ is the first edge containing v within the edge order. Let $\mathsf{DP}_{\downarrow}[\mathtt{n}]$ be the probability such that the configure of i-th subset becomes that of $\mathtt{n}$. Then

Algorithm 5.1: Procedures for constructing diagram to solve CSNR problem.

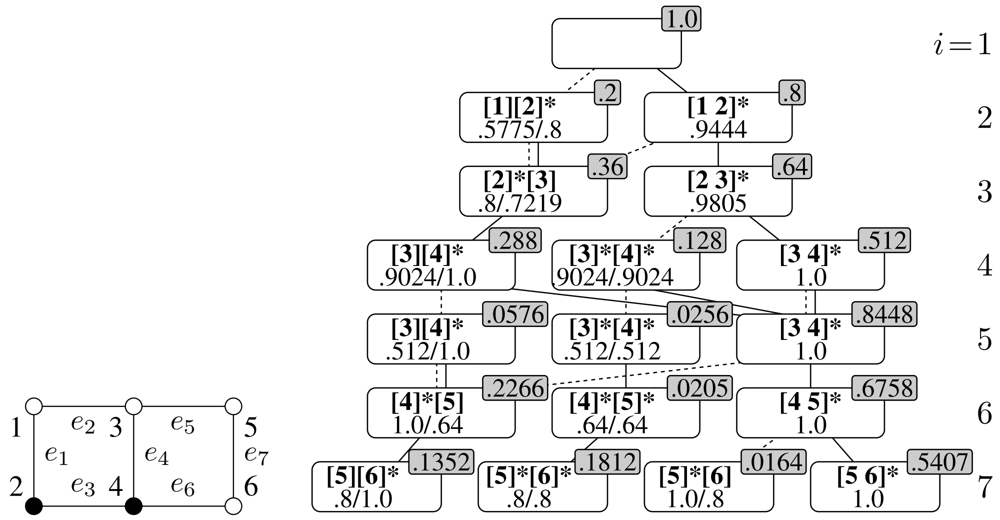
```
procedure $CSNR_T$.Root():
    return $\langle 1, (\emptyset, \emptyset)\rangle$
procedure $CSNR_T$.Child($\langle i, (\mathcal{P}, \mathsf{mark})\rangle, f$):
    foreach $u \in \{v, v'\} \setminus F_i$ do                        // $e_i = \{v, v'\}$
        Insert $[u]$ as a new block of $\mathcal{P}$     // $u$:vertex entering into frontier
        $\mathsf{mark}([u]) \leftarrow$ true if $u \in T$; false otherwise
    if $f =$ hi and $\mathsf{B}_v^{\mathcal{P}} \neq \mathsf{B}_{v'}^{\mathcal{P}}$ then
        Make new block $B$ by merging $\mathsf{B}_v^{\mathcal{P}}$ and $\mathsf{B}_{v'}^{\mathcal{P}}$
        $\mathsf{mark}(B) \leftarrow \mathsf{mark}(\mathsf{B}_v^{\mathcal{P}}) \vee \mathsf{mark}(\mathsf{B}_{v'}^{\mathcal{P}})$
        Remove blocks $\mathsf{B}_v^{\mathcal{P}}$, $\mathsf{B}_{v'}^{\mathcal{P}}$ from $\mathcal{P}$ and corresponding entries $\mathsf{mark}(\mathsf{B}_v^{\mathcal{P}})$,
          $\mathsf{mark}(\mathsf{B}_{v'}^{\mathcal{P}})$
    foreach $u \in \{v, v'\} \setminus F_{i+1}$ do         // $u$:vertex leaving from frontier
        $B \leftarrow \mathsf{B}_u^{\mathcal{P}}$
        Remove $u$ from $B$
        if $B$ is empty then
            if $\mathsf{mark}(B) =$ true then
                return $\langle m+1, 0\rangle$       // some vertices in $T$ are left isolated
            else
                Remove block $B$ from $\mathcal{P}$ and corresponding entry $\mathsf{mark}(B)$
    return $\langle i+1, (\mathcal{P}, \mathsf{mark})\rangle$
```

we can rewrite $R(T \cup \{v\})$ as follows:

$$\begin{aligned} R(T \cup \{v\}) &= \Pr(v \text{ is connected with all } v' \in T) \\ &= \textstyle\sum_{\mathtt{n} \in \mathtt{L}_i} \mathsf{DP}_{\downarrow}[\mathtt{n}] \cdot \Pr(v \text{ is connected to all } v' \in T \mid \mathtt{n}), \end{aligned} \tag{5.1}$$

where the second term is the conditional probability given that the configure of i-th subset $E_{<i}$ becomes that of $\mathtt{n}$.

Regarding the second term, we define the probability $\mathsf{DP}_{\uparrow}[\mathtt{n}, B]$ that, given dnode $\mathtt{n}$ and block B of the partition inside the configure of $\mathtt{n}$, vertices in block B is connected to all the vertices in T with the edges in the subset of $E_{\geq i}$ (unproceeed edges). Such probability can be consistently defined due to the following observation. Given two i-th subsets $X, Y \subseteq E_{<i}$ whose corresponding configures are identical, for any subset Z of $E_{\geq i}$, $X \cup Z$ and $Y \cup Z$ are equivalent in whether v is connected to all the vertices in T on the induced subgraph. This can be verified by imaginarily marking block $\mathsf{B}_v^{\mathcal{P}_{\mathtt{n}}}$, the block of $\mathcal{P}_{\mathtt{n}}$ containing vertex v, and then applying the HH method [Hardy *et al.*, 2007; Herrmann, 2010], where $\mathcal{P}_{\mathtt{n}}$ is the partition inside the configure of $\mathtt{n}$. Moreover, based on the above argument, the probability of the second term is the same even if v is replaced with any other vertex in $\mathsf{B}_v^{\mathcal{P}_{\mathtt{n}}}$.



(a) Graph (b) Built diagram

Figure 5.2: (a) Example of graph. Filled vertices indicate that they are in T. (b) Diagram of proposed method with the graph presented in (a) given that $T = \{2, 4\}$ and the probability calculation along with this diagram when $p_i = 0.8$ for all edges. Solid and dashed lines indicate that the edge is working and failing, respectively. Here $\perp$-pruned partitions are omitted.

Using this, (5.1) can be rewritten as

$$R(T \cup \{v\}) = \sum_{\mathtt{n} \in \mathsf{L}_i} \mathsf{DP}_{\downarrow}[\mathtt{n}] \cdot \mathsf{DP}_{\uparrow}[\mathtt{n}, \mathsf{B}_v^{\mathcal{P}_{\mathtt{n}}}] \quad (v \in F_i). \tag{5.2}$$

5.3.3 Top-Down and Bottom-Up Dynamic Programming

The remaining discussion is how to compute $\mathsf{DP}_{\downarrow}[\mathtt{n}]$ and $\mathsf{DP}_{\uparrow}[\mathtt{n}, B]$. Regarding the former, $\mathsf{DP}_{\downarrow}[\mathtt{n}]$ can be represented by the $\mathsf{DP}_{\downarrow}$ values of the parent dnodes of $\mathtt{n}$. Here, the parent dnodes of dnode $\mathtt{n}$ is the dnodes such that at least one of lo- and hi-child dnodes is $\mathtt{n}$. Specifically, the following equation holds, where $i = \mathsf{lb}(\mathtt{n})$:

$$\mathsf{DP}_{\downarrow}[\mathtt{n}] = (1 - p_{e_{i-1}}) \cdot \sum_{\mathtt{n}' \in \mathsf{L}_{i-1} : \mathsf{lo}(\mathtt{n}') = \mathtt{n}} \mathsf{DP}_{\downarrow}[\mathtt{n}'] + p_{e_{i-1}} \cdot \sum_{\mathtt{n}' \in \mathsf{L}_{i-1} : \mathsf{hi}(\mathtt{n}') = \mathtt{n}} \mathsf{DP}_{\downarrow}[\mathtt{n}']. \tag{5.3}$$

By starting with $\mathsf{DP}_{\downarrow}[\mathtt{r}] = 1$ for root dnode $\mathtt{r}$, all $\mathsf{DP}_{\downarrow}$ values can be computed using (5.3) in a top-down manner.

As for the latter, we formally consider the correspondence between blocks of successive dnodes.

Definition 5.4. Given partition $\mathcal{P}$ inside the configure of dnode $\mathtt{n}$ and $f \in \{\mathrm{lo}, \mathrm{hi}\}$, assume that f-child of $\mathtt{n}$ is not $\perp$. Let $\mathcal{P}'$ be the partition inside the configure of the f-child dnode of $\mathtt{n}$. For block $B \in \mathcal{P}$, we define an *f-child block* of B as follows: (1) If B contains vertex v in F_{i+1}, it is defined as $\mathsf{B}_v^{\mathcal{P}'}$, i.e., the block of $\mathcal{P}'$ containing v. (2) If no such vertex exists, the lo-child block is defined as $\emptyset$, i.e., lo-child block is not exist. For hi-child block, let $e_i = \{u, w\}$. If $w \in F_{i+1}$ and $u \in B$, it is defined as $\mathsf{B}_w^{\mathcal{P}'}$. If $u \in F_{i+1}$ and $w \in B$, it is defined as $\mathsf{B}_u^{\mathcal{P}'}$. Otherwise, it is defined as $\emptyset$. We write the lo- and hi-child blocks of B as $\mathsf{lo}(B)$ and $\mathsf{hi}(B)$.

For example, let us focus on the partition of dnode of level 5 $\mathcal{P} = [3][4]$ in Figure 5.2. For $B = [4]$, $\mathsf{lo}(B) = \mathsf{hi}(B) = [4]$. For $B = [3]$, $\mathsf{lo}(B) = \emptyset$ since vertex 3 is not in F_6. However, $\mathsf{lo}(B) = [5]$ because $e_5 = \{3, 5\}$ and vertex 5 is in F_6. Now we observe that if e_i is absent, block B of the partition inside the dnodes in L_i is connected to all T vertices iff $\mathsf{lo}(B)$ is connected to all T vertices. If e_i is present, $\mathsf{lo}(B)$ of the above argument is replaced with $\mathsf{hi}(B)$. Thus, $\mathsf{DP}_\uparrow[\mathtt{n}, B]$ can be written as

$$\mathsf{DP}_\uparrow[\mathtt{n}, B] = (1 - p_{e_{\mathsf{lb}(\mathtt{n})}}) \cdot \mathsf{DP}_\uparrow[\mathsf{lo}(\mathtt{n}), \mathsf{lo}(B)] + p_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{DP}_\uparrow[\mathsf{hi}(\mathtt{n}), \mathsf{hi}(B)], \quad (5.4)$$

if both $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$ are not $\perp$; here, we let $\mathsf{DP}_\uparrow[\cdot, \emptyset] = 0$. This enables us to compute $\mathsf{DP}_\uparrow[\mathtt{n}, B]$ in a bottom-up manner.

If $\mathsf{lo}(\mathtt{n}) = \perp$, $\mathsf{DP}_\uparrow[\mathsf{lo}(\mathtt{n}), \mathsf{lo}(B)]$ in (5.4) is replaced with 0 except for the case when all T vertices are appeared in $E_{<i}$ and B is the only marked block in $\mathcal{P}$. In such a case, all T vertices are already connected to B and thus $\mathsf{DP}_\uparrow[\mathsf{lo}(\mathtt{n}), \mathsf{lo}(B)]$ is replaced with 1. The same argument also holds for the case where $\mathsf{hi}(\mathtt{n}) = \perp$. There are corner cases for computing $\mathsf{DP}_\uparrow$ when $\mathsf{hi}(\mathtt{n}) = \perp$ and neither endpoint of e_i is in F_{i+1}. Let $e_i = \{u, w\}$ and assume that all T vertices are appeared in $E_{<i}$ and that $\mathsf{B}_u^{\mathcal{P}} \neq \mathsf{B}_w^{\mathcal{P}}$. First, if $\mathsf{B}_u^{\mathcal{P}}$ is the only marked block, $\mathsf{DP}_\uparrow[\mathtt{n}, \mathsf{B}_u^{\mathcal{P}}] = 1$ and $\mathsf{DP}_\uparrow[\mathtt{n}, \mathsf{B}_w^{\mathcal{P}}] = p_{e_{\mathsf{lb}(\mathtt{n})}}$ since the vertices in $\mathsf{B}_w^{\mathcal{P}}$ can eventually connected to all the T vertices only when $e_{\mathsf{lb}(\mathtt{n})}$ is present. Second, if $\mathsf{B}_u^{\mathcal{P}}$ and $\mathsf{B}_w^{\mathcal{P}}$ are the only two marked blocks, then $\mathsf{DP}_\uparrow[\mathtt{n}, \mathsf{B}_u^{\mathcal{P}}] = \mathsf{DP}_\uparrow[\mathtt{n}, \mathsf{B}_w^{\mathcal{P}}] = p_{e_{\mathsf{lb}(\mathtt{n})}}$ since the vertices in these blocks can eventually connected to all the T vertices only when $e_{\mathsf{lb}(\mathtt{n})}$ is present.

Figure 5.2 depicts $\mathsf{DP}_\downarrow[\mathtt{n}]$ and $\mathsf{DP}_\uparrow[\mathtt{n}, B]$ for all dnodes and blocks when $p_i = 0.8$ for all edges. The shaded box in the upper-right corner indicates $\mathsf{DP}_\downarrow[\mathtt{n}]$, and $\mathsf{DP}_\uparrow[\mathtt{n}, B]$ is written inside the white box. By focusing on $\mathtt{L}_5$, we can compute $R(T \cup \{3\})$ as $.288 \cdot .9024 + .128 \cdot .9024 + .512 \cdot 1.0 = .8874$. Other reliability values can be computed with different levels. Note that even if there are multiple levels whose frontier vertices contain v, we can choose any one of them to compute $R(T \cup \{v\})$ because the computed value will be identical. We also note that (5.2) holds even for $v \in T$, which yields the value of $R(T)$.

Algorithm 5.2 is the pseudocode for the DP and level-wise reliability computations. Lines 1–8 compute $\mathsf{DP}_{\downarrow}[\mathtt{n}]$ via top-down DP and lines 9–20 compute $\mathsf{DP}_{\uparrow}[\mathtt{n}, B]$ via bottom-up DP. The reliability computation is performed in lines 22–25. Finally, $\mathsf{r}[v]$ stores the value $R_{G,p}(T \cup \{v\})$ for each destination v.

5.3.4 Complexity

We first consider the size of the search diagram and then analyze the complexity of our algorithm. We again use the frontier width defined as $W_F = \max_i |F_i|$.

Lemma 5.5. *The number of dnodes in the diagram is at most $O(m \cdot 2^{W_F} \cdot E_{W_F})$, where E_l is the l-th Bell number.*

Proof. At level i, there are at most $E_{|F_i|}$ patterns on the blocks and $2^{|F_i|}$ patterns on the marks. Thus, the number of i-th level partitions is at most $2^{|F_i|} \cdot E_{|F_i|}$. The overall size is bounded by $\sum_{i=1}^{m} 2^{|F_i|} \cdot E_{|F_i|} = O(m \cdot 2^{W_F} \cdot E_{W_F})$. □

This leads to the complexity of our proposed algorithm.

Theorem 5.6. *Algorithms 5.1 and 5.2 can solve CSNR problem in $O(m \cdot W_F \cdot 2^{W_F} \cdot E_{W_F})$ time.*

Proof. This follows from the fact that for each dnode $\mathtt{n}$, the processing of configures in Algorithm 5.1 is performed in $O(|F_i|)$ time, and the DP computations of $\mathsf{DP}_{\downarrow}[\mathtt{n}]$ and $\mathsf{DP}_{\uparrow}[\mathtt{n}, \cdot]$ are also in $O(|F_i|)$ time. The former holds because all operations in Algorithm 5.1 can be performed in $O(|F_i|)$ time for $\mathtt{n}$. The latter can be verified because there are at most $O(|F_i|)$ values ($\mathsf{DP}_{\downarrow}[\mathtt{n}]$ and $\mathsf{DP}_{\uparrow}[\mathtt{n}, \cdot]$) to be computed and they are each computed in constant time via (5.3) and (5.4). □

The complexity of Theorem 5.6 is the same as that of the HH method for solving K-NR. Regarding the CSNR problem, the proposed method is $O(n)$ times faster than the HH method. Moreover, when $k = |T| + 1$ is assumed to be a constant, a better time complexity can be achieved.

Theorem 5.7. *If $k = |T| + 1$ is assumed to be a constant, Algorithms 5.1 and 5.2 can solve CSNR problem in $O(m \cdot W_F^k \cdot E_{W_F})$ time.*

Algorithm 5.2: DP and level-wise reliability computation.

```
DP↓[r] ← 1                                                // Start with root dnode r
for $i \leftarrow 2$ to $m$ do
    foreach n ∈ $L_i$ do DP↓[n] ← 0                       // Init DP↓
for $i \leftarrow 1$ to $m$ do                                    // Top-down DP
    foreach n ∈ $L_i$ do
        foreach f ∈ {lo, hi} do
            if f(n) ≠ ⊥ then
                DP↓[f(n)] += f($p_i$) · DP↓[n]                // (5.3)
                                  // here lo($p_i$) = $1 - p_i$ and hi($p_i$) = $p_i$
for $i \leftarrow m$ to 2 do                                      // Bottom-up DP
    foreach n ∈ $L_i$ do
        for $B \in \mathcal{P}_n$ do DP↑[n, $B$] ← 0                // Init DP↑
        foreach f ∈ {lo, hi} do
            if f(n) = ⊥ then
                if ($A_i \supseteq T$) and ($\mathcal{P}_n$ contains only one marked block) then
                    DP↑[n, $B^*$] += f($p_i$) for marked block $B^*$
                if ($A_i \supseteq T$) and (f = hi) and ($|F_i \setminus F_{i+1}| = 2$) then
                    ProcessCornerCase($i$, n)
            else                                              // f(n) ≠ ⊥
                for $B \in \mathcal{P}_n$ do
                    if f($B$) ≠ ∅ then
                        DP↑[n, $B$] += f($p_i$) · DP↑[f(n), f($B$)]     // (5.4)
foreach $v \in V \setminus T$ do                          // Level-wise Rel. Computation
    Choose $i \in \{1, \ldots, m\}$ s.t. $v \in F_i$
    r[$v$] ← 0
    foreach n ∈ $L_i$ do
        r[$v$] += DP↓[n] · DP↑[n, $B_v^{\mathcal{P}_n}$]                   // (5.2)
procedure ProcessCornerCase($i$, n):
    if $B_u^{\mathcal{P}_n} \neq B_w^{\mathcal{P}_n}$ then                             // $e_i = \{u, w\}$
        if $B_u^{\mathcal{P}_n}$ is the only marked block or $B_u^{\mathcal{P}_n}, B_w^{\mathcal{P}_n}$ are the only two marked blocks
          then
            DP↑[n, $B_u^{\mathcal{P}_n}$] += $p_i$
        if $B_w^{\mathcal{P}_n}$ is the only marked block or $B_u^{\mathcal{P}_n}, B_w^{\mathcal{P}_n}$ are the only two marked blocks
          then
            DP↑[n, $B_w^{\mathcal{P}_n}$] += $p_i$
```

Proof. For this scenario, we can say that the size of the search diagram is at most $O(m \cdot W_F^{k-1} \cdot E_{W_F})$. This is because, at level i, the number of patterns of the marks is at most $\sum_{t=0}^{\min\{k-1,|F_i|\}} \binom{|F_i|}{t} = O(W_F^{k-1})$. Following the proof of Theorem 5.6, the overall complexity is $O(m \cdot W_F^{k} \cdot E_{W_F})$. □

Theorem 5.7 includes the computation of 2-NRs for multiple destinations as a special case ($k = 1$). When $k = |K|$ is constant, we can prove that the HH method [Hardy *et al.*, 2007; Herrmann, 2010] could evaluate K-NR in $O(m \cdot W_F^{k+1} \cdot E_{W_F})$ time in the same way as the proof of Corollary 4.11. This means that, when k is small, although our method can compute reliabilities for $n-k+1$ destinations, the time complexity is $O(W_F)$ times faster than the single reliability computation with the HH method. This difference comes from the fact that our method marks the blocks with T, while the HH method marks them with $T \cup \{v\}$.

As described in Section 4.3.2, the frontier width W_F is closely related to the path-width W_p and the path decomposition.

5.4 Experiments

We compared the proposed method and the HH method [Hardy *et al.*, 2007; Herrmann, 2010] with respect to the consumed time for solving CSNR problem. We also compared the performance for evaluating K-NR to examine the effect of complexity results in Section 5.3.4. The proposed method is implemented in C++11. For HH method, we used TdZdd (available at `https://github.com/kunisura/TdZdd`), which implements the improved variant [Herrmann, 2010]. All codes were compiled by g++-9.1.0 with `-O3 -DNDEBUG` options. Experiments were conducted on a single thread of a Linux machine with Intel Xeon E7-8880 2.20 GHz CPU and 3072 GB RAM; note that, as described later, both methods used less than 65 GB of memory, except for one instance. The implementation and all data used in this section are available at `https://github.com/nttcslab/cs-reliability`.

We used both synthetic and real graphs as tested instances. The synthetic graphs are the grid graphs; Grid-wxh denotes a grid graph with $w \times h$ vertices. For these, we used the edge ordering of Iwashita et al. [Iwashita *et al.*, 2013], which is known to be better for DP on grid graphs. The real graphs are from the Internet Topology Zoo [Knight *et al.*, 2011] and Rocketfuel [Spring *et al.*, 2004] datasets. For these, we extracted the largest connected component, removed self-loops, and recursively removed the degree 1 vertices. We decided the edge ordering according to the beam-search based method [Inoue and Minato, 2016]. For each graph, we computed the betweenness central-

Table 5.1: Consumed time for solving CSNR problem: proposed method vs. TdZdd, a modern implementation of the HH method.

						CSNR (sec.)					
	original		preprocessed			$k = 2(\|T\| = 1)$		$k = 6(\|T\| = 5)$		$k = 11(\|T\| = 10)$	
Instance	n	m	n	m	W_F	Ours	TdZdd	Ours	TdZdd	Ours	TdZdd
Grid-7x14	98	175	98	175	7	**0.17**	17.69	**0.57**	22.42	**0.45**	18.95
Grid-7x28	196	357	196	357	7	**0.31**	90.84	**1.04**	124.18	**1.21**	114.14
Grid-7x42	294	539	294	539	7	**0.53**	221.00	**1.81**	324.94	**1.78**	298.82
Grid-8x8	6 64	112	64	112	8	**0.24**	22.89	**0.71**	24.23	**0.95**	19.71
Grid-10x10	100	180	100	180	10	**10.78**	>1h	**48.76**	>1h	**63.96**	>1h
Grid-12x12	144	264	144	264	12	**320.74**	>1h	**2309.55**	>1h	**3465.40**	>1h
Interoute	110	146	102	138	7	**0.02**	2.12	**0.04**	2.13	**0.05**	1.88
TataNld	145	186	136	177	6	**0.01**	0.99	**0.02**	1.11	**0.01**	0.51
Kdl	754	895	680	821	12	**6.55**	>1h	**24.48**	>1h	**26.82**	>1h
Rocketfuel-1221	318	758	178	618	12	**15.38**	>1h	**12.52**	2032.51	**12.43**	2619.35
Rocketfuel-1755	172	381	146	355	12	**22.18**	>1h	**105.30**	>1h	**48.65**	>1h
Rocketfuel-3257	240	404	139	303	12	**69.08**	>1h	**400.97**	>1h	**611.06**	>1h
Rocketfuel-3967	201	434	166	399	13	**128.33**	>1h	**277.61**	>1h	**213.04**	>1h
Rocketfuel-6461	182	294	100	212	10	**6.21**	1565.17	**9.51**	733.77	**3.89**	182.64

Table 5.2: Consumed time for evaluating K-NR: proposed method vs. TdZdd.

	original		preprocessed			K-NR (sec.)					
						$k = 2$		$k = 6$		$k = 11$	
Instance	n	m	n	m	W_F	Ours	TdZdd	Ours	TdZdd	Ours	TdZdd
Grid-7x14	98	175	98	175	7	**0.19**	0.33	0.51	**0.35**	0.39	**0.30**
Grid-7x28	196	357	196	357	7	**0.29**	0.49	0.95	**0.64**	1.03	**0.59**
Grid-7x42	294	539	294	539	7	**0.47**	0.79	1.66	**1.04**	1.59	**1.01**
Grid-8x8	6 64	112	64	112	8	**0.21**	0.44	0.65	**0.49**	0.80	**0.35**
Grid-10x10	100	180	100	180	10	**9.18**	42.56	**44.39**	56.23	57.97	**54.37**
Grid-12x12	144	264	144	264	12	**284.11**	1535.82	**2105.08**	>1h	**3139.19**	>1h
Interoute	110	146	102	138	7	**0.02**	0.05	**0.04**	**0.04**	0.04	**0.03**
TataNld	145	186	136	177	6	**0.01**	0.02	0.02	**0.01**	**0.01**	**0.01**
Kdl	754	895	680	821	12	**5.57**	164.55	**19.38**	248.36	**23.95**	247.99
Rocketfuel-1221	318	758	178	618	12	**12.55**	30.88	**10.83**	13.32	**11.06**	12.20
Rocketfuel-1755	172	381	146	355	12	**19.80**	116.32	87.36	**70.23**	44.85	**42.71**
Rocketfuel-3257	240	404	139	303	12	**56.70**	108.63	**360.49**	784.34	496.69	**224.91**
Rocketfuel-3967	201	434	166	399	13	**103.58**	346.81	**223.50**	399.75	**177.51**	242.20
Rocketfuel-6461	182	294	100	212	10	**5.59**	19.07	7.56	**3.81**	3.33	**2.01**

ity [Freeman, 1977] of each vertex and chose $|T| = 1, 5, 10$ vertices with higher centrality as sources T of the CSNR problem, because sources should be deployed in an easily accessible "center" of the network. For the K-NR problem, we chose $k = 2, 6, 11$ vertices with higher betweenness centrality as terminals K. Since the original data [Knight *et al.*, 2011; Spring *et al.*, 2004] do not include the working probability p_i of each e_i, it was chosen uniformly at random from $[0.9, 0.95]$ according to the literature [Elshqeirat *et al.*, 2015; Botev *et al.*, 2012; Xiao *et al.*, 2009; Nishino *et al.*, 2018; Inoue, 2019]. We set the time limit of each run to 1 hour.

Table 5.1 shows the results for CSNR problem. The results demonstrate that our method can solve CSNR problem 30–400 times faster than the existing method. Notably, our method successfully computed all reliability measures within 1 hour even for larger graphs, e.g., those having more than 200 edges and $W_F = 12, 13$. Although the K-NR problem on Rocketfuel graphs with more than 200 edges has not yet been solved to our knowledge, our method can solve more difficult CSNR problem in a reasonable time. Regarding the results of Grid-7xh, the computation time is roughly $O(h)$ for our method and $O(h^2)$ for TdZdd. This is because the complexity for a graph with constant W_F is $O(m)$ for our method and $O(mn)$ for the HH method. As for the memory usage, in solving a Grid-12x12 ($|T| = 10$) instance, our method used 341.2 GB while TdZdd used 177.3 GB when the computation expired. The second largest peak memory usage was recorded for Rocketfuel-3257, in which our method used 64.2 GB and TdZdd used 61.3 GB.

As for the K-NR problem, we summarize the results in Table 5.2. Our method is faster than TdZdd for all instances with $k = 2$ and graphs having large W_F with $k = 6, 11$. This reflects the complexity result of Theorem 5.7: When k is small, our method is $O(W_F)$ times faster than the HH method. On the other hand, since the HH method has pruning techniques that are not used in our methods, namely 1-pruning when all terminals are connected and 0-pruning when v is disconnected, the HH method is sometimes faster in solving K-NR when k is large or W_F is small.

5.5 Related Work

As described in Chapter 4, Hardy *et al.* [2007] proposed a basic method, and Herrmann [2010] improved the equivalence decision of states, but their computational complexity is the same. Several reliability problems extending K-NR have been studied: e.g., reliability optimization [Nishino *et al.*, 2018], reliability for any k terminals of n vertices [Li *et al.*, 2009], reliability under routing constraints [Hayashi and Abe, 2008], and reliability under disjoint-paths constraints [Inoue, 2019]. Despite that rich body of literature, no work has studied CSNR problem.

To reduce the computation costs, approximate solutions like a Monte Carlo simulation have been studied [Gertsbakh and Shpungin, 2009], but other works [Nishino *et al.*, 2018; Inoue, 2019] showed that the deviation of the Monte Carlo approach is often more than double. Canale *et al.* [2014] proposed F-Monte Carlo, which estimates the probability of rare events accurately, but depends on an unrealistic assumption, i.e., that all links are likely to fail with equal probability. Khan *et al.* [2014] approximately enumerated destinations whose reliability for any source exceeds a given threshold. The problem seems similar to ours, but reliability values are not given, so we cannot quantitatively assess the risk of each destination.

5.6 Conclusion

The work in this chapter made two contributions: (1) An efficient method was proposed for CSNR problem, with a time complexity that is the same as the existing method for K-NR, that is, our method can solve CSNR problem $O(n)$ times faster than the existing method; and (2) We revealed the method's practical efficiency with real networks that have hundreds of edges. In future work, we will elaborate our method for specific infrastructures.

Chapter 6

Fast Evaluation for the Expected Number of Connected Nodes

Several network infrastructures are required to keep all nodes connected, although these nodes are occasionally disconnected due to failures. Thus, the *expected number* of connected node pairs (ECP) during an operation period is a reasonable reliability measure in network design. However, no work has studied ECP due to its computational hardness; we have to solve the tough reliability evaluation problem for $O(n^2)$ times where n is the number of nodes in a network. This chapter proposes an efficient method that exactly computes ECP. Our method performs dynamic programming just once without explicit repetition for each node pair and obtains an exact ECP value weighted by the number of users at each node. A thorough complexity analysis reveals that our method is faster than the method in Chapter 5, which can be transferred to ECP computation, by $O(n)$. Numerical experiments using real topologies show great efficiency; e.g., our method computes the ECP of an 821-link network in ten seconds; the existing method and the method in Chapter 5 cannot complete it in an hour. This chapter also presents two applications: critical link identification and optimal server placement.

6.1 Introduction

This chapter categorizes network infrastructures into two types: point-to-point (P2P) infrastructures, which allow their users to access each other (e.g., telecommunication and transportation), and client-server type (CS) infrastructures, which let users access a shared server (e.g., cloud computing and power). Infrastructures are required to keep all users connected to each other or a server [Nojo and Watanabe, 1987, 1993; Tollar and Bennett, 1995], but network components (e.g., links) can occasionally fail, disconnecting some users. Thus, the number of users that were able to connect during operation is a reasonable performance measure, and its *expected number* is a good reliability measure in network design. This chapter mainly studies the expected number of connected node pairs (ECP) for P2P infrastructures and also examines the expected number of connected nodes (ECN) for CS infrastructures. Infrastructures can be effectively enhanced by reinforcing links that have the greatest impact on the ECP or ECN [Frank, 1992]. The performance of CS infrastructure can be optimized by adding another server to a location where ECN is maximized [Sahoo *et al.*, 2016]. Since network infrastructures are usually very large and important, evaluation methods of ECN and ECP must be scalable and accurate.

6.1.1 Literature Review

Historically, the all-terminal network reliability (All-NR), i.e., the probability of connecting all nodes (see Section 4.2), has been studied as a reliability measure of an entire network [Hardy *et al.*, 2005; Chaturvedi, 2016; Gaur *et al.*, 2021]. However, All-NR often yields counter-intuitive reliability values. Network (a) in Figure 6.1 is reliable overall, although only the right-most node is likely to be disconnected. All-NR is affected by this right-most node, resulting in a very poor value, while ECP is reasonably high, reflecting the reliable clique on the left side. Network (b) in Figure 6.1 is vulnerable due to an unreliable bridge, and the ECP becomes fairly low. ECP distinguishes networks by considering the number of affected node pairs.

The connected nodes and connected pairs have often been counted in network vulnerability analysis [Albert *et al.*, 2000; Cohen *et al.*, 2001; Magoni, 2003; Smith and Song, 2020; Neumayer and Modiano, 2010, 2011; Agarwal *et al.*, 2010; Rahnamay-Naeini *et al.*, 2011; Manzano *et al.*, 2011; Oostenbrink and Kuipers, 2017; Natalino *et al.*, 2017; Al Mtawa *et al.*, 2021]. These past studies focused on severe events such as natural disasters, wars, or epidemics to evaluate the impact on the number of connected nodes. However, they ignored the average utility of the network in the long run. On the contrary, this chapter studies the *expected number* of connected nodes/pairs during a long operation period of ordinary times.

To the best of our knowledge, no work has studied ECP assuming long operation periods, perhaps due to very high computational cost. As described in Section 4.1, infrastructures in ordinary times are usually modeled as independent failure model [Moskowitz, 1958], in which each edge fails

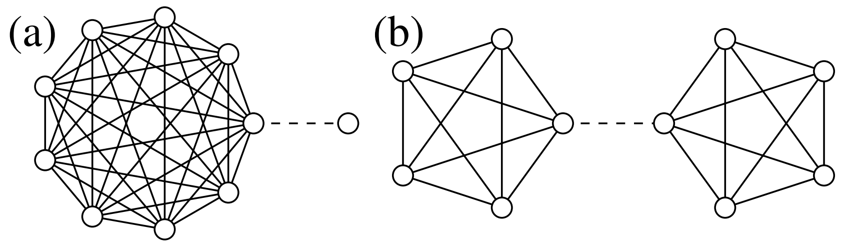


Figure 6.1: Example P2P infrastructures to illustrate the difference between All-NR and ECP. Networks (a) and (b) have ten nodes (45 node pairs). Solid and dashed links fail independently with 10% and 50% probabilities, respectively. All-NR is 0.4999 for network (a) and 0.4994 for network (b), almost identical. ECP is 40.49 (0.8998 as normalized by pair counts) for network (a) and 32.49 (0.7720) for network (b), distinguishing them.

independently with a given probability. Under this model, the ECP can be obtained as the sum of 2-NRs for all vertex pairs, i.e., $\frac{1}{2}\sum_{u,v\in V:u\neq v} R(\{u,v\})$, where V is the set of vertices and $R(\{u,v\})$ is 2-NR (the probability of connecting two vertices, u and v). Unfortunately, as described in Section 4.2.1, even computing single $R(\{u,v\})$ is a computationally tough problem known as #P-complete [Valiant, 1979]. It is very hard to solve the #P-complete problem (2-NR) for all pairs $O(n^2)$ times where n is the number of vertices, and this difficulty prevented past studies from examining the ECP. Note that past studies on severe events described in the previous paragraph only dealt with a polynomial number of network states that could occur after the event.

We here mention that the method proposed in Chapter 5, which we call CSNR method in this chapter, can be used for computing ECP. By setting the sources as $T = \{u\}$, the CSNR problem (Problem 5.1) becomes computing $R(\{u,v\})$ for every $v \in V$. Thus, by solving CSNR problems with $T = \{u\}$ for every u, we can compute $R(\{u,v\})$ for all vertex pairs, yielding the ECP value. Ultimately, the CSNR method allows us to compute ECP in $O(mnW_F^2B_W)$ time by running it $O(n)$ times, where W_F is the frontier width. Using the CSNR method, we conducted preliminary experiments to compute ECP for the 821-link network. However, we were not able to complete it even with a one-hour time limit.

6.1.2 Our Contribution

This chapter proposes an efficient method that exactly computes ECP and presents two applications.

- This chapter develops a method that computes ECP in $O(mW_F^2B_W)$

time. Since the proposed method is equivalent to $O(n^2)$ 2-NRs, it is faster than the HH method by $O(n^2 W_F)$ and the CSNR method by $O(n)$. In addition, our method is designed to specify node weights to represent user counts at each node, which is a favorable feature for practical use. Our method's computation time was experimentally measured with real communication network topologies. The results show that it computed the ECP of an 821-link network in just ten seconds.

- Application 1: the critical link identification problem. Using the proposed method with an automatic differentiation technique [Griewank and Walther, 2008], we can compute the criticality of each link in a P2P infrastructure. Our criticality, defined based on the ECP, was compared with the All-NR's criticality [Kuo *et al.*, 2007; Inoue, 2019] and well-known network centrality indices [Koschützki *et al.*, 2005]. Numerical evaluation with actual topologies reveals that the existing measures likely identify links less critical for ECP, implying that our criticality measure is required to effectively reinforce networks.

- Application 2: the optimal server placement problem. By exploiting our ECP method, which can compute ECNs for every node, we can efficiently select the optimal node on which another server is added to a CS infrastructure. Numerical experiments with actual topologies show that our method is 70–10,000 times faster than existing methods.

The rest of this chapter is organized as follows. Section 6.2 defines a problem common to ECP computation and two applications. Section 6.3 develops a method that exactly and efficiently solves the problem, and Section 6.4 analyzes the time complexity. Section 6.5 describes how to utilize the method for the applications, and Section 6.6 numerically evaluates our method with its applications. Sections 6.7 and 6.8 discuss related work and our conclusions.

6.2 Problem Statement

We use an independent failure model described in Section 4.1. However, to be suited for our purpose, we define a network reliability measure a bit different from the K-NR. We define $R_0(T, v)$ of vertices $T \subseteq V$ and vertex $v \in V$ as the probability that v is connected to at least one of the vertices in T on the subgraph of G induced by present edges. $R_0(T, v)$ is different from K-NR $R(T \cup \{v\})$ in that the former only requires that v is connected to at least one vertex in T, while the latter requires that v is connected to all the vertices in T. Note that $R_0(\{u\}, v)$ is the 2-NR of vertices u and v. Let $\mathcal{E}_{T,v} \subseteq 2^E$ be a family of subgraphs such that v is connected to at least one vertex in T. Then, $R_0(T, v)$ can be written as

$$R_0(T, v) = \sum_{E' \in \mathcal{E}_{T,v}} \Pr(E') = \sum_{E' \in \mathcal{E}_{T,v}} \left[\prod_{e \in E'} p_e \cdot \prod_{e \in E \setminus E'} (1 - p_e) \right]. \tag{6.1}$$

Note that we define $R_0(T, v) = 1$ if $v \in T$.

For our setting, we are additionally given node weight w_v for each node $v \in V$. We define the (*weighted*) *expected number of connected nodes* (*ECN*) of T, denoted by $S_0(T)$, as

$$S_0(T) \coloneqq \sum_{v \in V} w_v \cdot R_0(T, v). \tag{6.2}$$

Moreover, we can define the (*weighted*) *expected number of connected node pairs* (*ECP*) as

$$\frac{1}{2} \sum_{u,v \in V : u \neq v} w_u w_v R(\{u, v\}) = \frac{1}{2} \sum_{u \in V} \left(w_u S_0(\{u\}) - w_u^2 \right). \tag{6.3}$$

We also define a *normalized ECP* (*NECP*) as the value of ECP divided by $\frac{1}{2} \sum_{u,v \in V : u \neq v} w_u w_v$. Since the value of NECP is at most 1 regardless of n, it is convenient for comparisons with different topologies. Now the core problem we consider, called the *ECN+ problem*, can be described as follows.

Problem 6.1 (ECN+)**.** We are given each vertex's weight w_v for $v \in V$ and vertex set $T' \subseteq V$, which might be an empty set. The task is to compute $S_0(T' \cup \{u\})$ for every $u \in V \setminus T'$.

By solving ECN+ with $T' = \emptyset$, we can obtain the value of ECP by (6.3). Case $T' \neq \emptyset$ is needed for solving the optimal server placement problem, as described in Section 6.5.

6.3 Method

In the CSNR method of Chapter 5, which computes the 2-NR between a vertex and every other vertex, each network reliability value is decomposed as a sum over the partial subsets of the edges (*i-th subsets*), and bidirectional dynamic programming (DP) (*top-down* and *bottom-up* DP) are performed to compute their values. The proposed method in this chapter employs a similar approach, but we have a different DP table and an elaborated dynamic programming (DP) computations for the ECN+ problem. Hereafter, we derive mathematical formulas for solving the ECN+ problem in Sections 6.3.1 and 6.3.2. The whole procedure of our proposed algorithm is described in Section 6.3.3 and its preprocessing in Section 6.3.4. Finally, we summarize the key points of our method and briefly compare it with the existing methods in Section 6.3.5.

6.3.1 Partition and Level-wise Formula

Decomposition of $R_0(T' \cup \{u\}, v)$ over Partitions

Before considering the computation of $S_0(T' \cup \{u\})$, we return to individual $R_0(T' \cup \{u\}, v)$. A naive way to compute $R_0(T' \cup \{u\}, v)$ is to enumerate all the subsets in $\mathcal{E}_{T' \cup \{u\}, v}$. Then, $R_0(T' \cup \{u\}, v)$ can be computed by (6.1).

In a similar way as Section 5.3.2, we introduce another view for computing $R_0(T' \cup \{u\}, v)$. Let $T' + u \sim v$ be a (probabilistic) event where v is connected to u or at least one vertex in T'. By definition, $R_0(T' \cup \{u\}, v) = \Pr(T' + u \sim v)$. We also consider an event where the present edges in $E_{<i}$ are $X_{<i} \subseteq E_{<i}$. Since such events are mutually exclusive for different $X_{<i}$, we have

$$R_0(T' \cup \{u\}, v) = \sum_{X \subseteq E_{<i}} \Pr(X) \cdot \Pr(T' + u \sim v | X), \tag{6.4}$$

where the first factor is the probability that the present edges in $E_{<i}$ are just X and the second factor is the conditional probability given that the present edges in $E_{<i}$ are X.

We cannot directly compute the second factor. However, if we can define equivalence classes for the i-th subsets where conditional probability $\Pr(T' + u \sim v|X)$ is identical within the same equivalence class, we can rewrite (6.4) as

$$R_0(T' \cup \{u\}, v) = \sum_{\mathtt{n} \in \mathtt{L}_i} \Pr(\mathtt{n}) \cdot \Pr(T' + u \sim v|\mathtt{n}), \tag{6.5}$$

where $\mathtt{L}_i$ is a set of the equivalence classes of the i-th subsets, the first factor is the probability that the working edges in $E_{<i}$ are in equivalence class $\mathtt{n}$, and the second factor is the conditional probability given class $\mathtt{n}$. We eventually show we can define such equivalence class by again considering *configure* for i-th subsets.

We can say $\Pr(T' + u \sim v|X) = \Pr(T' + u \sim v|Y)$ for $X, Y \in E_{<i}$ if for every $Z \subseteq E_{\geq i}$, $X \cup Z$ and $Y \cup Z$ are equivalent in that they are included in $\mathcal{E}_{T' \cup \{u\}, v}$. Since $\mathcal{E}_{T' \cup \{u\}, v}$ only considers the connectivity among vertices, X and Y are equivalent if their connectivities among vertices are identical, i.e.,

their induced connected components are equal. Moreover, even in this case we can focus on the connectivity among frontier vertices. This is because every vertex v that does not appear in $E_{\geq i}$ can be identified with a frontier vertex v' if v is connected to v', or can be ignored if it is not connected to any frontier vertices. This introduces a partition very similar to that in Sections 4.3 and 5.3 (i.e., Definitions 4.4 and 5.2). However, to be suited for our problem where the vertices in T' have a particular roll, the definition here is different from them in that the vertices connected to at least one vertex in T' are regarded as *special.*

Definition 6.2. A *partition* for i-th subset $X \subseteq E_{<i}$ is a partitioning of frontier vertices F_i into *blocks* that consist of exactly one possibly empty *special block* and other *normal blocks.* Vertex $v \in F_i$ is in a special block if and only if v is connected to at least one vertex in T' on the subgraph induced by X. Two vertices $x, y \in F_i$, which are not in the special block, are in the same normal block if and only if they are connected on the subgraph induced by X. For partition $\mathcal{P}$, we describe the special block as $\mathsf{S}^{\mathcal{P}}$ and the block containing $v \in F_i$ as $\mathsf{B}_v^{\mathcal{P}}$.

A partition is represented as a list of blocks. The special block and each normal block are represented by curly brackets ({ }) and square brackets ([ ]) with vertex ids inside them. For example, F_3, i.e., the third frontier vertices of the graph of Figure 6.2a, are vertices 2 and 3, and the third subsets and their corresponding partitions are drawn in Figure 6.2b. Here the partitions of three third subsets $\{\}$, $\{e_1\}$, and $\{e_2\}$ are equal, $\{2\}[3]$, and their conditional probabilities are identical. Now each partition constitutes an equivalence class for the i-th subsets, and (6.5) holds where L_i is a set of partitions of i-th subsets.

Formula for ECN $S_0(T' \cup \{u\})$

Next we decompose $S_0(T' \cup \{u\}) = \sum_{v \in V} w_v R_0(T' \cup \{u\}, v)$ into factors related to the partitions and blocks explained above. By using (6.5) and exchanging the summation order, we have

$$S_0(T' \cup \{u\}) = \sum_{\mathcal{P} \in \mathsf{L}_i} \sum_{v \in V} w_v \cdot \Pr(\mathcal{P}) \cdot \Pr(T' + u \sim v | \mathcal{P}). \qquad (6.6)$$

Now we focus on an integer i such that $u \in F_i$. Note that hereafter we assume that graph G has no self-loops and that the degree of each vertex in $V \setminus T'$ is at least 2; by proper preprocessing described in Section 6.3.4, any network can satisfy this assumption. If so, there is at least one level i such that $u \in F_i$

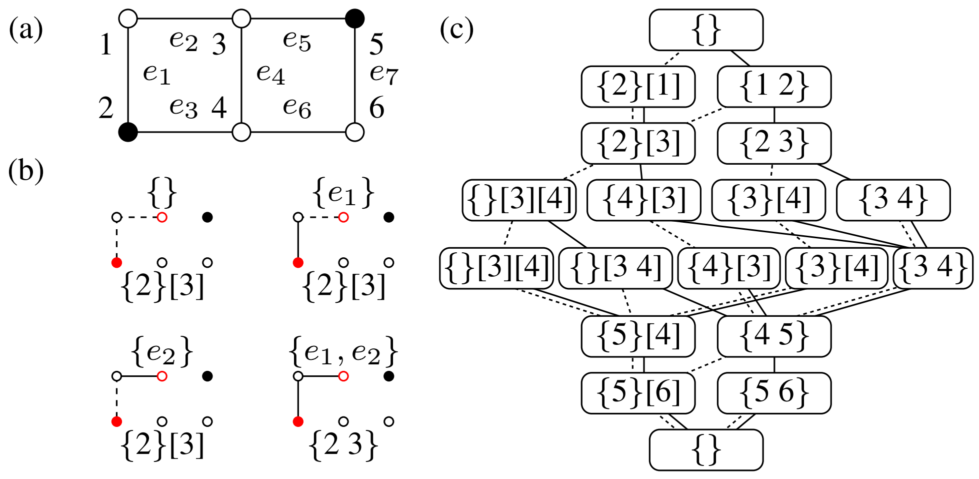


Figure 6.2: (a) Example of graph G: Filled vertices are in T'. (b) Third subsets for graph (a) and their corresponding partitions. Red vertices are in F_3. (c) All possible partitions generated from graph of (a). Solid and dashed lines indicate the connection from P to P^{HI} and P^{LO}.

in the same argument as Section 5.2. We further decompose the sum over $v \in V$ in (6.6) into that over $v \in F_i \cup B_i$ and that over $v \in V \setminus (F_i \cup B_i) = A_i$; recall that A_i and B_i are the vertices that do not appear in $E_{\geq i}$ and $E_{<i}$, respectively. By case analysis on whether $v \in F_i \cup B_i$ or $v \in A_i$, we can represent $\Pr(\mathcal{P}) \cdot \Pr(T' + u \sim v|\mathcal{P})$ with terms related to $\mathsf{B}_u^{\mathcal{P}}$, the block of P containing u. Hereafter, we write the present edges in $E' \subseteq E$ as present(E'). Moreover, we write the subgraph of G induced by $E' \subseteq E$ as $G[E']$. These notations are used for the sake of simplicity in the explanation

For $v \in B_i \cup F_i$, when $v \in F_i$, v is connected to u or some vertices in T' on $G[\text{present}(E_{<i})]$ if and only if $\mathsf{B}_v^{\mathcal{P}} = \mathsf{B}_u^{\mathcal{P}}$ or $\mathsf{B}_v^{\mathcal{P}} = \mathsf{S}^{\mathcal{P}}$. Otherwise, v may be connected to u or some vertices in T' with $G[\text{present}(E)] = G[\text{present}(E_{<i}) \cup \text{present}(E_{\geq i})]$, i.e., when present edges in $E_{\geq i}$ are added to $G[\text{present}(E_{<i})]$. This happens if and only if v is connected to one of the vertices in B_u^{P} or T' by the addition of present$(E_{\geq i})$. We write the event where v is connected to some vertices in a block B or T' as $T' + B \sim_{\geq i} v$, where the subscript means that they are connected by the addition of present$(E_{\geq i})$. Finally, we have

$$\Pr(\mathcal{P}) \cdot \Pr(T' + u \sim v|\mathcal{P}) = \Pr(\mathcal{P}) \cdot \Pr(T' + \mathsf{B}_u^{\mathcal{P}} \sim_{\geq i} v|\mathcal{P}).$$

Here we define $\Pr(T' + \mathsf{B}_u^{\mathcal{P}} \sim_{\geq i} v|\mathcal{P}) = 1$ when $v \in F_i$ and $\mathsf{B}_v^{\mathcal{P}} = \mathsf{B}_u^{\mathcal{P}}$ or $\mathsf{B}_v^{\mathcal{P}} = \mathsf{S}^{\mathcal{P}}$.

For $v \in A_i$, there are the following three cases depending on present$(E_{<i})$. First, if v is connected to some vertices in T' on $G[\text{present}(E_{<i})]$, we can say

v is connected to the special block S^P. Second, if v is not connected to any vertices in T' but connected to some vertices in F_i on $G[\text{present}(E_{<i})]$, v may eventually be connected to u or some vertices in T' depending on $G[\text{present}(E_{\geq i})]$. Otherwise, v is never connected to u or any vertices in T'. Here v is connected to at most one block of $\mathcal{P}$ on $G[\text{present}(E_{<i})]$ since if v is connected to multiple blocks, these blocks should be interconnected via v that contradicts the definition of partition. Therefore, $\Pr(\mathcal{P}) \cdot \Pr(T' + u \sim v|\mathcal{P})$ equals

$$\sum_{B \in \mathcal{P}} \Pr(\mathcal{P}) \cdot \Pr(v \sim_{<i} B|\mathcal{P}) \cdot \Pr(T' + \mathsf{B}_u^{\mathcal{P}} \sim_{\geq i} B|\mathcal{P}).$$

Here, $v \sim_{<i} B$ is the event where v is connected with block B on the subgraph $G[\text{present}(E_{<i})]$ and $T' + B \sim_{\geq i} B'$ is the event where block B' is eventually connected to block B or any vertices in T' on $G[\text{present}(E)]$, i.e., by adding $\text{present}(E_{\geq i})$ into $G[\text{present}(E_{<i})]$. To reflect the first case, we define $\Pr(T' + \mathsf{B}_u^{\mathcal{P}} \sim_{\geq i} \mathsf{S}^{\mathcal{P}}|\mathcal{P}) = 1$. Note that $\Pr(\mathcal{P}) \cdot \Pr(v \sim_{<i} B|\mathcal{P}) = \Pr(\mathcal{P}, v \sim_{<i} B)$, which is the joint probability that the partition of i-th subset $\text{present}(E_{<i})$ is $\mathcal{P}$ and $v \sim_{<i} B$.

Finally, by taking the summation over $v \in A_i$ and $v \in F_i \cup B_i$, we have the following formula.

Theorem 6.3. *We define the following values for partition $\mathcal{P}$ of an i-th subset and its blocks $B, B' \in \mathcal{P}$:*

$$\mathsf{p}_{\downarrow}^{i}(\mathcal{P}) := \Pr(\mathcal{P}), \tag{6.7}$$

$$\mathsf{q}_{\uparrow}^{i}(\mathcal{P}, B, B') := \Pr(T' + B \sim_{\geq i} B'|\mathcal{P}), \tag{6.8}$$

$$\mathsf{R}_{\downarrow}^{i}(\mathcal{P}, B) := \sum_{v \in A_i} w_v \cdot \Pr(\mathcal{P}, v \sim_{<i} B), \tag{6.9}$$

$$\mathsf{T}_{\uparrow}^{i}(\mathcal{P}, B) := \sum_{v \in F_i \cup B_i} w_v \cdot \Pr(T' + B \sim_{\geq i} v|\mathcal{P}). \tag{6.10}$$

Then $S_0(T' \cup \{u\})$ equals

$$\sum_{\mathcal{P} \in \mathsf{L}_i} \left[\sum_{B \in \mathcal{P}} \left(\mathsf{R}_{\downarrow}^{i}(\mathcal{P}, B) \cdot \mathsf{q}_{\uparrow}^{i}(\mathcal{P}, \mathsf{B}_u^{\mathcal{P}}, B) \right) + \mathsf{p}_{\downarrow}^{i}(\mathcal{P}) \cdot \mathsf{T}_{\uparrow}^{i}(\mathcal{P}, \mathsf{B}_u^{\mathcal{P}}) \right]. \tag{6.11}$$

Note that we can also obtain other metrics from (6.7)–(6.10). One can easily show that $S_0(T') = \sum_{v \in V} w_v R_0(T', v)$ and $R_0(T', v)$ for $v \in V \setminus T'$ are represented as

$$S_0(T') = \mathsf{R}_{\downarrow}^{m+1}(\mathcal{P}^{\text{si}}, \mathsf{S}^{\mathcal{P}^{\text{si}}}) \quad (\mathcal{P}^{\text{si}} \in \mathsf{L}_{m+1}), \tag{6.12}$$

$$R_0(T', v) = \sum_{\mathcal{P} \in \mathsf{L}_i} \mathsf{p}_{\downarrow}^{i}(\mathcal{P}) \cdot \mathsf{q}_{\uparrow}^{i}(\mathcal{P}, \mathsf{S}^{\mathcal{P}}, \mathsf{B}_v^{\mathcal{P}}) \quad (v \in F_i), \tag{6.13}$$

where $\mathcal{P}^{\mathrm{si}}$ is the only one $(m+1)$-st partition. These values will be used in the preprocessing; see Section 6.3.4.

6.3.2 DP Formulas

If $\mathsf{p}_{\downarrow}^{i}$, $\mathsf{q}_{\uparrow}^{i}$, $\mathsf{R}_{\downarrow}^{i}$, and $\mathsf{T}_{\uparrow}^{i}$ are computed for all i, Theorem 6.3 enables us to solve ECN+ without repetitively starting the computation from scratch for each $u \in V \setminus T'$. This is achieved by choosing level i_u such that $u \in F_{i_u}$ for every $u \in V \setminus T'$ and computing $S_0(T' \cup \{u\})$ with (6.11). To compute the $\mathsf{p}_{\downarrow}$, $\mathsf{q}_{\uparrow}$, $\mathsf{R}_{\downarrow}$, and $\mathsf{T}_{\uparrow}$ values, we first derive the relations among the partitions and their blocks of i-th and $(i+1)$-st subsets. Note that from the viewpoint of the top-down construction of diagram (Section 2.5), it is equivalent to consider the Child procedure where the configure is the partition defined in Definition 6.2. Using them, we derive DP formulas for computing all these values.

Child Partitions and Blocks

For an i-th subset X, we can consider two successive $(i+1)$-st subsets, X and $X \cup \{e_i\}$. In a similar manner, we derive two *child partitions* from a partition.

Definition 6.4. Let $\mathcal{P}$ be a partition of i-th subset $X \subseteq E_{<i}$. We define the *lo-child partition* $\mathsf{lo}(\mathcal{P})$ and *hi-child partition* $\mathsf{hi}(\mathcal{P})$ of P as the partitions of $(i+1)$-st partitions $X_{<i}$ and $X_{<i} \cup \{e_i\}$, respectively.

$\mathsf{lo}(\mathcal{P})$ and $\mathsf{hi}(\mathcal{P})$ are uniquely determined regardless of the choice of X in Definition 6.4. This is because if X and Y have identical connectivity among F_i, both (X, Y) and $(X \cup \{e_i\}, Y \cup \{e_i\})$ have identical connectivity among F_{i+1} for each pair. Note that this corresponds to the configure's condition (II) in Section 4.3.

We also define the correspondence between blocks of successive partitions in a similar manner as Chapter 5 (i.e., Definition 5.4).

Definition 6.5. Let $\mathcal{P}$ be a partition and let $B \in \mathcal{P}$ be a block. We define *lo-child block* $\mathsf{lo}(B)$ and *hi-child block* $\mathsf{hi}(B)$ as follows: For a special block, we define $\mathsf{f}(\mathsf{S}^{\mathcal{P}}) = \mathsf{S}^{\mathsf{f}(\mathcal{P})}$ for $\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}$. For normal block B, if it contains at least one vertex v in F_{i+1}, $\mathsf{f}(v) := \mathsf{B}_v^{\mathsf{f}(\mathcal{P})}$ for $\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}$, i.e., the block of $\mathsf{f}(\mathcal{P})$ containing v. Otherwise, $\mathsf{lo}(B) = \mathsf{hi}(B) := \emptyset$, i.e., no child blocks, except for the following case: If B contains one endpoint of e_i and another endpoint v' is in F_{i+1}, $\mathsf{hi}(B) := \mathsf{B}_{v'}^{\mathsf{hi}(\mathcal{P})}$.

Note that the difference comes from the existence of special block in Definition 6.2: the child blocks of special block are always special blocks.

Top-down DP

Now we focus on $\mathsf{p}_\downarrow$ and $\mathsf{R}_\downarrow$. For the first subsets, since $F_1 = \emptyset$, we have only one partition $\mathcal{P}^{\mathrm{ro}} = \{\}$ consisting of an empty special block. For this, $\mathsf{p}_\downarrow^1(\mathcal{P}^{\mathrm{ro}}) = 1$ and $\mathsf{R}_\downarrow^1(\mathcal{P}^{\mathrm{ro}}, \mathsf{S}^{\mathcal{P}^{\mathrm{ro}}}) = 0$ by definition. We compute all $\mathsf{p}_\downarrow$ and $\mathsf{R}_\downarrow$ values from $\mathsf{p}_\downarrow^1$, $\mathsf{R}_\downarrow^1$ to $\mathsf{p}_\downarrow^{m+1}$, $\mathsf{R}_\downarrow^{m+1}$, i.e., in a top-down manner, by the formulas for computing $\mathsf{p}_\downarrow^{i+1}$ and $\mathsf{R}_\downarrow^{i+1}$ from $\mathsf{p}_\downarrow^i$ and $\mathsf{R}_\downarrow^i$.

First, $\mathsf{p}_\downarrow$ can be computed in the same way as $\mathsf{DP}_\downarrow$ in Chapter 5. The partition of $(i+1)$-st subset present$(E_{<i+1})$ is $\mathcal{P}$ if and only if (i) the partition of i-th subset present$(E_{<i})$ is $\mathcal{P}'$ such that $\mathsf{lo}(\mathcal{P}') = \mathcal{P}$ and e_i is absent, or (ii) that is $\mathcal{P}'$ such that $\mathsf{hi}(\mathcal{P}') = \mathcal{P}$ and e_i is present. Thus, we have

$$\mathsf{p}_\downarrow^{i+1}(\mathcal{P}) = \sum_{\mathsf{f}\in\{\mathsf{lo},\mathsf{hi}\}} \left[\mathsf{f}(p_{e_i}) \cdot \sum_{\mathcal{P}'\in\mathsf{L}_i:\mathsf{f}(\mathcal{P}')=\mathcal{P}} \mathsf{p}_\downarrow^i(\mathcal{P}') \right], \tag{6.14}$$

where we define $\mathsf{hi}(p_e) = 1 - \mathsf{lo}(p_e) = p_e$ for $e \in E$.

Next we focus on $\mathsf{R}_\downarrow$. Given partition $\mathcal{P}$ of $(i+1)$-st subset present$(E_{<i+1})$ and its block B, we have similar formula as (6.14):

$$\Pr(\mathcal{P}, v \sim_{<i+1} B) = \sum_{\mathsf{f}\in\{\mathsf{lo},\mathsf{hi}\}} \left[\mathsf{f}(p_{e_i}) \cdot \sum_{\mathcal{P}'\in\mathsf{L}_i:\mathsf{f}(\mathcal{P}')=\mathcal{P}} \sum_{B'\in\mathcal{P}':\mathsf{f}(B')=B} \Pr(\mathcal{P}', v \sim_{<i} B') \right]. \tag{6.15}$$

Note that we here use the fact that events $\mathcal{P}, v \sim_{<i} B_1$ and $\mathcal{P}, v \sim_{<i} B_2$ for $B_1, B_2 \in \mathcal{P}$, $B_1 \neq B_2$ are mutually exclusive since v is connected to at most one block on $G[\text{present}(E_{<i})]$.

To compute $\mathsf{R}_\downarrow^{i+1}$, we decompose the sum over $v \in A_{i+1}$ in (6.9) into that over $v \in A_i$ and that over $v \in A_{i+1} \setminus A_i$. The former can be represented with $\mathsf{R}_\downarrow^i$ by using (6.15). For the latter, $\Pr(\mathcal{P}', v \sim_{<i} B') = \Pr(\mathcal{P}')$ if $B' = \mathsf{B}_v^{\mathcal{P}'}$ and 0 otherwise. Finally, we have

$$\begin{aligned}\mathsf{R}_\downarrow^{i+1}(\mathcal{P}, B) = &\sum_{\mathsf{f}\in\{\mathsf{lo},\mathsf{hi}\}} \left[\mathsf{f}(p_{e_i}) \cdot \sum_{\mathcal{P}'\in\mathsf{L}_i:\mathsf{f}(\mathcal{P}')=\mathcal{P}} \sum_{B'\in\mathcal{P}':\mathsf{f}(B')=B} \mathsf{R}_\downarrow^i(\mathcal{P}', B') \right] \\ &+ \sum_{v\in A_{i+1}\setminus A_i} \sum_{\mathsf{f}\in\{\mathsf{lo},\mathsf{hi}\}} \left[w_v \cdot \mathsf{f}(p_{e_i}) \cdot \sum_{\mathcal{P}'\in\mathsf{L}_i:\mathsf{f}(\mathcal{P}')=\mathcal{P},\mathsf{f}(\mathsf{B}_v^{\mathcal{P}'})=B} \mathsf{p}_\downarrow^i(\mathcal{P}') \right],\end{aligned} \tag{6.16}$$

where the first and second terms are derived from the former and the latter. Note that there are corner cases where $v \in A_{i+1} \setminus A_i$ has only degree 1. In such cases, since $v \notin F_i$, we cannot define block $\mathsf{B}_v^{\mathcal{P}'}$ of partition $\mathcal{P}'$. Here, since $v \in T'$ by assumption, we can define $\mathsf{f}(\mathsf{B}_v^{\mathcal{P}'}) = \mathsf{S}^{\mathsf{f}(\mathcal{P}')}$ for computing the second term of (6.16).

Bottom-up DP

Next we focus on $\mathsf{q}_\uparrow$ and $\mathsf{T}_\uparrow$. For the $(m+1)$-st subsets, since $F_{m+1} = \emptyset$, we have only one partition $\mathcal{P}^{\text{si}} = \{\}$ consisting of only an empty special block. For this, $\mathsf{q}_\uparrow^{m+1}(\mathcal{P}^{\text{si}}, \mathsf{S}^{\mathcal{P}^{\text{si}}}, \mathsf{S}^{\mathcal{P}^{\text{si}}}) = 1$ and $\mathsf{T}_\uparrow^{m+1}(\mathcal{P}^{\text{si}}, \mathsf{S}^{\mathcal{P}^{\text{si}}}) = 0$ by definition. Our goal now is to compute all $\mathsf{q}_\uparrow$ and $\mathsf{T}_\uparrow$ values from $\mathsf{q}_\uparrow^{m+1}$, $\mathsf{T}_\uparrow^{m+1}$ to $\mathsf{q}_\uparrow^1$, $\mathsf{T}_\uparrow^1$, i.e., in a bottom-up manner.

We first consider the computation of $\mathsf{q}_\uparrow^i$ from $\mathsf{q}_\uparrow^{i+1}$. Given two blocks $B, B' \in \mathcal{P}$ where $\mathcal{P}$ is a partition of an i-th subset, basically $T' + B \sim_{\geq i} B'$ if (i) e_i is absent and $T' + \mathsf{lo}(B) \sim_{\geq i+1} \mathsf{lo}(B')$, or (ii) e_i is present and $T' + \mathsf{hi}(B) \sim_{\geq i+1} \mathsf{hi}(B')$. By cooperating with the trivial case that $B' = B$ or $B' = \mathsf{S}^{\mathcal{P}}$ (in such cases $T' + B \sim_{\geq i} B'$ must hold),

$$\mathsf{q}_\uparrow^i(\mathcal{P}, B, B') = \begin{cases} 1 & (B' \in \{B, \mathsf{S}^{\mathcal{P}}\}) \\ \sum_{\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}} \mathsf{f}(p_{e_i}) \cdot \mathsf{q}_\uparrow^{i+1}(\mathsf{f}(\mathcal{P}), \mathsf{f}(B), \mathsf{f}(B')) & (\text{otherwise}). \end{cases} \tag{6.17}$$

In the second case, we define $\mathsf{q}_\uparrow^i(\mathcal{P}, \emptyset, B') = \mathsf{q}_\uparrow^i(\mathcal{P}, \mathsf{S}^{\mathcal{P}}, B')$ for any $B' \in \mathcal{P}$ and $\mathsf{q}_\uparrow^i(\mathcal{P}, B, \emptyset) = 0$ for any $B \in \mathcal{P} \cup \{\emptyset\}$; such values can easily be derived from the definition (6.8).

Note that, as with the case with the computation of $\mathsf{DP}_\uparrow$ in Chapter 5, there is a corner case for computing $\mathsf{q}_\uparrow^i$. When e_i is present, e_i connects two different normal blocks B and B', and $\mathsf{hi}(B)$ and $\mathsf{hi}(B')$ are both $\emptyset$, B and B' are eventually connected on $G[\text{present}(E)]$, but this case is not considered in (6.17). In such a case, we set $\mathsf{q}_\uparrow^i(\mathcal{P}, B, B') = p_{e_i}$.

Finally, we derive the formula for $\mathsf{T}_\uparrow$. We can have similar analyses as $\mathsf{q}_\uparrow$ for $\Pr(T' + B \sim_{\geq i} v | \mathcal{P})$:

$$\Pr(T' + B \sim_{\geq i} v | \mathcal{P}) = \sum_{\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}} \mathsf{f}(p_{e_i}) \cdot \Pr(T' + \mathsf{f}(B) \sim_{\geq i+1} v | \mathsf{f}(\mathcal{P})). \tag{6.18}$$

To compute $\mathsf{T}_\uparrow^i$ from $\mathsf{T}_\uparrow^{i+1}$, we decompose the sum over $v \in F_i \cup B_i$ in (6.10) into that over $v \in F_{i+1} \cup B_{i+1}$ and $v \in (F_i \cup B_i) \setminus (F_{i+1} \cup B_{i+1}) = A_{i+1} \setminus A_i$. The former can be represented with $\mathsf{T}_\uparrow^{i+1}$ values by using (6.18). For the latter,

$$\Pr(T' + B \sim_{\geq i} v | \mathcal{P}) = \Pr(T' + B \sim_{\geq i} \mathsf{B}_v^{\mathcal{P}} | \mathcal{P}) = \mathsf{q}_\uparrow^i(\mathcal{P}, B, \mathsf{B}_v^{\mathcal{P}}).$$

Finally, we have

$$\mathsf{T}_\uparrow^i(\mathcal{P}, B) = \sum_{\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}} \mathsf{f}(p_{e_i}) \cdot \mathsf{T}_\uparrow^{i+1}(\mathsf{f}(\mathcal{P}), \mathsf{f}(B)) + \sum_{v \in A_{i+1} \setminus A_i} w_v \cdot \mathsf{q}_\uparrow^i(\mathcal{P}, B, \mathsf{B}_v^{\mathcal{P}}). \tag{6.19}$$

Algorithm 6.1: Procedures for constructing diagram of partitions

```
procedure $ECNP_{T'}$.Root():
    return $\langle 1, \mathcal{P}^{\mathrm{ro}} := \{\} \rangle$
procedure $ECNP_{T'}$.Child($\langle i, \mathcal{P} \rangle, f$):
    foreach $x \in \{v, v'\} \setminus F_i$ do                                   // $e_i = \{v, v'\}$
        if $x \in T'$ then Insert $x$ into $\mathsf{S}^{\mathcal{P}}$, the special block of $\mathcal{P}$
        else Insert $[x]$ as a new normal block of $\mathcal{P}$
    if $f = \mathsf{hi}$ then
        Merge $\mathsf{B}_v^{\mathcal{P}}$ and $\mathsf{B}_{v'}^{\mathcal{P}}$; if either is a special block, the merged one is also special
    foreach $x \in \{v, v'\} \setminus F_{i+1}$ do
        Remove $x$ from $\mathsf{B}_x^{\mathcal{P}}$
    Remove all empty normal blocks from $P^f$
    return $\langle i + 1, \mathcal{P} \rangle$
```

Note that in computing the first term of (6.19), we set $\mathsf{T}_{\uparrow}^{i}(\mathcal{P}, \emptyset) = \mathsf{T}_{\uparrow}^{i}(\mathcal{P}, \mathsf{S}^{\mathcal{P}})$ by considering the definition (6.10). As with $\mathsf{R}_{\downarrow}$, if $v \in A_{i+1} \setminus A_i$ has only degree 1, we define $\mathsf{B}_v^{\mathcal{P}} = \mathsf{S}^{\mathcal{P}}$ for computing the second term of (6.19).

6.3.3 Procedures of Algorithm

To compute the $\mathsf{p}_{\downarrow}$, $\mathsf{q}_{\uparrow}$, $\mathsf{R}_{\downarrow}$, and $\mathsf{T}_{\uparrow}$ values by (6.14)–(6.19), we should know set L_i of the partitions of i-th subsets for every i. We should also compute the child partitions and blocks for every partition. However, it is not known in advance. Therefore, we enumerate all possible partitions from a given graph. Thanks to the similarity in the definition of partitions, we can achieve these computations with the diagram construction framework described in Section 2.5 although the resultant diagram is neither a BDD nor a ZDD. We describe the procedures for the diagram construction in Algorithm 6.1. Since similar procedures are employed for the HH and CSNR methods (Algorithms 4.1 and 5.1), we omit these details. For example, Figure 6.2c depicts the diagram of partitions generated from the graph of Figure 6.2a. Here the correspondences between partitions are also drawn.

After that, the $\mathsf{p}_{\downarrow}$ and $\mathsf{R}_{\downarrow}$ values are computed from $i = 1$ to $m + 1$ by (6.14) and (6.16), the $\mathsf{q}_{\uparrow}$ and $\mathsf{T}_{\uparrow}$ values are computed from $i = m + 1$ to 1 by (6.17) and (6.19), and then every $S_0(T' \cup \{u\})$ value is computed by (6.11).

6.3.4 Preprocessing

In Section 6.3.1, we have assumed that the graph has no self-loops and the degree of every vertex in $V \setminus T'$ is at least 2. Here we deal with the case where self-loops or such vertices exist. First, since the self-loops do not affect the connectivity among vertices, we can safely remove them from G.

Now we consider the case where $x \in V \setminus T'$ has degree 1. Let $e_x = \{x, y\}$ be the only edge incident to x, and let $G - x$ be the graph obtained by removing x from G. We can represent $R_0^G(T' \cup \{u\}, v)$, R_0 on graph G, with p_{e_x} and $R_0^{G-x}(\cdot,\cdot)$, R_0 on $G - x$, by the analysis like Moskowitz [1958]:

$$R_0^G(T' \cup \{u\}, v) = \begin{cases} 1 & (u = v = x) \\ p_{e_x} \cdot R_0^{G-x}(T' \cup \{y\}, v) \\ \quad + (1 - p_{e_x}) R_0^{G-x}(T', v) & (v \neq u = x) \\ p_{e_x} \cdot R_0^{G-x}(T' \cup \{u\}, y) & (u \neq v = x) \\ R_0^{G-x}(T' \cup \{u\}, v) & (\text{otherwise}). \end{cases}$$

This enables us to represent $S_0(T' \cup \{u\})$ on G, denoted by $S_0^G(T' \cup \{u\})$, with the values of R_0^{G-x}. That is,

$$\begin{aligned} S_0^G(T' \cup \{u\}) &= w_x \cdot R_0^G(T' \cup \{u\}, x) + \sum_{v \in V \setminus \{x\}} w_v \cdot R_0^G(T' \cup \{u\}, v) \\ &= \begin{cases} w_x + p_{e_x} \cdot \displaystyle\sum_{v \in V \setminus \{x\}} w_v \cdot R_0^{G-x}(T' \cup \{y\}, v) & (u = x) \\ w_x p_{e_x} \cdot R_0^{G-x}(T' \cup \{u\}, y) \\ \quad + \displaystyle\sum_{v \in V \setminus \{x\}} w_v \cdot R_0^{G-x}(T' \cup \{u\}, v) & (u \neq x). \end{cases} \end{aligned}$$

Using this, $S_0^G(T' \cup \{u\})$ can be computed by solving ECN+ on $G - x$ as follows: We define new vertex weight values for $v \in V \setminus \{x\}$ as

$$w'_v = \begin{cases} w_y + w_x p_{e_x} & (v = y) \\ w_v & (v \neq y), \end{cases} \tag{6.20}$$

and compute $(S')_0^{G-x}(T' \cup \{u\}) = \sum_{v \in V \setminus \{x\}} w'_v R_0^{G-x}(T' \cup \{u\}, v)$ for every $u \in (V \setminus T') \setminus \{x\}$. Then we can prove that $S_0^G(T' \cup \{u\})$ equals $(S')_0^{G-x}(T' \cup \{u\})$ for $u \in (V \setminus T') \setminus \{x\}$, and $S_0^G(T' \cup \{x\})$ equals

$$\begin{aligned} &w_x[1 - p_{e_x}^2 - p_{e_x}(1 - p_{e_x}) R_{G-x}(T', y)] \\ &\qquad + p_{e_x} \cdot (S')_0^{G-x}(T' \cup \{y\}) + (1 - p_{e_x})(S')_0^{G-x}(T'). \end{aligned} \tag{6.21}$$

Note that $(S')_0^{G-x}(T')$ and $R_0^{G-x}(T', y)$ can be obtained from (6.12) and (6.13).

There are cases such that $(V \setminus T') \setminus \{x\}$ still contains vertices with degree 1. If so, we recursively remove such a vertex and set new weights as (6.20). After recursion stops, we can use our proposed algorithm. Finally, the answer can be recovered by recursively applying (6.21).

6.3.5 Summary of Our Proposed Method

We summarize the key points of our method. Our method relies on excellent ideas from two methods: the subset equivalency based on connectivity among frontier vertices (HH method in [Hardy *et al.*, 2007]), and the i-th subsets used to compute the reliability for every vertex (CSNR method in Chapter 5). However, the proposed method's design is completely new to introduce our new building blocks, which are the accumulated probabilities ($\mathsf{R}_\downarrow$ and $\mathsf{T}_\uparrow$) and related DP formulas (6.16) and (6.19). They enable us to compute *ECN* for *every vertex.* To incorporate them, we entirely renovated the whole procedure, e.g., the decomposition formula (6.11) and other DP computations.

6.4 Complexity

Here we analyze the time complexity of our algorithm. As in previous chapters, let $W_F = \max_i |F_i|$ be the maximum number of frontier vertices. Before considering the time complexity, we bound the maximum number $M = \max_i |\mathsf{L}_i|$ of partitions of i-th subsets.

Lemma 6.6. *When $T' \neq \emptyset$, $M = O(W_F E_{W_F})$, where E_{W_F} is the W_F-th Bell number. When $T' = \emptyset$, $M = O(B_{W_F})$.*

Proof. From the definition, the i-th partition is a partitioning of F_i into a special block and normal blocks. The number of partitioning of F_i is at most $B_{|F_i|} \leq E_{W_F}$. For each, the number of candidates for the special block is at most $W_F + 1$ (including the case the special block is empty). Thus, M is bounded by $O(W_F E_{W_F})$.

When $T' = \emptyset$, the special block is always empty, and thus the i-th partition is truly a partitioning of F_i into normal blocks. Thus, in such a case, M is bounded by E_{W_F}. □

Now we bound the running time of our algorithm.

Theorem 6.7. *Our proposed method runs in $O(mW_F^2 M)$ time.*

Proof. With the maintenance of partitions like the existing method [Hardy *et al.*, 2007], we can enumerate each partition in $O(W_F)$ time. Since at most $O(mM)$ partitions are generated in total, the overall cost is bounded by $O(mW_F M)$. For each $\mathcal{P}' \in \mathsf{L}_i$, $\mathsf{p}_\downarrow^i(\mathcal{P}')$ appears two times in total on the right-hand side of (6.14) (when computing $\mathsf{p}_\downarrow^{i+1}(\mathsf{lo}(\mathcal{P}'))$ and $\mathsf{p}_\downarrow^{i+1}(\mathsf{hi}(\mathcal{P}'))$). Since there are $O(mM)$ values for $\mathsf{p}_\downarrow$, (6.14) costs $O(mM)$ time in total. For computing $\mathsf{R}_\downarrow$, there are $O(mW_FM)$ values to compute since each partition has at most W_F+1 blocks. For each $\mathcal{P}' \in \mathsf{L}_i$ and $B' \in \mathcal{P}'$, $\mathsf{R}^i_\downarrow(\mathcal{P}', B')$ appears at most two times in the first term of (6.16). Thus, the first term of (6.17) costs $O(mW_FM)$ time in total. The second term costs $O(nM)$ time in total since for every $v \in V$, it costs at most $O(M)$ time in total. Since G is connected and thus $n = O(m)$, the overall cost for computing $\mathsf{R}_\downarrow$ is bounded by $O(mW_FM)$. Computing $\mathsf{q}_\uparrow$ and $\mathsf{T}_\uparrow$ costs $O(mW_F^2M)$ and $O(mW_FM)$, since there are $O(mW_F^2M)$ and $O(mW_FM)$ values to compute, and each value can be computed in constant time by (6.17) and (6.19). Finally, (6.11) can be computed in $O(W_FM)$ time for each $u \in V$, and thus $O(nW_FM) = O(mW_FM)$ time in total. Thus, the overall time complexity is $O(mW_F^2M)$. □

Corollary 6.8. *Our proposed algorithm runs in $O(mW_F^3E_{W_F})$ time. When $T' = \emptyset$, it runs in $O(mW_F^2E_{W_F})$ time.*

We now compare it with the existing method. We consider four problems: 2-NR, which computes single $R(\{u,v\})$ for given $u, v \in V$; CSNR problem with $|T| = 1$, which computes $R(\{u,v\})$ for given $v \in V$ and all $u \in V$; ECN+ problem with $T' = \emptyset$; and ECN+ problem with $T' \neq \emptyset$. Table 6.1 compares the complexity. Since computing an ECP value can be done by solving ECN+ problem with $T' = \emptyset$, we also consider such a problem denoted as "ECP" in Table 6.1. The proposed method in this chapter can solve ECN+ with $T' = \emptyset$, i.e., ECP, in $O(n^2W_F)$ times faster than the HH method [Hardy *et al.*, 2007] and $O(n)$ times faster than the CSNR method (Chapter 5).

Note that by using the HH method and the operations on BDDs, we can solve the ECN+ problem with $T' \neq \emptyset$ as follows. First, we fix $v \in V \setminus T'$ and compute $\mathtt{B}_{\{t'\},v}$, which is the BDD that represents $\mathcal{E}_{\{t'\},v}$, by the HH method for each $t' \in T'$. Second, we compute $\mathtt{B}_{T',v} := \bigvee_{t' \in T'} \mathtt{B}_{\{t'\},v}$, which is the disjunction (logical OR) of all $\mathtt{B}_{\{t'\},v}$, by Apply algorithm [Bryant, 1986] on BDDs. Here BDD $\mathtt{B}_{T'v}$ represents $\mathcal{E}_{T',v}$. Finally, for every $u \in V \setminus T'$, $R_0(T' \cup \{u\}, v)$ is computed by building BDD $\mathtt{B}_{\{u\},v}$ representing $\mathcal{E}_{\{u\},v}$ by the HH method, building $\mathtt{B}_{T',v} \vee \mathtt{B}_{\{u\},v}$ representing $\mathcal{E}_{T' \cup \{u\},v}$ by the Apply algorithm, and performing DP on this BDD. We call this the HH+ method. Unfortunately, analyzing its time complexity is difficult, denoted by (unknown) in Table 6.1, since the HH+ method involves BDD operations, whose time complexity heavily depends on the given data.

As described in Section 4.3.2, the frontier width W_F is closely related to the path-width W_p and the path decomposition.

Table 6.1: Comparison of time complexity.

	2-NR	CSNR ($\lvert T\rvert=1$)	ECN+ ($T'=\emptyset$) ECP	ECN+ ($T'\neq\emptyset$)
HH	$O(mW_F^3E_{W_F})$	$O(mnW_F^3E_{W_F})$	$O(mn^2W_F^3E_{W_F})$	(unknown)
CSNR	$O(mW_F^2E_{W_F})$	$O(mW_F^2E_{W_F})$	$O(mnW_F^2E_{W_F})$	-
Proposed	$O(mW_F^2E_{W_F})$	$O(mW_F^2E_{W_F})$	$O(mW_F^2E_{W_F})$	$O(mW_F^3E_{W_F})$

6.5 Applications

6.5.1 Critical Link Identification

This subsection defines three criteria for measuring link criticality with respect to ECP. The *essentiality* of edge e represents the impact of e's failure on a reliability measure R_* [Kuo *et al.*, 2007]. The general definition is $M_{R_*} - R_*|_{\neg e}$, where M_{R_*} is the possible maximum value of R_* for the topology and $R_*|_{\neg e}$ is the value of R_* given that edge e is absent. Thus, the essentiality of edge e with respect to ECP is defined as $C_e^{\mathrm{ECP}}(e) := \frac{1}{2}\sum_{u\neq v} w_u w_v - \mathrm{ECP}|_{\neg e}$. In the following, we show the computation method and time complexity of computing $C_e^{\mathrm{ECP}}(e)$ values for all $e \in E$.

Theorem 6.9. *The values of $C_e^{\mathrm{ECP}}(e)$ for all $e \in E$ can be computed in $O(mW_F^2E_{W_F})$ time, which is identical with just computing the ECP value.*

We only describe how to compute $\mathrm{ECP}|_{\neg e}$. Seeing $R_0(T, v)$ in (6.1) as a function of p_e by fixing all other $p_{e'}$ values as constants, it becomes just a linear function of p_e, and so are $S_0(T)$ and ECP. Thus, ECP can be represented as $\frac{\partial \mathrm{ECP}}{\partial p_e} \cdot p_e + \mathrm{ECP}|_{\neg e}$, and we have $\mathrm{ECP}|_{\neg e} = \mathrm{ECP} - \frac{\partial \mathrm{ECP}}{\partial p_e} \cdot p_e$ for every $e \in E$. Next, we discuss $\frac{\partial \mathrm{ECP}}{\partial p_e}$. Since our algorithm never performs non-differentiable operations (such as taking max) in the DP computation, we can use automatic differentiation [Griewank and Walther, 2008], which automatically constructs a procedure for computing derivatives from an algorithm, to get $\frac{\partial \mathrm{ECP}}{\partial p_e}$ values for every $e \in E$: that is, the partial differential coefficient of ECP with respect to p_e. Since ECP is scalar-valued, automatic differentiation can be performed in the same complexity as computing ECP due to the cheap gradient principle [Griewank and Walther, 2008].

The other criticality metrics are defined as follows [Inoue, 2019]. The *augmentability* of e, which indicates the impact of edge augmentation on ECP, is given as $C_a^{\mathrm{ECP}}(e) := \mathrm{ECP}|_e$, where $\mathrm{ECP}|_e$ is the ECP value given that e works, which can be computed by $\mathrm{ECP}+\frac{\partial \mathrm{ECP}}{\partial p_e}(1-p_e)$. The *contribution* of e to ECP is $C_c^{\mathrm{ECP}}(e) := \frac{p_e \cdot \mathrm{ECP}|_e}{\mathrm{ECP}}$. These criticality metrics for every $e \in E$ can also be computed in $O(mW^2 E_{W_F})$ time.

6.5.2 Optimal Server Placement

Although there are many criteria for assessing the reliability of server placement in a CS infrastructure, the most fundamental requirement for each user is the connectivity to servers, i.e., the existence of a path between a user and any server. In this view, the reliability of server placement under probabilistic CS network can be measured by the expected number of connected users, i.e., the ECN. Therefore, we define the server placement problem to determine where a new server is placed for maximizing the reliability for the new set of servers.

Problem 6.10. Given $T' \subseteq V$, the *server placement problem* is to compute $u \in V \setminus T'$ that maximizes $S_0(T' \cup \{u\})$.

The server placement problem can be solved as an ECN+ problem by choosing vertex u that maximizes $S_0(T' \cup \{u\})$. Due to Corollary 6.8, when the first server is added to the infrastructure, the time complexity is as follows.

Theorem 6.11. *The server placement problem of $T' = \emptyset$ can be solved in $O(mW_F^2 E_{W_F})$ time.*

When some servers are already placed and we want to add another one, the time complexity is as follows.

Theorem 6.12. *The server placement problem of $T' \neq \emptyset$ can be solved in $O(mW_F^3 E_{W_F})$ time.*

6.6 Experiments

We numerically evaluated our method with two applications. Our proposed method and the existing methods were implemented in C++11 and compiled by g++-4.8.5 with `-O3` option. Experiments were conducted on a single thread of a Linux machine with AMD EPYC 7763 2.45 GHz CPU and 2048 GB RAM. We set the time limit of every run to 1 hour. We set all vertex weights w_v to 1 throughout this section.

6.6.1 Elapsed Time for ECN+ ($T' = \emptyset$) and ECP

We first compared our method, the CSNR method, and the HH method with respect to the elapsed time for solving ECN+ with $T' = \emptyset$, i.e., computing ECP. As described later, for real topology graphs, we prepared two implementations for the HH method, HH(1) and HH(2), differing in how they determine edge ordering. All methods involved the preprocessing of recursively removing degree 1 vertices. Additionally, for the HH methods, since each $R_0(u, v)$ is computed separately, we also conducted the degree 2 vertex elimination in Section 4.2.1 other than u and v for every u, v pair, which may drastically decrease the computation time for the HH method. We implemented the HH method with TdZdd (`https://github.com/kunisura/TdZdd`).

We used all the topology graphs used in Chapter 5 as test instances, including synthetic graphs and real communication networks. Grid-wxh is a grid graph with $w \times h$ vertices. The real networks are from the Internet Topology Zoo [Knight *et al.*, 2011] and Rocketfuel [Spring *et al.*, 2004] datasets. For them, we extracted the largest connected component and removed all the self-loops. For the grid graphs, we used the edge ordering of [Iwashita *et al.*, 2013], which is known to be better for the DP on grid graphs, for all the methods: Ours, CSNR, and HH(1). For the real networks, we used beam-search based heuristics [Inoue and Minato, 2016] to determine the edge ordering, and its consumed time is included in the computational time. Ours, CSNR, and HH(1) computed the edge ordering only once after removing the degree 1 vertices, while HH(2) computed it for every u, v pair; HH(2) optimized the edge ordering for every u, v pair, but it repeatedly executed the time-consuming beam-search. Each p_e was chosen uniformly at random from $[0.9, 0.95]$ according to the literature [Elshqeirat *et al.*, 2015; Botev *et al.*, 2012; Xiao *et al.*, 2009; Nishino *et al.*, 2018; Inoue, 2019] because original data do not include them.

Table 6.2 shows the result. Here we also show the number of vertices (n), the number of edges (m), and the frontier width of the original graph (W_F) computed by beam-search [Inoue and Minato, 2016]. The proposed method computed ECP for all the graphs within an hour, and the other methods could not. Even for graphs solvable by the existing methods, our method is about 4–400 times faster than the CSNR method and about 70–4000 times faster than the HH method. This clearly indicates the extreme efficiency of our method. In addition, our method runs faster for the graphs with smaller W_F value. This reflects the complexity analysis of Section 6.4. Note that peak memory usage, 160.7 GB, of the proposed method was recorded for Rocketfuel-3257, where CSNR, HH(1), and HH(2) used 182.9, 151.5, and 218.1 GB (resp.) when time expired.

Table 6.2: Computed values of NECP and All-NR, and elapsed times for solving ECN+ with $T' = \emptyset$.

				Computed value		$T' = \emptyset$ (sec.)			
Instance	n	m	W_F	NECP	All-NR	Proposed	CSNR	HH(1)	HH(2)
Grid-7x14	98	175	7	0.9989	0.9543	**0.08**	9.34	554.79	-
Grid-7x28	196	357	7	0.9993	0.9391	**0.18**	47.57	>1h	-
Grid-7x42	294	539	7	0.9994	0.9254	**0.27**	113.39	>1h	-
Grid-8x8	64	112	8	0.9988	0.9670	**0.17**	12.50	468.14	-
Grid-10x10	100	180	10	0.9992	0.9651	**4.74**	677.95	>1h	-
Grid-12x12	144	264	12	0.9992	0.9485	**149.28**	>1h	>1h	-
Interoute	110	146	7	0.9572	0.2922	**0.10**	0.43	7.27	118.72
TataNld	145	186	7	0.9475	0.2027	**0.14**	0.67	19.39	290.54
Kdl	754	895	12	0.8759	3.365e-6	**8.61**	>1h	>1h	>1h
Rocketfuel-1221	318	758	12	0.9072	9.581e-6	**11.81**	2358.27	>1h	>1h
Rocketfuel-1755	172	381	12	0.9714	0.1143	**20.39**	>1h	>1h	>1h
Rocketfuel-3257	240	404	14	0.9038	2.116e-4	**659.20**	>1h	>1h	>1h
Rocketfuel-3967	201	434	16	0.9649	3.828e-2	**494.95**	>1h	>1h	>1h
Rocketfuel-6461	182	294	10	0.9147	9.032e-4	**0.45**	54.27	3160.04	1758.24

Table 6.3: Elapsed times for solving ECN+ with $T' \neq \emptyset$.

				$T' \neq \emptyset$ (sec.)							
				$k=2$ ($\|T'\|=1$)		$k=3$ ($\|T'\|=2$)		$k=4$ ($\|T'\|=3$)		$k=5$ ($\|T'\|=4$)	
Instance	n	m	W	Proposed	HH+	Proposed	HH+	Proposed	HH+	Proposed	HH+
Grid-7x14	98	175	7	**0.21**	2078.82	**0.35**	2267.53	**0.31**	2228.58	**0.31**	2223.23
Grid-7x28	196	357	7	**0.67**	>1h	**0.65**	>1h	**0.96**	>1h	**0.96**	>1h
Grid-7x42	294	539	7	**0.65**	>1h	**1.23**	>1h	**1.19**	>1h	**1.19**	>1h
Grid-8x8	64	112	8	**0.21**	1754.92	**0.58**	1847.40	**0.57**	1816.48	**0.57**	1848.01
Grid-10x10	100	180	10	**13.53**	>1h	**10.16**	>1h	**22.05**	>1h	**23.84**	>1h
Grid-12x12	144	264	12	**214.41**	>1h	**713.55**	>1h	**707.97**	>1h	**709.16**	>1h
Interoute	110	146	7	**0.10**	71.35	**0.10**	71.22	**0.11**	77.42	**0.11**	78.44
TataNld	145	186	7	**0.15**	184.80	**0.15**	184.89	**0.15**	142.12	**0.15**	141.03
Kdl	754	895	12	**9.23**	>1h	**9.18**	>1h	**11.83**	>1h	**10.91**	>1h
Rocketfuel-1221	318	758	12	**12.53**	>1h	**17.50**	>1h	**17.63**	>1h	**38.54**	>1h
Rocketfuel-1755	172	381	12	**22.47**	>1h	**26.02**	>1h	**90.92**	>1h	**92.08**	>1h
Rocketfuel-3257	240	404	14	**683.25**	>1h	**684.78**	>1h	**683.86**	>1h	**684.19**	>1h
Rocketfuel-3967	201	434	16	**2223.59**	>1h	**796.63**	>1h	**2292.83**	>1h	**2293.51**	>1h
Rocketfuel-6461	182	294	10	**1.41**	>1h	**1.53**	>1h	**1.51**	>1h	**1.51**	>1h

6.6.2 Elapsed Time for ECN+ ($T' \neq \emptyset$) with Server Placement

We next measured the elapsed time for solving ECN+ when $T' \neq \emptyset$. We compared our method, the factoring approach of AboElFotoh et al. [AboElFotoh *et al.*, 2005], and the HH+ method described in Section 6.4. We implemented the Apply algorithm in the HH+ method with SAPPOROBDD (`https://github.com/Shin-ichi-Minato/SAPPOROBDD`).

We used the same graphs, the edge ordering, and the p_e values as Section 6.6.1. Assume a scenario where servers are added to a CS infrastructure one by one (Section 6.5.2). The server set after k-th step ($k = 1, 2, \ldots$) was denoted by T'_k, and we defined $T'_0 = \emptyset$. At the k-th step, we selected the optimal vertex $u^* \in V \setminus T'_{k-1}$ by solving the server placement problem maximizing $S_0(T'_{k-1} \cup \{u\})$, and set $T'_k = T'_{k-1} \cup \{u^*\}$. We measured the elapsed time to determine T'_k as shown in Table 6.3.

First, since the factoring approach [AboElFotoh *et al.*, 2005] could not solve any instances within an hour, we omitted it from Table 6.3. Our method again solved all the instances within an hour, and the other method could not. Our method is 70–10,000 times faster than the HH+ method even for the instances solved by it. Compared to case $T' = \emptyset$, our method is up to just six times slower. This reflects the complexity result of Corollary 6.8 where our method runs $O(W_F)$ times slower when $T' \neq \emptyset$. Note that peak memory usage, 407.0 GB, of the proposed method was recorded for Rocketfuel-3967 and $k = 5$ ($|T'| = 4$).

6.6.3 ECP Values and Critical Links

Finally, we compared the ECP values computed in Section 6.6.1 and the All-NR values. In addition, we computed the link criticality with respect to ECP and compared it with other criticality measures.

Table 6.2 shows the NECP and All-NR values. Here we used NECP because we compared it with different topologies. There are some topology pairs such that All-NR is larger for one topology and NECP is larger for another (e.g., Grid-7x14 and Grid-7x42, or TataNld and Rocketfuel-1755). This indicates that although computing All-NR is easier than ECP, All-NR cannot be an alternative measure for ECP. We also observe that the All-NR values are too small for some real topologies. This is because these topologies have some pendant edges, i.e., the edges connected to degree 1 vertices. Such edges have less impact on NECP since their failure only disconnects one vertex from the other.

For the criticality, we considered a situation where we wanted to select a few links to be reinforced to improve ECP. For this purpose, ECP augmentability C_a^{ECP} is appropriate since it indicates the impact of edge augmentation on ECP. To observe the difference of the ECP augmentability and other criticality measures, we selected three real networks, Interoute, TataNld, and Rocketfuel-6461, computed their criticality measures, and selected top-5 or top-10 critical edges with respect to these criticality measures. Other criticality measures included All-NR augmentability [Inoue, 2019] (denoted as C_a^{NR}) and current-flow betweenness centrality [Brandes and Fleischer, 2005] (denoted as C^{CBC}). Intuitively, C_a^{NR} indicates the impact of edge augmentation on the probability that all the vertices are connected, and C^{CBC} indicates the amount of traffic of links when all the vertex pairs are communicated. To implement automatic differentiation, we used Adept [Hogan, 2014].

Figure 6.3 depicts the result. The edges selected by C_a^{NR} and C^{CBC} are generally different from those selected by C_a^{ECP}. To improve ECP, the following two factors should be considered: (a) the number of vertex pairs whose paths pass through this edge, and (b) the absence of alternative reliable paths when this edge fails. C_a^{NR} fails to consider (a) since All-NR only considers whether all vertices are connected, and so typically C_a^{NR} chooses pendant edges. C^{CBC} fails to consider (b), and so typically C^{CBC} chooses some "center" edges that have many alternative reliable paths. C_a^{ECP} can consider both factors: For example, in Interoute (left column of Figure 6.3), the rightmost part is only connected by red edges, and thus these are selected as critical by C_a^{ECP}, although other critical measures do not select them. In Rocketfuel-6461 (right column of Figure 6.3), the leftmost part is only connected by the red "bridge" edge and thus it is selected as critical by C_a^{ECP}.

Note that during the computation of C_a^{ECP} for every edge of these graphs, less than triple computational time elapsed compared with just computing the ECP value.

6.7 Related Work

Network reliability evaluation has been studied for decades, as described in Section 4.2.1. However, evaluation methods for $R_0(T, v)$ in (6.1) with $|T| > 1$ have rarely been investigated. To the best of our knowledge, only AboElFotoh *et al.* [2005] studied it, although they relied on an inefficient factoring approach, as shown in the experiments.

Several papers have studied network behaviors under severe events. Researchers in the field of complex networks revealed that scale-free networks, including network infrastructures, are quite vulnerable (i.e., divided into

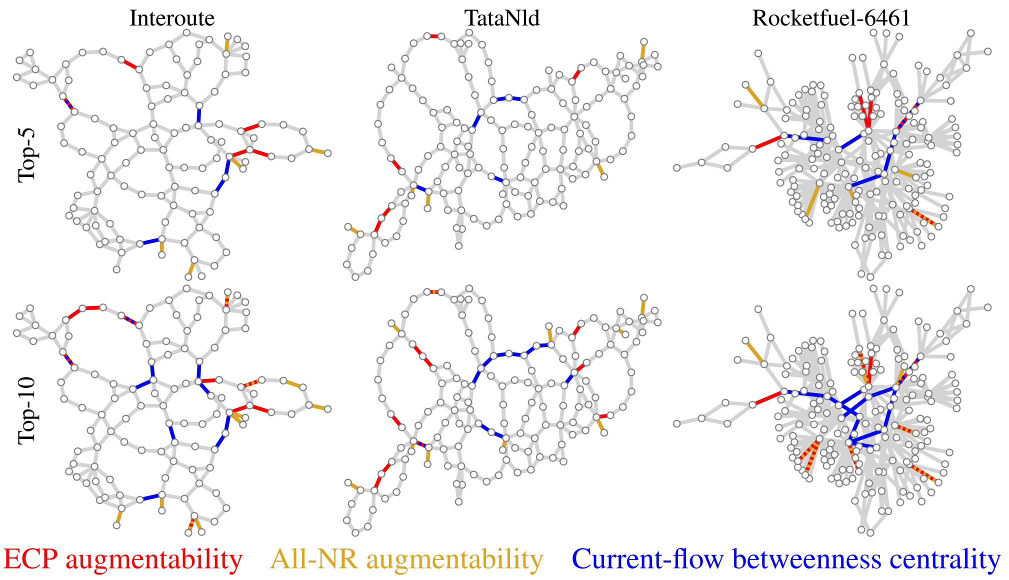


Figure 6.3: Top-5 and top-10 critical edges chosen by three criticality measures: ECP augmentability (red), All-NR augmentability (yellow), and current-flow betweenness centrality (blue).

small connected components) against attacks on high-degree nodes [Albert *et al.*, 2000; Cohen *et al.*, 2001; Magoni, 2003]. Researchers of operations research studied the interdiction problem, which brings down a network (e.g., degrades its shortest path or its maximum flow) with minimum attacks [Smith and Song, 2020]. In the field of communication networks, researchers investigated network conditions after severe events such as natural disasters [Neumayer and Modiano, 2010, 2011; Agarwal *et al.*, 2010; Rahnamay-Naeini *et al.*, 2011; Manzano *et al.*, 2011; Oostenbrink and Kuipers, 2017; Natalino *et al.*, 2017; Al Mtawa *et al.*, 2021]. These studies often counted the number of connected nodes to measure the damage. They used normalized reliability measures, e.g., the average content accessibility (the ratio of clients connected to a resource (server)) [Natalino *et al.*, 2017] or the average two-terminal reliability (the ratio of node pairs connected to each other) [Neumayer and Modiano, 2010, 2011; Agarwal *et al.*, 2010; Rahnamay-Naeini *et al.*, 2011; Manzano *et al.*, 2011; Oostenbrink and Kuipers, 2017; Al Mtawa *et al.*, 2021]. These reliability measures have similarity with ECN and ECP in their definitions, but the past studies only examined a few (a polynomial number of) failed states of a network that could happen after the event and could not compute the *expected number* of connected nodes/pairs during a long operation period.

Although a rich body of literature exists on optimal resource (server) placement problems [Sahoo *et al.*, 2016], they have not been investigated from the viewpoint of network reliability due to its computational hardness. Migov [2019] studied an optimal placement problem in which the locations of data sinks are determined based on the connectivity with sensors in a wireless sensor network. Unfortunately, they relied on an inefficient factoring approach [AboElFotoh *et al.*, 2005] and only examined a network with fewer than 100 edges.

Methods to identify critical links in a probabilistic network have been studied [Kuo *et al.*, 2007; Inoue, 2019], but they are based on traditional network reliability measures, not ECP. Network centrality indices, which were introduced to identify influential nodes or links in a complex network, are usually defined based on paths and node degrees [Koschützki *et al.*, 2005]. As shown in the experiments, these indices cannot represent the criticality against ECP.

6.8 Conclusion

This chapter proposed an efficient method that exactly computes ECP, the expected number of connected node pairs, under an independent failure model. The proposed method is more efficient than the CSNR method in Chapter 5 by $O(n)$. This chapter presents two important applications in network design: critical link identification and optimal server placement. Numerical evaluation shows that our method outperforms the state-of-the-art by several orders of magnitude in terms of computational efficiency. Future work will include an extension to support node failures, an analysis using node weights with population, and an unified failure model that represents both ordinary times and severe events.

We have provided public access to their code and data at `https://github.com/nttcslab/fast-ecp-ecn`.

Chapter 7

Variance Analysis on Network Reliability

Traditionally, network reliability has been evaluated on the assumption that the availability of each link is precisely given. However, in reality, it may be given with a degree of uncertainty, e.g., with *variance*, which must be propagated into the network reliability. To the best of our knowledge, no literature has investigated this issue because, since even computing the reliability itself belongs to a computationally tough class, computing the variance seems much harder.

This chapter proposes an efficient algorithm to compute the variance of the network reliability given the variance in link availability. We experimentally verify the performance of the proposed algorithm and show that it can compute the variance of network reliability within 0.1 seconds for real topologies with nearly 200 links. We also perform extensive analyses on the variance of network reliability and reveal that network reliability tends to be accurate and that the variance in reliability does not significantly exceed the variance in link availability. Even when some links have a substantial variance in availability, the impact on network reliability is marginal.

7.1 Introduction

7.1.1 Background

As described in Section 4.2, network reliability is defined as the probability that a path of working links exists between any pair of specified nodes assuming each link fails independently and stochastically [Moskowitz, 1958; Boesch *et al.*, 2009]. Here, network reliability has traditionally been evaluated under the assumption that the *availability* of each link is given precisely (link availability is the probability that the link is working). In reality, however, it may be given with a degree of inaccuracy or estimated with some error; e.g., the lifetime of an optical fiber is estimated as a distribution [Aso *et al.*, 2012], and a degree of confidence is often given to regression predictions made by machine learning. If the link availability has errors, e.g., *variance*, the network reliability evaluated based on them should also have variance, but no literature has investigated this issue. Previous studies only dealt with the reliability value itself without an accuracy measure like variance. In practice, the accuracy of network reliability can be vital: Assuming two networks, one with 99% reliability and a standard deviation of 1% and another with 98.5% reliability and a standard deviation of 0.1%, which is a more stable infrastructure? The latter is probably preferred. Unfortunately, such a comparison was impossible in the past because the accuracy was not evaluated.

7.1.2 Literature Review

Although "variance" has been studied in the context of network reliability [Cancela and El Khadiri, 1995; Cancela *et al.*, 2014; Robledo *et al.*, 2020], these studies were only concerned with it as a sampling error in Monte Carlo estimation of network reliability. Therefore, even if networks have links with significant uncertainty in availability, they basically ignore the impact on network reliability.

Agrawal *et al.* [2019] investigated network survivability (i.e., network reliability in the broad sense) using real disaster statistics. Although this study relied on different statistics, the uncertainty was not considered in reliability analysis.

In recent years, the research community on network reliability has often addressed the problem where the routes of newly added links are decided to minimize the damage factor and cost given a geographic damage rate [Tapolcai *et al.*, 2021; Oostenbrink and Kuipers, 2021]. This chapter complements these studies. Although the availability of additional links is subject to uncertainty due to a lack of operational experience, the work of this chapter allows us to evaluate their impact on network reliability.

7.1.3 Research Challenges and Our Contributions

Currently, there are two challenges related to the accuracy of network reliability given uncertainty in link availability.

- Behavior of the variance of network reliability (*VoR*): The most critical issue is that no one knows the accuracy of the network reliability when the link availability has variance. It is also unknown how the variance of link availability is related to VoR. Does VoR tend to be larger than availability variance of each link? How does VoR behave when the link availability variance or the network size changes? How small should the link availability variance be to keep VoR within a specified range?

- VoR computation: There is no known method for evaluating VoR. Is it possible to develop a method with sufficient scalability for practical use? The computation of network reliability belongs to a computationally tough class called #P-complete [Valiant, 1979] as described in Section 4.2.1, and VoR computation is unlikely to be easier, although its hardness is unknown.

In this chapter, we propose an algorithm that computes VoR and analyze it from several perspectives. The contributions are summarized as follows.

- Computation method: This chapter proposes an efficient algorithm for computing VoR using BDD and DP. The HH method [Hardy *et al.*, 2007] in Section 4.3, which compute the reliability itself, utilize DP with BDD dnodes as states; we compute VoR by designing novel DP with BDD dnodes *pairs* as states. This chapter also analyzes the computational complexity, which is much smaller than that of the naive method for the sparse topologies common in infrastructures. Numerical experiments show that VoR can be computed within 0.1 seconds even for a large real topology with nearly 200 links.

- Empirical analyses of VoR: This chapter empirically finds a good property of network reliability where VoR does not greatly exceed the variance of the link availability. This means that the network reliability is as accurate as each link availability, notwithstanding the influence of several links. This chapter also finds another property where even if the variance of availability is quite significant for some links, the impact on VoR is marginal. This means that the network reliability tends to be robust against some links' significant uncertainty in availability. Overall, network reliability tends to be accurate even under uncertainty in link availability, which is a desirable property for network design.

7.2 Problem Statement

We use an independent failure model described in Section 4.1 and a standard K-terminal network reliability in Section 4.2. The difference comes from how the edge availability, the probability that an edge $e \in E$ is present, is defined. In standard definition (Section 4.1), it is given as a real value $p_e \in [0,1]$, meaning that the probability of presence is exactly p_e. In this problem, to reflect the uncertainty, we let an edge availability follow a probability distribution of mean p_e and variance σ_e^2. That is, the edge availability of edge $e \in E$ is given as a random variable $\mathbf{P}_e$ with mean p_e and variance σ_e^2. To ensure independence among the edges' states (present or absent), it is assumed that $\mathbf{P}_e$ and $\mathbf{P}_{e'}$ are statistically independent for $e \neq e'$. By substituting p_e with $\mathbf{P}_e$ in the definition of K-terminal network reliability ((4.2) in Section 4.2), the K-NR also becomes a random variable $\mathbf{R}(K)$ defined as

$$\mathbf{R}(K) = \sum_{E' \in \mathcal{E}_K} \left[\prod_{e \in E'} \mathbf{P}_e \cdot \prod_{e \in E \setminus E'} (1 - \mathbf{P}_e) \right]. \tag{7.1}$$

Recall that $\mathcal{E}_K$ is the family of subgraphs of G such that the vertices in K are interconnected.

Hereafter, we use E[$\cdot$], Var[$\cdot$], and Cov[$\cdot,\cdot$] as expectation, variance, and covariance with respect to $\mathbf{P}_1, \ldots, \mathbf{P}_m$. Then by fixing $\mathbf{P}_i = p_i$, the conventional K-NR equals E[$\mathbf{R}(K)$]. The problem solved in this chapter can be posed as follows.

Problem 7.1 (VoR computation)**.** Given terminals K and each edge's p_i and σ_i^2, the VoR computation problem is to compute VoR Var[$\mathbf{R}(K)$], which represents the uncertainty in network reliability given the variances of edge availabilities.

Let us consider a simple example of triangle graph: G is a graph with 3 vertices $V = \{1,2,3\}$ and 3 edges $E = \{e_1 = (1,2), e_2 = (2,3), e_3 = (1,3)\}$, and $K = \{1,2\}$. Then, $\mathbf{R}(K) = \mathbf{P}_{e_1} + (1 - \mathbf{P}_{e_1})\mathbf{P}_{e_2}\mathbf{P}_{e_3}$. Suppose that all mean and variance values of link availabilities are identical, i.e., $p_{e_i} = p$ and $\sigma_{e_i}^2 = \sigma^2$ for all i. Even under such assumption, VoR Var[$\mathbf{R}(K)$] becomes complicated:

$$\sigma^2(1 - 4p^3 + 3p^4) + \sigma^4(1 - 2p + 3p^2) + \sigma^6.$$

This example shows us that even for small networks, the computation of VoR becomes a cumbersome task, and the behavior of VoR value is hard to grasp.

A naive way to compute VoR is to use (7.1) to decompose it:

$$\mathrm{Var}[\mathbf{R}(K)] = \mathrm{Var}[\textstyle\sum_{X \in \mathcal{E}_K} \mathbf{P}(X)] = \displaystyle\sum_{X \in \mathcal{E}_K} \sum_{Y \in \mathcal{E}_K} \mathrm{Cov}[\mathbf{P}(X), \mathbf{P}(Y)],$$

where $\mathbf{P}(E')$ for $E' \subseteq E$ is defined as

$$\mathbf{P}(E') = \prod_{e \in E'} \mathbf{P}_e \cdot \prod_{e \in E \setminus E'} (1 - \mathbf{P}_e).$$

Since each $\mathrm{Cov}[\mathbf{P}(X), \mathbf{P}(Y)]$ can be computed in $O(m)$ time with p_i and σ_i^2, $\mathrm{Var}[\mathbf{R}(K)]$ can be computed in $O(m|\mathcal{E}_K|^2)$ time once all the states in $\mathcal{E}_K$ are enumerated. However, such a method becomes intractable for graphs with only dozens of links since $|\mathcal{E}_K|$ grows exponentially with m.

7.3 Method

The proposed method can be divided into two parts: (i) constructing a BDD [Bryant, 1986] that represents $\mathcal{E}_K$ by the HH method Hardy *et al.* [2007] and (ii) computing $\mathrm{E}[\mathbf{R}(K)]$ and $\mathrm{Var}[\mathbf{R}(K)]$ using the built BDD. Since we have already explained the HH method in Section 4.3, we omit the procedures for the former part, but we first introduce an important notion, *random variables associated with dnodes*. Then we describe the latter part in Section 7.3.2 and analyze the computational complexity of the proposed method in Section 7.3.3.

7.3.1 Random Variables for Dnodes

As shown in Section 4.3, we can construct BDD $\mathtt{B}_K$ representing $\mathcal{E}_K$ in a top-down manner by calling $\textsc{Construct}(\mathit{Connected}_K)$ (see Algorithms 2.1 and 4.1). Note that we may further reduce the size of $\mathtt{B}_K$ by reduce operation [Bryant, 1986]. After construction, we consider associating a random variable for every dnode in $\mathtt{B}_K$. Recall that in the standard problem setting, the K-NR value can be obtained by a bottom-up DP on $\mathtt{B}_K$ and examine the root value since evaluating K-NR is equivalent to solve the weighted model counting problem. In this algorithm, we compute a kind of partial sum for every dnode $\mathtt{n}$ and store it as $\mathsf{DP}[\mathtt{n}]$. More specifically, we conduct the following bottom-up DP computation:

$$\begin{aligned} \mathsf{DP}[\top] &= 1, \quad \mathsf{DP}[\bot] = 0, \\ \mathsf{DP}[\mathtt{n}] &= (1 - p_{e_{\mathsf{lb}(\mathtt{n})}})\mathsf{DP}[\mathsf{lo}(\mathtt{n})] + p_{e_{\mathsf{lb}(\mathtt{n})}}\mathsf{DP}[\mathsf{hi}(\mathtt{n})]. \end{aligned} \tag{7.2}$$

Eventually, K-NR $R(K)$ equals $\mathsf{DP}[\mathtt{r}]$ of root dnode $\mathtt{r}$.

Similar to this, we associate random variable $\mathbf{R}_\mathtt{n}$ for every dnode $\mathtt{n}$ satisfying the following equations. For $\mathtt{n} = \top, \bot$, which mean the terminals are already disconnected or all connected, respectively, we define

$$\mathbf{R}_\top = 1, \quad \mathbf{R}_\bot = 0. \tag{7.3}$$

For internal node $\mathtt{n}$, we define

$$\mathbf{R}_\mathtt{n} = (1 - \mathbf{P}_{e_{\mathsf{lb}(\mathtt{n})}})\mathbf{R}_{\mathsf{lo}(\mathtt{n})} + \mathbf{P}_{e_{\mathsf{lb}(\mathtt{n})}}\mathbf{R}_{\mathsf{hi}(\mathtt{n})}. \tag{7.4}$$

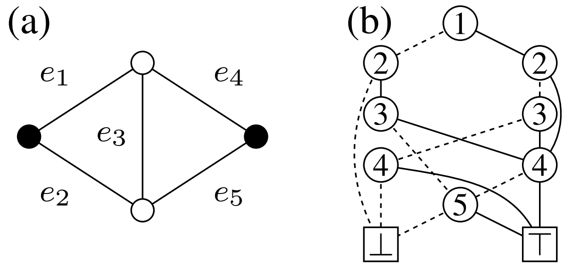


Figure 7.1: (a) Example of undirected graph G. Black vertices indicate they are terminals. (b) $\mathtt{B}_K$ for graph and terminals in (a). Here lo- and hi-arcs are drawn with dashed and solid lines, respectively, and label of each dnode is drawn inside the circle.

Since (7.3) and (7.4) imitates the standard DP computation (7.2), eventually $\mathbf{R}(K) = \mathbf{R}_{\mathtt{r}}$ holds where $\mathtt{r}$ is the root dnode.

7.3.2 Proposed Method

The proposed method computes VoR $\mathrm{Var}[\mathbf{R}(K)]$ using the BDD $\mathtt{B}_K$. The key point is that we consider the covariances among the random variables associated with dnodes, i.e., $\mathrm{Cov}[\mathbf{R}_{\mathtt{n}}, \mathbf{R}_{\mathtt{n}'}]$ for the pair of BDD dnodes $\mathtt{n}, \mathtt{n}'$. Since $\mathbf{R}_{\mathtt{r}}$ equals $\mathbf{R}(K)$, its variance can be rewritten as $\mathrm{Var}[\mathbf{R}_{\mathtt{r}}] = \mathrm{Cov}[\mathbf{R}_{\mathtt{r}}, \mathbf{R}_{\mathtt{r}}]$. The proposed method computes $\mathrm{Cov}[\mathbf{R}_{\mathtt{n}}, \mathbf{R}_{\mathtt{n}'}]$ by decomposing it into the covariances of the child dnodes, e.g., $\mathrm{Cov}[\mathbf{R}_{\mathsf{lo}(\mathtt{n})}, \mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]$, which can be seen as a DP whose states are pairs of BDD dnodes.

Before proceeding to our method, we derive two equations for the covariances. Lemma 7.2 is a well-known formula, and Lemma 7.3 generalizes the variance of two independent random variables, which we will prove.

Lemma 7.2. *Let* $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$*, and* $\mathbf{W}$ *be random variables. Then,*

$$\begin{aligned}\mathrm{Cov}[\mathbf{X}+\mathbf{Y}, \mathbf{Z}+\mathbf{W}] = {} & \mathrm{Cov}[\mathbf{X}, \mathbf{Z}] + \mathrm{Cov}[\mathbf{X}, \mathbf{W}] \\ & + \mathrm{Cov}[\mathbf{Y}, \mathbf{Z}] + \mathrm{Cov}[\mathbf{Y}, \mathbf{W}].\end{aligned} \tag{7.5}$$

Lemma 7.3. *Let* $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$*, and* $\mathbf{W}$ *be random variables such that* $\mathbf{X}$ *and* $\mathbf{Z}$*,* $\mathbf{Y}$ *and* $\mathbf{W}$*, and* $\mathbf{XY}$ *and* $\mathbf{ZW}$ *are pairwise independent for each. Then,*

$$\begin{aligned}\mathrm{Cov}[\mathbf{XZ}, \mathbf{YW}] = {} & (\mathrm{E}[\mathbf{X}]\mathrm{E}[\mathbf{Y}] + \mathrm{Cov}[\mathbf{X}, \mathbf{Y}])\mathrm{Cov}[\mathbf{Z}, \mathbf{W}] \\ & + \mathrm{Cov}[\mathbf{X}, \mathbf{Y}]\mathrm{E}[\mathbf{Z}]\mathrm{E}[\mathbf{W}].\end{aligned} \tag{7.6}$$

Proof. Left-hand side equals

$$\begin{aligned}&\mathrm{E}[\mathbf{XYZW}] - \mathrm{E}[\mathbf{XZ}]\mathrm{E}[\mathbf{YW}] \\ &= \mathrm{E}[\mathbf{XY}]\mathrm{E}[\mathbf{ZW}] - \mathrm{E}[\mathbf{X}]\mathrm{E}[\mathbf{Y}]\mathrm{E}[\mathbf{Z}]\mathrm{E}[\mathbf{W}],\end{aligned}$$

where we use independentness of the random variables. Right-hand side also equals

$$\begin{aligned}&\mathrm{E}[\mathbf{XY}]\mathrm{Cov}[\mathbf{Z},\mathbf{W}]+\mathrm{Cov}[\mathbf{X},\mathbf{Y}]\mathrm{E}[\mathbf{Z}]\mathrm{E}[\mathbf{W}]\\&=\mathrm{E}[\mathbf{XY}](\mathrm{E}[\mathbf{ZW}]-\mathrm{E}[\mathbf{Z}]\mathrm{E}[\mathbf{W}])\\&\quad+(\mathrm{E}[\mathbf{XY}]-\mathrm{E}[\mathbf{X}]\mathrm{E}[\mathbf{Y}])\mathrm{E}[\mathbf{Z}]\mathrm{E}[\mathbf{W}]\\&=\mathrm{E}[\mathbf{XY}]\mathrm{E}[\mathbf{ZW}]-\mathrm{E}[\mathbf{X}]\mathrm{E}[\mathbf{Y}]\mathrm{E}[\mathbf{Z}]\mathrm{E}[\mathbf{W}].\end{aligned} \qquad \square$$

Using them, we derive equations for the decomposition of $\mathrm{Cov}[\mathbf{R}_{\mathtt{n}},\mathbf{R}_{\mathtt{n}'}]$. First of all, if $\mathtt{n}\in\{\top,\bot\}$, $\mathrm{Cov}[\mathbf{R}_{\mathtt{n}},\mathbf{R}_{\mathtt{n}'}]=0$ since $\mathbf{R}_\top$ and $\mathbf{R}_\bot$ are just constants. The same argument holds for the case $\mathtt{n}'\in\{\top,\bot\}$. Hereafter, we consider the case that both $\mathtt{n}$ and $\mathtt{n}'$ are internal nodes. Let $\mathbf{Q}_e=1-\mathbf{P}_e$ ($e\in E$) be a random variable. Then,

$$\begin{aligned}&\mathrm{E}[\mathbf{P}_e]=p_e,\quad \mathrm{E}[\mathbf{Q}_e]=1-p_e,\\&\mathrm{Var}[\mathbf{P}_e]=\mathrm{Var}[\mathbf{Q}_e]=\sigma_e^2,\quad \mathrm{Cov}[\mathbf{P}_e,\mathbf{Q}_e]=-\sigma_e^2.\end{aligned} \tag{7.7}$$

When $\mathtt{n}$ and $\mathtt{n}'$ have identical label $i=\mathsf{lb}(\mathtt{n})=\mathsf{lb}(\mathtt{n}')$, we see

$$\begin{aligned}&\mathrm{Cov}[\mathbf{R}_{\mathtt{n}},\mathbf{R}_{\mathtt{n}'}]\\&=\mathrm{Cov}[\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n})}+\mathbf{P}_{e_i}\mathbf{R}_{\mathsf{hi}(\mathtt{n})},\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}+\mathbf{P}_{e_i}\mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]\\&=\mathrm{Cov}[\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n})},\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}]+\mathrm{Cov}[\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n})},\mathbf{P}_{e_i}\mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]\\&\quad+\mathrm{Cov}[\mathbf{P}_{e_i}\mathbf{R}_{\mathsf{hi}(\mathtt{n})},\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}]+\mathrm{Cov}[\mathbf{P}_{e_i}\mathbf{R}_{\mathsf{hi}(\mathtt{n})},\mathbf{P}_{e_i}\mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]\end{aligned}$$

from (7.4) and Lemma 7.2. Here we observe that $\mathbf{R}_{\mathsf{lo}(\mathtt{n})}$ and $\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}$ are polynomials of $\mathbf{P}_{e_j}$ and $\mathbf{Q}_{e_j}$ ($j=i+1,\ldots,m$) since BDDs have the ordered property. Therefore, they are independent of $\mathbf{Q}_{e_i}$ and $\mathbf{Q}_{e_i}^2$, enabling us to use Lemma 7.3 to decompose $\mathrm{Cov}[\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n})},\mathbf{Q}_{e_i}\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}]$. Other terms can be decomposed in the same manner, and we finally have

$$\begin{aligned}&\mathrm{Cov}[\mathbf{R}_{\mathtt{n}},\mathbf{R}_{\mathtt{n}'}]\\&=(q_{e_i}^2+\sigma_{e_i}^2)\mathrm{Cov}[\mathbf{R}_{\mathsf{lo}(\mathtt{n})},\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}]\\&\quad+(p_{e_i}q_{e_i}-\sigma_{e_i}^2)(\mathrm{Cov}[\mathbf{R}_{\mathsf{lo}(\mathtt{n})},\mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]+\mathrm{Cov}[\mathbf{R}_{\mathsf{hi}(\mathtt{n})},\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}])\\&\quad+(p_{e_i}^2+\sigma_{e_i}^2)\mathrm{Cov}[\mathbf{R}_{\mathsf{hi}(\mathtt{n})},\mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]\\&\quad+\sigma_{e_i}^2(\mathrm{E}[\mathbf{R}_{\mathsf{hi}(\mathtt{n})}]-\mathrm{E}[\mathbf{R}_{\mathsf{lo}(\mathtt{n})}])(\mathrm{E}[\mathbf{R}_{\mathsf{hi}(\mathtt{n}')}]-\mathrm{E}[\mathbf{R}_{\mathsf{lo}(\mathtt{n}')}]),\end{aligned} \tag{7.8}$$

where $q_e=1-p_e$. Note that $\mathrm{E}[\mathbf{R}_{\mathtt{n}}]$ can be computed in a bottom-up manner with the following equation, which is obtained by taking expectation of both sides of (7.4):

$$\mathrm{E}[\mathbf{R}_{\mathtt{n}}]=q_{e_{\mathsf{lb}(\mathtt{n})}}\mathrm{E}[\mathbf{R}_{\mathsf{lo}(\mathtt{n})}]+p_{e_{\mathsf{lb}(\mathtt{n})}}\mathrm{E}[\mathbf{R}_{\mathsf{hi}(\mathtt{n})}]. \tag{7.9}$$

Algorithm 7.1: VoR computation using BDD.

```
   Input: BDD $\mathtt{B}_K = (\mathtt{N}, \mathtt{A})$ and mean $p_e$ and variance $\sigma_e^2$ of $e$'s availability.
   Output: Variance of network reliability $\mathrm{Var}[R(K)]$.
1  $\mathsf{e}[\top] \leftarrow 1.0$, $\mathsf{e}[\bot] \leftarrow 0.0$                         // $\mathsf{e}[\mathtt{n}] := \mathrm{E}[\mathbf{R}_\mathtt{n}]$
2  foreach $\mathtt{n} \in \mathtt{N} \setminus \{\top, \bot\}$ in bottom-up order do
3      $\mathsf{e}[\mathtt{n}] \leftarrow q_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{e}[\mathsf{lo}(\mathtt{n})] + p_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{e}[\mathsf{hi}(\mathtt{n})]$      // (7.9); $q_e = 1 - p_e$
4  return $\mathtt{Cov}(\mathtt{r}, \mathtt{r})$                                   // r: root node of $\mathtt{B}_K$
5  function $\mathtt{Cov}(\mathtt{n}, \mathtt{n}')$:                           // Computes $\mathrm{Cov}[\mathbf{R}_\mathtt{n}, \mathbf{R}_{\mathtt{n}'}]$
6      if $(\mathtt{n} \in \{\top, \bot\}) \vee (\mathtt{n}' \in \{\top, \bot\})$ then                // Base case
7          return 0
8      if $\mathsf{c}[(\mathtt{n}, \mathtt{n}')]$ or $\mathsf{c}[(\mathtt{n}', \mathtt{n})]$ exists then                // Already computed
9          return $\mathsf{c}[(\mathtt{n}, \mathtt{n}')]$ or $\mathsf{c}[(\mathtt{n}', \mathtt{n})]$              // Return cached value
10     $i \leftarrow \min\{\mathsf{lb}(\mathtt{n}), \mathsf{lb}(\mathtt{n}')\}$
11     if $\mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathtt{n}')$ then                                // $\mathsf{lb}(\mathtt{n})$ is smaller
12         $\mathsf{c}[(\mathtt{n}, \mathtt{n}')] \leftarrow q_{e_i} \cdot \mathtt{Cov}(\mathsf{lo}(\mathtt{n}), \mathtt{n}') + p_{e_i} \cdot \mathtt{Cov}(\mathsf{hi}(\mathtt{n}), \mathtt{n}')$          // (7.10)
13     else if $\mathsf{lb}(\mathtt{n}) > \mathsf{lb}(\mathtt{n}')$ then                           // $\mathsf{lb}(\mathtt{n}')$ is smaller
14         $\mathsf{c}[(\mathtt{n}, \mathtt{n}')] \leftarrow q_{e_i} \cdot \mathtt{Cov}(\mathtt{n}, \mathsf{lo}(\mathtt{n}')) + p_{e_i} \cdot \mathtt{Cov}(\mathtt{n}, \mathsf{hi}(\mathtt{n}'))$          // (7.10)
15     else                                          // n and n' have the same label
16         $\mathsf{c}[(\mathtt{n}, \mathtt{n}')] \leftarrow (q_{e_i}^2 + \sigma_{e_i}^2) \cdot \mathtt{Cov}(\mathsf{lo}(\mathtt{n}), \mathsf{lo}(\mathtt{n}'))$
             $+ (p_{e_i} q_{e_i} - \sigma_{e_i}^2) \cdot (\mathtt{Cov}(\mathsf{lo}(\mathtt{n}), \mathsf{hi}(\mathtt{n}')) + \mathtt{Cov}(\mathsf{hi}(\mathtt{n}), \mathsf{lo}(\mathtt{n}')))$
             $+ (p_{e_i}^2 + \sigma_{e_i}^2) \cdot \mathtt{Cov}(\mathsf{hi}(\mathtt{n}), \mathsf{hi}(\mathtt{n}'))$
             $+ \sigma_{e_i}^2 \cdot (\mathsf{e}[\mathsf{hi}(\mathtt{n})] - \mathsf{e}[\mathsf{lo}(\mathtt{n})])(\mathsf{e}[\mathsf{hi}(\mathtt{n}')] - \mathsf{e}[\mathsf{lo}(\mathtt{n}')])$        // (7.8)
17     return $\mathsf{c}[(\mathtt{n}, \mathtt{n}')]$                            // Return computed value
```

When $i = \mathsf{lb}(\mathtt{n}) < \mathsf{lb}(\mathtt{n}')$, we can derive more simple formula. By just applying (7.4) for $\mathbf{R}_\mathtt{n}$ and using Lemmas 7.2 and 7.3, we have

$$\mathrm{Cov}[\mathbf{R}_\mathtt{n}, \mathbf{R}_{\mathtt{n}'}] = q_{e_i}\mathrm{Cov}[\mathbf{R}_{\mathsf{lo}(\mathtt{n})}, \mathbf{R}_{\mathtt{n}'}] + p_{e_i}\mathrm{Cov}[\mathbf{R}_{\mathsf{hi}(\mathtt{n})}, \mathbf{R}_{\mathtt{n}'}]. \tag{7.10}$$

The case $\mathsf{lb}(u) > \mathsf{lb}(v)$ is treated similarly. By recursively using (7.8) and (7.10), we can decompose $\mathrm{Var}[\mathbf{R}(K)] = \mathrm{Cov}[\mathbf{R}_\mathtt{r}, \mathbf{R}_\mathtt{r}]$ into small parts. We apply such decompositions until at least one node of the pair becomes $\top$ or $\bot$. To speed-up computation, we store the value of $\mathrm{Cov}[\mathbf{R}_\mathtt{n}, \mathbf{R}_{\mathtt{n}'}]$ for dnode pair $(\mathtt{n}, \mathtt{n}')$ once it is computed.

The pseudocode for our algorithm is given in Algorithm 7.1. Lines 1–3 compute $\mathsf{e}[\mathtt{n}] = \mathrm{E}[\mathbf{R}_\mathtt{n}]$ for each $\mathtt{n} \in \mathtt{N}$; here the expectation of reliability $\mathrm{E}[\mathbf{R}(K)] = \mathsf{e}[\mathtt{r}]$ is also computed. The main part is $\mathtt{Cov}(\mathtt{n}, \mathtt{n}')$: a recursive procedure for computing $\mathrm{Cov}[\mathbf{R}_\mathtt{n}, \mathbf{R}_{\mathtt{n}'}]$. We use cache c to store the computed value of $\mathrm{Cov}[\mathbf{R}_\mathtt{n}, \mathbf{R}_{\mathtt{n}'}]$ in $\mathsf{c}[(\mathtt{n}, \mathtt{n}')]$. Line 6 deals with the base case; if at least one of $\mathtt{n}$ and nDD' is a terminal, the covariance is zero. Line 8 is the case where $\mathrm{Cov}[\mathbf{R}_\mathtt{n}, \mathbf{R}_{\mathtt{n}'}]$ was already computed. Lines 11 and 13 use (7.10) since their labels are different, and Line 15 uses (7.8) since they are identical.

7.3.3 Complexity

Here we analyze the computational complexity of the proposed method.

Theorem 7.4. *Given BDD* $\mathtt{B}_K$*, Algorithm 7.1 runs in* $O(|\mathtt{B}_K|^2)$ *time.*

Proof. The computation of $\mathtt{e}$ in Lines 1–3 costs $O(|\mathtt{B}_K|)$ time. The value of each cache entry $\mathtt{c}[(\mathtt{n}, \mathtt{n}')]$ can be computed in constant time without the recursive calls of $\mathtt{Cov}(\cdot, \cdot)$. Since there are at most $O(|\mathtt{B}_K|^2)$ entries to compute, the overall running time is bounded by $O(|\mathtt{B}_K|^2)$. □

Combining Theorem 7.4 and Lemma 4.9 gives the time complexity for computing VoR: $O(m^2 4^{W_F} E^2_{W_F})$ time where m is the number of edges, W_F is the frontier width, and E_{W_F} is the W_F-th Bell number. It seems that the computational time of VoR with our algorithm is proportional to the square of the number of links when the frontier width is assumed to be a constant. However, we can sharpen this bound since Lemma 4.9 just provides the sizes of *normalized* BDD. A BDD is normalized if for every $\mathtt{n} \in \mathtt{N} \setminus \{\top, \bot\}$, both $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$ are either a terminal node or a node whose label is $\mathsf{lb}(\mathtt{n}) + 1$. Given normalized BDD $\mathtt{B}_K$, Algorithm 7.1 computes $\mathtt{c}[(\mathtt{n}, \mathtt{n}')]$ for only pairs of nodes whose labels are identical. This is verified as follows: First, $\mathtt{Cov}(\mathtt{r}, \mathtt{r})$ is called, in which $\mathsf{lb}(\mathtt{r}) = \mathsf{lb}(\mathtt{r})$ holds. Next, when $\mathtt{Cov}(\mathtt{n}, \mathtt{n}')$ with $\mathsf{lb}(\mathtt{n}) = \mathsf{lb}(\mathtt{n}')$ is called, all recursive calls $\mathtt{Cov}(\mathsf{lo}(\mathtt{n}), \mathsf{lo}(\mathtt{n}'))$, $\mathtt{Cov}(\mathsf{lo}(\mathtt{n}), \mathsf{hi}(\mathtt{n}'))$, $\mathtt{Cov}(\mathsf{hi}(\mathtt{n}), \mathsf{lo}(\mathtt{n}'))$, and $\mathtt{Cov}(\mathsf{hi}(\mathtt{n}), \mathsf{hi}(\mathtt{n}'))$ in Line 16 satisfy that both arguments of $\mathtt{Cov}$ has an identical node or they involve terminal nodes. This leads to sophisticated complexity of Algorithm 7.1.

Theorem 7.5. *If the given BDD* $\mathtt{B}_K$ *is normalized, Algorithm 7.1 runs in* $O(\sum_{i=1}^{m} |\mathtt{L}_i|^2)$ *time, where* $|\mathtt{L}_i|$ *is the number of dnodes with label* i*.*

Recalling the proofs of Lemma 4.9 and Corollaries 4.11 and 4.12, we can say $|\mathtt{L}_i| = O(2^{W_F} E_{W_F})$ for general cases, $|\mathtt{L}_i| = O(W_F^2 E_{W_F})$ for $|K| = 2$, and $|\mathtt{L}_i| = O(E_{W_F})$ for $|K| = n$. This leads to the following result.

Corollary 7.6. *Given graph* G*, terminals* K*, and edge order whose frontier width is* W_F*, we can compute the variance* $\mathrm{Var}[\mathbf{R}(K)]$ *of the reliability in* $O(m 4^{W_F} E^2_{W_F})$ *time. When* $|K| = 2$*, the time complexity is bounded by* $O(m W_F^4 E^2_{W_F})$*, and when* $|K| = n$*, it is bounded by* $O(m E^2_{W_F})$*.*

This suggests that for graphs with a bounded path-width W_p, the computational time of VoR is linear to the number of edges, since we can determine edge order with $W_F = W_p$ ([Inoue and Minato, 2016]; also see Section 4.3.2). We later experimentally confirm this in Section 7.4.1.

7.3.4 Preprocessing

We consider preprocessing the input graph G before constructing the BDD $\mathtt{B}_K$ to accelerate our method. That is, the degree 1 vertex elimination. Since this operation sometimes decreases the path-width of G, it may lead to a substantial speed-up of our method.

Suppose w is a degree 1 vertex in G and let $e_w = \{u, w\}$ be the only edge incident to w. Then, the network reliability (7.1) on G (denoted by $\mathbf{R}^G(\cdot)$) can be represented by that on $G - w$ (denoted by $\mathbf{R}^{G-w}(\cdot)$), the graph obtained by deleting w and e_w from G:

$$\mathbf{R}^G(K) = \begin{cases} \mathbf{P}_{e_w}\mathbf{R}^{G-w}(K \cup \{u\} \setminus \{w\}) & (w \in K), \\ \mathbf{R}^{G-w}(K) & (w \notin K). \end{cases} \tag{7.11}$$

By recursively removing degree 1 vertices, we obtain a graph with only degree ≥ 2 vertices or a graph with a single vertex; we denote this by G'. The recursive application of (7.11) gives the following form: $\mathbf{R}^G(K) = \mathbf{P}_{e_{i_1}} \cdot \cdots \cdot \mathbf{P}_{e_{i_l}}\mathbf{R}^{G'}(K')$. After computing the mean and variance of $\mathbf{R}^{G'}(K')$ by the proposed method, those of $\mathbf{R}^G(K)$ can immediately be computed with the following well-known formulas: given mutually independent random variables $\mathbf{X}$ and $\mathbf{Y}$, $\mathrm{E}[\mathbf{XY}] = \mathrm{E}[\mathbf{X}]\mathrm{E}[\mathbf{Y}]$ and $\mathrm{Var}[\mathbf{XY}] = \mathrm{Var}[\mathbf{X}](\mathrm{E}[\mathbf{Y}])^2 + (\mathrm{E}[\mathbf{X}])^2\mathrm{Var}[\mathbf{Y}] + \mathrm{Var}[\mathbf{X}]\mathrm{Var}[\mathbf{Y}]$.

7.4 Experiments

7.4.1 Computational Time

We first measured the elapsed time to compute VoR to confirm the efficiency of our algorithm and the complexity results in Section 7.3.3. The proposed method and the naive method described in Section 7.2 were implemented in C++11. Note that both our method and the naive method employed the preprocessing described in Section 7.3.4. All the codes were compiled by g++-4.8.5 with `-O3 -DNDEBUG` options. The experiments were conducted on a single thread of Linux machine with AMD EPYC 7763 2.45 GHz CPU and 2048 GB RAM; note that we used less than 8 GB of memory during our experiments. To precisely measure the consumed time of VoR computation, we measured the time after building BDD $\mathtt{B}_K$ for the proposed method and the time after enumerating the states in $\mathcal{E}_K$ for the naive method. Note that for our method, the cost of building BDD $\mathtt{B}_K$ with HH method was less than 250 and 13 ms for grid and real instances (described later), respectively, and was negligible compared to the running time of Algorithm 7.1.

We used both grid and real graphs as tested instances. Grid-wxh denotes a grid graph with $w \times h$ vertices. For them, we used the edge ordering of [Iwashita *et al.*, 2013], which is based on a path-decomposition of width $\min\{w, h\}$ and is known to build smaller BDDs for grid graphs. The real ones are from Internet Topology Zoo [Knight *et al.*, 2011] datasets. For them, we extracted the largest connected component, removed the self-loops, and selected graphs with more than 100 and less than 200 edges. We determined the edge ordering based on a beam search based path-width optimization [Inoue and Minato, 2016] because no better rigid ordering is known for them. For each graph, we considered three k values, that is, the number of terminals: two-terminals ($k = 2$), half-terminals ($k = \lfloor |V|/2 \rfloor$), and all-terminals ($k = |V|$). For each k, we made 20 terminal sets $K_1, \ldots, K_{20}$ by randomly choosing k vertices for each, computed $\mathrm{Var}[\mathbf{R}(K_j)]$ for each j, and then aggregated the average and standard deviation of the computational times. Note that for $k = |V|$, we simply computed $\mathrm{Var}[\mathbf{R}(V)]$ 20 times and aggregated the computational times. Since the original data [Knight *et al.*, 2011] do not include mean p_e and variance σ_e of the availability of e, p_e was chosen uniformly at random from $[0.9, 0.95]$ based on the literature [Elshqeirat *et al.*, 2015; Botev *et al.*, 2012; Xiao *et al.*, 2009; Nishino *et al.*, 2018; Inoue, 2019], and σ_e was set to 0.01. The implementation and all data used in this section are available at `https://github.com/nttcslab/variance-net-reliability`.

First, the naive method did not finish the computation of VoR for all terminals of the Grid-4x4 graph (the smallest graph we tested) within an hour even after all the states in $\mathcal{E}_K$ are enumerated. On the other hand, our method computed VoR within 20 seconds for every instance. Since this clearly shows our algorithm's efficiency over the naive method, hereafter we only focus on the results of our method.

Table 7.1 shows the result, where W_F is the frontier width. It can be observed that VoR can be computed in less than 0.1 seconds for all the graphs from Internet Topology Zoo. We can also see that although the Grid-6x60 graph has many vertices and edges, VoR can be computed in a reasonable time because its frontier width remains small. In Table 7.1, the standard deviation of the two-terminal setting ($k = 2$) is basically larger than those of other settings. This is because when $k = 2$, the BDD size $|\mathtt{B}_K|$ changes drastically in proportion to how far the selected two terminals are in the original graph G. Since the computational time heavily depends on the BDD size, it also deviates.

Figure 7.2 depicts the computational time for the Grid-6xh graphs. Since the frontier width of the Grid-6xh graph can be always 6 regardless of h and the number of edges is $O(h)$, it is expected from Corollary 7.6 that the computational complexity is also $O(h)$. Figure 7.2 clearly indicates this

Table 7.1: Consumed time (mean and standard deviation in seconds) for computing VoR with the proposed method.

Instance	n	m	W_F	$k = 2$	$k = \lfloor \lvert V \rvert /2 \rfloor$	$k = \lvert V \rvert$
Grid-4x4	16	24	4	<1 ms	<1 ms	<1 ms
Grid-5x5	25	40	5	0.004±0.002	0.009±0.005	0.002±0.000
Grid-6x6	36	60	6	0.116±0.084	0.290±0.136	0.033±0.001
Grid-6x30	180	324	6	3.331±2.204	3.198±0.398	0.283±0.004
Grid-6x60	360	654	6	6.991±2.931	6.388±0.671	0.569±0.004
Grid-7x7	49	84	7	4.969±2.953	8.519±2.642	0.764±0.006
Interoute	110	146	6	0.007±0.004	0.023±0.015	0.005±0.000
Ion	125	146	5	<1 ms	0.002±0.000	<1 ms
DialtelecomCz	84	151	5	<1 ms	<1 ms	<1 ms
Deltacom	113	161	6	0.008±0.005	0.023±0.008	0.006±0.000
TataNld	145	186	6	0.017±0.010	0.020±0.007	0.010±0.000
UsCarrier	158	189	5	<1 ms	0.001±0.001	<1 ms

tendency; the computational time is roughly $O(h)$ rather than $O(h^2)$ (from Theorem 7.4) for all the settings.

7.4.2 Analyses on Variance of Reliability

In this section, we analyzed VoR with the graphs used in Section 7.4.1. First, we plotted in Figure 7.3 the values of the standard deviation of reliability, which is the square root of VoR and is hereafter abbreviated as $\sqrt{\text{VoR}}$, computed in the experiments in Section 7.4.1. In Section 7.4.1, the standard deviation σ_e of the availability was set to 0.01 for every edge. Here $\sqrt{\text{VoR}}$ does not exceed 0.01 for a large portion of reliability measures, and it never exceeds twice of 0.01. It is shown that network reliability has roughly the same accuracy as the availabilities of edges, and it becomes more accurate for many cases. Figure 7.3 also suggests that there is no clear correspondence between the number of terminals k and $\sqrt{\text{VoR}}$.

Next, we observed the robustness of VoR as follows: We computed $\sqrt{\text{VoR}}$ for all terminals ($k = |V|$) with the same p_e and σ_e values in Section 7.4.1, except that one σ_{e^*} value was set to 0.1, 10 times larger than the original value, which means the variance of the availability of $e^* \in E$ became quite significant. We conducted the above computation for every edge $e^* \in E$ and drew how many times $\sqrt{\text{VoR}}$ increased as a heatmap. Figure 7.4 depicts such heatmaps for Grid-6x6, Grid-6x30, and the graphs from Internet Topology

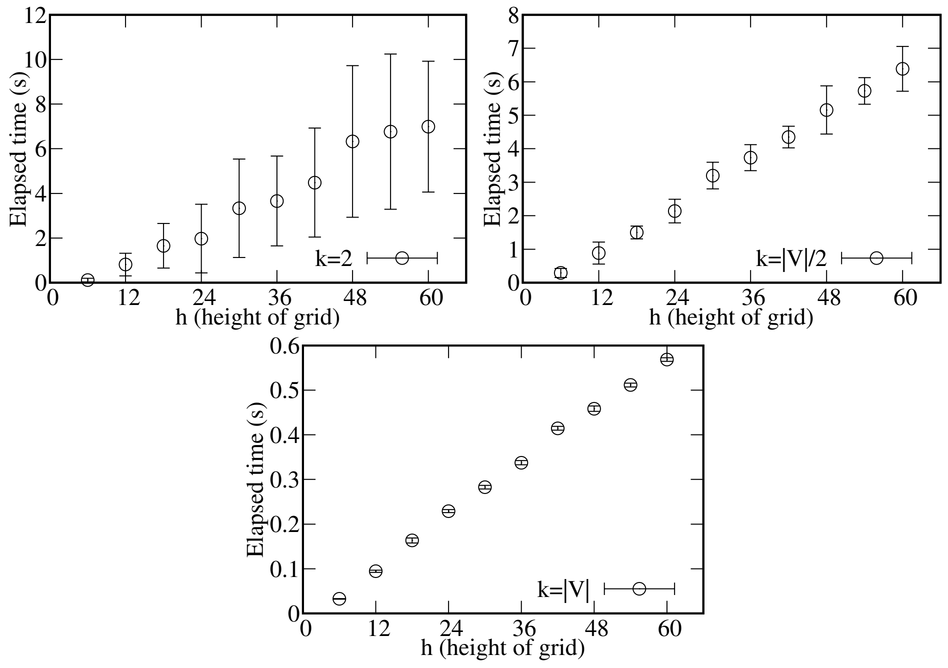


Figure 7.2: Consumed time (mean and standard deviation in seconds) for computing VoR of Grid-6xh graphs with our method.

Zoo; a darker color means that $\sqrt{\text{VoR}}$ increased more when its availability's variance increased. Figure 7.4 shows that for almost all edges e^*, $\sqrt{\text{VoR}}$ only increased marginally even if σ_{i^*} became 10 times larger, which indicates the robustness of VoR. It is also observed that the edges with a greater impact on VoR correspond to the critical edges, e.g., a bridge edge whose failure immediately disconnected some terminals.

Finally, we analyzed the effect of the p_e and σ_e values on VoR. We observed the expectation and the VoR for all terminals when all p_e have identical value p and all σ_e also have identical value σ. We conducted the above computation for various p and σ values. Figure 7.5 plots the reliability and $\sqrt{\text{VoR}}$ for all terminals as a function of p when σ is set to $0.01, \ldots, 0.04$. Figure 7.5 shows a tendency that $\sqrt{\text{VoR}}$ became larger when the reliability value rapidly increased with respect to p, but it again suggests that its value remains in the same degree as σ, the uncertainty in each edge's availability. We also see that $\sqrt{\text{VoR}}$ is almost proportional to σ for every p value.

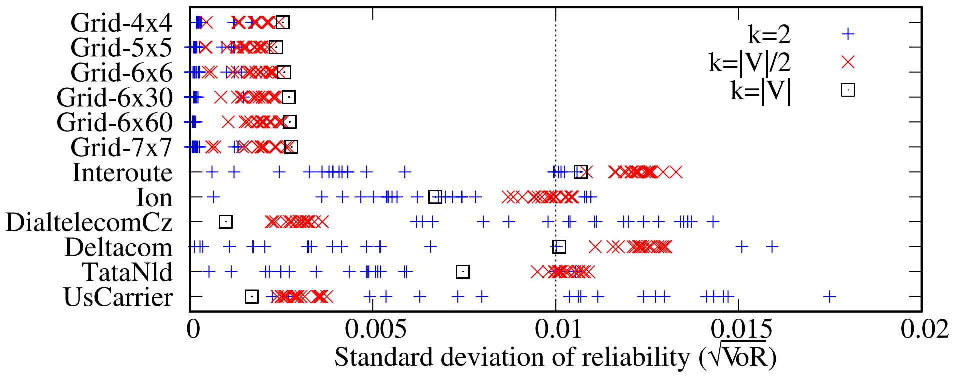


Figure 7.3: Values of standard deviation of reliability ($\sqrt{\text{VoR}}$) computed in experiment of Section 7.4.1. Standard deviation of each edge's availability (0.01) is indicated by a dashed line.

7.5 Conclusion

This chapter made the following two contributions: (i) We proposed an algorithm that computes VoR with analyses of computational complexities. Its practical utility was confirmed by experiments, in which our algorithm computed VoR within 0.1 seconds for real network topologies with nearly 200 links. (ii) We conducted empirical analyses for VoR. Such analyses revealed its good properties. Its value is the same degree as the variance of the link availability, and it exhibits robustness against some links' significant variance of availability.

As future works, we will develop algorithms for computing VoR under other variants of network models. One direction is to consider vertex failures as well as link failures [Kawahara *et al.*, 2019]. Another direction is to deal with dependent failures of links and vertices by getting rid of the assumption of independentness, e.g., [Xiao *et al.*, 2009; Botev *et al.*, 2012].

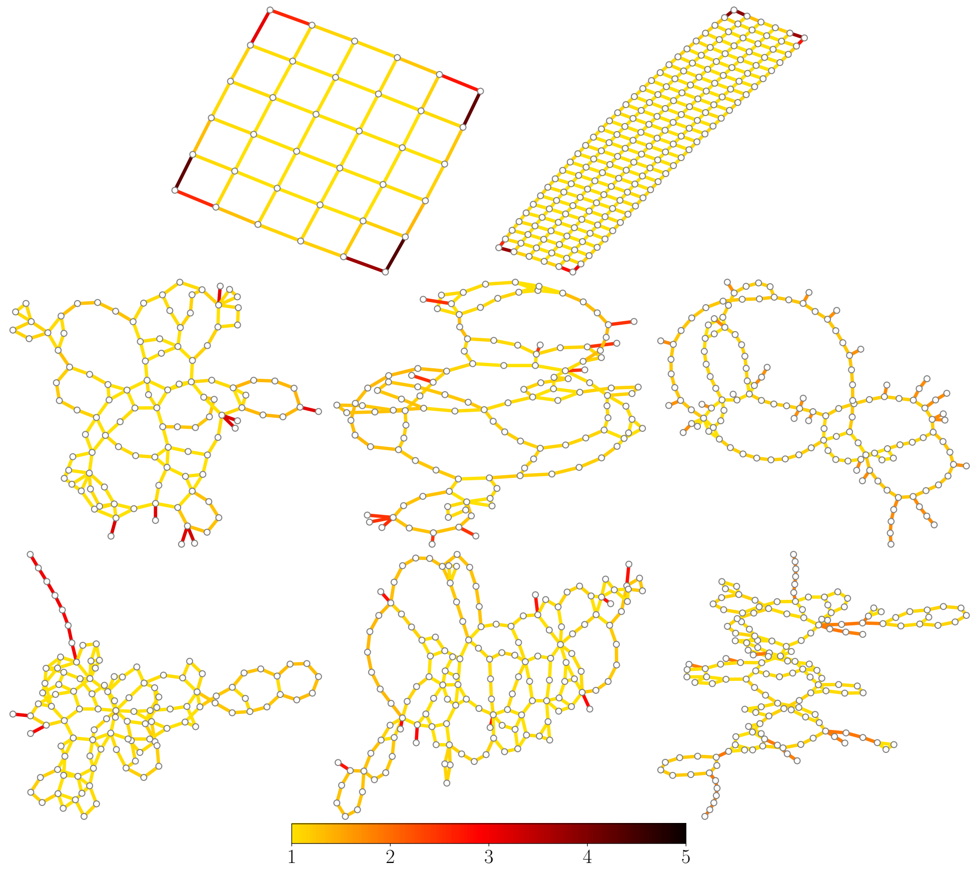


Figure 7.4: Heatmaps indicating how many times $\sqrt{\text{VoR}}$ for all terminals increased when standard deviation of availability of an edge became 10 times larger (from 0.01 to 0.1): Grid-6x6, Grid-6x30 (first row), Interoute, Ion, DialtelecomCz (second row), Deltacom, TataNld and UsCarrier (third row).

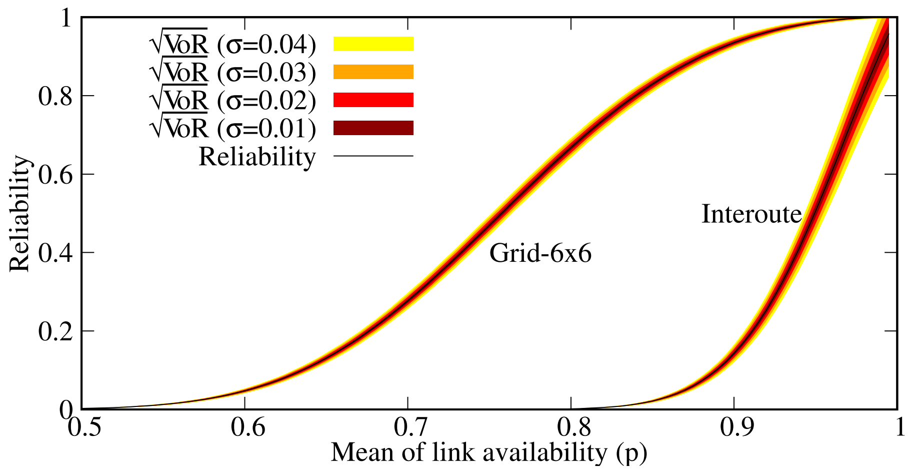


Figure 7.5: All-terminal reliability and its standard deviation ($\sqrt{\text{VoR}}$) when mean (p) and standard deviation (σ) of availabilities are all identical. Width of vertical slice represents twice of $\sqrt{\text{VoR}}$.

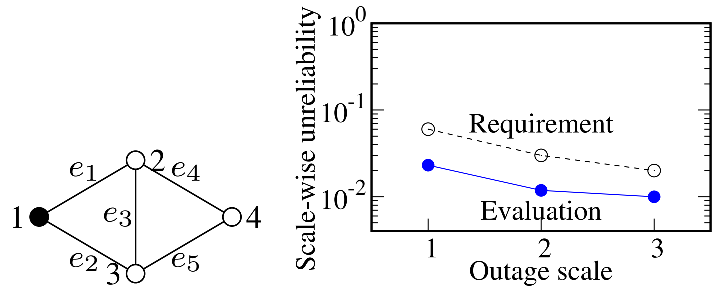
Chapter 8

Efficient Computation of Scale-wise Network Unreliability

In communication networks, the significance of an outage is measured mainly by its scale (number of disconnected nodes). To avoid serious outages, operators design their networks so that the reliability meets the specification for each outage scale, where the more significant the outage, the less likely it is to occur. Although such a scale-wise unreliability has been evaluated with rough approximation, sixth-generation (6G) mobile communication requires more accurate reliability evaluation with seven 9's accuracy. Unfortunately, accurate scale-wise reliability evaluation is a computationally very tough problem, so no previous literature has studied evaluation methods rigorous enough. This chapter proposes an efficient algorithm to exactly compute the probability for each number of disconnected nodes. The proposed algorithm performs the scale-wise unreliability evaluation in a DP manner without redundant repetition for each outage scale. Numerical experiments using real network topologies show its great efficiency, e.g., our algorithm computes exact probabilities for every outage scale in just two hours for a network with nearly 200 links. We also provide several interesting insights on the reliability of real topologies from the scale-wise perspective, since our work is the first to present the scale-wise unreliability of real large topologies.

8.1 Introduction

The significance of a communication network outage is traditionally measured by the duration and scale (number of affected users) [Tollar and Ben-



(a) Example network (b) Scale-wise unreliability

Figure 8.1: (a) Example communication network. (b) Scale-wise unreliability of network (a).

nett, 1995; Matsukawa and Funakoshi, 2010]. For example, an outage that affects 900,000 user minutes in the US or that affects 30,000 users for one hour in Japan is considered "significant" and must be reported to the network's supervisory agency [Federal Communications Commission; Ministry of Internal Affairs and Communications]. To avoid significant outages, operators first specify a reliability requirement, i.e., the distribution of admissible outage probability for each scale [Nojo and Watanabe, 1987; Fukuda and others, 2020] (Figure 8.1b). Then, operators design their networks to meet the requirement, i.e., restrict the reliability of a network under design to a level within the requirement [Nojo and Watanabe, 1993].

Conventionally, scale-wise network reliability has been evaluated approximately assuming a limited number of outage scenarios [Taka and Abe, 1994; Watanabe *et al.*, 2003]. However, the requirements of sixth-generation (6G) mobile communications feature the "extreme high reliability" up to 99.99999% [Alwis *et al.*, 2021; NTT Docomo, 2022], which requires network reliability evaluation methods to be rigorous without approximation. In addition, accurate reliability evaluation has to be performed for every outage scale. To our knowledge, no work has successfully performed scale-wise reliability evaluation in a precise manner.

Let us see an example of network shown in Figure 8.1a, where every edge fails with 10% probability and we assume an independent failure model. The All-NR, i.e., the probability that all nodes are connected, is 97.686%. However, the traditional definition of network reliability does not take into account outage scales.

The problem studied in this chapter is also conceptually outlined in Figure 8.1. The red node 1 plays the role of a "server" (e.g., a packet gateway or edge server), and the other nodes (e.g., edge/aggregation switches or base stations) need to be connected to the server. Network outage occurs if some nodes are disconnected from the server. The outage scale is defined by the number of disconnected nodes in this chapter. In Figure 8.1, if e_1 and e_2 are absent and the others are present, nodes 2, 3, and 4 are disconnected, so the outage scale is 3 and the state probability is $0.9^3 \times 0.1^2 = 0.00729$. Here, there are 8 states where the outage scale is 3, and the total probability of scale-3 outage states is 0.01. Our problem is to draw the probability distribution per outage scale, as shown in Figure 8.1b. The probability distribution is called the *scale-wise unreliability* in this dissertation.

The contributions of this chapter are summarized as follows.

- This chapter analyzes why existing network reliability evaluation methods are unsuitable for scale-wise unreliability evaluation. The existing methods evaluate reliability only for a specified subset of nodes, so we need to perform an existing method for each of an exponential number of node combinations.

- Based on the above analysis, we develop an algorithm to exactly evaluate the scale-wise unreliability. The proposed algorithm is designed as a DP to track connected client counts without explicitly managing an exponential number of combinations. The time complexity analysis shows that our algorithm does not depend on the exponential factor of node counts.

- Numerical experiments using real communication network topologies show that our algorithm successfully evaluates the scale-wise unreliability for a large network of nearly 200 links within a few minutes in most cases and within 100 minutes at most. In addition, because this is the first work to present the scale-wise unreliability of real topologies, we offer interesting insights on network reliability from the scale-wise perspective.

Note that this chapter focuses on scale-wise unreliability evaluation, so how to specify reliability requirements is out of the study's scope [Nojo and Watanabe, 1987; Fukuda and others, 2020] (dashed line in Figure 8.1b).

8.2 Problem Statement

We use an independent failure model described in Section 4.1. For the problem in this chapter, vertex sets $T, C \subseteq V$ called *server vertex set* and *client vertex set* are given. Here, our focus is on the number of client vertices that fail to connect with any vertex in T. Let $U(n')$ $(n' = 0, \dots, |C|)$ be the probability that *exactly* n' vertices out of the vertices in C fail to connect with the vertices in T. This probability can be written as

$$U(n') = \sum_{E' \in \mathcal{E}^{=n'}} \Pr(E') = \sum_{E' \in \mathcal{E}^{=n'}} \left[\prod_{e \in E'} p_e \cdot \prod_{e \in E \setminus E'} (1 - p_e) \right], \tag{8.1}$$

where $\mathcal{E}^{=n'}$ is the family of edge subsets $E' \subseteq E$ such that the number of vertices in C that do not connect to any vertex in T on the subgraph induced by E' is exactly n'. In other words, $E' \in \mathcal{E}^{=n'}$ if and only if n' client vertices are disconnected from servers when the edges in E' are present and the others are absent. The *scale-wise unreliability*, denoted by $S(n')$ for $n' = 1, \ldots, |C|$, is the probability that *more than or equal to* n' vertices out of the vertices in C fail to connect with any vertex in T, and it can be represented as

$$S(n') = \sum_{k=n'}^{|C|} U(k). \tag{8.2}$$

Formally, the problem we solve in this chapter can be described as follows.

Problem 8.1 (Scale-wise unreliability computation)**.** Given server vertex set T and client vertex set C, the task is to compute scale-wise unreliability $S(n')$ for every $n' = 1, \ldots, |C|$.

For example, for the graph of Figure 8.2a and $T = \{1\}$, $C = V$, $\mathcal{E}^{=3} = \{\{\}, \{e_3\}, \{e_4\}, \{e_3, e_4\}, \{e_5\}, \{e_3, e_5\}, \{e_4, e_5\}, \{e_3, e_4, e_5\}\}$. If every edge has an availability 0.9, we have $S(3) = U(3) = 0.01$. Similarly, we have $S(2) = U(2) + U(3) = 0.0118$ and $S(1) = 0.02314$.

8.2.1 Straightforward Approach Using HH Method

For evaluating $S(n')$ with scale $n' = 1$, when $|T| = 1$, $U(0) = 1 - S(1)$ equals the $(C \cup T)$-terminal network reliability, i.e. the probability that all the vertices in $C \cup T$ are interconnected. Since computing it is #P-complete [Valiant, 1979], this connection reveals us that the computation of scale-wise unreliability is #P-hard, a computationally challenging class. In contrast, it can be computed by the HH method described in Section 4.3. This method can easily be adopted for the case $|T| \geq 2$ by combining with Apply algorithm [Bryant, 1986] on BDDs. Similar to this, each client vertex's probability of connecting to servers, i.e., $R_0(T, v)$ for client vertex v, can also be computed by their methods or the method in Chapter 6. Note that $R_0(T, v)$ is defined at (6.1) in Section 6.2.

However, computing $R_0(T, v)$ for every client v is not sufficient to evaluate $S(n')$ for $n' \geq 2$. Consider the probabilistic event that vertex $v \in C$ is connected to some vertices in T. Such events are stochastically dependent on different vertices $v \neq v'$. For example, for the graph of Figure 8.2a, if vertices 2 and 3 are not connected to vertex 1, vertex 4 cannot be connected to vertex 1. Therefore, running HH method $O(n)$ times or running the method in Chapter 6 is not sufficient to evaluate the scale-wise unreliability.

A straightforward way to solve it is to examine, for each *subset* $V' \subseteq C$ of vertices, the probability that the vertices in V' are connected to servers and other client vertices are disconnected from servers. This probability can also be computed by the HH method combined with the Apply algorithm [Bryant, 1986] on BDDs. All $S(n')$ values can be computed by conducting this computation for all of the subsets $V' \subseteq C$. However, since there are $2^{|C|}$ vertex subsets, the computational cost is prohibitively large. We empirically compare this method, as the baseline method, with the proposed method in Section 8.4.

8.3 Method

We try to compute $U(n')$ values for every $n' = 0, 1, \ldots, |C|$ since the scale-wise unreliability value $S(\cdot)$ can easily be recovered by (8.2). Now, recall the HH method to compute K-NR. In HH method, configures that is enough to distinguish whether all the vertices in K are interconnected are designed, and the top-down construction method described in Section 2.5 is used. As a result, BDD $\mathtt{B}_K$ representing $\mathcal{E}_K$ is built. The resulting diagram collects the edge subsets in $\mathcal{E}_K$ as the paths from root $\mathtt{r}$ to $\top$ terminal, and in fact, the other edge subsets are collected as the path from root to $\perp$ terminal Therefore, we can see $\mathtt{B}_K$ as a data structure to classify the edge subsets into those in $\mathcal{E}_K$ and the others.

The proposed method extends this idea: we try to construct a data structure to classify the edge subsets with respect to the number of client vertices disconnected from any servers. Since there are $|C|+1$ cases, from 0 to $|C|$, for the number of client vertices, this can be achieved by preparing $|C|+1$ kinds of terminals instead of only two terminals ($\top$ and $\perp$). To build such data structures, we modify the top-down construction framework of Section 2.5. Below we start our detailed explanation with a naive exhaustive enumeration for the sake of completeness since the built data structure is no longer a standard BDD.

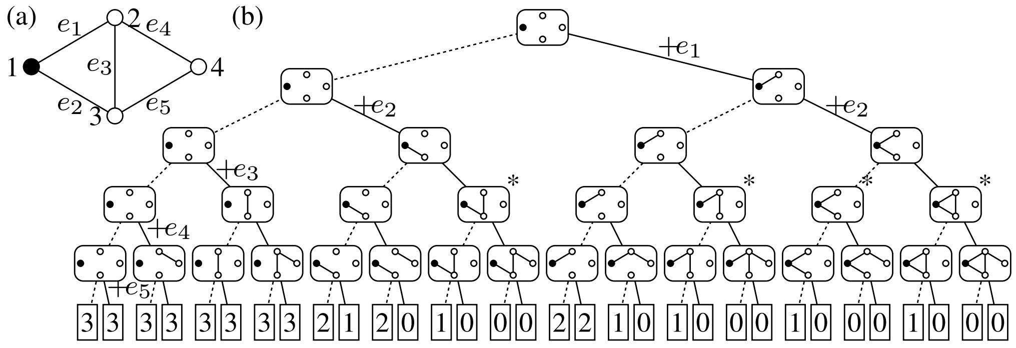


Figure 8.2: (a) Example of graph G: Filled vertices are in T. Here all vertices are client vertices. (b) Exhaustive enumeration of edge subsets on graph (a), where it forms a complete binary tree. Dashed and solid lines indicate e_i is excluded and included, respectively.

8.3.1 Naive Exhaustive Enumeration

We start with a naive way to compute $U(n')$ for every n'. Given predefined edge order $e_1, \dots, e_m$, we choose whether e_i is present or absent one by one according to this order. This process can be seen as traversing a binary decision tree like Figure 8.2b. After determining every edge's state, we can count the number of client vertices disconnected from the server vertices as in the terminals, i.e. the bottom, of the tree like Figure 8.2b. The probability that edges in $E' \subseteq E$ are present and the others are absent can be computed by the following *path product*: From root to terminal, if a solid line is traversed at the i-th step, p_{e_i} is multiplied, and if a dashed line is traversed at the i-th step, $(1 - p_{e_i})$ is multiplied. The probability $U(n')$ can be obtained from the sum of path products of all the paths from root to n' terminals. For example, the probability that edges $\{e_1, e_3, e_5\}$ are present and the others are absent equals the path product of the path (solid)-(dashed)-(solid)-(dashed)-(solid) from the root, $p_{e_1}(1-p_{e_2})p_{e_3}(1-p_{e_4})p_{e_5}$, which is added to $U(0)$ because this path arrives at "0" terminal.

8.3.2 Equivalence to Reduce Computation

Since there are 2^m edge subsets, such a procedure is intractable even for graphs with dozens of edges. To accelerate computation, we identify some edge subsets by defining configures. For example, on the 4th level of Figure 8.2b, four vertices marked with an asterisk (*) have an identical pattern of terminals, "1 0 0 0." We show that such an equivalence can reduce the computation.

We now define the equivalency. As in Chapter 4, let $E_{<i} := \{e_1, \ldots, e_{i-1}\}$, $E_{\geq i} := \{e_i, \ldots, e_m\}$, and we call the subset of $E_{<i}$ an *i-th subset*. All of the i-th subsets are now enumerated at the i-th level of a binary decision tree like Figure 8.2b. We consider the following condition for the i-th subsets X and Y:

> (*) For any $Z \subseteq E_{\geq i}$, the number of client vertices disconnected from any server with edges in $X \cup Z$ and that with edges in $Y \cup Z$ are equal.

Note that this can be seen as an extension of condition (i) on configures to construct standard BDD described in Section 2.5. Condition (i) requires distinguishing two cases whether the subset is in a desired family, while condition (*) requires distinguishing $|C| + 1$ cases.

Given i-th subset X, let $\mathcal{E}_X^{=n'} \subseteq 2^{E_{\geq i}}$ be the family of subsets $Z \subseteq E_{\geq i}$ such that the number of clients disconnected from servers on the subgraph induced by $X \cup Z$ is exactly n', and let $\mathcal{X} = \{X_1, \ldots, X_t\}$ be the i-th subsets satisfying condition (*) with each other. Condition (*) tells us that $\mathcal{E}_{X_j}^{=n'} = \mathcal{E}_{X_k}^{=n'}$ for $j \neq k$ and $n' = 0, \ldots, |C|$. So, we let $\mathcal{E}_{\mathcal{X}}^{=n'} := \mathcal{E}_{X_j}^{=n'}$ by arbitrarily choosing j. Thus, on the binary decision tree like Figure 8.2b, the subtrees rooted at $X_1, \ldots, X_t$ are identical.

Now we consider merging these subtrees into one rooted at a new dnode $\mathcal{X}$ to make a diagram. Even after merging them, $U(n')$ equals the sum of path products from root to n' terminals. This is because merging them does not change the set of paths from root to n' terminals; the path in the binary decision tree passing through X_i corresponds to that in the DAG passing through $\mathcal{X}$.

Moreover, we can reduce the computation by this diagram. Among all the path products contributing to $U(n')$, those passing through any of $X \in \mathcal{X}$ can be represented as

$$\begin{aligned}
&\sum_{X\in\mathcal{X}} \left[\prod_{e\in X} p_e \cdot \prod_{e\in E_{<i}\setminus X}(1-p_e) \cdot \sum_{Z\in\mathcal{E}_X^{=n'}} \left[\prod_{e\in Z} p_e \cdot \prod_{e\in E_{\geq i}\setminus Z}(1-p_e)\right]\right] \\
&= \sum_{X\in\mathcal{X}} \left[\prod_{e\in X} p_e \cdot \prod_{e\in E_{<i}\setminus X}(1-p_e)\right] \cdot \sum_{Z\in\mathcal{E}_{\mathcal{X}}^{=n'}} \left[\prod_{e\in Z} p_e \cdot \prod_{e\in E_{\geq i}\setminus Z}(1-p_e)\right], \qquad (8.3)
\end{aligned}$$

since $\mathcal{E}_{X_j}^{=n'} = \mathcal{E}_{\mathcal{X}}^{=n'}$ for all j. By considering the diagram, the first factor is the sum of path products from root to $\mathcal{X}$, and the second factor is that of path products from $\mathcal{X}$ to any n' terminal. Here, the point is that the second factor of (8.3) needs to be computed only once instead of computed for every X_j.

In the following, Section 8.3.3 discusses the configure to detect such equivalence and Section 8.3.4 describes the procedures of the proposed algorithm.

8.3.3 Partition and Number-map

In the following, we define two concepts, *partition* and *number-map*, for an i-th subset. Our goal is to prove that if the partition and number-map are identical for different i-th subsets, they satisfy condition (*) (Theorem 8.4). In other words, our goal is to prove that they can be used as a configure. Eventually, the definition of partition is precisely the same as in Chapter 6 (Definition 6.2), while the number-map is a newly proposed concept in this chapter.

It is obvious that two i-th subsets X and Y satisfy condition (*) if the connectivities among vertices are identical, i.e., for any two vertices $u, v \in V$, whether u and v are connected is equivalent for two subgraphs (V, X) and (V, Y). However, such a condition is a bit excessive. Here we focus on an i-th subset X. We again use frontier vertices F_i, processed vertices A_i, and unprocessed vertices B_i defined in Definition 4.3. We extract the connectivity among frontier vertices as the partition (Definition 8.2). In addition, since condition (*) counts the number of vertices disconnected from servers, we also keep track of the number of connected vertices of each frontier vertex, which constitutes the number-map (Definition 8.3).

First, we define the partition.

Definition 8.2. A *partition* for the i-th subset $X \subseteq E_{<i}$ is a partitioning of the i-th frontier vertices F_i into *blocks* consisting of exactly one *special block* and other *normal blocks.* Vertex $v \in F_i$ is in a special block if and only if v is connected to at least one vertex in T on the subgraph induced by X. vertices $u, v \in F_i$ not contained in a special block are in the same normal block if and only if they are interconnected on the subgraph induced by X. For partition $\mathcal{P}$, its special block is denoted by $\mathsf{S}^{\mathcal{P}}$ and the block containing vertex $v \in F_i$ is denoted by $\mathsf{B}_v^{\mathcal{P}}$.

Next, we define the number-map.

Definition 8.3. For the i-th subset X whose partition is $\mathcal{P}$, let num be a mapping from each block of $\mathcal{P}$ to an integer value defined as follows: For special block $\mathsf{S}^{\mathcal{P}}$, $\mathsf{num}(\mathsf{S}^{\mathcal{P}})$ is the number of vertices in C connected with some vertices in T on the subgraph induced by X. For normal block B, $\mathsf{num}(B)$ is the number of vertices in C included in the corresponding connected component on the subgraph induced by X. We call this num a *number-map.*

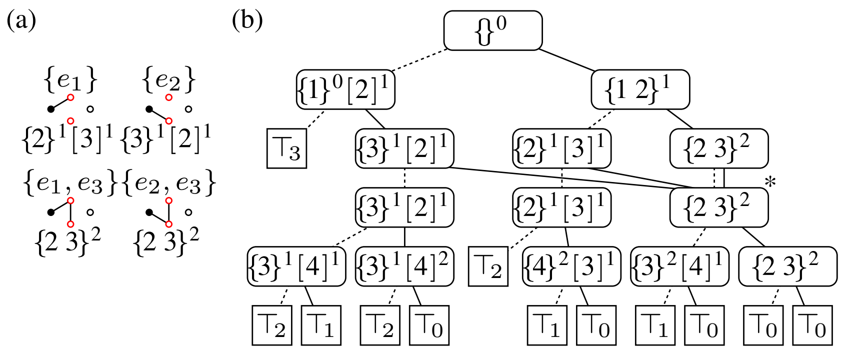


Figure 8.3: (a) Examples of 4th subsets of graph (a) and their partitions and number-maps. Red vertices are 4th frontier vertices. (b) Decision diagram with partitions and number-maps on the graph of Figure 8.2(a).

The special block and each normal block are represented by curly brackets ($\{\}$) and square brackets ([ ]) with vertex ids inside them, respectively. The value of number-map of each block is written on the upper-right of it; for example, $\{1\ 4\}^2[3]^5$ means that the partition is $\{1\ 4\}[3]$ and the number-map is $\mathsf{num} = \{\{1\ 4\} \mapsto 2, [3] \mapsto 5\}$. Figure 8.3a depicts some examples of the 4th subsets and its corresponding partitions and number-maps. Note that $F_4 = \{2, 3\}$. Here, the partitions and number-maps of $\{e_1, e_2\}$ and $\{e_1, e_3\}$ are identical.

Now we prove the following.

Theorem 8.4. *For two i-th subsets X and Y, suppose that their partitions and number-maps are identical for each. Then, condition (*) holds.*

Proof. Given $W \subseteq E_{<i}$ and $Z \subseteq E_{\geq i}$, let $V_{W,Z} \subseteq V \setminus A_i$ be the vertices that are not connected to any server with the edges in W but connected to servers by adding the edges in Z. Since X and Y have the same partition, the connectivity among $V \setminus A_i$ is identical, i.e., for $u, v \in V \setminus A_i$, u and v are connected on the subgraph induced by X if and only if they are connected on the subgraph induced by Y. Moreover, since the edges in $E_{\geq i}$ are not incident to the vertices in A_i, $V_{X,Z} = V_{Y,Z}$ for any $Z \subseteq E_{<i}$.

Let $(\mathcal{P}, \mathsf{num})$ be the partition and the number-map of X and Y, and let $\mathcal{B}$ be the set of the blocks of $\mathcal{P}$ containing at least one vertex in $V_{X,Z}(= V_{Y,Z})$. Then, the number of client vertices connected to servers with the edges in $X \cup Z$ equals

$$\mathsf{num}(\mathsf{S}^{\mathcal{P}}) + \sum_{B \in \mathcal{B}} \mathsf{num}(B) + |(V_{X,Z} \setminus F_i) \cap C|, \qquad (8.4)$$

where the first term is the number of already connected clients on the subgraph induced by X, the second factor is that of newly connected clients in $A_i \cup F_i$ by adding the edges in Z, and the third factor is that of newly connected clients in $V \setminus (A_i \cup F_i)$. Since $V_{X,Z} = V_{Y,Z}$, the number of client vertices connected to servers with the edges in $Y \cup Z$ also equals (8.4). $\square$

Theorem 8.4 suggests that two subsets $X, Y \subseteq E_{<i}$ can be identified if the partition and the number-map are identical. Moreover, we can easily derive the following.

Lemma 8.5. *Let X, Y be the i-th subsets whose partitions and number-maps are identical. Then, the partitions and number-maps of X, Y and those of $X \cup \{e_i\}, Y \cup \{e_i\}$ (seen as $(i+1)$-st subsets) are also identical for each.*

This can be proved in a similar way as Theorem 8.4: X and Y have the same connectivity among $V \setminus A_i$, so X and Y (or $X \cup \{e_i\}$ and $Y \cup \{e_i\}$) also have the same connectivity among $V \setminus A_{i+1}$. The number of client vertices connected to each block is also the same for X and Y (or $X \cup \{e_i\}$ and $Y \cup \{e_i\}$) since the number-map is identical. Theorem 8.4 and Lemma 8.5 constitutes a basis for using the pair of partition and number-map as a configure to construct a diagram.

8.3.4 Proposed Method

First, we describe the top-down construction of diagram. Since we construct a diagram with many terminals, we slightly modify the top-down construction framework. We prepare $|C| + 1$ kinds of terminals, $\top_0, \top_1, \ldots, \top_{|C|}$. $\top_k$ means that the number of disconnected clients from servers is exactly k. The top-down construction framework is altered so that we set $\top_k$ instead of setting $\top$ or $\bot$ when receiving $\langle m+1, \cdot \rangle$. The modified framework is in Algorithm 8.1.

The next step is to design ROOT and CHILD procedures. Since the partition's definition is identical to that in Chapter 6, we can borrow the same procedure for updating partition as Algorithm 6.1. The maintenance of number-map is similar to that of mark defined in Definition 5.3: the mark maintains whether each block is connected to any vertex in T, while the number-map maintains the number of connected clients to each block.

We describe the procedure for constructing diagram in Algorithm 8.2. Finally, CONSTRUCTMULTI($\mathit{ScaleUnreliability}_{T,C}$) constructs the diagram. Figure 8.3b is the built diagram when the input graph is Figure 8.2a.

The remaining is how to compute all the $U(n')$ values using this diagram. This is almost identical to the top-down DP computation demonstrated in

Algorithm 8.1: Top-down construction framework.

```
procedure ConstructMulti($\mathcal{S}$):
    $\langle i_0, s_0\rangle \leftarrow \mathcal{S}$.Root()
    Create root dnode $\mathtt{r} \in \mathtt{L}_{i_0}$ whose configure is $s_0$
    for $i = i_0$ to $m$ do
        foreach $\mathtt{n} \in \mathtt{L}_i$ do
            foreach $f \in \{\mathrm{lo}, \mathrm{hi}\}$ do
                $s \leftarrow$ ($\mathtt{n}$'s configure)
                $\langle i', s'\rangle \leftarrow \mathcal{S}$.Child($\langle i, s\rangle, f$)
                if $\langle i', s'\rangle = \langle m+1, k\rangle$ for some $k$ then Set $\top_k$ to the $f$-child of $\mathtt{n}$
                else
                    if $\exists \mathtt{n}' \in \mathtt{L}_{i'}$ s.t. configure is $s'$ then
                        Set $\mathtt{n}'$ to the $f$-child of $\mathtt{n}$
                    else
                        Create node $\mathtt{n}'' \in \mathtt{L}_{i'}$ whose configure is $s'$
                        Set $\mathtt{n}''$ to the $f$-child of $\mathtt{n}$
```

Algorithm 8.2: Constructing decision diagram.

```
procedure $ScaleUnreliability_{T,C}$.Root():
    return $\langle 1, (\mathcal{P}^{\mathrm{r}} \coloneqq \{\}, \{\mathsf{S}^{\mathcal{P}^{\mathrm{r}}} \mapsto 0\})\rangle$
procedure $ScaleUnreliability_{T,C}$.Child($\langle i, (\mathcal{P}, \mathsf{num})\rangle, f$):
    foreach $x \in \{v, v'\} \setminus F_i$ do                                  // $e_i = \{v, v'\}$
        if $x \in T$ then
            Insert $x$ into $\mathsf{S}^{\mathcal{P}}$
            if $x \in C$ then $\mathsf{num}(\mathsf{S}^{\mathcal{P}})$ += 1
        else
            Insert $[x]$ as a new normal block of $\mathcal{P}$
            $\mathsf{num}([x]) \leftarrow 1$ if $x \in C$; 0 otherwise
    if $f = \mathsf{hi}$ and $\mathsf{B}^{\mathcal{P}}_v \neq \mathsf{B}^{\mathcal{P}}_{v'}$ then
        Merge $\mathsf{B}^{\mathcal{P}}_v$ and $\mathsf{B}^{\mathcal{P}}_{v'}$ into a new block $B$; if either is a special block, $B$ is
          special
        $\mathsf{num}(B) \leftarrow \mathsf{num}(\mathsf{B}^{\mathcal{P}}_v) + \mathsf{num}(\mathsf{B}^{\mathcal{P}}_{v'})$
        Remove entries $\mathsf{num}(\mathsf{B}^{\mathcal{P}}_v)$ and $\mathsf{num}(\mathsf{B}^{\mathcal{P}}_{v'})$
    foreach $x \in \{v, v'\} \setminus F_{i+1}$ do                              // Completed vertices
        Remove $x$ from $\mathsf{B}^{\mathcal{P}}_x$
    Remove all empty normal blocks from $\mathcal{P}$ and all corresponding entries from
      $\mathsf{num}$
    if all vertices in $T$ are processed and $\mathsf{S}^{\mathcal{P}}$ is empty then
        return $\langle m+1, |C| - \mathsf{num}(\mathsf{S}^{\mathcal{P}})\rangle$
    else
        return $\langle i+1, (\mathcal{P}, \mathsf{num})\rangle$
```

Algorithm 8.3: DP computation.

```
1 Set all DP and r values to zero except that DP[r] ← 1 for root r
2 for $i \leftarrow 1$ to $m$ do
3     foreach $\mathtt{n} \in \mathtt{L}_i$ do
4         foreach $\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}$ do
5             $\mathsf{DP}[\mathsf{f}(\mathtt{n})] \mathrel{+}= \mathsf{f}(p_{e_i}) \cdot \mathsf{DP}[\mathtt{n}]$        // (8.5); $\mathsf{hi}(p_e) = 1 - \mathsf{lo}(p_e) = p_e$
6 Output every $\mathsf{DP}[\top_{n'}]$ value as $U(n')$
```

Chapters 5 and 6. Recall that $U(n')$ can be computed as the sum of path products of all the paths from root $\mathtt{r}$ to $\top_{n'}$ terminal. Thus, for dnode $\mathtt{n}$, let $\mathsf{DP}[\mathtt{n}]$ be the sum of path products of all the paths from root $\mathtt{r}$ to $\mathtt{n}$. Similar to $\mathsf{DP}_\downarrow$ in Chapter 5, we have the following recursive formula, where $i = \mathsf{lb}(\mathtt{n})$:

$$\mathsf{DP}[\mathtt{n}] = (1-p_{e_{i-1}}) \cdot \sum_{\mathtt{n}' \in \mathtt{L}_{i-1} : \mathsf{lo}(\mathtt{n}') = \mathtt{n}} \mathsf{DP}[\mathtt{n}'] + p_{e_{i-1}} \cdot \sum_{\mathtt{n}' \in \mathtt{L}_{i-1} : \mathsf{hi}(\mathtt{n}') = \mathtt{n}} \mathsf{DP}[\mathtt{n}']. \tag{8.5}$$

Similarly, $U(n')$ equals

$$\sum_i \left[(1-p_{e_i}) \cdot \sum_{\mathtt{n} \in \mathtt{L}_i : \mathsf{lo}(\mathtt{n}) = \top_{n'}} \mathsf{DP}[\mathtt{n}] + p_{e_i} \cdot \sum_{\mathtt{n} \in \mathtt{L}_i : \mathsf{hi}(\mathtt{n}) = \top_{n'}} \mathsf{DP}[\mathtt{n}] \right]. \tag{8.6}$$

Algorithm 8.3 performs these computations.

Finally, the proposed algorithm can be summarized as follows: First, we construct a decision diagram by ConstructMulti($\mathit{ScaleUnreliability}_{T,C}$) with Algorithms 8.1 and 8.2. Then, we compute every $U(n')$ value with Algorithm 8.3.

8.3.5 Complexity

We derive a time complexity bound for the proposed algorithm. We again use the frontier width $W_F = \max_i |F_i|$ in the analysis.

Theorem 8.6. *The proposed algorithm runs in $O(mW_F^2 E_{W_F}(|C|+1)^{W_F})$ time, where E_{W_F} is W_F-th Bell number.*

Proof. The number of possible partitions of F_i is at most $(|F_i|+1)B_{|F_i|} = O(W_F E_{W_F})$ because the number of divisions of F_i is at most $E_{|F_i|}$ and for each division the number of candidates for the special block is at most $|F_i|+1$ (including the case where the special block is empty). Similarly, the number of possible number-maps for each partition is at most $(|C|+1)^{W_F}$, since each num value ranges from 0 to $|C|$. Thus, there are at most $O(mW_F E_{W_F}(|C| +$

$1)^{W_F}$) pairs of partition and number-map for the graph. For each dnode, i.e., for each pair of partition and number-map, Algorithm 8.2 costs $O(W_F)$ time and Algorithm 8.3 costs $O(1)$ time. $\square$

Generally, the frontier width W_F is far smaller than n and m, i.e., the numbers of vertices and edges. This makes the proposed algorithm much faster than the baseline method.

8.4 Experiments

8.4.1 Network Topologies Used in Experiments

We first mention the topologies used in the experiments. The network topologies were derived from the Internet Topology Zoo [Knight *et al.*, 2011] dataset. From each topology, we extracted the largest connected component and removed all the self-loops that do not affect the connectivity among vertices. Topologies of more than 30 edges were taken as instances, except for one overly large graph with 895 edges (Kdl).

Servers were determined in the same way as Chapter 5: For each graph, we computed the betweenness centrality [Freeman, 1977] of each vertex and chose $|T| = 1, 5, 10$ vertices with higher centrality as servers T. This is because servers, e.g., packet gateways and edge servers, are located so that they are easily accessible from clients [Zhang *et al.*, 2021]. Clients were always set to $C = V$, i.e., all the vertices, because this maximizes the computational load for both methods. Each working probability p_e is decided as follows: (I) If both endpoints of edge e have information of the latitude and longitude, we computed geodesic distance d (km) between them and set $p_e = 1 - 4.863d \times 10^{-6}$, following the reliability statistics of optical fiber and amplifier [Segovia *et al.*, 2008]. (II) Otherwise, p_e is set to the average availability of edges of type (I). Note that 82.4% of the edges are type (I).

8.4.2 Computation Time

We evaluated the proposed method and the baseline method (Section 8.2.1) by the computation time needed to evaluate all $U(n')$ values. All methods were implemented in C++11 and compiled by g++-4.8.5 with `-O3` option. For the baseline method, we used TdZdd (`https://github.com/kunisura/TdZdd`) for implementing [Hardy *et al.*, 2007] and SAPPOROBDD (`https://github.com/Shin-ichi-Minato/SAPPOROBDD`) for implementing BDD manipulations. Throughout the experiment, we used a single thread of a Linux machine with AMD EPYC 8863 2.45 GHz CPU and 2048 GB of RAM; note that the proposed method used less than 64 GB of memory except for one topology (Interoute). We set the time limit of each run to 12 hours. The edge order is commonly determined for both methods according to beam-search based heuristics [Inoue and Minato, 2016].

Table 8.1 shows the results. Here TO indicates that the computation was not finished within the time limit. It is obvious from Table 8.1 that the proposed method is far more efficient than the baseline method. For topologies of less than 100 edges, each computation of the proposed method was finished within a second, while that of the baseline exceeded the time limit in about half of the instances. Even for topologies with more than 100 vertices and edges, the proposed method computed all $U(n')$ values within two hours for Interoute and within 20 minutes for other topologies. Here the consumed time of the proposed method heavily depends on W_F rather than n and m; it is consistent with the complexity results in Section 8.3.5. It is also observed that both methods ran faster when $|T|$ is large. For the baseline method, this is because the server vertices can be safely ignored when considering all vertex subsets of clients. For the proposed method, the reason is as follows: When there are more servers, more additional vertices tend to be connected with servers, which means that they are in the special block. This decreases the variety of the pair of partition and number-map and thus the computational cost also decreases.

Typically, the proposed method was also efficient in terms of memory usage; e.g., for Intranetwork $|T| = 1$ instance, the proposed method used less than 32 MB while the baseline needs more than 6 GB. Note that the peak usage of the proposed method was 200.2 GB recorded by Interoute $|T| = 1$.

8.4.3 Scale-Wise Unreliability in Real Networks

Here, for the first time, we present scale-wise unreliability for the real topologies; it has never been published before, including approximate evaluation. Figure 8.4 plots the scale-wise unreliability $S(n')$, i.e., the probability that *more than or equal to* n' vertices are disconnected from servers, for each scenario of topologies with more than 100 edges. The results show that the scale-wise unreliability greatly varies among topologies. Deltacom and UsCarrier show high unreliability for small n' (x-axis), but this decreases rapidly with n'. This indicates that large-scale failures are unlikely to occur. On the contrary, TataNld, Ion, and DialtelecomCz show a rapid decrease in unreliability for small n' but show high plateaus in large n'. This indicates that they usually have few outages but can have significant failures with a low probability. Our algorithm is the first to show differences in scale-wise unreliability, and it opens up a new avenue in the research of network relia-

Table 8.1: Computation time for scale-wise unreliability in seconds.

				$\|T\| = 1$		$\|T\| = 5$		$\|T\| = 10$	
Instance	n	m	W_F	**Ours**	Base	**Ours**	Base	**Ours**	Base
Funet	26	30	3	<**0.1**	2.8	<**0.1**	1.0	<**0.1**	0.1
Darkstrand	28	31	3	<**0.1**	9.4	<**0.1**	1.3	<**0.1**	0.2
Sunet	26	32	3	<**0.1**	2.9	<**0.1**	1.2	<**0.1**	0.1
Shentel	28	35	3	<**0.1**	9.8	<**0.1**	1.1	<**0.1**	0.2
Bren	37	38	2	<**0.1**	6478.2	<**0.1**	1472.2	<**0.1**	328.6
NetworkUsa	35	39	3	<**0.1**	1215.1	<**0.1**	118.6	<**0.1**	9.7
IowaStatewideFiberMap	33	41	4	<**0.1**	486.3	<**0.1**	111.7	<**0.1**	19.0
PionierL1	36	41	3	<**0.1**	2570.7	<**0.1**	308.3	<**0.1**	89.5
LambdaNet	42	46	3	<**0.1**	TO	<**0.1**	11855.5	<**0.1**	723.2
Intranetwork	39	51	3	<**0.1**	19364.3	<**0.1**	1533.9	<**0.1**	122.5
RoedunetFibre	48	52	4	<**0.1**	TO	<**0.1**	TO	<**0.1**	TO
Ntelos	47	58	4	<**0.1**	TO	<**0.1**	TO	<**0.1**	17237.5
Palmetto	45	64	5	**0.6**	TO	**0.1**	TO	<**0.1**	TO
UsSignal	61	78	4	**0.2**	TO	**0.1**	TO	<**0.1**	TO
Missouri	67	83	4	**0.1**	TO	<**0.1**	TO	<**0.1**	TO
Switch	74	92	5	**0.7**	TO	**0.1**	TO	<**0.1**	TO
VtlWavenet2008	88	92	3	**0.1**	TO	**0.1**	TO	<**0.1**	TO
RedBestel	84	93	3	<**0.1**	TO	<**0.1**	TO	<**0.1**	TO
Intellifiber	73	95	5	**0.5**	TO	**0.1**	TO	<**0.1**	TO
VtlWavenet2011	92	96	3	**0.1**	TO	**0.1**	TO	<**0.1**	TO
Oteglobe	83	99	4	**0.2**	TO	**0.1**	TO	<**0.1**	TO
Interoute	110	146	7	**4602.7**	TO	**977.1**	TO	**11.2**	TO
Ion	125	146	5	**16.7**	TO	**13.0**	TO	**0.8**	TO
DialtelecomCz	138	151	5	**9.2**	TO	**2.1**	TO	**2.0**	TO
Deltacom	113	161	6	**74.9**	TO	**6.5**	TO	**5.3**	TO
TataNld	145	186	6	**775.6**	TO	**80.8**	TO	**24.5**	TO
UsCarrier	158	189	5	**19.8**	TO	**18.1**	TO	**4.8**	TO

bility.

Here, we provide further detailed discussion of the results.

- Probabilities in Figure 8.4 are plotted with log-scale, so they tell us that $S(n')$ decays roughly exponentially with respect to the outage scale. However, their decay stagnates at some points. This suggests that interpolation among them may lead to a significant under- or over-estimate of probabilities, even when $S(n')$ values are sparsely obtained at some points.

- For all topologies, the effect of $|T|$ is not significant when the outage scale is relatively small. This is because these real topology include many pendant edges, i.e., the edges connecting vertices with degree 1. Failure of such a edge immediately disconnects one vertex, so its outage

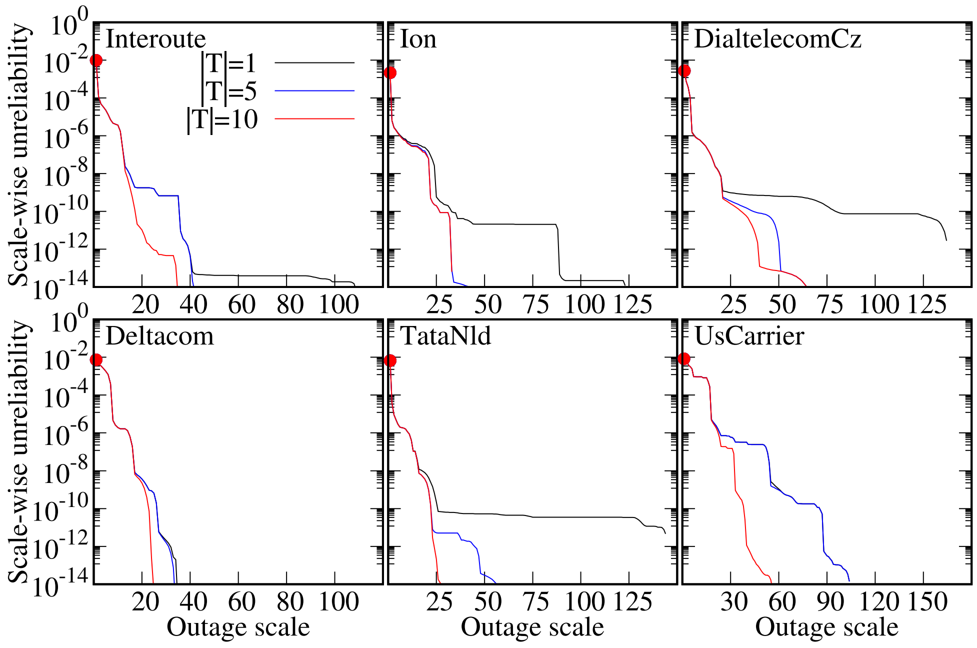


Figure 8.4: Scale-wise unreliability, i.e., probabilities that more than or equal to n' clients are not connected to servers. Filled circle indicates $S(1)$ value.

would be difficult to resolve even if we put more servers.

- Although longer edges have lower availability (Section 8.4.1), Interoute, the largest network across Europe and North America, is not the least reliable, whereas DialtelecomCz, the smallest one within the Czech Republic, is not the most reliable for the whole outage scale. This indicates that network reliability strongly depends on topological properties, not their reach.

8.5 Related Work

The research community has studied network reliability evaluation methods for many years, as described in Section 4.2.1. The fastest CSNR method (in Chapter 5) does not construct BDDs, which are required for the baseline method in Section 8.2.1, so we use HH method [Hardy *et al.*, 2007] for a baseline; note that even if we could use the CSNR method for the baseline, the large gap of several orders of magnitude in Section 8.4 could not be compensated.

Some studies have examined outage scales. Researchers on complex networks revealed that scale-free networks, including the Internet, can be divided into small connected components against attacks on high-degree nodes [Albert *et al.*, 2000; Cohen *et al.*, 2001; Magoni, 2003]. Researchers on communication networks investigated devastating network statuses after severe accidents [Neumayer and Modiano, 2010; Oostenbrink and Kuipers, 2017; Al Mtawa *et al.*, 2021; Yano *et al.*, 2022]. Although these studies often counted the number of connected nodes to measure the damage, they only considered a limited number of failure scenarios and did not evaluate a scale-wise unreliability. The algorithm in Chapter 6 can be used for evaluating the *expected number* of connected nodes, which can be regarded as computing the expected number of outage scale in our problem. However, as demonstrated in the experiments, the distribution of scale-wise unreliability cannot be recovered from only the expected number of outage scale.

8.6 Conclusion

This chapter studied an efficient algorithm to precisely evaluate scale-wise unreliability. The proposed algorithm computed the probability of every outage scale using DP. Numerical experiments show that our algorithm successfully dealt with real large-scale topologies within two hours. Future work includes extending the algorithm to support an outage scale based on user counts. We have provided public access to their code and data at `https://github.com/nttcslab/scale-wise-unrel`.

Chapter 9

Framework for Simultaneous Subgraph Counting under Connectivity Constraints

The subgraph counting problem is a graph counterpart of the weighted model counting problem, which computes the number of subgraphs of a given graph that satisfy some constraints. Among various constraints imposed on a graph, those regarding the connectivity of vertices, such as “these two vertices must be connected,” have great importance since they are indispensable for determining various graph substructures, e.g., paths, Steiner trees, and rooted spanning forests. In this view, the subgraph counting problem under connectivity constraints is also important because counting such substructures often corresponds to measuring the importance of a vertex in network infrastructures. However, we must solve the subgraph counting problems multiple times to compute such an importance measure for every vertex. Conventionally, they are solved separately by constructing a BDD or a ZDD for each problem. However, even solving a single subgraph counting is a computationally hard task, preventing us from solving it multiple times in a reasonable time. In this chapter, we propose a DP framework that simultaneously counts subgraphs for every vertex by focusing on similar connectivity constraints. Experimental results show that the proposed method solved multiple subgraph counting problems about 10–20 times faster than the existing approach for many problem settings.

9.1 Introduction

Given graph $G = (V, E)$, the *subgraph counting problem* computes the (possibly weighted) count of the subgraphs of G that satisfy some constraints such as each vertex's degree and the existence of cycles. More specifically, given edge weights $w_e^+, w_e^- \in \mathbb{R}$ for $e \in E$, this problem (exactly) computes the following value:

$$W(\mathcal{E}) := \sum_{E' \in \mathcal{E}} \left[\prod_{e \in E'} w_e^+ \cdot \prod_{e \in E \setminus E'} w_e^- \right], \tag{9.1}$$

where $\mathcal{E} \subseteq 2^E$ is a family of the subsets of edges, i.e., *subgraphs*, that satisfy given constraints. This problem can be seen as a graph counterpart of the weighted model counting problem defined in Problem 2.4.

When $\mathcal{E} = \mathcal{E}_K$, i.e., the family of subgraphs such that all the vertices in K are interconnected, and we set $w_e^+ = p_e$ and $w_e^- = 1 - p_e$, this is identical to computing K-NR. Thus, the K-NR computation is a specific case of subgraph counting problem. Other than network reliability computation, this problem has been studied as a fundamental task in computer science [Bax, 1994; Alon *et al.*, 1997; Flum and Grohe, 2004; Curticapean, 2018].

Here, we focus on a connectivity constraint, which is a topological constraint requiring that some pairs of vertices are connected and other pairs are disconnected. A kind of connectivity constraint is imposed when we compute K-NR. A connectivity constraint is also fundamental in determining various graph substructures, such as paths, Steiner trees, spanning trees, and rooted spanning forests, in combination with other constraints, as described in Section 9.2. Counting these substructures also has great importance, especially for evaluating the importance of a vertex. For example, as seen in Chapter 3, paths and Steiner trees on communication networks correspond to the routing of point-to-point and multi-site communication. Thus, the number of paths or Steiner trees passing vertex v is an importance measure for v in this communication network, since its failure causes the lost of this number of communication routing. A cycle passing through source vertex s and another vertex v constitutes a vertex-disjoint two paths between s and v, and counting such cycles corresponds to the number of non-blocking pairs of communication routings from s to v [Inoue, 2019]. A rooted spanning forest (RSF) rooted at $r_1, \dots, r_k$ corresponds to a (electrical) distribution network whose substations are located at $r_1, \dots, r_k$ [Inoue *et al.*, 2015]. When we add a new substation to v, the number of RSFs rooted at $r_1, \dots, r_k, v$ (given other constraints such as electric constraints) provides flexibility of the distribution network.

In evaluating such an importance measure for every vertex v, we generally have to solve the subgraph counting problem for every v. That is, we must compute multiple count values $W(\mathcal{E}_{v_1}), \ldots, W(\mathcal{E}_{v_n})$ for different families of subgraphs $\mathcal{E}_{v_1}, \ldots, \mathcal{E}_{v_n}$. However, the subgraph counting problem is generally #P-complete as seen in Section 4.2.1 (saying that K-NR computation is #P-complete). Even a practically fast algorithm for computing $W(\mathcal{E})$ described below may take several minutes or more for a graph with hundreds of edges. Subgraph counting for every vertex described above seems computationally much more difficult since we have to repeatedly solve cumbersome counting tasks.

This chapter extends the CSNR algorithm in Chapter 5 to propose a practically fast algorithm for *simultaneously* counting subgraphs for every vertex (formally defined in Section 9.2). Here, "simultaneously" means that we build only one data structure for obtaining all count values $W(\mathcal{E}_{v_1}), \ldots, W(\mathcal{E}_{v_n})$. Our contribution is summarized as follows:

- We formally defined the subgraph counting problem for every vertex under similar connectivity constraints.
- The proposed method enables us to simultaneously count such graph substructures as paths, cycles, Steiner trees, and RSFs by sophisticated dynamic programming (DP) on the built data structure.
- Complexity analyses show that the proposed method solves subgraph counting for every vertex $O(n)$ times faster than the baseline method that separately solves each counting, where n is the number of vertices.
- We empirically confirmed that the proposed algorithm solved subgraph counting for every vertex around 10–20 times faster than the baseline method.

9.1.1 Related Works

We have already give the literatures on computing K-NR, a kind of subgraph counting problem, in Chapter 4. Thus, we here examine literatures for other subgraph counting problems. The simplest method enumerates all the substructures [Read and Tarjan, 1975; Shioura *et al.*, 1997] such as paths and spanning trees. Especially, the enumeration of spanning trees [Shioura *et al.*, 1997] corresponds to computing the Tutte polynomial of a graph, which can be used for many kinds of graph counting problems. However, since there might be an exponential number of substructures, enumeration suddenly becomes intractable with the growth of graph size. For practically fast counting, indexing the constrained subgraphs into BDD-like structures has also been studied, as described in Section 2.5. Sekine *et al.* [1995] designed an algorithm to build a BDD representing all the spanning trees to compute the Tutte polynomial. Knuth [2011] proposed a very efficient scheme called Simpath that indexes all the simple paths in a ZDD, which can be used for counting simple paths. By expanding such research, Kawahara *et al.* [2017b] proposed a *frontier-based search* (FBS), which can build a ZDD of various graph substructures. Since ZDD also allows a simple DP for counting as seen in Section 2.4, FBS can be used for practically fast subgraph counting. The proposed algorithm is also based on FBS. Subgraph counting is also studied in the context of parameterized complexity theory [Flum and Grohe, 2004; Curticapean, 2018], although their interest often focuses on theoretical aspects. To the best of our knowledge, no works have outperformed the BDD/ZDD-based methods in practically solving subgraph counting problems, including network reliability evaluations.

The CSNR algorithm in Chapter 5 essentially solves counting problems for every vertex and runs much faster than the baseline where each client's reliability is computed separately. However, it can only deal with the constraints of the form "all the specified vertices are connected" and cannot accept the constraints of disconnection and others. The proposed algorithm in this chapter can be seen as an extension of CSNR algorithm. Technically, the proposed method can deal with these constraints by utilizing the ZDD structures [Minato, 1993] and the FBS [Kawahara *et al.*, 2017b]. While the CSNR algorithm relies on a BDD-like structure that is not truly a BDD, we build a legitimate ZDD by FBS, enabling us to combine such existing ZDD algorithms as Apply [Minato, 1993] and subsetting [Iwashita and Minato, 2013].

9.1.2 Organization of Chapter

The rest of this chapter is organized as follows. Section 9.2 describes the preliminary and the formal statement of the problem we solved. Section 9.3 gives the overview of the proposed method. Section 9.4 introduces the ZDD and the frontier-based search that are used in the proposed method. Section 9.5 details the proposed method, and Section 9.6 analyzes the computational complexity of it. Section 9.7 empirically compares the proposed method with the baseline in terms of computational time, and Section 9.8 gives a conclusion.

9.2 Problem Statement

Before proceeding to our problem statement, we introduce a notion that represents the connectivity constraint in the same manner as [Kawahara *et al.*, 2017b]. As described in Section 9.1, a connectivity constraint requires that some pairs of vertices are connected and other pairs are disconnected. We represent the connectivity constraint by *subpartition* $\mathcal{C}$ of vertex set V of graph $G = (V, E)$, where a subpartition of V is a set of pairwise disjoint subsets of V. $\mathcal{C}$ imposes the following constraints: (i) for any pair of vertices v, v' in the same set in P, they must be connected, and (ii) for any pair of vertices v, v' in different sets in P, they must not be connected.

We extend the notion by introducing exactly one *wildcard* $*$, which will be replaced by a vertex to represent various connectivity constraints. Let $\mathcal{C}^*$ be a subpartition of $V \cup \{*\}$ that must contain exactly one $*$. For $v \in V$, let $\mathcal{C}^*[v]$ be the connectivity constraint obtained by substituting $*$ in $\mathcal{C}^*$ with v. Additionally, let $\mathcal{C}^*[]$ be the connectivity constraint obtained by simply removing $*$ from $\mathcal{C}$. Note that if $v \in V$ is already present in $\mathcal{C}^*$, $\mathcal{C}^*[v]$ equals (i) $\mathcal{C}^*[]$ if v is in the same set in $\mathcal{C}^*$ that contains $*$; or (ii) an inconsistent constraint. For example, for $\mathcal{C}^* = \{\{v_1, v_2, *\}, \{v_3\}\}$, $\mathcal{C}^*[v_4] = \{\{v_1, v_2, v_4\}, \{v_3\}\}$, $\mathcal{C}^*[v_2] = \mathcal{C}^*[] = \{\{v_1, v_2\}, \{v_3\}\}$, and $\mathcal{C}^*[v_3]$ is an inconsistent constraint.

Now we proceed to the problem definition. In our problem, we are given connected undirected graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, connectivity constraint $\mathcal{C}^*$ containing exactly one wildcard $*$, family $\mathcal{F}$ of subgraphs, i.e., subsets of edges, and weights $w_e^+, w_e^- \in \mathbb{R}$ for each $e \in E$. For connectivity constraint $\mathcal{C}$, let $\mathcal{E}(\mathcal{C})$ be the family of subgraphs satisfying $\mathcal{C}$. Our goal is to compute the following value

$$\mathsf{count}(v) := W(\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[v])) \quad \text{for every } v \in V. \tag{9.2}$$

The problems described in Section 9.1 can be covered by our problem.

- *Path*: Given $s, t \in V$, we set $\mathcal{C}^* = \{\{s, t, *\}\}$ and let $\mathcal{F}$ be a family of subgraphs where (i) the degree of s and t is 1, and the others have degree 0 or 2, and (ii) there are no cycles. Then $\mathsf{count}(v)$ equals the number of simple s, t-paths that pass through v.

- *Cycle*: Given $s \in V$, we set $\mathcal{C}^* = \{\{s, *\}\}$, and let $\mathcal{F}$ be a family of subgraphs where (i) the degree of each vertex is 0 or 2, and (ii) there is exactly one connected component. Then $\mathsf{count}(v)$ equals the number of cycles starting from s that pass through v.

- *Steiner tree*: Given $T \subseteq V$, we set $\mathcal{C}^* = \{T \cup \{*\}\}$ and let $\mathcal{F}$ be a family of subgraphs where there are no cycles and exactly one connected component. Then $\mathsf{count}(v)$ equals the number of T-Steiner trees containing v, where T-Steiner trees are the trees connecting all the T vertices.

- *Rooted spanning forest*: Given $T = \{r_1, \ldots, r_k\} \subseteq V$, we set $\mathcal{C}^* = \{\{r_1\}, \ldots, \{r_k\}, \{*\}\}$, and let $\mathcal{F}$ be a family of subgraphs where (i) every vertex has degree at least 1, (ii) there are no cycles, and (iii) there are exactly $(k+1)$ connected components. Then $\mathsf{count}(v)$ equals the number of rooted spanning forests rooted at $r_1, \ldots, r_k$ and v.

In addition, the CSNR problem in Chapter 5 (Problem 5.1) essentially counts the subgraphs where given vertex set $T \subseteq V$ and vertex v are connected for every v and can be recovered by $\mathcal{C}^* = \{T \cup \{*\}\}$ and $\mathcal{F} = 2^E$. Although here we list only topological constraints for $\mathcal{F}$, we can also impose non-topological constraints with $\mathcal{F}$, such as knapsack constraints.

9.3 Overview and High-level Idea

The proposed method can be seen as an extension of CSNR method in Chapter 5 combined with FBS. Indeed, the proposed method is similar to CSNR method in that we first construct a decision diagram in a top-down manner and then perform a DP using configures. However, as described in Section 9.1.1, a legitimate ZDD is built by the proposed method, while a diagram not truly a BDD or a ZDD is built by CSNR method. The configure of FBS is more elaborated than that of the CSNR method (Definitions 5.2 and 5.3), so we should have more elaborated DP than the CSNR method. Moreover, since the computed values are no longer probabilities in the problem in this chapter, we completely renew the argument not to rely on probability. At first, we give an overview of the proposed algorithm. Then, we describe the intuition and high-level idea behind the proposed algorithm.

9.3.1 Overview of the Proposed Algorithm

First, we explain case $\mathcal{F} = 2^E$. Given connectivity constraint $\mathcal{C}^*$, the baseline method, which separately computes the count value for every v by FBS [Kawahara *et al.*, 2017b], builds a ZDD with connectivity constraint $\mathcal{C}^*[v]$ for every $v \in V$. By FBS with $\mathcal{C}^*[v]$, a ZDD representing $\mathcal{E}(\mathcal{C}^*[v])$ is built and allows a simple DP for computing $\mathsf{count}(v) = W(\mathcal{E}(\mathcal{C}^*[v]))$. The details of FBS are explained in Section 9.4.

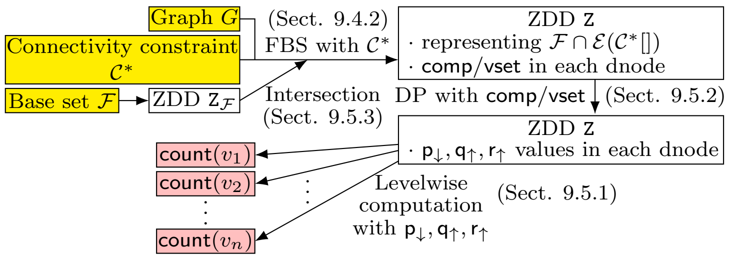

Graph G
Connectivity constraint $\mathcal{C}^*$
Base set $\mathcal{F}$
ZDD $\mathsf{Z}_\mathcal{F}$
(Sect. 9.4.2) FBS with $\mathcal{C}^*$
Intersection (Sect. 9.5.3)
ZDD Z
· representing $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[])$
· comp/vset in each dnode
DP with comp/vset
(Sect. 9.5.2)
ZDD Z
· $\mathsf{p}_\downarrow, \mathsf{q}_\uparrow, \mathsf{r}_\uparrow$ values in each dnode
Levelwise computation with $\mathsf{p}_\downarrow, \mathsf{q}_\uparrow, \mathsf{r}_\uparrow$
(Sect. 9.5.1)
$\mathsf{count}(v_1)$
$\mathsf{count}(v_2)$
$\mathsf{count}(v_n)$

Figure 9.1: Overview of proposed algorithm.

More specifically, the proposed method builds a ZDD by FBS. However, unlike the baseline method, we build only one ZDD Z for $\mathcal{C}^*$, which represents $\mathcal{E}(\mathcal{C}^*[])$. Instead, we retain the information used in the FBS for building Z, comp and vset in each dnode of Z, both of which are discarded after the FBS in the baseline method. Since comp and vset provide rich information for connectivity among vertices, we fully exploit them to perform a more elaborated DP, yielding for each dnode of Z three kinds of values, $\mathsf{p}_\downarrow$, $\mathsf{q}_\uparrow$, and $\mathsf{r}_\uparrow$. Their definitions are described in Section 9.5.1. By using them, we can compute $\mathsf{count}(v)$ values for every $v \in V$. Since the computation of $\mathsf{p}_\downarrow$, $\mathsf{q}_\uparrow$, $\mathsf{r}_\uparrow$, and $\mathsf{count}(v)$ can be performed in time proportional to the execution of FBS, the proposed algorithm runs faster than the baseline. We fully describe the computation of the $\mathsf{count}(v)$ values in Section 9.5.1 and those of $\mathsf{p}_\downarrow$, $\mathsf{q}_\uparrow$, and $\mathsf{r}_\uparrow$ (the DP procedure) in Section 9.5.2.

Finally, we deal with case $\mathcal{F} \neq 2^E$. For it, we first construct ZDD $\mathsf{Z}_\mathcal{F}$ by the existing methods. Then, we construct one ZDD Z that represents $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[])$ whose dnodes have comp and vset. This can be performed by exploiting existing techniques of constructing a ZDD of set intersection, such as Apply [Minato, 1993] and subsetting [Iwashita and Minato, 2013]. After Z is built, we can perform the same DP scheme as above. We describe taking the set intersection in Section 9.5.3. An overview of the proposed method is given in Figure 9.1.

By changing base set $\mathcal{F}$ and connectivity constraint $\mathcal{C}^*$, the proposed algorithm can solve various subgraph counting problems, as in Section 9.2. We named our proposed algorithm *compDP* since it fully uses information comp.

9.3.2 Intuition and Idea

We describe the high-level idea behind the proposed algorithm. As described later, ZDD, which is a rooted and layered directed acyclic graph, represents a family of subgraphs as the set of paths from the root to a terminal dnode.

By defining the *path product* of a path by the weights along this path (precise definition is later), the count value equals the sum of path products of these paths. Note that the argument based on path product instead of probability is suited for the problem considered in this chapter. The intuitive for the proposed algorithm is as follows: Let $\mathcal{E}_v = \mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[v])$. To compute $\mathsf{count}(v) = W(\mathcal{E}_v)$ for every $v \in V$, it is sufficient to build a ZDD representing $\mathcal{E}_v$ for every v. However, since $\mathcal{E}_v$ and $\mathcal{E}_w$ $(v \neq w)$ are similar families stemming from the common constraint $\mathcal{C}^*$, the ZDDs representing them also are expected to exhibit similar structures. We use such similarities to reduce the computation.

More specifically, we use the following step-by-step ideas: First, since $\mathcal{E}(\mathcal{C}^*[v]) \subseteq \mathcal{E}(\mathcal{C}^*[])$ for any $v \in V$, $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[v])$ is represented by the subset of the paths within the ZDD representing $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[])$. Second, these paths can be decomposed into former and latter parts; the former is the paths from the root to a specific layer, and the latter is the paths from the specific layer to the terminal. The count value $\mathsf{count}(v)$ can be represented by the sums of path products of the former part and those of the latter part. Third, when considering such a decomposition for every $v \in V$, we can reuse the values of "the sums of path products of the former part" ($\mathsf{p}_\downarrow$ in the proposed algorithm) and "those of the latter part" ($\mathsf{q}_\uparrow$ and $\mathsf{r}_\uparrow$ in the proposed algorithm). Thus, by pre-computing them by a DP, we can compute $\mathsf{count}(v)$ for every $v \in V$ with these values.

9.4 Path Product and Frontier-based Search

As a preliminary, we first define the path product on ZDD. Then, we explain the FBS in detail.

9.4.1 Path Product on ZDD

Let $\mathtt{Z} = (\mathtt{N}, \mathtt{A})$ be a ZDD representing the family of edge subsets, i.e., subgraphs. We define some notions for representing paths on $\mathtt{Z}$. For $\mathtt{n} \in \mathtt{N}$, let $\mathsf{lo\text{-}arc}(\mathtt{n})$ and $\mathsf{hi\text{-}arc}(\mathtt{n})$ be the lo- and hi-arcs outgoing from $\mathtt{n}$. For $\mathtt{n}, \mathtt{n}' \in \mathtt{N}$, $\mathcal{R}_\mathtt{Z}(\mathtt{n}, \mathtt{n}')$ denotes the set of paths from $\mathtt{n}$ to $\mathtt{n}'$. Recall that, given a predefined order of edges $e_1, \ldots, e_m$, ZDD $\mathtt{Z}$ represents a family of subgraphs $\mathcal{E}_\mathtt{Z} \subseteq 2^E$ by $\mathcal{R}_\mathtt{Z}(\mathtt{r}, \top)$ where $\mathtt{r} \in \mathtt{N}$ is the root dnode. For path $\mathtt{R}$ in $\mathtt{Z}$, we associate subset $E(\mathtt{R}) \subseteq E$ where $e_i \in E(\mathtt{R})$ if and only if $\mathtt{R}$ traverses a hi-arc outgoing from a dnode with label i. We say $E(\mathtt{R})$ the corresponding subgraph of $\mathtt{R}$. Then $\mathcal{E}_\mathtt{Z} = \{E(\mathtt{R}) \mid \mathtt{R} \in \mathcal{R}_\mathtt{Z}(\mathtt{r}, \top)\}$. For example, in Figure 9.2b, path 1-(hi)-2-(lo)-3-(lo)-4-(hi)-5-(hi)-$\top$ indicates that $\{e_1, e_4, e_5\} \in \mathcal{E}_\mathtt{Z}$.

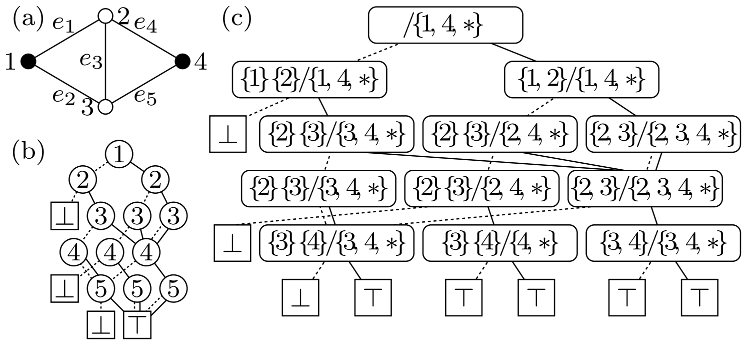


Figure 9.2: (a) Example of graph. (b) Example of (normalized) ZDD. Dashed and solid lines indicate lo- and hi-arcs, and an integer inside a dnode represents a label. (c) ZDD made by FBS where $\mathcal{C}^* = \{\{1, 4, *\}\}$. Two subpartitions of vertices inside a dnode represent comp and vset.

We here derive the *normalized* property for ZDDs in the same way as Section 7.3.3. A ZDD is normalized if for every $\mathtt{n} \in \mathtt{N} \setminus \{\top, \bot\}$, both $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$ are either a terminal dnode or a dnode whose label is $\mathsf{lb}(\mathtt{n}) + 1$. For normalized ZDD, we define the path product as follows. Let $\mathtt{Z}$ be a normalized ZDD, and let $\mathtt{R} \in \mathcal{R}_{\mathtt{Z}}(\mathtt{n}, \mathtt{n}')$ be an arbitrarily chosen path. Given weights $w_e^+, w_e^- \in \mathbb{R}$ for every $e \in E$, we define the *path product* of $\mathtt{R}$ by

$$\Pi_{\mathtt{R}} := \prod_{e \in E(\mathtt{R})} w_e^+ \cdot \prod_{e \in \{e_{\mathsf{lb}(\mathtt{n})}, \ldots, e_{\mathsf{lb}(\mathtt{n}')-1}\} \setminus E(\mathtt{R})} w_e^- . \tag{9.3}$$

In other words, $\Pi_{\mathtt{R}}$ is the product of w_e^+ for the edges in $E(R)$ and w_e^- for the edges not in $E(R)$. We also define value $\mathsf{p}_\downarrow(\mathtt{n})$ for ZDD dnode $\mathtt{n}$ by the sum of path products of the paths in $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \mathtt{n})$. By definition, $W(\mathcal{E}_{\mathtt{Z}})$ equals $\mathsf{p}_\downarrow(\top)$. Moreover, although $\mathsf{p}_\downarrow(\mathtt{n})$ is defined as the sum of a possibly exponential number of path products, its value can efficiently be computed by a DP. Simple calculation shows the following equation for a dnode other than $\mathtt{r}$ or $\bot$:

$$\mathsf{p}_\downarrow(\mathtt{n}) = \sum_{\mathtt{n}' : \mathsf{lo}(\mathtt{n}') = \mathtt{n}} w^-_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{p}_\downarrow(\mathtt{n}') + \sum_{\mathtt{n}' : \mathsf{hi}(\mathtt{n}') = \mathtt{n}} w^+_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{p}_\downarrow(\mathtt{n}'). \tag{9.4}$$

Starting with $\mathsf{p}_\downarrow(\mathtt{r}) = 1$, by applying (9.4) in a top-down manner along $\mathtt{Z}$, we can compute the value of $\mathsf{p}_\downarrow(\top) = W(\mathcal{E}_{\mathtt{Z}})$ in time proportional to the number of dnodes in $\mathtt{Z}$.

9.4.2 Frontier-based Search

A frontier-based search (FBS) [Kawahara *et al.*, 2017b] is an efficient method for constructing a normalized ZDD that represents a family of subgraphs satisfying some constraints. Below we explain its procedure of given connectivity constraint $\mathcal{C}$ (without $*$) using the top-down construction framework of Section 2.5.

Similar to the partition used in previous chapters, we again focus on the connectivity among a limited subset of vertices. More specifically, we focus on the frontier vertices F_i defined in Definition 4.3 and the vertices appearing in the connectivity constraint $\mathcal{C}$. Let $V_{\mathcal{C}}$ be the set of vertices appearing in connectivity constraint $\mathcal{C}$. The goal of the configure design for FBS is to represent the connectivity among $F_i \cup V_{\mathcal{C}}$.

In FBS, the connectivity among $F_i \cup V_{\mathcal{C}}$ is maintained by two subpartitions of vertices: comp and vset. Here comp maintains the connectivity among F_i while vset maintains the connectivity among the sets in comp and the vertices in $V_{\mathcal{C}}$. More specifically, comp is a partition of F_i such that $v, v' \in F_i$ are in the same set if and only if they are connected, and vset maintains the connectivity constraint such that the vertices in the same set must be connected and those in different sets must be disconnected. Here $\mathsf{comp}[v]$ denotes the set in comp containing v, and as is the same with $\mathsf{vset}[v]$. We use the pair of comp and vset as a configure.

The ROOT and CHILD procedures of FBS is given in Algorithm 9.1. At root, we have an empty comp since $F_1 = \emptyset$ and $\mathsf{vset} = \mathcal{C}$ requiring connectivity constraint $\mathcal{C}$ (Lines 1–2). In CHILD procedure, let $e_i = \{v, v'\}$ be the current edge. Let V_{vset} be the vertices appearing in vset and $\mathsf{vset}[v]$ ($v \in V_{\mathsf{vset}}$) be the set in vset containing v. If we include e_i but both endpoints of e_i are in V_{vset} and the sets in vset containing these endpoints are different (Line 4), we must not connect them and thus we immediately return $\bot$ as a child (Line 5). Then, for every vertex u that are newly entered into frontier, we add a component consisting of only u as a new component of comp (Lines 6–7). If we include e_i and the components that contains the endpoints are different (Line 8), we merge these two components (Line 9). Here, if one endpoint u appears in vset and the other u' does not appear in vset, we add the vertices in the component containing u' one into the set in vset containing u, since now they are connected and thus u' and the vertices in the same component must satisfy the same connectivity constraint as u (Lines 10–11). Then, for every vertex u leaving from frontier (Line 12), we must care the case where the component including u is left isolated (Line 13). If u appears in vset but $\{u\}$ does not appear in vset, this means that the set in vset containing u has another vertex w. Eventually u must be connected to w, but this cannot be

Algorithm 9.1: Frontier-based search for connectivity constraint $\mathcal{C}$. Underlined part in is added if it is used for a subroutine of proposed method.

```
procedure $FBS_{\mathcal{C}}$.Root():
    return $\langle 1, (\emptyset, \mathcal{C}) \rangle$
procedure $FBS_{\mathcal{C}}$.Child($\langle i, (\mathsf{comp}, \mathsf{vset}) \rangle, f$):     // $e_i = \{v, v'\}$
    if $f = \text{hi}$ and $v, v' \in V_{\mathsf{vset}}$ and $\mathsf{vset}[v] \neq \mathsf{vset}[v']$ then
        return $\langle m+1, 0 \rangle$                  // $v$ and $v'$ must not be connected
    foreach $u \in \{v, v'\} \setminus F_i$ do                 // Vertices entering the frontier
        $\mathsf{comp} \leftarrow \mathsf{comp} \cup \{\{u\}\}$                  // Add $u$ as an isolated vertex
    if $f = \text{hi}$ and $\mathsf{comp}[v] \neq \mathsf{comp}[v']$ then       // Connecting two components
        Merge $\mathsf{comp}[v]$ and $\mathsf{comp}[v']$ into one
        if $v \in V_{\mathsf{vset}}$ and $v' \notin V_{\mathsf{vset}}$ then Add vertices in $\mathsf{comp}[v']$ to $\mathsf{vset}[v]$
        if $v \notin V_{\mathsf{vset}}$ and $v' \in V_{\mathsf{vset}}$ then Add vertices in $\mathsf{comp}[v]$ to $\mathsf{vset}[v']$
    foreach $u \in \{v, v'\} \setminus F_{i+1}$ do             // Vertices leaving from frontier
        if $\{u\} \in \mathsf{comp}$ then          // Component containing $u$ leaves frontier
            if $u \in V_{\mathsf{vset}}$ and $\{u\} \notin \mathsf{vset}$ $\underline{\text{and } \{u, *\} \notin \mathsf{vset}}$ then
                return $\langle m+1, 0 \rangle$        // $u$ must be connected to $w \in \mathsf{vset}[u] \setminus \{u\}$
        Remove $u$ from $\mathsf{comp}$ and $\mathsf{vset}$ if exists
        Remove empty subset from $\mathsf{comp}$ and $\mathsf{vset}$ if exists
        if $i = m$ then return $\langle m+1, 1 \rangle$        // All conditions are satisfied
        else return $\langle i+1, (\mathsf{comp}, \mathsf{vset}) \rangle$
```

satisfied since the component including u is left isolated. Thus, in this case we immediately return $\perp$ as a child (Line 15). Afterwards, u is removed from comp and vset (Lines 16–17). Finally, the resulting pair of comp and vset is returned as an f-child configure except that when $i = m$, all constraints are satisfied and thus we return $\top$ (Lines 18–19).

For example, Figure 9.2c is the result of FBS given the graph in Figure 9.2a and the constraint that vertices 1 and 4 must be connected. Note that we here ignore the $*$ mark in Figure 9.2c. We now focus on the left, 2nd level dnode: "{1}{2}/{1,4}". Here comp $= \{1\}\{2\}$ denotes two components, the one including 1 and the one including 2, and vset $= \{1, 4\}$ represents that 1 and 4 must be connected. If we exclude $e_2 = \{1, 3\}$, the component including 1 is left isolated because $F_3 = \{2, 3\}$ does not include 1. However, since this contradicts that 1 and 4 must be connected, the lo-child is $\perp$ (pruned). If we include e_2, the component including 1 becomes one that includes 3 at the next level. The constraint that 1 and 4 must be connected can be rewritten as that 3 and 4 must be connected. Thus, hi-child's comp is {2}{3} and vset is {3,4}.

9.5 Details of Proposed Method

First, we assume $\mathcal{F} = 2^E$; this assumption is removed in Section 9.5.3. As in Section 9.3, the proposed method first builds ZDD `Z` representing $\mathcal{E}(\mathcal{C}^*[])$. To explain the meaning and procedure for this, we observe the relationship between $\mathcal{C}^*[]$ and $\mathcal{C}^*[v]$. Let $V^*_{\mathcal{C}^*}$ be the vertices in the set in $\mathcal{C}^*$ containing $*$, and let $V^{\neg}_{\mathcal{C}^*}$ be the other vertices present in $\mathcal{C}^*$. From the definition, $\mathcal{C}^*[v]$ imposes the following additional constraint on $\mathcal{C}^*[]$:

> ($\#_v$) v must be connected with the vertices in $V^*_{\mathcal{C}^*}$, and v must be disconnected from the vertices in $V^{\neg}_{\mathcal{C}^*}$.

That is, $\mathcal{E}(\mathcal{C}^*[v]) = \{E' \mid E' \in \mathcal{E}(\mathcal{C}^*[]),\ E' \text{ satisfies } (\#_v)\}$. Now the constraint $\mathcal{C}^*[v]$ is decomposed into ($\#_v$) and $\mathcal{C}^*[]$, where ($\#_v$) involves only the connectivity around v and $\mathcal{C}^*[]$ represents the other constraints. The fact that ($\#_v$) concerns only the connectivity around v enables us to compute $\mathsf{count}(v)$ for every v with only one ZDD `Z` representing $\mathcal{E}(\mathcal{C}^*[])$, as described in the subsequent sections.

Meanwhile, during the procedure of FBS, we must remember which set in `vset` corresponds to the set in $\mathcal{C}$ containing $*$ since we use it for the subsequent computation. To achieve this, we just consider $*$ in $\mathcal{C}^*$ a special vertex. More specifically, we let the root's `vset` as $\mathcal{C}^*$ (instead of $\mathcal{C}$) in Line 2 of Algorithm 9.1 and add the underlined part of Line 14. By adding the underlined part, we simply discards $*$ even if `vset`$[v]$ contains $*$. Thus, `Z` finally represents $\mathcal{E}(\mathcal{C}^*[])$, while each `vset` has at most one set containing $*$. For example, Figure 9.2c is the result of FBS given the graph in Figure 9.2a and the constraint $\mathcal{C}^* = \{\{1, 4, *\}\}$.

9.5.1 Computation with Intermediate Level of Diagram

Let $\mathcal{R}_v$ be the set of the paths in $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \top)$ whose corresponding subgraphs satisfy ($\#_v$). As stated above, $\mathsf{count}(v)$ equals the sum of path products of the paths in $\mathcal{R}_v$. Here we focus on i-th level $\mathtt{L}_i$ of `Z` where $v \in F_i$.[1] For dnode $\mathtt{n} \in \mathtt{L}_i$ with label i, let $\mathcal{R}_{v,\mathtt{n}}$ be the set of paths in $\mathcal{R}_v$ passing through `n`. Since `Z` is normalized, every $\mathtt{r}$-$\top$ path in `Z` passes exactly one dnode in $\mathtt{L}_i$. Thus, we have $\mathcal{R}_v = \bigcup_{\mathtt{n}\in\mathtt{L}_i} \mathcal{R}_{v,\mathtt{n}}$ and $\mathcal{R}_{v,\mathtt{n}} \cap \mathcal{R}_{v,\mathtt{n}'} = \emptyset$ for $\mathtt{n} \neq \mathtt{n}'$, meaning that

$$\mathsf{count}(v) = \sum_{\mathtt{R}\in\mathcal{R}_v} \Pi_{\mathtt{R}} = \sum_{\mathtt{n}\in\mathtt{L}_i} \sum_{\mathtt{R}\in\mathcal{R}_{v,\mathtt{n}}} \Pi_{\mathtt{R}}.$$

[1]If v's degree is more than 1, there is at least one i such that $v \in F_i$, as described in Section 5.3.2. The treatment of degree 1 vertices is in Section 9.9.1.

We further decompose $\sum_{\mathtt{R}\in\mathcal{R}_{v,\mathtt{n}}}\Pi_{\mathtt{R}}$ by focusing on $\mathtt{n}\in\mathtt{L}_i$. Hereafter, we write comp and vset retained in dnode $\mathtt{n}$ as $\mathtt{n}.\mathsf{comp}$ and $\mathtt{n}.\mathsf{vset}$. Since $\mathtt{n}.\mathsf{comp}$ maintains the connectivity among F_i, the sets in $\mathtt{n}.\mathsf{comp}$ are indeed connected components. Because the connectivity around v can be translated into that around connected component $B = \mathtt{n}.\mathsf{comp}[v]$, $(\#_v)$ can be restated as a constraint on $B = \mathtt{n}.\mathsf{comp}[v]$:

> $(\#'_B)$ Connected component B must be connected with the vertices in $V^*_{\mathcal{C}^*}$, and B must be disconnected from the vertices in $V^{\neg}_{\mathcal{C}^*}$.

Let $\mathcal{R}_{\mathtt{n},B}\subseteq\mathcal{R}_{\mathsf{Z}}(\mathtt{n},\top)$ be the set of paths such that $\mathtt{R}'\in\mathcal{R}_{\mathtt{n},B}$ if and only if $E(\mathtt{R})\cup E(\mathtt{R}')$ satisfies $(\#'_B)$ for an arbitrarily chosen $\mathtt{R}\in\mathcal{R}_{\mathsf{Z}}(\mathtt{r},\mathtt{n})$. $\mathcal{R}_{\mathtt{n},B}$ is well-defined, i.e., kept unchanged regardless of the choice of $\mathtt{R}$ because $E(\mathtt{R})$'s connectivity among components and the vertices in $V^*_{\mathcal{C}^*}\cup V^{\neg}_{\mathcal{C}^*}$ is completely determined in $\mathtt{n}.\mathsf{vset}$. This means that $\mathcal{R}_{v,\mathtt{n}}$ can be written as *direct product* $\mathcal{R}_{\mathsf{Z}}(\mathtt{r},\mathtt{n})\sqcup\mathcal{R}_{\mathtt{n},\mathtt{n}.\mathsf{comp}[v]}$ where $A\sqcup B := \{a\cup b \mid a\in A, b\in B\}$. In other words, every path $\mathtt{R}\in\mathcal{R}_{v,\mathtt{n}}$ can be decomposed into $\mathtt{R}'\in\mathcal{R}_{\mathsf{Z}}(\mathtt{r},\mathtt{n})$ and $\mathtt{R}''\in\mathcal{R}_{\mathtt{n},\mathtt{n}.\mathsf{comp}[v]}$. From the definition of path product, $\Pi_{\mathtt{R}}=\Pi_{\mathtt{R}'}\Pi_{\mathtt{R}''}$. Thus, by defining $\mathsf{q}_\uparrow(\mathtt{n},B)$ as the sum of path product of the paths in $\mathcal{R}_{\mathtt{n},B}$, the following holds:

$$\sum_{\mathtt{R}\in\mathcal{R}_{v,\mathtt{n}}}\Pi_{\mathtt{R}} = \sum_{\mathtt{R}'\in\mathcal{R}_{\mathsf{Z}}(\mathtt{r},\mathtt{n})}\ \sum_{\mathtt{R}''\in\mathcal{R}_{\mathtt{n},\mathtt{n}.\mathsf{comp}[v]}}\Pi_{\mathtt{R}'}\Pi_{\mathtt{R}''} = \mathsf{p}_\downarrow(\mathtt{n})\cdot\mathsf{q}_\uparrow(\mathtt{n},\mathtt{n}.\mathsf{comp}[v]). \tag{9.5}$$

Finally, $\mathsf{count}(v)$ can be represented as

$$\mathsf{count}(v) = \sum_{\mathtt{n}\in\mathtt{L}_i}\sum_{\mathtt{R}\in\mathcal{R}_{v,\mathtt{n}}}\Pi_{\mathtt{R}} = \sum_{\mathtt{n}\in\mathtt{L}_i}\mathsf{p}_\downarrow(\mathtt{n})\cdot\mathsf{q}_\uparrow(\mathtt{n},\mathtt{n}.\mathsf{comp}[v]). \tag{9.6}$$

By choosing i such that $v\in F_i$ for every v, we can compute $\mathsf{count}(v)$ for every v by (9.6) if $\mathsf{p}_\downarrow$ and $\mathsf{q}_\uparrow$ are computed. In the next section, we show that $\mathsf{q}_\uparrow$ can be computed by DP.

9.5.2 Dynamic Programming

First, we define a correspondence of the components in comp between $\mathtt{n}$ and its child dnodes in a similar manner as the correspondence between blocks of partitions (see Definition 5.4).

Definition 9.1. Let $\mathtt{n}\in\mathtt{L}_i$ be a dnode of a normalized ZDD whose label is i, and let f be either lo or hi. Assuming that the f-child of $\mathtt{n}$ is not a terminal, for $B\in\mathtt{n}.\mathsf{comp}$, we define the f-child of B as follows: (i) If B contains vertex

v in F_{i+1}, the f-child is $\mathtt{n}'.\mathsf{comp}[v]$, where $\mathtt{n}'$ is the f-child of $\mathtt{n}$. (ii) If no such vertex exists, the lo-child of B is $\emptyset$, i.e., no corresponding component. For hi-child, let v, v' be the endpoints of e_i. If $v' \in F_i$ and $v \in B$, the hi-child of B is $\mathsf{hi}(\mathtt{n}).\mathsf{comp}[v']$. If $v \in F_i$ and $v' \in B$, the hi-child of B is $\mathsf{hi}(\mathtt{n}).\mathsf{comp}[v]$. Otherwise, the hi-child of B is $\emptyset$. We write the lo- and hi-child of B as $\mathsf{lo}(\mathtt{n})$ and $\mathsf{hi}(\mathtt{n})$.

Intuitively, for $\mathsf{f} \in \{\mathsf{lo}, \mathsf{hi}\}$, $\mathsf{f}(B)$ is a component in $\mathsf{f}(\mathtt{n}).\mathsf{comp}$ that represents the same component as B.

We derive a formula for $\mathsf{q}_{\uparrow}(\mathtt{n}, B)$ by decomposing the set of paths $\mathcal{R}_{\mathtt{n},B}$. Since every path in $\mathcal{R}_{\mathtt{n},B}$ passes either $\mathsf{lo\text{-}arc}(\mathtt{n})$ or $\mathsf{hi\text{-}arc}(\mathtt{n})$, we have a case analysis. Let $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B)$ ($\mathsf{q}_{\uparrow}^{+}(\mathtt{n}, B)$) be the sum of path products of the paths in $\mathcal{R}_{\mathtt{n},B}$ that traverse $\mathsf{lo\text{-}arc}(\mathtt{n})$ ($\mathsf{hi\text{-}arc}(\mathtt{n})$). We have $\mathsf{q}_{\uparrow}(\mathtt{n}, B) = \mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B) + \mathsf{q}_{\uparrow}^{+}(\mathtt{n}, B)$.

We now focus on $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B)$, which means that e_i is excluded. If $\mathsf{lo}(\mathtt{n}) = \bot$, constraint $\mathcal{E}(\mathcal{C}^*[])$ is not satisfied, so no path in $\mathcal{R}_{\mathtt{n},B}$ passes through $\mathsf{lo\text{-}arc}(\mathtt{n})$. Thus, $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B) = 0$. Otherwise, $\mathsf{lo}(B)$ is defined as in Definition 9.1. If $\mathsf{lo}(B) \neq \emptyset$, since $\mathsf{lo}(B)$ is the same component as B, constraint $(\#'_B)$ is satisfied if and only if constraint $(\#'_{\mathsf{lo}(B)})$ for $\mathsf{lo}(\mathtt{n})$ is satisfied. Thus, the set of paths in $\mathcal{R}_{\mathtt{n},B}$ that traverse $\mathsf{lo\text{-}arc}(\mathtt{n})$ can be written as $\{\mathsf{lo\text{-}arc}(\mathtt{n})\} \sqcup \mathcal{R}_{\mathsf{lo}(\mathtt{n}),\mathsf{lo}(B)}$. This means that $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B) = w_{e_i}^{-} \cdot \mathsf{q}_{\uparrow}(\mathsf{lo}(\mathtt{n}), \mathsf{lo}(B))$.

The remaining case is $\mathsf{lo}(\mathtt{n}) \neq \bot$ and $\mathsf{lo}(B) = \emptyset$. In this case, the component B does not exist in the next level $\mathtt{L}_{i+1}$ and thus we cannot translate constraint $(\#'_B)$ into the one concerning the lo-child $\mathsf{lo}(\mathtt{n})$. In other words, we must judge whether constraint $(\#'_B)$ is satisfied with only the information on $\mathtt{n}$. Fortunately, it is possible because $\mathtt{n}.\mathsf{vset}$ completely determines the connectivity among $B \in \mathtt{n}.\mathsf{comp}$ and $V_{\mathcal{C}^*}^{*} \cup V_{\mathcal{C}^*}^{\neg}$.

We have case analysis on how B is connected with $V_{\mathcal{C}^*}^{*} \cup V_{\mathcal{C}^*}^{\neg}$; how to distinguish these cases are described later. If B is connected with some (but not all) vertices in $V_{\mathcal{C}^*}^{*}$, it violates the constraint $\mathcal{C}^*[]$ that all the vertices in $V_{\mathcal{C}^*}^{*}$ are connected. If B is connected with both the vertices in $V_{\mathcal{C}^*}^{*}$ and those in $V_{\mathcal{C}^*}^{\neg}$, it again violates the constraint $\mathcal{C}^*[]$ that the vertices in the different sets of $\mathcal{C}^*[]$ are disconnected. Therefore, since at least $\mathcal{C}^*[]$ is not violated by $\mathtt{R}' \in \mathcal{R}_{\mathsf{Z}}(\mathtt{r}, \mathtt{n})$, only one of the three cases must hold: (i) B is disconnected from any vertex in $\mathcal{C}^*$, (ii) B is connected with all the vertices in $V_{\mathcal{C}^*}^{*}$, and (iii) B is connected with some vertices in $V_{\mathcal{C}^*}^{\neg}$. When $V_{\mathcal{C}^*}^{*} \neq \emptyset$, only case (ii) satisfies $(\#'_B)$. When $V_{\mathcal{C}^*}^{*} = \emptyset$, case (i) also satisfies $(\#'_B)$. For both scenarios, the set of paths in $\mathcal{R}_{\mathtt{n},B}$ that traverse $\mathsf{lo\text{-}arc}(\mathtt{n})$ can be written as $\{\mathsf{lo\text{-}arc}(\mathtt{n})\} \sqcup \mathcal{R}_{\mathsf{Z}}(\mathsf{lo}(\mathtt{n}), \top)$, since $(\#'_B)$ is always satisfied regardless of the choice of the path from $\mathsf{lo}(\mathtt{n})$ to $\top$. By defining $\mathsf{r}_{\uparrow}(\mathtt{n})$ as the sum of path products of the paths in $\mathcal{R}_{\mathsf{Z}}(\mathtt{n}, \top)$ for any dnode $\mathtt{n}$, $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B) = w_{e_i}^{-} \cdot \mathsf{r}_{\uparrow}(\mathsf{lo}(\mathtt{n}))$

for these cases. To sum up, the following holds:

$$\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, B) = \begin{cases} w^{-}_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{q}_{\uparrow}(\mathsf{lo}(\mathtt{n}), \mathsf{lo}(B)) & (\mathsf{lo}(\mathtt{n}) \neq \bot,\ \mathsf{lo}(B) \neq \emptyset) \\ w^{-}_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{r}_{\uparrow}(\mathsf{lo}(\mathtt{n})) & (\mathsf{lo}(\mathtt{n}) \neq \bot,\ \mathsf{lo}(B) = \emptyset, \\ & \quad \text{case (ii) or (case (i) and } V^{*}_{\mathcal{C}^*} = \emptyset)) \\ 0 & (\text{otherwise}) \end{cases} . \tag{9.7}$$

Note that the recurrence formula for $\mathsf{r}_{\uparrow}$ can easily be derived from the definition in the same way as the formula (9.4) for $\mathsf{p}_{\downarrow}$:

$$\mathsf{r}_{\uparrow}(\top) = 1,\ \mathsf{r}_{\uparrow}(\bot) = 0,\ \mathsf{r}_{\uparrow}(\mathtt{n}) = w^{-}_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{r}_{\uparrow}(\mathsf{lo}(\mathtt{n})) + w^{+}_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{r}_{\uparrow}(\mathsf{hi}(\mathtt{n})). \tag{9.8}$$

The remaining is how to distinguish the cases (i)–(iii). Since the connectivity among B and the vertices in $\mathcal{C}^*$ is stored in $\mathtt{n}.\mathsf{vset}$, it can be achieved by the comparison of B and $\mathtt{n}.\mathsf{vset}$. Similar to the notions $V^{*}_{\mathcal{C}^*}$ and $V^{\neg}_{\mathcal{C}^*}$, let $V^{*}_{\mathtt{n}.\mathsf{vset}}$ be the vertices in the set in $\mathtt{n}.\mathsf{vset}$ containing $*$ and let $V^{\neg}_{\mathtt{n}.\mathsf{vset}}$ be the other vertices present in $\mathtt{n}.\mathsf{vset}$. Then, case (i) holds when the vertices in B do not exist in $\mathtt{n}.\mathsf{vset}$. Case (ii) holds when B has a dnode in $V^{*}_{\mathtt{n}.\mathsf{vset}}$. Case (iii) holds when B has a dnode in $V^{\neg}_{\mathtt{n}.\mathsf{vset}}$. Let us see the example by Figure 9.2c. If we perform FBS with $\mathcal{C}^* = \{\{1, 4, *\}\}$, the center dnode of 5th level, say $\mathtt{n}$, becomes $\{3\}\{4\}/\{4, *\}$. When traversing $\mathsf{lo\text{-}arc}(\mathtt{n})$, both $\{3\}$ and $\{4\}$ leave from the frontier. Here $\{3\}$ falls into case (i) and $\{4\}$ falls into case (ii), thus we have $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, \{3\}) = 0$ and $\mathsf{q}_{\uparrow}^{-}(\mathtt{n}, \{4\}) = w^{-}_{e_5} \cdot \mathsf{r}_{\uparrow}(\mathsf{lo}(\mathtt{n})) = w^{-}_{e_5} \cdot \mathsf{r}_{\uparrow}(\top) = w^{-}_{e_5}$.

Almost the same equation holds:

$$\mathsf{q}_{\uparrow}^{+}(\mathtt{n}, B) = \begin{cases} w^{+}_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{q}_{\uparrow}(\mathsf{hi}(\mathtt{n}), \mathsf{hi}(B)) & (\mathsf{hi}(\mathtt{n}) \neq \bot,\ \mathsf{hi}(B) \neq \emptyset) \\ w^{+}_{e_{\mathsf{lb}(\mathtt{n})}} \cdot \mathsf{r}_{\uparrow}(\mathsf{hi}(\mathtt{n})) & (\mathsf{hi}(\mathtt{n}) \neq \bot,\ \mathsf{hi}(B) = \emptyset, \\ & \quad \text{case (ii) or (case (i) and } V^{*}_{\mathcal{C}^*} = \emptyset)) \\ 0 & (\text{otherwise}) \end{cases} , \tag{9.9}$$

except for the following corner case. Let $e_i = \{v, v'\}$. We consider the case where $v, v' \notin F_{i+1}$ and $\mathtt{n}.\mathsf{comp}[v]$ leaves the frontier with case (i). When $V^{*}_{\mathcal{C}^*} \neq \emptyset$, if $v' \in V^{*}_{\mathtt{n}.\mathsf{vset}}$ or $\mathtt{n}.\mathsf{comp}[v']$ leaves the frontier with case (ii), $\mathsf{q}_{\uparrow}^{+}(\mathtt{n}, \mathtt{n}.\mathsf{comp}[v]) = w^{+}_{e_i} \cdot \mathsf{r}_{\uparrow}(\mathsf{hi}(\mathtt{n}))$ since $\mathtt{n}.\mathsf{comp}[v]$ is finally connected with $V^{*}_{\mathcal{C}^*}$. Similarly, when $V^{*}_{\mathcal{C}^*} = \emptyset$, if $v' \in V^{\neg}_{\mathtt{n}.\mathsf{vset}}$ or $\mathtt{n}.\mathsf{comp}[v']$ leaves the frontier with case (iii), $\mathsf{q}_{\uparrow}^{+}(\mathtt{n}, \mathtt{n}.\mathsf{comp}[v]) = 0$ since $\mathtt{n}.\mathsf{comp}[v]$ is finally connected with $V^{\neg}_{\mathcal{C}^*}$.

Algorithm 9.2 describes the pseudocode for DP. After $\mathsf{p}_{\downarrow}$, $\mathsf{q}_{\uparrow}$ and $\mathsf{r}_{\uparrow}$ values are computed by Algorithm 9.2, the $\mathsf{count}(v)$ value for every v can be obtained by (9.6).

Algorithm 9.2: CompDP: dynamic programming with information of comp.

```
r.p↓ ← 1, set all other p↓ values to 0, ⊤.r↑ ← 1, ⊥.r↑ ← 0, set all q↑ values to 0
for $i \leftarrow 1$ to $m$ do                                              // Top-down DP
    foreach n ∈ L_i do
        if lo(n) ≠ ⊥ then p↓(lo(n)) += $w^-_{e_i} \cdot$ p↓(n)                // (9.4)
        if hi(n) ≠ ⊥ then p↓(hi(n)) += $w^+_{e_i} \cdot$ p↓(n)
for $i \leftarrow m$ to 1 do                                              // Bottom-up DP
    foreach n ∈ L_i do
        r↑(n) ← $w^-_{e_i} \cdot$ r↑(lo(n)) + $w^+_{e_i} \cdot$ r↑(hi(n))              // (9.8)
        foreach $B \in$ n.comp do
            if lo(n) ≠ ⊥ and lo($B$) ≠ ∅ then                          // (9.7), 1st case
                q↑⁻(n, $B$) ← $w^-_{e_i} \cdot$ q↑(lo(n), lo($B$))
            else if lo(n) ≠ ⊥ and ($B \cap V^*_{\text{n.vset}} \neq \emptyset$ or ($V^*_{\mathcal{C}^*} = \emptyset$ and $B \cap V^-_{\text{n.vset}} = \emptyset$))
              then
                q↑⁻(n, $B$) ← $w^-_{e_i} \cdot$ r↑(lo(n))                       // (9.7), 2nd case
            if hi(n) ≠ ⊥ and hi($B$) ≠ ∅ then                          // (9.9), 1st case
                q↑⁺(n, $B$) ← $w^+_{e_i} \cdot$ q↑(hi(n), hi($B$))
            else if hi(n) ≠ ⊥ and ($B \cap V^*_{\text{n.vset}} \neq \emptyset$ or ($V^*_{\mathcal{C}^*} = \emptyset$ and $B \cap V^-_{\text{n.vset}} = \emptyset$))
              then
                q↑⁺(n, $B$) ← $w^+_{e_i} \cdot$ r↑(lo(n))                       // (9.7), 2nd case
        Process corner cases
```

9.5.3 Intersection with Base Set

Next we generalize for case $\mathcal{F} \neq 2^E$. In the previous sections, we used the fact that $\mathcal{C}^*[v]$ is a constraint made by adding another constraint $(\#_v)$ to $\mathcal{C}^*[]$. This also holds even if $\mathcal{F}$ is constrained, i.e., $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[v]) = \{E' \mid E' \in \mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[]),\ E' \text{ satisfies } (\#_v)\}$. Therefore, by constructing ZDD Z representing $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[])$ with the information of comp and vset, we can reuse the discussions in Sections 9.5.1 and 9.5.2 and run Algorithm 9.2 on Z to obtain count(v) for every $v \in V$. More specifically, let Z′ be the normalized ZDD of $\mathcal{E}(\mathcal{C}^*[])$ built by FBS with $\mathcal{C}^*$. Then Z should be a normalized ZDD representing $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[])$ that satisfies the following condition for every i: for any i-th subgraph $E' \subseteq E_{<i}$, if E' corresponds to i-th level dnodes n in Z and n′ in Z′, n must have the same comp and vset as n′.

After building $\mathtt{Z}_{\mathcal{F}}$ that represents $\mathcal{F}$ by some means, Z can be built by combining FBS with the existing methods. One approach is to use Apply [Minato, 1993]. First, we build Z′ representing $\mathcal{E}(\mathcal{C}^*[])$ by FBS. Then by taking the set intersection of $\mathtt{Z}_{\mathcal{F}}$ and Z′ with Apply while keeping the information of comp and vset, we can construct Z. The other is to use subsetting [Iwashita and Minato, 2013]. It enables us to directly construct Z from $\mathtt{Z}_{\mathcal{F}}$ in a similar

Algorithm 9.3: Frontier-based search for connectivity constraint $\mathcal{C}$ along with subsetting on $\mathtt{Z}_{\mathcal{F}}$.

```
procedure $\mathit{FBSsubset}_{\mathcal{C},\mathtt{Z}_{\mathcal{F}}}$.Root():
    return $\langle 1, (\emptyset, \mathcal{C}, \mathtt{r}_{\mathcal{F}}) \rangle$                // $\mathtt{r}_{\mathcal{F}}$: $\mathtt{Z}_{\mathcal{F}}$'s root
procedure $\mathit{FBSsubset}_{\mathcal{C},\mathtt{Z}_{\mathcal{F}}}$.Child($\langle i, (\mathsf{comp}, \mathsf{vset}, \mathsf{base}) \rangle, f$):   // $e_i = \{v, v'\}$
    if $f = \mathrm{hi}$ and $i < \mathsf{lb}(\mathsf{base})$ then
        return $\langle m+1, 0 \rangle$                      // No further subgraphs in $\mathcal{F}$
    if $i = \mathsf{lb}(\mathsf{base})$ then
        base $\leftarrow$ (base's $f$-child)                  // base proceeds to child dnode
        if base $= \perp$ then
            return $\langle m+1, 0 \rangle$                  // No further subgraphs in $\mathcal{F}$
    if $f = \mathrm{hi}$ and $v, v' \in V_{\mathsf{vset}}$ and $\mathsf{vset}[v] \neq \mathsf{vset}[v']$ then
        return $\langle m+1, 0 \rangle$                      // $v$ and $v'$ must not be connected
    foreach $u \in \{v, v'\} \setminus F_i$ do               // Vertices entering the frontier
        comp $\leftarrow$ comp $\cup \{\{u\}\}$                // Add $u$ as an isolated vertex
    if $f = \mathrm{hi}$ and $\mathsf{comp}[v] \neq \mathsf{comp}[v']$ then   // Connecting two components
        Merge $\mathsf{comp}[v]$ and $\mathsf{comp}[v']$ into one
        if $v \in V_{\mathsf{vset}}$ and $v' \notin V_{\mathsf{vset}}$ then Add vertices in $\mathsf{comp}[v']$ to $\mathsf{vset}[v]$
        if $v \notin V_{\mathsf{vset}}$ and $v' \in V_{\mathsf{vset}}$ then Add vertices in $\mathsf{comp}[v]$ to $\mathsf{vset}[v']$
    foreach $u \in \{v, v'\} \setminus F_{i+1}$ do           // Vertices leaving from frontier
        if $\{u\} \in$ comp then           // Component containing $u$ leaves frontier
            if $u \in V_{\mathsf{vset}}$ and $\{u\} \notin$ vset and $\{u, *\} \notin$ vset then
                return $\langle m+1, 0 \rangle$     // $u$ must be connected to $w \in \mathsf{vset}[u] \setminus \{u\}$
        Remove $u$ from comp and vset if exists
        Remove empty subset from comp and vset if exists
        if $i = m$ then return $\langle m+1, 1 \rangle$         // All conditions are satisfied
        else return $\langle i+1, (\mathsf{comp}, \mathsf{vset}) \rangle$
```

manner as the FBS. For the sake of completeness, we explain the subsetting approach in detail.

The FBS with subsetting can be described as Algorithm 9.3. Here the red part is newly added elements that are not included in Algorithm 9.1. Given ZDD $\mathtt{Z}_{\mathcal{F}}$ that represents the base set $\mathcal{F}$ and connectivity constraint $\mathcal{C}$, it constructs a ZDD that represents $\mathcal{F} \cap \mathcal{E}(\mathcal{C})$. We add base to a configure, which records the current position of dnode in $\mathtt{Z}_{\mathcal{F}}$. Starting with $\mathtt{r}_{\mathcal{F}}$ where $\mathtt{r}_{\mathcal{F}}$ is the root dnode of $\mathtt{Z}_{\mathcal{F}}$, we traverse the lo-arc of $\mathtt{Z}_{\mathcal{F}}$ if e_i is excluded in the FBS and its hi-arc of if e_i is included in the FBS. We now record the present dnode in $\mathtt{Z}_{\mathcal{F}}$ as $\mathtt{n}.\mathsf{base}$ for each dnode $\mathtt{n}$. If it reaches $\perp$ in $\mathtt{Z}_{\mathcal{F}}$, there are no further subgraphs in $\mathcal{F}$, and so pruning is executed. Note that Algorithm 9.3 a bit complicated than the above explanation since we also cope with the case where $\mathtt{Z}_{\mathcal{F}}$ is not normalized.

9.6 Complexity Analysis

We here conduct a complexity analysis of the proposed algorithm. For connectivity constraint $\mathcal{C}$ possibly including $*$, let $\mathsf{Z}_{\mathrm{FBS}(\mathcal{C})}$ be a ZDD built by FBS with $\mathcal{C}$, let $c_{\mathcal{C}}$ be the number of sets in P, and let $v_{\mathcal{C}}$ be the number of vertices in $\mathcal{C}$ (excluding $*$). We also use frontier width $W_F = \max_i |F_i|$.

First, we bound the running time of our algorithm by the ZDD size.

Proposition 9.2. *Our proposed algorithm runs in $O(W_F \cdot |\mathsf{Z}_{\mathcal{F}}||\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}|)$ time.*

Proof. By storing comp and vset as an integer sequence whose length is W_F as described in Chapter 4, FBS with an intersection runs in $O(W_F \cdot |\mathsf{Z}|)$ time where Z is the resultant ZDD. Since $|\mathsf{Z}|$ can be bounded by the product of $|\mathsf{Z}_{\mathcal{F}}|$ and $|\mathsf{Z}_{\mathrm{FBS}(P)}|$ [Minato, 1993], $|\mathsf{Z}| = O(|\mathsf{Z}_{\mathcal{F}}||\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}|)$ in the proposed algorithm. For each dnode $\mathtt{n}$, $\mathsf{p}_{\downarrow}(\mathtt{n})$, $\mathsf{r}_{\uparrow}(\mathtt{n})$, and $\mathsf{q}_{\uparrow}(\mathtt{n}, B)$ can be computed in constant time, and there are at most W_F sets in comp. Thus, the DP computation is completed in $O(W_F \cdot |\mathsf{Z}|)$ time. The complete computation of $\mathsf{count}(v)$ can be done in $O(|\mathsf{Z}|)$ time; by choosing i such that e_i is the first edge containing v for every v, each level of Z is scanned at most twice for computing $\mathsf{count}(v)$ for every v. □

Next we bound the ZDD sizes. The bound of $|\mathsf{Z}_{\mathrm{FBS}(\mathcal{C})}|$ for $\mathcal{C}$ excluding $*$ is given in Proposition 9.3 and that of $|\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}|$ for $\mathcal{C}^*$ including $*$ is in Proposition 9.4.

Proposition 9.3. *The size of $\mathsf{Z}_{\mathrm{FBS}(\mathcal{C})}$ for connectivity constraint $\mathcal{C}$ excluding $*$ is bounded by $O(mE_{W_F} \cdot \min\{(c_{\mathcal{C}}+1)^{W_F}, (W_F+1)^{v_{\mathcal{C}}}\})$, where E_{W_F} is the W_F-th Bell number.*

Proof. We consider the number of possible patterns for the comp and vset pair. We focus on an i-th level. Since comp is simply a partition of F_i, the number of possible patterns for it is $E_{|F_i|}$. The number of possible patterns for vset can be bounded in two ways. First, vset retains the information of how the components in vset are connected to each set in P. Each component of comp is connected to at most one set in $\mathcal{C}$, since if more than two sets are connected, connectivity constraint $\mathcal{C}$ is violated. Since there are at most $|F_i|$ components, the number of vset patterns is bounded by $(c_{\mathcal{C}}+1)^{|F_i|}$, where $+1$ deals with the case where no component is connected to any sets in $\mathcal{C}$. Second, vset can be seen as retaining the information of how the vertices in $\mathcal{C}$ are connected to the component in comp. Thus, the number of vset patterns is bounded by $(|F_i|+1)^{v_{\mathcal{C}}}$, where $+1$ deals with the case where a vertex in $\mathcal{C}$ is not connected to any component in comp. To sum up, the number of patterns of the comp and vset pair can be bounded by $O(E_{|F_i|} \cdot \min\{(c_{\mathcal{C}}+1)^{|F_i|}, (|F_i|+1)^{vc}\})$.

Since there are m levels and $|F_i| \leq W_F$, the overall size is bounded by $O(mE_{W_F} \cdot \min\{(c_{\mathcal{C}}+1)^{W_F}, (W_F+1)^{vc}\})$. □

Proposition 9.4. *For connectivity constraint $\mathcal{C}^*$ including $*$, $|\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}| \leq c_{\mathcal{C}^*}|\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*[])}|$.*

Proof. Since running Algorithm 9.1 with $\mathcal{C}^*$ and $\mathcal{C}^*[]$ yields the same representing family of sets, $\mathcal{E}(\mathcal{C}^*[])$, we only have to address the number of patterns of comp and vset. The only difference is that when running FBS with $\mathcal{C}^*$, we must determine which set in vset has $*$. Since there are at most $c_{\mathcal{C}^*}$ sets in vset, there will be at most $c_{\mathcal{C}^*}$ patterns of vset for a dnode in $\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*[])}$ when running FBS with $\mathcal{C}^*$. Thus, $|\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}| \leq c_{\mathcal{C}^*}|\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*[])}|$ holds. □

Combining Propositions 9.2–9.4 yields the following theorem.

Theorem 9.5. *The proposed algorithm runs in $O(W_F \cdot c_{\mathcal{C}^*}|\mathsf{Z}_{\mathcal{F}}||\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*[])}|)$ time, which is bounded by $O(|\mathsf{Z}_{\mathcal{F}}| \cdot mc_{\mathcal{C}^*} \cdot W_F \cdot D_{W_F} \cdot \min\{(c_{\mathcal{C}^*}+1)^{W_F}, (W_F+1)^{vc^*}\})$.*

If W_F can be considered as a constant, the proposed algorithm runs in $O(mc_{\mathcal{C}^*}|\mathsf{Z}_{\mathcal{F}}|)$ time.

We compare this complexity with the baseline method where we separately build a ZDD representing $\mathcal{F} \cap \mathcal{E}(\mathcal{C}^*[v])$ by FBS. The overall complexity is $O(W_F \cdot \sum_v |\mathsf{Z}_{\mathcal{F}}||\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*[v])}|)$, analyzed in the same way as Proposition 9.2, which is bounded by $O(|\mathsf{Z}_{\mathcal{F}}| \cdot mn \cdot W_F \cdot D_{W_F} \cdot \min\{(c_{\mathcal{C}^*}+1)^{W_F}, (W_F+1)^{vc^*+1}\})$. If W_F is constant, it is $O(|\mathsf{Z}_{\mathcal{F}}|mn)$. Compared with Theorem 9.5, the proposed method runs faster by an $O(n)$ factor.

Here we mention the ZDD sizes. The complexity bounds of the proposed and baseline methods heavily depend on $\mathsf{Z}_{\mathcal{F}}$'s size. Here $|\mathsf{Z}_{\mathcal{F}}|$ also remains small for various constraints if W_F is small. For example, the constraints appeared in the example of Section 9.2, e.g., the degree constraints and the existence of cycles, can all be represented as a ZDD whose size is proportional to m if W_F is constant [Sekine *et al.*, 1995; Knuth, 2011; Kawahara *et al.*, 2017b]. This boosts the effectiveness of both the proposed and baseline methods for practical use because W_F is often much smaller than n and m for graphs in real worlds. Moreover, $|\mathsf{Z}|$ is often much smaller than expected from the above analysis, as demonstrated by Kawahara *et al.* [2017b].

We close this section by mentioning the space complexity of the proposed and the baseline methods. The proposed algorithm uses at most $O(W_F \cdot |\mathsf{Z}_{\mathcal{F}}||\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}|)$ words of space, since it retains $O(W_F)$ words of information for each dnode of $\mathsf{Z}_{\mathrm{FBS}(\mathcal{C}^*)}$. The baseline method typically uses at most $O(W_F \cdot |\mathtt{Z}_{\mathcal{F}}||\mathtt{Z}_{\mathrm{FBS}(\mathcal{C}^*[v])}|)$ words of space for the computation of $\mathtt{count}(v)$. If it is assumed that $|\mathtt{Z}_{\mathrm{FBS}(\mathcal{C}^*[v])}|$ is close to $|\mathtt{Z}_{\mathrm{FBS}(\mathcal{C}^*[])}|$, the space complexity of the baseline method is at most only $O(c_{\mathcal{C}^*})$ times smaller due to Proposition 9.4.

9.7 Experiments

We empirically compared the proposed and the baseline methods with respect to the computational time. Here the baseline method is to separately build a ZDD by FBS for each constraint. Both methods were implemented in C++ and compiled by g++ with -O3 option. We used TdZdd (`https://github.com/kunisura/TdZdd`) for the baseline method, which is a highly optimized C++ library for FBS. We also used TdZdd for the proposed method to construct ZDD $\mathtt{Z}_{\mathcal{F}}$ of base set. Experiments are conducted on a single thread of a Linux machine with AMD EPYC 7763 2.45 GHz CPU and 2048 GB RAM; note that we used less than 256 GB of memory during the experiments. We set the time limit of every run to 600 seconds.

We used both synthetic graphs and real benchmarks as tested graphs. The synthetic ones are grid graphs; Grid-wxh represents a grid graph with $w \times h$ vertices. For the others, we used the Rocketfuel [Spring *et al.*, 2004] and Romegraph datasets (retrieved from `http://www.graphdrawing.org/data.html`). Rocketfuel was also used in Chapters 5 and 6. From Romegraph, we chose all the graphs with $n = 100$: there were 140 such graphs. Identical edge ordering was used for both methods, and it was decided as follows: For the grid graphs, we used the edge ordering of Iwashita *et al.* [2013], which is better for the DP on grid graphs. For the other graphs, we used beam-search heuristics [Inoue and Minato, 2016] to determine the edge ordering.

We evaluated four problem settings in Section 9.2: path, cycle, Steiner tree, and rooted spanning forest (RSF). The given vertices for these settings were determined as follows. Let $d(v, v')$ be the shortest distance between vertices v and v'. For the path problem, we chose the most distant vertex pair as s, t, i.e., s, t satisfies $d(s, t) = \max_{v,v'} d(v, v')$. For the cycle problem, we chose the graph center as s, i.e., $s \in \mathrm{argmin}_v \max_{v'} d(v, v')$. For the other problems, we chose four vertices as T such that the sum of the distances between distinct vertices, $\sum_{v,v' \in T : v \neq v'} d(v, v')$, is maximized.

Table 9.1 shows the result for the grid graphs and the Rocketfuel dataset. For all the graphs and problem settings solved by both methods within the time limit, the proposed method ran about 10–20 times faster than the baseline method. The complexity analyses in Section 9.6 suggest that the proposed method becomes faster than the existing method when n is large.

Table 9.1: Computational time for grid graphs and Rocketfuel dataset in seconds

				Path		Cycle		Steiner tree		RSF	
Instance	n	m	W_F	Ours	Base	Ours	Base	Ours	Base	Ours	Base
Grid-8x8	64	112	8	**0.06**	0.48	**0.06**	0.54	**4.93**	29.87	**8.87**	38.17
Grid-8x16	128	232	8	**0.16**	2.54	**0.16**	2.81	**14.16**	183.16	**25.72**	234.17
Grid-8x24	192	352	8	**0.26**	6.29	**0.27**	6.86	**23.27**	470.60	**42.81**	>600
Grid-8x32	256	472	8	**0.36**	11.73	**0.38**	12.65	**32.55**	>600	**60.28**	>600
Grid-9x9	81	144	9	**0.22**	2.13	**0.22**	2.34	**40.13**	298.57	**62.44**	397.57
Grid-10x10	100	180	10	**0.78**	9.70	**0.89**	11.69	**284.17**	>600	**430.04**	>600
Grid-11x11	121	220	11	**2.88**	42.26	**3.22**	50.97	>600	>600	>600	>600
Grid-12x12	144	264	12	**11.24**	183.87	**15.25**	241.94	>600	>600	>600	>600
Grid-13x13	169	312	13	**45.51**	>600	**56.25**	>600	>600	>600	>600	>600
Rocketfuel-1221	318	758	10	**157.01**	>600	**111.52**	>600	**181.38**	>600	>600	>600
Rocketfuel-1755	172	381	12	**43.94**	>600	**32.48**	>600	>600	>600	>600	>600
Rocketfuel-6461	182	294	10	**3.66**	65.64	**4.80**	128.86	**71.10**	>600	**95.47**	>600

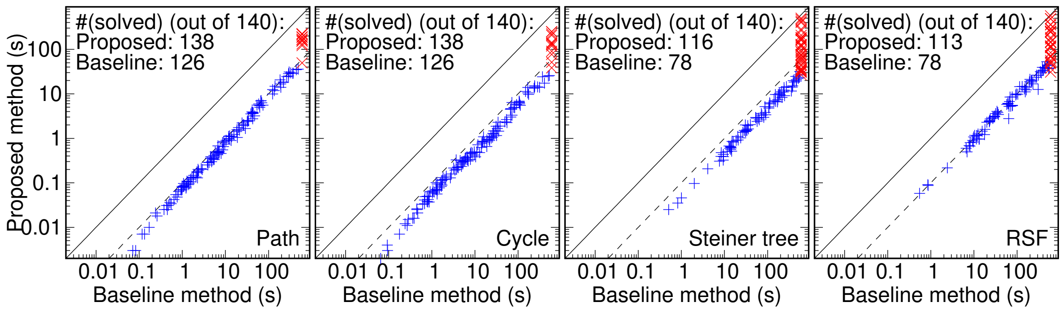


Figure 9.3: Computational time for Romegraph dataset: Blue points indicate instances solved by both methods, and red points indicate those solved only by proposed method. Solid black lines indicate elapsed time for both methods is identical, and dashed lines indicate proposed method is 10 times faster than baseline method.

Table 9.1 exhibits such a tendency. For example, for the Grid-8xh graphs, the proposed method becomes much faster than the baseline method when $n = 8h$ is increased. In addition, both methods ran faster for graphs with smaller W_F value, reflecting the complexity analyses.

Figure 9.3 plots the result for the Romegraph dataset and also describes the number of graphs solved by each method within the time limit. Here each point corresponds to a graph, where the blue ones are those solved by both methods and the red ones are those solved only by the proposed method. Although the computational time itself varied from less than 0.01 to 600 seconds, for almost all the graphs the proposed method ran about 10–20 times faster than the existing method. This ratio is kept because the graphs all have the same number of vertices: 100. We give detailed results for Romegraph in Section 9.9.2.

9.8 Conclusion

We proposed a novel framework, compDP, for solving multiple subgraph counting problems with similar connectivity constraints simultaneously. A complexity analysis showed that the proposed method ran $O(n)$ times faster than the baseline approach, and the experiments revealed the efficiency of the proposed method.

As a future work, we will consider dealing with the reachability in directed graphs. There are approaches for building BDDs of reachability constraints [Maehara *et al.*, 2017; Suzuki *et al.*, 2018], and we want to consider whether they can be incorporated into our framework.

9.9 Appendix

9.9.1 Treatment of Degree 1 Vertices

As in Section 9.5.1, we can compute the $\mathsf{count}(v)$ value by focusing on the i-th level of $\mathtt{Z}$ where $v \in F_i$. However, if v's degree is 1, no such i exists. Let $e_i = \{v, v'\}$ be the only edge incident to v. Then for $i' < i$, v is not present in $E_{\geq i'}$, and for $i' \geq i$, it is not present in $E_{<i'}$, so no frontier contains v. Therefore, we have an alternative formula for computing $\mathsf{count}(v)$ like (9.6). Let us focus on i-th level where $e_i = \{v, v'\}$ is the only edge incident to v.

First, we address the case where the other endpoint, v', is in F_i. Let $\mathtt{n} \in \mathsf{L}_i$ be an arbitrarily chosen i-th level dnode of $\mathtt{Z}$. Since e_i is the only edge incident to v, if e_i is excluded, v remains as an isolated vertex. Thus, if $V^*_{\mathcal{C}^*} \neq \emptyset$, constraint $(\#_v)$ will never be satisfied. Otherwise, if $V^*_{\mathcal{C}^*} = \emptyset$, constraint $(\#_v)$ is always satisfied. In this case, the set of paths in $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \top)$ that passes $\mathtt{n}$ whose corresponding subgraph satisfies $(\#_v)$, i.e., $\mathcal{R}_{v,\mathtt{n}}$, can be written as $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \mathtt{n}) \sqcup \{\mathsf{lo\text{-}arc}(\mathtt{n})\} \sqcup \mathcal{R}_{\mathtt{Z}}(\mathsf{lo}(\mathtt{n}), \top)$. The sum of their path products is $\mathsf{p}_{\downarrow}(\mathtt{n}) \cdot w^-_{e_i} \cdot \mathsf{r}_{\uparrow}(\mathsf{lo}(\mathtt{n}))$ given that $\mathsf{lo}(\mathtt{n}) \neq \perp$. If e_i is included, v is connected with v', and so condition $(\#_v)$ is met if and only if condition $(\#'_B)$ for $B = \mathtt{n}.\mathsf{comp}[v']$ is met. In this case, the set of paths in $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \top)$ that passes $\mathtt{n}$ whose corresponding subgraph satisfies $(\#_v)$ can be written as $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \mathtt{n}) \sqcup \mathcal{R}^+_{\mathtt{n},\mathtt{n}.\mathsf{comp}[v']}$, where $\mathcal{R}^+_{\mathtt{n},B}$ is the set of paths in $\mathcal{R}_{\mathtt{n},B}$ that passes through $\mathsf{hi\text{-}arc}(\mathtt{n})$. The sum of their path products is $\mathsf{p}_{\downarrow}(\mathtt{n}) \cdot \mathsf{q}^+_{\uparrow}(\mathtt{n}, \mathtt{n}.\mathsf{comp}[v'])$ using the notion $\mathsf{q}^+_{\uparrow}$ introduced in Section 9.5.2. The value $\mathsf{count}(v)$ can be computed by their sum over $\mathtt{n} \in \mathsf{L}_i$:

$$
\begin{aligned}
\mathsf{count}(v) = &\sum_{\mathtt{n} \in \mathsf{L}_i} \mathsf{p}_{\downarrow}(\mathtt{n}) \cdot \mathsf{q}^+_{\uparrow}(\mathtt{n}, \mathtt{n}.\mathsf{comp}[v']) \\
&+ \begin{cases} 0 & (V^*_{\mathcal{C}^*} \neq \emptyset) \\ \sum_{\mathtt{n} \in \mathsf{L}_i : \mathsf{lo}(\mathtt{n}) \neq \perp} \mathsf{p}_{\downarrow}(\mathtt{n}) \cdot w^-_{e_i} \cdot \mathsf{r}_{\uparrow}(\mathsf{lo}(\mathtt{n})) & (V^*_{\mathcal{C}^*} = \emptyset) \end{cases} .
\end{aligned}
\tag{9.10}
$$

The remaining issue is how to cope with case $v' \notin F_i$. For it, we can assume $v' \in F_{i+1}$; otherwise, v' is also a degree 1 vertex that means graph G consists of only e_i since G is connected, which is trivial. The case where e_i is excluded is treated in the same way as above. If e_i is included, let $\mathtt{n}$ be an arbitrary i-th level dnode of $\mathtt{Z}$. Since v is connected with v', constraint $(\#_v)$ is met if and only if constraint $(\#'_B)$ for $B = \mathsf{lo}(\mathtt{n}).\mathsf{comp}[v']$ is met. Therefore, the set of paths in $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \top)$ that passes $\mathtt{n}$ whose corresponding subgraph satisfies $(\#_v)$ can be written as $\mathcal{R}_{\mathtt{Z}}(\mathtt{r}, \mathtt{n}) \sqcup \{\mathsf{hi\text{-}arc}(\mathtt{n})\} \sqcup \mathcal{R}_{\mathsf{hi}(\mathtt{n}),\mathsf{hi}(\mathtt{n}).\mathsf{comp}[v']}$, given that $\mathsf{hi}(\mathtt{n}) \neq \perp$. The sum of their path products is

Table 9.2: Comparison of the number of solved graphs in Romegraph dataset within time limit for each frontier width.

		Path		Cycle		Steiner tree		RSF	
W_F	#(graphs)	Ours	Base	Ours	Base	Ours	Base	Ours	Base
6	3	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**
7	7	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
8	26	**26**	**26**	**26**	**26**	**26**	**26**	**26**	**26**
9	28	**28**	**28**	**28**	**28**	**28**	27	**28**	**28**
10	33	**33**	**33**	**33**	**33**	**33**	15	**33**	14
11	25	**25**	24	**25**	24	**18**	0	**15**	0
12	10	**10**	5	**10**	5	**1**	0	**1**	0
13	6	**6**	0	**6**	0	0	0	0	0
14	2	0	0	0	0	0	0	0	0
Total	140	**138**	126	**138**	126	**116**	78	**113**	78

$\mathsf{p}_\downarrow(\mathtt{n}) \cdot w_{e_i}^{+} \cdot \mathsf{q}_\uparrow(\mathsf{hi}(\mathtt{n}), \mathsf{hi}(\mathtt{n}).\mathsf{comp}[v'])$. The value $\mathsf{count}(v)$ can be computed by

$$\begin{aligned}\mathsf{count}(v) = &\sum_{\mathtt{n}\in \mathsf{L}_i : \mathsf{hi}(\mathtt{n})\neq\bot} \mathsf{p}_\downarrow(\mathtt{n}) \cdot w_{e_i}^{+} \cdot \mathsf{q}_\uparrow(\mathsf{hi}(\mathtt{n}), \mathsf{hi}(\mathtt{n}).\mathsf{comp}[v']) \\ &+ \begin{cases} 0 & (V_{\mathcal{C}^*}^* \neq \emptyset) \\ \sum_{\mathtt{n}\in \mathsf{L}_i : \mathsf{lo}(\mathtt{n})\neq\bot} \mathsf{p}_\downarrow(\mathtt{n}) \cdot w_{e_i}^{-} \cdot \mathsf{r}_\uparrow(\mathsf{lo}(\mathtt{n})) & (V_{\mathcal{C}^*}^* = \emptyset) \end{cases}.\end{aligned} \tag{9.11}$$

9.9.2 Detailed Results for Romegraph Dataset

Next we describe a detailed experimental results for the Romegraph dataset, which has 140 graphs whose number of dnodes is exactly 100. With beam-search heuristics [Inoue and Minato, 2016], the frontier width W_F of each graph ranges from 6 to 14. The number of graphs per value of W_F is described in Table 9.2.

Table 9.2 also shows the number of graphs solved within the time limit for each method, each problem setting, and each frontier width value. In addition, Figure 9.4 plots the computational time for the Romegraph dataset aggregated by frontier width W_F. For both methods, the graphs with a larger W_F value are clearly difficult to solve, i.e., time-consuming; this outcome reflects the complexity results in Section 9.6. However, our proposed method can also treat graphs with a larger W_F value than the baseline method. This again clearly indicates the efficiency of our proposed method.

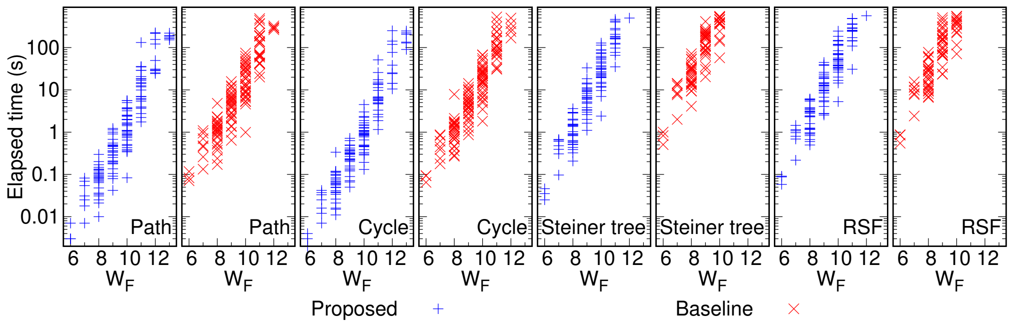


Figure 9.4: Computational time for Romegraph dataset aggregated by frontier width W_F. Blue points indicate results of proposed method, and red points indicate results of baseline method.

Chapter 10

Conclusion

In this dissertation, we have delved into three areas of network analysis problems: combinatorial congestion games, network reliability analysis, and subgraph counting problems. Although all of the problems dealt in this dissertation fall into computationally difficult complexity classes such as NP-hard and #P-complete, we proposed a practically fast algorithm for every problem by fully using the BDDs and similar structures. First, we summarize the contributions and possible future directions of each chapter.

Chapter 3: Equilibrium Computation of Combinatorial Congestion Games. We have developed a practically fast iterative equilibrium computation method for general continuous-player combinatorial congestion games. The proposed method combines ZDDs for representing strategy sets with Frank–Wolfe-style iterative optimization algorithms. Experiments revealed the practical efficiency of the proposed method for realistic situations.

Future directions include the *optimization* of equilibrium to reach better social cost by modifying the parameters within network infrastructure design. This problem is addressed by the subsequent work [Sakaue and Nakamura, 2021], where the proposed algorithm again uses ZDDs to represent strategy sets along with a differentiable modification of Frank–Wolfe-style algorithm.

Chapter 5: Network Reliability Evaluation for Client-Server Model. An efficient algorithm for the CSNR problem, which is equivalent to compute K-NR $O(n)$ times, has been developed whose time complexity is the same as just computing single K-NR with the existing method. Its practical efficiency was verified by experiments with grid and real-world topologies with hundreds of edges. As a future work, it should be examined to extend the work to compute network reliability for every *pair* of vertices, which corresponds to the network reliability evaluation for point-to-point infrastructures.

Chapter 6: Fast Evaluation for the Expected Number of Connected Nodes. An extremely efficient algorithm for computing ECP or equivalently multiple ECNs has been developed. Experiments confirmed its practical efficiency; the proposed method can compute ECP of complicated real-world network topologies with hundreds of edges within around ten minutes. Applications for critical link identification and server placement have also been exhibited. Future direction includes the support of vertex failures and analyses of ECP and ECN values using real-world population data.

Chapter 7: Variance Analysis on Network Reliability. We have developed an efficient algorithm for computing VoR with BDDs; it computed the VoR of real-world topologies with around 200 edges within 0.1 seconds. We have also conducted empirical analyses on VoR, which reveals us the smallness and robustness of VoR that are desirable for network design. Computing VoR under different network models such as vertex failure and dependent failure should be the future works of this chapter.

Chapter 8: Efficient Computation of Scale-wise Network Unreliability. We have established an efficient algorithm to compute scale-wise network unreliability by constructing multi-terminal variant of BDDs. Numerical experiments verified that the scale-wise unreliability of the real-world topologies with up to 200 edges can be computed within 1.3 hours. We have also plotted the scale-wise unreliability of real topologies, which unveils some interesting properties. As a future work, the support for an outage scale with user count instead of vertex count is in demand.

Chapter 9: Framework for Simultaneous Subgraph Counting under Connectivity Constraints. We have proposed a novel framework for solving multiple subgraph counting with similar connectivity constraints simultaneously. The computational time of the proposed algorithm is analyzed both theoretically and empirically. Specifically, the proposed algorithm runs an order-of-magnitude faster than the existing approach for graphs with 100 vertices for various graph constraints. As a future direction, it should be considered whether reachability constraints in directed graphs can be handled in similar way as this work.

10.1 Essence and Future Direction

Now we conclude this dissertation by summarizing the essences of preceding chapters and showing the future directions related to all of the chapters.

Generally, the size of a BDD or a ZDD representing a family of subsets is exponential with respect to the size of the base item set. However, when the desired family stems from a graph with smaller path-width, the BDD's size can be bounded theoretically, and it remains small empirically. This property makes BDDs and similar structures desirable for network analysis problems since the network is often modeled as a sparse graph whose path-width is typically small. By developing an elaborated procedure working on the decision diagram structures that represent the family of subgraphs stemming from a sparse graph, we have proposed a practically efficient algorithm for every problem. The time complexity of every algorithm is theoretically bounded with the frontier-width value W_F that can be equal to the path-width W_p.

From the viewpoint of algorithmic aspects, the algorithms proposed in this dissertation are accomplished by expanding both the construction procedures of data structures and the dynamic programming procedures on them. In constructing decision diagram structures, we are basically based on the top-down construction method described in Section 2.5 since it is suitable for constructing a DD structure of a family of subgraphs. However, in some chapters, we carefully manipulate the construction procedure to be suited for the subsequent dynamic programming computation. For example, in Chapter 8, we construct a multi-terminal variant of BDD to compute the scale-wise unreliability. In dynamic programming procedures, we are focusing on the fact that the decision diagram structures represent the family of subgraphs in a recursive manner as in Chapter 2. That is, as well as the whole decision diagram structure represents a family of subgraphs, each dnode in the structure also represents a family of subgraphs. This enables us to develop elaborated dynamic programming procedures working on them by keeping track of what is computed for every dnode. For example, in Chapter 7, we associate every dnode with a random variable corresponding to the family of subgraphs represented by this dnode and compute the covariances among these random variables by dynamic programming. Other examples are Chapters 5, 6, and 9: here, we associate every dnode with probabilities, accumulated probabilities, or more generally count values corresponding to the represented family of subgraphs and develop top-down and bottom-up dynamic programming procedures. These ingredients make the decision diagram structures suitable for practically solving network analysis problems in an exact manner.

The future direction of this dissertation is as follows. First, although we only deal with edge-induced subgraphs, we should also deal with vertex-induced subgraphs. For example, in network reliability analysis, the failure of a network node can be modeled with an absence of a vertex, which arouses the treatment for vertex-induced subgraphs. Although there are some works for treating vertex-induced subgraphs with BDDs and ZDDs (e.g., [Kawahara *et al.*, 2019]), whether such works can be combined with the algorithms proposed in this dissertation should be verified for each. Second, although all of the proposed algorithms in this dissertation deal with undirected graphs, the support for directed graphs is in demand. For example, the network reliability analysis on directed graphs is equivalent to the analysis of information spread [Maehara *et al.*, 2017], which is important for the analysis of social networks. There are some works for constructing a BDD or a ZDD representing a family of subgraphs of a directed graph, e.g., [Maehara *et al.*, 2017; Suzuki *et al.*, 2018]. The algorithms in Chapters 3 and 7 may work for the BDDs constructed by the above algorithm. However, we should consider whether the top-down and bottom-up DP framework in Chapters 5, 6, and 9 can be incorporated into these works. Since we use the fact that the connectivity in an undirected graph is an equivalence relation in this framework and the reachability in a directed graph is not an equivalence relation, it may be not straightforward. Also, we should newly consider the decision diagram construction considering the outage scale on a directed graph like Chapter 8.

Bibliography

Jacob D. Abernethy and Jun-Kun Wang. On Frank–Wolfe and equilibrium computation. In *Proc. of Advances in Neural Information Processing Systems 30* (*NIPS 2017*), pages 6584–6593. Curran Associates, Inc., 2017.

Hosam M. F. AboElFotoh, Sundararaja S. Iyengar, and Krishnendu Chakrabarty. Computing reliability and message delay for cooperative wireless distributed sensor networks subject to random failures. *IEEE Transactions on Reliability*, 54(1):145–155, 2005.

Pankaj K. Agarwal, Alon Efrat, Shashidhara K. Ganjugunte, David Hay, Swaminathan Sankararaman, and Gil Zussman. Network vulnerability to single, multiple, and probabilistic physical attacks. In *Proc. of IEEE Military Communications Conference* (*MILCOM 2010*), pages 1824–1829, 2010.

Anuj Agrawal, Vimal Bhatia, and Shashi Prakash. Network and risk modeling for disaster survivability analysis of backbone optical communication networks. *Journal of Lightwave Technology*, 37(10):2352–2362, 2019.

Yaser Al Mtawa, Anwar Haque, and Hanan Lutfiyya. Migrating from legacy to software defined networks: A network reliability perspective. *IEEE Transactions on Reliability*, 70(4):1525–1541, 2021.

Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

Eitan Altman, Thomas Boulogne, Rachid El-Azouzi, Tania Jiménez, and Laura Wynter. A survey on networking games in telecommunications. *Computers and Operations Research*, 33(2):286–311, 2006.

Chamitha De Alwis, Anshuman Kalla, Quoc-Viet Pham, Pardeep Kumar, Kapal Dev, Won-Joo Hwang, and Madhusanka Liyanage. Survey on 6G frontiers: Trends, applications, requirements, technologies and future research. *IEEE Open Journal of the Communications Society*, 2:836–886, 2021.

Osamu Aso, Toshio Matsufuji, Takuya Ishikawa, Masateru Tadakuma, Soichiro Otosu, Takeshi Yagi, and Masato Oku. Inference of the optical fiber lifetime for mechanical reliability. *Furukawa Review*, 42:1–6, 2012.

Hillel Bar-Gera. Origin-based algorithm for the traffic assignment problem. *Transportation Science*, 36(4):398–417, 2002.

Eric T. Bax. Algorithms to count paths and cycles. *Information Processing Letters*, 52(5):249–252, 1994.

Martin Beckmann, C. B. McGuire, and Christopher B. Winsten. *Studies in the Economics of Transportation.* Yale University Press, 1956.

Avrim Blum, Eyal Even-Dar, and Katrina Ligett. Routing without regret: On convergence to Nash equilibria of regret-minimizing algorithms in routing games. In *Proc. of the 25th Annual ACM Symposium on Principles of Distributed Computing* (*PODC 2006*), pages 45–52. ACM, 2006.

Francis T. Boesch, Appajosyula Satyanarayana, and Charles L. Suffel. A survey of some network reliability analysis and synthesis results. *Networks*, 54(2):99–107, 2009.

Zdravko I. Botev, Pierre L'Ecuyer, and Bruno Tuffin. Dependent failures in highly reliable static networks. In *Proc. of the 2012 Winter Simulation Conference* (*WSC 2012*), pages 1–12, 2012.

Ulrik Brandes and Daniel Fleischer. Centrality measures based on current flow. In *Proc. of Annual Symposium on Theoretical Aspects of Computer Science* (*STACS 2005*), pages 533–544. Springer, 2005.

Gábor Braun, Sebastian Pokutta, and Daniel Zink. Lazifying conditional gradient algorithms. *The Journal of Machine Learning Research*, 20(71):1–42, 2019.

Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

Eduardo Canale, Franco Robledo, Pablo Romero, and Pablo Sartor. Monte Carlo methods in diameter-constrained reliability. *Optical Switching and Networking*, 14:134–148, 2014.

Héctor Cancela and Mohamed El Khadiri. A recursive variance-reduction algorithm for estimating communication-network reliability. *IEEE Transactions on Reliability*, 44(4):595–602, 1995.

Héctor Cancela, Mohamed El Khadiri, Gerardo Rubino, and Bruno Tuffin. Balanced and approximate zero-variance recursive estimators for the network reliability problem. *ACM Transactions on Modeling and Computer Simulation*, 25(1), 2014.

Sanjay Kumar Chaturvedi. *Network reliability: measures and evaluation*. John Wiley & Sons, 2016.

Reuven Cohen, Keren Erez, Daniel Ben-Avraham, and Shlomo Havlin. Breakdown of the internet under intentional attack. *Physical review letters*, 86(16):3682, 2001.

José R. Correa and Nicolás E. Stier-Moses. Wardrop equilibria. In *Wiley Encyclopedia of Operations Research and Management Science*. Wiley Online Library, 2011.

Radu Curticapean. Counting problems in parameterized complexity. In *Proc. of the 13th International Symposium on Parameterized and Exact Computation* (*IPEC 2018*), pages 1:1–1:18, 2018.

Basima Elshqeirat, Sieteng Soh, Suresh Rai, and Mihai Lazarescu. Topology design with minimal cost subject to network reliability constraint. *IEEE Transactions on Reliability*, 64(1):118–131, 2015.

Alex Fabrikant, Christos Papadimitriou, and Kunal Talwar. The complexity of pure Nash equilibria. In *Proc. of the 36th Annual ACM Symposium on Theory of Computing* (*STOC 2004*), pages 604–612. ACM, 2004.

Federal Communications Commission. Network Outage Reporting System (NORS). `https://www.fcc.gov/network-outage-reporting-system-nors`.

Simon Fischer and Berthold Vöcking. On the evolution of selfish routing. In *Proc. of the 12th European Symposium on Algorithms* (*ESA 2004*), pages 323–334. Springer-Verlag, 2004.

Simon Fischer, Herald Räcke, and Berthold Vöcking. Fast convergence to Wardrop equilibria by adaptive sampling methods. *SIAM Journal on Computing*, 39(8):3700–3735, 2010.

Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1–2):95–110, 1956.

András Frank. Augmenting graphs to meet edge-connectivity requirements. *SIAM Journal on Discrete Mathematics*, 5(1):25–53, 1992.

Luigi Fratta and Ugo Montanari. A Boolean algebra method for computing the terminal reliability in a communication network. *IEEE Transactions on Circuit Theory*, 20(3):203–211, 1973.

Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.

Narumi Fukuda et al. On network quality design. In *Japan Network Operators Group Meeting 46*, 2020. `https://www.janog.gr.jp/meeting/janog46/wp-content/uploads/2020/06/janog46-quality.pdf#page=31`, in Japanese.

Vaibhav Gaur, Om Prakash Yadav, Gunjan Soni, and Ajay Pal Singh Rathore. A literature review on network reliability analysis and its engineering applications. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 235(2):167–181, 2021.

Ilya B. Gertsbakh and Yoseph Shpungin. *Models of network reliability: analysis, combinatorics, and Monte Carlo*. CRC press, 2009.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, 2008.

Zhu Han, Dusit Niyato, Walid Saad, Tamer Başar, and Are Hjørungnes. *Game Theory in Wireless and Communication Networks: Theory, Models, and Applications*. Cambridge University Press, 1st edition, 2012.

Gary Hardy, Corinne Lucet, and Nikolaos Limnios. Computing all-terminal reliability of stochastic networks with binary decision diagrams. In *11th International Symposium on Applied Stochastic Models*, pages 1469–74, 2005.

Gary Hardy, Corinne Lucet, and Nikolaos Limnios. K-terminal network reliability measures with binary decision diagrams. *IEEE Transactions on Reliability*, 56(3):506–515, 2007.

Masahiro Hayashi and Takeo Abe. Evaluating reliability of telecommunications networks using traffic path information. *IEEE Transactions on Reliability*, 57(2):283–294, 2008.

Johannes U. Herrmann. Improving reliability calculation with augmented binary decision diagrams. In *Proc. of the 24th IEEE International Conference on Advanced Information Networking and Applications* (*AINA 2010*), pages 328–333, 2010.

Robin J. Hogan. Fast reverse-mode automatic differentiation using expression templates in c++. *ACM Transactions on Mathematical Software*, 40(4):1–16, 2014.

Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

Yuma Inoue and Shin-ichi Minato. Acceleration of ZDD construction for subgraph enumeration via pathwidth optimization. Technical Report TCS-TR-A-16-80, Division of Computer Science, Hokkaido University, 2016.

Takeru Inoue, Norihito Yasuda, Shunsuke Kawano, Yuji Takenobu, Shin-ichi Minato, and Yasuhiro Hayashi. Distribution network verification for secure restoration by enumerating all critical failures. *IEEE Transactions on Smart Grid*, 6(2):843–852, 2015.

Takeru Inoue, Hiroaki Iwashita, Jun Kawahara, and Shin-ichi Minato. Graphillion: software library for very large sets of labeled graphs. *International Journal on Software Tools for Technology Transfer*, 18(1):57–66, 2016.

Takeru Inoue. Reliability analysis for disjoint paths. *IEEE Transactions on Reliability*, 68(3):985–998, 2019.

Hiroaki Iwashita and Shin-ichi Minato. Efficient top-down ZDD construction techniques using recursive specifications. Technical Report TCS-TR-A-13-69, Division of Computer Science, Hokkaido University, 2013.

Hiroaki Iwashita, Yoshio Nakazawa, Jun Kawahara, Takeaki Uno, and Shin-ichi Minato. Efficient computation of the number of paths in a grid graph with minimal perfect hash functions. Technical Report TCS-TR-A-13-64, Division of Computer Science, Hokkaido University, 2013.

Martin Jaggi. Revisiting Frank–Wolfe: Projection-free sparse convex optimization. In *Proc. of the 30th International Conference on Machine Learning* (*ICML 2013*), volume 28, pages 427–435. PMLR, 2013.

Olaf Jahn, Rolf H. Möhring, Andreas S. Schulz, and Nicolas E. Stier-Moses. System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations Research*, 53(4):600–616, 2005.

Jun Kawahara, Takashi Horiyama, Keisuke Hotta, and Shin-ichi Minato. Generating all patterns of graph partitions within a disparity bound. In *Proc. of the 11th International Conference and Workshops on Algorithms and Computation* (*WALCOM 2017*), pages 119–131, 2017.

Jun Kawahara, Takeru Inoue, Hiroaki Iwashita, and Shin-ichi Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E100-A(9):1773–1784, 2017.

Jun Kawahara, Koki Sonoda, Takeru Inoue, and Shoji Kasahara. Efficient construction of binary decision diagrams for network reliability with imperfect vertices. *Reliability Engnierring & System Safety*, 188:142–154, 2019.

Thomas Kerdreux, Fabian Pedregosa, and Alexandre d'Aspremont. Frank–Wolfe with subsampling oracle. In *Proc. of the 35th International Conference on Machine Learning* (*ICML 2018*), volume 80, pages 2591–2600. PMLR, 2018.

Arijit Khan, Francesco Bonchi, Aristides Gionis, and Francesco Gullo. Fast reliability search in uncertain graphs. In *Proc. of the 17th International Conference on Extending Database Technology* (*EDBT 2014*), pages 535–546, 2014.

Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications*, 29:1765–1775, 2011.

Donald E. Knuth. *The art of computer programming: Vol. 4A. Combinatorial Algorithms, Part 1.* Addison-Wesley Professional, 2011.

Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. Centrality indices. In *Network analysis*, pages 16–61. Springer, 2005.

Rahul G. Krishnan, Simon Lacoste-Julien, and David Sontag. Barrier Frank–Wolfe for marginal inference. In *Proc. of Advances in Neural Information Processing Systems 28* (*NIPS 2015*), pages 532–540. Curran Associates, Inc., 2015.

Sy-Yen Kuo, Fu-Min Yeh, and Hung-Yau Lin. Efficient and exact reliability evaluation for networks with imperfect vertices. *IEEE Transactions on Reliability*, 56(2):288–300, 2007.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of Frank–Wolfe optimization variants. In *Proc. of Advances in Neural Information Processing Systems 28* (*NIPS 2015*), pages 496–504. Curran Associates, Inc., 2015.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate Frank–Wolfe optimization for structural SVMs. In *Proc. of the 30th International Conference on Machine Learning* (*ICML 2013*), volume 28, pages 53–61. PMLR, 2013.

Ruiying Li, Ning Huang, and Rui Kang. A new parameter and its algorithm for network connection reliability: k/N-terminal reliability. In *Proc. of the 1st International Conference on Future Information Networks* (*ICFIN 2009*), pages 259–262, Oct 2009.

Takanori Maehara, Hirofumi Suzuki, and Masakazu Ishihata. Exact computation of influence spread by binary decision diagrams. In *Proc. of the 26th International World Wide Web Conference* (*WWW 2017*), pages 947–956, 2017.

Damien Magoni. Tearing down the internet. *IEEE Journal on Selected Areas in Communications*, 21(6):949–960, 2003.

Marc Manzano, Eusebi Calle, and David Harle. Quantitative and qualitative network robustness analysis under different multiple failure scenarios. In *Proc. of 3rd International Congress on Ultra Modern Telecommunications and Control Systems and Workshops* (*ICUMT 2011*), pages 1–7, 2011.

Tatsuya Matsukawa and Hiroyuki Funakoshi. Analyzing failure frequency and severity in communication networks. In *Proc. of the Annual Reliability and Maintainability Symposium* (*RAMS 2010*), pages 1–6, 2010.

Denis A. Migov. New index for wireless sensor network reliability analysis. In *Proc. of the 2019 IEEE 2nd International Conference on Automation,* *Electronics and Electrical Engineering* (*AUTEEE 2019*), pages 334–337, 2019.

Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. of the 30th ACM/IEEE Design Automation Conference* (*DAC 1993*), pages 272–277, 1993.

Ministry of Internal Affairs and Communications. Reporting System for Telecommunication Outages. `https://www.soumu.go.jp/main_content/000665005.pdf#page=48`. in Japanese.

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.

Carlos Natalino, Aysegul Yayimli, Lena Wosinska, and Marija Furdek. Content accessibility in optical cloud networks under targeted link cuts. In *Proc. of the 2017 International Conference on Optical Network Design and Modeling* (*ONDM 2017*), pages 1–6, 2017.

Sebastian Neumayer and Eytan Modiano. Network reliability with geographically correlated failures. In *Proc. of the 29th Conference on Computer Communications* (*INFOCOM 2010*), pages 1–9, 2010.

Sebastian Neumayer and Eytan Modiano. Network reliability under random circular cuts. In *Proc. of the 2011 IEEE Global Telecommunications Conference* (*GLOBECOM 2011*), pages 1–6, 2011.

Sebastian Neumayer and Eytan Modiano. Network reliability under geographically correlated line and disk failure models. *Computer Networks*, 94:14–28, 2016.

Masaaki Nishino, Takeru Inoue, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Optimizing network reliability via best-first search over decision diagrams. In *Proc. of the 2018 IEEE Conference on Computer Communications* (*INFOCOM 2018*), pages 1817–1825, 2018.

Satoshi Nojo and Hitoshi Watanabe. Reliability specification for communication networks based on the failure-influence. In *Proc. of the IEEE Global Communications Conference* (*GLOBECOM 1987*), pages 1135–1139, 1987.

Satoshi Nojo and Hitoshi Watanabe. Incorporating reliability specifications in the design of telecommunication networks. *IEEE Communications Magazine*, 31(6):40–43, 1993.

NTT Docomo. 5G evolution and 6G. *Whitepaper*, 2022.

Jorik Oostenbrink and Fernando Kuipers. Computing the impact of disasters on networks. *SIGMETRICS Performance Evaluation Review*, 45(2):107–110, oct 2017.

Jorik Oostenbrink and Fernando Kuipers. Going the extra mile with disaster-aware network augmentation. In *Proc. of the 2021 IEEE Conference on Computer Communications* (*INFOCOM 2021*), pages 1–10, 2021.

Mahshid Rahnamay-Naeini, Jorge E. Pezoa, Ghady Azar, Nasir Ghani, and Majeed M. Hayat. Modeling stochastic correlated failures and their effects on network reliability. In *Proc. of the 20th International Conference on Computer Communications and Networks* (*ICCCN 2011*), pages 1–6, 2011.

R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

Neil Robertson and P. D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

Franco Robledo, Pablo Romero, Pablo Sartor, Luis Stabile, and Omar Viera. A survivable and reliable network topological design model. In *Reliability and Maintenance*, chapter 6. IntechOpen, 2020.

Arnie Rosenthal. Computing the reliability of complex networks. *SIAM Journal on Applied Mathematics*, 32(2):384–393, 1977.

Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

Tim Roughgarden. *Selfish Routing and the Price of Anarchy.* The MIT Press, 2005.

Jagruti Sahoo, Mohammad A Salahuddin, Roch Glitho, Halima Elbiaze, and Wessam Ajib. A survey on replica server placement algorithms for content delivery networks. *IEEE Communications Surveys & Tutorials*, 19(2):1002–1026, 2016.

Shinsaku Sakaue and Kengo Nakamura. Differentiable equilibrium computation with decision diagrams for Stackelberg models of combinatorial congestion games. In *Proc. of the 35th Conference on Neural Information Processing Systems* (*NeurIPS 2021*), pages 9416–9428, 2021.

William H. Sandholm. Potential games with continuous player sets. *Journal of Economic Theory*, 97(1):81–108, 2001.

Juan Segovia, Eusebi Calle, Pere Vila, Jose Marzo, and Janos Tapolcai. Topology-focused availability analysis of basic protection schemes in optical transport networks. *Journal of Optical Networking*, 7(4):351–364, Apr 2008.

Kyoko Sekine, Hiroshi Imai, and Seiichiro Tani. Computing the Tutte polynomial of a graph of moderate size. In *Proc. of the 6th International Symposium on Algorithms and Computation* (*ISAAC 1995*), pages 224–233, 1995.

Srinivas Shakkottai, Eitan Altman, and Anurag Kumar. Multihoming of users to access points in WLANs: A population game perspective. *IEEE Journal on Selected Areas in Communications*, 25(6):1207–1215, 2007.

Akiyoshi Shioura, Akihisa Tamura, and Takeaki Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM Journal on Computing*, 26(3):678–692, 1997.

Akhilesh Shrestha, Liudong Xing, Yan Sun, and Vinod M. Vokkarane. Infrastructure communication reliability of wireless sensor networks considering common-cause failures. *Intenational Journal of Performability Engineering*, 8(2):141–150, 2012.

J. Cole Smith and Yongjia Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020.

Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking*, 12(1):2–16, 2004.

Hirofumi Suzuki, Masakazu Ishihata, and Shin-ichi Minato. Exact computation of strongly connected reliability by binary decision diagrams. In *Proc. of the 12th Annual International Conference on Combinatorial Optimization and Applications* (*COCOA 2018*), pages 281–295, 2018.

Masahiro Taka and Takeo Abe. Network reliability design techniques to improve customer satisfaction. *IEEE Communications Magazine*, 32(10):64–68, 1994.

János Tapolcai, Zsombor L. Hajdú, Alija Pašić, Pin-Han Ho, and Lajos Rónyai. On network topology augmentation for global connectivity under regional failures. In *Proc. of the 2021 IEEE Conference on Computer Communications* (*INFOCOM 2021*), pages 1–10, 2021.

Jerome Thai. *On learning Game-Theoretical models with Application to Urban Mobility.* PhD thesis, UC Berkeley, 2017. ProQuest ID: Thai_berkeley_0028E_17598. Merritt ID: ark:/13030/m59s6nbq. Retrieved from https://escholarship.org/uc/item/3b61v84v.

Eric S. Tollar and Jay M. Bennett. Network outage impact measures for telecommunications. In *Proc. of the 1st IEEE Symposium on Computers and Communications* (*ISCC 1995*), pages 120–126, 1995.

Shuji Tsukiyama, Isao Shirakawa, Hiroshi Ozaki, and Hiromu Ariyoshi. An algorithm to enumerate all cutsets of a graph in linear time per cutset. *Journal of the ACM*, 27(4):619–632, 1980.

William T. Tutte. *Graph Theory.* Cambridge Mathematical Library. Cambridge University Press, 2001.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

Dirck V. Vliet. The Frank-Wolfe algorithm for equilibrium traffic assignment viewed as a variational inequality. *Transportation Research Part B: Methodological*, 21(1):87–89, 1987.

Hitoshi Watanabe, Minetoshi Kudo, Kazuhiro Kawanishi, and Koji Yamasaki. A reliability design method for private networks. In *Proc. of the Annual Reliability and Maintainability Symposium* (*RAMS 2003*), pages 237–243, 2003.

O. Wing and P. Demetriou. Analysis of probabilistic networks. *IEEE Transactions on Communication Technology*, 12:38–40, 1964.

R. Kevin Wood. Factoring algorithms for computing K-network network reliability. *IEEE Transactions on Reliability*, 35(3):269–278, 1986.

Yufeng Xiao, Xin Li, Yuhong Li, and Shanzhi Chen. Evaluate reliability of wireless sensor networks with OBDD. In *Proc. of the 2009 IEEE International Conference on Communications* (*ICC 2009*), pages 1–5, 2009.

Liudong Xing. An efficient binary-decision-diagram-based approach for network reliability and sensitivity analysis. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 38(1):105–115, 2008.

Hiroki Yano, Sumihiro Yoneyama, and Hiroyoshi Miwa. Reliable network design problem by improving node reliability. In *Proc. of the 10th International Conference on Emerging Internet, Data & Web Technologies* (*EIDWT 2022*), pages 42–51, 2022.

Zesen Zhang, Alexander Marder, Ricky Mok, Bradley Huffaker, Matthew Luckie, K C Claffy, and Aaron Schulman. Inferring regional access network topologies: Methods and applications. In *Proc. of the 21st ACM Internet Measurement Conference* (*IMC 2021*), pages 720–738, 2021.