



Apprentissage d'heuristiques pour la résolution de problèmes d'optimisation combinatoire dans les réseaux

Chen Dang

► To cite this version:

Chen Dang. Apprentissage d'heuristiques pour la résolution de problèmes d'optimisation combinatoire dans les réseaux. Autre [cs.OH]. Université Paris sciences et lettres, 2023. Français. <NNT : 2023UPSLD044>. <tel-04457253>

HAL Id: tel-04457253

<https://theses.hal.science/tel-04457253v1>

Submitted on 14 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization



THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris-Dauphine

**Heuristics Learning for Solving
Network Optimization Problems**

Soutenue par
Chen DANG
Le 08/12/2023

École doctorale n°543
SDOSE

Spécialité
Informatique



Composition du jury :

Cristina BAZGAN Professeur LMSADE - Université Paris Dauphine-PSL	<i>Directrice de thèse</i>
Hind CASTEL-TALEB Professeur Télécom SudParis	<i>Rapporteuse</i>
Tristan CAZENAVE Professeur LMSADE - Université Paris Dauphine-PSL	<i>Co-encadrant de thèse</i>
Morgan CHOPIN Chercheur Orange Innovation, Châtillon	<i>Co-directeur de thèse</i>
Bernard FORTZ Professeur Université de Liège	<i>Président</i>
Frédéric GIROIRE Directeur de recherche au CNRS I3S, Université Nice Sophia Antipolis	<i>Examineur</i>
Frédéric SAUBION Professeur LERIA - Université d'Angers	<i>Rapporteur</i>
Pierre-Henri WUILLEMIN Maître de conférences LIP6 - Sorbonne Université	<i>Co-encadrant de thèse</i>

Acknowledgements

Time has swiftly passed, bringing to an end my three-year PhD journey. I began my PhD during the lockdown caused by the COVID-19 pandemic, a period that was far from ideal. These years presented their fair share of challenges, but they also marked my most significant personal and professional growth. I wish to express my deep appreciation to everyone who has accompanied and supported me on this path.

Firstly, A warm thank you to Orange. I'm grateful for the funding of this project and even more for providing a research environment that was both free and challenging. In such an environment, I was not only able to dive deep into my topic but also had the chance to explore many different research directions. I'm also grateful for the inclusive and joyful team I was part of. We shared countless happy moments. A special shout-out to my team members Paolo Zappalà, John Jairo Quiroga Orozco, Alexis Schneider, Nirussan Sivanand, Samy Mokhtari, and many others. I believe that our ping-pong matches and the odd topics of conversation during our cafe breaks will remain some of my most treasured memories, no matter how many years pass. I also want to acknowledge Eric Gourdin, our manager, for his support and for leading such an incredible team.

My gratitude extends to my supervisors. I'm fortunate to have such responsible and understanding supervisors. Our weekly meetings kept me motivated and also gave me ample freedom to explore diverse areas. I would like to thank Cristina Bazgan for her invaluable guidance both on my research and in administrative and personal spheres. Morgan Chopin's technical expertise and support at the company were indispensable, and I have learned immeasurably from him. Furthermore, I appreciate Tristan Cazenave and Pierre-Henri Willemin for their insightful feedback, often pointing out aspects I might have overlooked. Without them, my PhD project could not have been successfully completed. I am deeply grateful to all my supervisors.

Next, I want to thank my friends. I may not always be the most outgoing, but I've been blessed to have so many great friends with me throughout these three years. A special thanks to Wang Ze and his occasional display of culinary prowess, to Yang Yang, Jingang Qu, San San for their cheerful company, and Jinfeng Zhong for our shared adventures outdoors. As I approached my graduation, Huairan Li, Aitong Li, and Liqiao Huang's companionship was especially heartening. A shout-out to those amazing friends with whom I've shared wonderful times even outside of Paris: Yang Lei, Quanshangze Du, Zixin Zhou and many others. I regret that I can't name all of you, but I love each one of you.

My deepest affection and respect go out to my parents. They taught me to be humble, honest, ambitious, and empathetic. They granted me the freedom to make significant life decisions, shaping me into the individual I am today. Every achievement of mine stands on the foundation

they laid for me. I also wish to thank my deepest love, Jia Xing, for her unwavering love and support. She understands me better than anyone and has been my strongest support, especially during tough times, always encouraging me to be the person I aspire to be. Lastly, I want to thank my two cats, Nana and Nico. They are, undoubtedly, the world's most affectionate and well-natured cats, though I do wish they'd skip the keyboard naps during my toughest coding sessions.

Résumé

Cette thèse s'intéresse aux problèmes d'optimisation issus du domaine de l'ingénierie de trafic dans les réseaux de télécommunications. En particulier, nous avons étudié le problème de minimisation de la congestion dans les réseaux IP reposant sur les protocoles de routage au plus court chemin (OSPF, IS-IS) d'une part et le segment routing d'autre part.

La principale contribution de la thèse est l'application de la méthode de recherche Monte Carlo Nested Rollout Policy Adaptation (NRPA) au problème d'optimisation des métriques (WEIGHT SETTING). L'approche proposée s'adapte bien à la taille de l'instance et peut être facilement étendue pour prendre en compte des contraintes ou des critères d'optimisation supplémentaires (délai bout en bout, unicité des chemins de routage, etc...).

En outre, cette thèse présente une version améliorée de NRPA, appelée Meta-NRPA, qui utilise la théorie de l'arrêt optimal pour mieux guider la recherche de NRPA. De plus, nous présentons plusieurs techniques exploratoires pour NRPA permettant une exploration plus efficace de l'espace des solutions. Nos expérimentations confirment l'amélioration significative des performances de NRPA.

Par ailleurs, cette thèse introduit l'algorithme Hierarchical Shortest Path Routing (HSPR), une version adaptée de l'algorithme Customizable Contraction Hierarchy (CCH). HSPR permet d'accélérer significativement la simulation des protocoles de routage au plus court chemin comparé aux méthodes traditionnelles basées sur l'algorithme de Dijkstra. Nous l'avons combiné avec l'algorithme Meta-NRPA pour résoudre le problème WEIGHT SETTING. Les résultats obtenus montrent une amélioration substantielle des performances de Meta-NRPA, surtout dans les réseaux de grande taille.

Enfin, nous étudions également plusieurs heuristiques pour résoudre le problème d'optimisation WAYPOINT OPTIMIZATION, lié au Segment Routing. Nous combinons une approche basée sur la recherche locale et la décomposition arborescente du réseau afin d'explorer plus efficacement l'espace des solutions. Nous proposons également d'autres approches basées sur l'apprentissage par renforcement et évaluons leur efficacité à travers une série d'expériences approfondies sur des données réelles et synthétiques.

Mots clés : Optimisation des Réseaux, Methodes de Monte Carlo, Ingénierie de Trafic, Segment Routing, Apprentissage par Renforcement

Abstract

This thesis examines optimization problems that arise from telecommunications traffic engineering. In particular, we have studied the problem of congestion minimization in IP networks based on shortest path routing protocols (OSPF, IS-IS) on the one hand, and segment routing on the other.

The main contribution of this thesis is the application of the Monte Carlo Nested Rollout Policy Adaptation (NRPA) method for solving the metric optimization problem `WEIGHT SETTING`. The proposed approach adapts well to the size of the instance and can be readily expanded to consider additional constraints or optimization criteria (end-to-end delay, uniqueness of routing paths, etc...).

Furthermore, this thesis introduces an enhanced version of NRPA, named Meta-NRPA, which leverages the optimal stopping theory to better guide the search of NRPA. Additionally, we present several exploratory techniques for NRPA, enabling a more efficient exploration of the search space. Our experiments confirm a significant improvement in NRPA's performance.

Subsequently, this thesis presents the Hierarchical Shortest Path Routing (HSPR) algorithm, an adapted version of the Customizable Contraction Hierarchy (CCH) algorithm. HSPR allows to significantly accelerate the simulation of shortest path routing protocols compared to traditional methods based on the Dijkstra algorithm. We combined it with the Meta-NRPA algorithm to address the `WEIGHT SETTING` problem. The results demonstrate a substantial enhancement in Meta-NRPA's performance, especially in large networks.

Lastly, we also investigate several heuristics for solving the optimization problem `WAYPOINT OPTIMIZATION` related to Segment Routing. We combine a local search based approach with tree decomposition to explore more effectively the solution space. We also propose other methods based on reinforcement learning and assess their effectiveness through a series of in-depth experiments on real and synthetic data.

Keywords : Network Optimization, Monte Carlo Methods, Traffic Engineering, Segment Routing, Reinforcement Learning

Contents

Acknowledgements	i
Résumé	iii
Abstract	iv
Contents	iv
1 General Introduction	1
1 Background and Problem Statement	2
2 Motivation and Relevance	3
3 Objective and Methodology	4
4 Contributions	5
2 Preliminaries	7
1 Graph Theory	8
2 Traffic Engineering	10
3 Monte Carlo Based Approaches	12
4 Reinforcement Learning	19
5 Dataset	22
3 Weight Setting Problem with Monte Carlo Approaches	31
1 Introduction	32
2 Weight Setting Problem	32
3 Modeling Based on Monte Carlo Algorithm	34
4 Experiments and Results	37
5 Conclusion	45
4 Hierarchical Shortest Path Routing	47
1 Introduction	48
2 Customizable Contraction Hierarchy	49
3 Extending CCH for Shortest Path Routing Computation	54
4 Experiments and Results	57
5 Application to Weight Setting Problem	60
6 Conclusion	63
5 Warm Starting NRPA	65
1 Introduction	66
2 The Secretary Problem	66

3	Warm Starting NRPA - Meta-NRPA	68
4	Experiments and Results	70
5	Conclusion	75
6	Waypoint Optimization Problem	77
1	Introduction	78
2	Waypoint Optimization Problem	78
3	Our Contributions	82
4	Experiments and Results	90
5	Conclusion	102
7	Conclusion and perspectives	103
1	Conclusion	103
2	Perspectives	104
	Résumé Français	106
1	Introduction	106
2	Problème de Métriques	116
3	Problème d’Optimisation des Waypoints	118
4	Nos Contributions	122
	List of Figures	124
	List of Tables	126
	Bibliography	127

Chapter 1

General Introduction

Objectifs

This chapter outlines the background, the motivation and the primary objective of developing heuristics-based approaches to solve network optimization problems. Additionally, the chapter summarizes the contributions and presents the structure of the thesis.

Contents

1	Background and Problem Statement	2
2	Motivation and Relevance	3
3	Objective and Methodology	4
4	Contributions	5

1 Background and Problem Statement

In recent years, the telecommunications industry has experienced a significant paradigm shift due to the emergence of next-generation network technologies. These revolutionary advancements offer increased data transfer speeds, reduced latency, and enhanced capacity, fundamentally altering the operation and management of telecommunication networks.

One major development in this field is the incorporation of software-defined networking (SDN) and network virtualization. This innovative approach decouples the physical hardware from the software layer that controls it, enabling the coexistence of multiple virtual networks on the same physical infrastructure. This not only allows for greater flexibility and scalability but also facilitates more efficient resource utilization, streamlined network deployment and maintenance, and improved fault tolerance. Additionally, optimization algorithms can be deployed and executed directly within network controllers (SDN controller), a marked departure from traditional methods.

These novel paradigms have led to a more centralized network architecture, which has given rise to new challenges regarding traffic routing, load balancing, and energy efficiency. These challenges are particularly pronounced in the domain of network virtualization and optimization, as maintaining quality of service (QoS) becomes increasingly critical for facing the growing traffic volume driven by high-definition streaming, cloud gaming, and other bandwidth-intensive applications.

Another important development is the Segment Routing, which has emerged as a key solution for simplifying network operations and enhancing scalability within the telecommunication industry. It is a source-based, path computation and traffic engineering mechanism that operates within the network layer, leveraging the existing routing protocols (e.g., OSPF, IS-IS, etc.) to establish efficient and flexible forwarding paths.

Segment Routing can be particularly beneficial in large-scale telecommunication networks, where it can optimize traffic flows, enhance network performance, and reduce operational complexity. By enabling more fine-grained control over the routing and forwarding of packets, segment routing empowers network operators to optimize network resources, minimize congestion, and ensure better QoS for end users.

To ensure that QoS is maintained in modern telecommunication networks, it is imperative to develop efficient algorithms capable of scaling with the network size and adapting to the unique demands of virtualization. A promising direction for designing optimization algorithms for virtualized networks involves the integration of Machine Learning (ML) and Combinatorial Optimization (CO). This combination offers the potential to create algorithms that can adapt to the dynamic nature of virtualized networks by capitalizing on modern network capabilities, giving rise to an emerging research area with significant opportunities for exploration and innovation.

In this thesis, we perform an in-depth exploration of heuristic algorithms for addressing some of the most notoriously hard traffic engineering problems, specifically the WEIGHT SETTING problem and the WAYPOINT OPTIMIZATION problem, which arises in the context of Segment

Routing. Heuristic algorithms are particularly well-suited for delivering high-quality results within a limited time frame for large telecommunication networks, making them ideal for tackling the challenges presented by emerging network technologies. Our primary focus is on mitigating telecommunication network congestion while maintaining requisite QoS parameters, such as delay, in an increasingly virtualized and dynamic environment.

By examining the potential of heuristic algorithms within the domain of telecommunication network optimization, this thesis aspires to contribute to the ongoing development of advanced solutions for managing and optimizing complex future networks. Through extensive research and experimentation on both synthetic and real-world networks, we will investigate the advantages and limitations of various heuristic methods, assess their adaptability to the rapidly evolving telecommunication networks. Additionally, we will exhibit key properties for optimizing large-scale networks, providing more insights into the optimization process for such problems and offering valuable knowledge for network operators and researchers alike.

Moreover, some of the methods presented in this thesis can be applied to a broad spectrum of problems, encompassing not only difficult combinatorial optimization problems like network traffic simulation but also more general problems such as the Traveling Salesman Problem, the snake-in-the-box problem, and others.

In summary, this thesis aims to uncover the potential of heuristic methods for telecommunication network optimization, aiding in addressing the challenges and capitalizing on the opportunities ushered in by next-generation network technologies. By investigating innovative heuristic techniques, we strive to advance the state-of-the-art in network optimization, ultimately contributing to the improvement of telecommunication networks' performance, efficiency, and reliability for the advantage of all users.

It is important to emphasize that this thesis exclusively discusses optimization at the algorithmic level and does not go into the details of network structure and protocol design and implementation. Consequently, in our work, network structure, protocols, and related aspects have been abstracted into abstract models and algorithms for algorithmic optimization purposes.

2 Motivation and Relevance

In this thesis, we address the telecommunication network WEIGHT SETTING problem and WAYPOINT OPTIMIZATION problem, which pose significant challenges in the field of telecommunication network optimization and hold wide-ranging implications for both the research community and various industries. Efficient management and optimization of telecommunication networks are crucial for sustaining high-quality services, particularly in an era characterized by rapidly escalating data traffic and evolving user demands. Tackling these problems through innovative algorithms and methodologies has the potential to substantially enhance network routing efficiency, optimize resource allocation, and refine decision-making algorithms for telecommunication networks.

Researchers have encountered numerous obstacles in solving the WEIGHT SETTING problem and the WAYPOINT OPTIMIZATION problem, including computational complexity, the demand

for scalable and adaptive solutions, among others. This thesis aims to confront these challenges by devising and evaluating heuristic algorithms and methodologies specifically tailored to the characteristics of these problems. By leveraging on well-known techniques such as Monte Carlo-based approaches, local search, and by proposing novel algorithms like Hierarchical Shortest Path Routing and warm starting Nested Rollout Policy Adaptation (Meta-NRPA), our research aspires to provide efficient methods for solving these two traffic engineering problems for large-scale networks. In addition to the contribution of the development of efficient optimization algorithms for solving the WEIGHT SETTING problem and WAYPOINT OPTIMIZATION problem, this thesis also explores their practical applications in real-world settings. Furthermore, these algorithms can serve as important building blocks to solve a broader class of optimization problems, extending beyond the telecommunication network traffic engineering field.

The practical applications of our research are manifold. By improving network routing efficiency, our algorithms can assist telecommunication providers in delivering superior quality of service to their customers, minimizing congestion, and managing their resources more effectively. Additionally, the methodologies we proposed may be adapted to other problem domains, such as transportation and logistics, wherein efficient routing and resource allocation are essential for minimizing operational costs, elevating service quality, and reducing the environmental impact of these endeavors.

3 Objective and Methodology

The primary objective of this thesis is to develop heuristic algorithms for addressing specific telecommunication traffic engineering problems, with the aim of ensuring the performance of large-scale telecommunication networks and enabling the scalability of solutions. To concentrate solely on their algorithmic aspects, we have simplified and modeled the traffic engineering problems as graph problems. The traffic engineering problems under consideration are the WEIGHT SETTING problem and the WAYPOINT OPTIMIZATION problem. Our foremost interest lies in optimizing network congestion, which, in our context, represents the ratio of traffic traversing the edge of the network to the capacity of that edge. By adeptly directing traffic within the network using optimization algorithms, we can minimize congestion and thereby deliver better QoS.

The algorithms proposed in this thesis are predominantly heuristic-based, encompassing Monte Carlo-based algorithms, local search techniques, and certain reinforcement learning algorithms. These algorithms are specifically designed to address the studied traffic engineering problem at scale and are tested on distinct datasets, including both real telecommunication networks and synthetic datasets. To demonstrate the efficacy of the proposed algorithms, empirical results on instances of various sizes are presented, substantiating their performance and highlighting their potential to contribute significantly to the telecommunication network optimization domain.

Furthermore, to strengthen the reliability of our findings, we perform comparative analyses between the proposed heuristic algorithms and existing optimization techniques, demonstrating

the advantages of our methods in terms of computational efficiency, adaptability, and scalability.

4 Contributions

In this thesis, we present several heuristics and empirical insights to improve the performance of the algorithm for solving large-scale traffic engineering problems.

The second chapter of this thesis focuses on the preliminaries required for comprehending the problems at hand and the primary approaches utilized in our work, specifically Monte Carlo-based approaches. This chapter introduces the fundamentals of graph theory, traffic engineering, and Monte Carlo algorithms.

The third chapter concentrates on employing Monte Carlo approaches for addressing the WEIGHT SETTING problem. We provide a formal introduction to the WEIGHT SETTING problem in this chapter, along with a detailed explanation of how this problem is modeled using Monte Carlo approaches. We assess three different approaches and propose general techniques to increase the performance of these approaches for large instances.

In the fourth chapter, we address the bottleneck of the simulation for the traffic engineering problem, which is the calculation of the shortest path and congestion for large telecommunication networks with constantly changing metrics. We propose the algorithm Hierarchical Shortest Path Routing (HSPR), which is based on Customizable Contraction Hierarchy (CCH) [34], which shows significant improvement compared to traditional approaches. While applied to the approaches proposed in chapter 3, the new algorithm yields better results. This approach can also be employed in other simulation-based optimization algorithms for the WEIGHT SETTING problem, as well as in other simulations necessitating frequent weight changes in the network.

In the fifth chapter, we present an improvement on the best-performing Monte Carlo approach that we tested in the third chapter: Nested Rollout Policy Adaptation (NRPA) [85]. This chapter deals with a critical issue that is frequently encountered in optimization problems: escaping the local minima. We propose the use of optimal stopping theory to warm start NRPA, estimating the performance of NRPA execution at an early stage and discarding under-performing solutions. The resulting algorithm, named Meta-NRPA, exhibits exceptional performance on three combinatorial optimization problems, including the WEIGHT SETTING problem.

In the sixth chapter, we focus on the WAYPOINT OPTIMIZATION (WPO) problem, which arises in the context of the Segment Routing (SR). We introduce GreedierWPO, a local search algorithm that outperforms existing methods in handling large instances. The chapter also explores techniques for reducing the search space, enhancing the efficiency of the solution process. We also briefly discussed the advantages of joint optimization between the two problems: the WEIGHT SETTING problem and the WAYPOINT OPTIMIZATION problem. Our work opens the possibility for future research in large scale optimization of network segment routing.

Chapter 2

Preliminaries

Objectifs

This chapter serves as an introduction to the topics of graph theory, traffic engineering problems, Monte Carlo based approaches, and reinforcement learning basics. It also offers a preview of the dataset that will be utilized in the subsequent chapters, along with an initial analysis of its topological properties. It provides the necessary background and preliminary knowledge required to understand the subsequent chapters.

Contents

1	Graph Theory	8
2	Traffic Engineering	10
2.1	Network	10
2.2	Routing Protocols	11
3	Monte Carlo Based Approaches	12
3.1	Preliminaries	12
3.2	UCT	14
3.3	NMCS	15
3.4	NRPA	16
4	Reinforcement Learning	19
4.1	Q-learning	19
4.2	SARSA	20
4.3	Eligibility Trace	20
4.4	SARSA(λ)	21
5	Dataset	22
5.1	Topological Properties	25

In this chapter, we provide the notions that are essential to comprehend the problematic and the approaches we mainly used to solve the problems in this thesis. We introduce here the preliminaries for graph theory, traffic engineering, several Monte Carlo based approaches, reinforcement learning basics, and the dataset we used in this thesis.

1 Graph Theory

A *directed graph* or *digraph*, denoted by $G = (V, A)$, consists of a set of nodes V and a set of ordered pairs of nodes, known as *arcs*, $A \subseteq V \times V$. Let $|V|$ be the number of nodes and $|A|$ be the number of arcs. For an arc $uv \in A$, we say that uv is directed from u to v . A *bidirected graph* is a *directed graph* where, for every arc $uv \in A$ in the graph, there is also an arc $vu \in A$. The *in-degree* of a node $v \in V$ is the number of arcs directed towards node v . The *out-degree* of a node $v \in V$ is the number of arcs originating from node v . And the *degree* of a node $v \in V$, denoted by $\deg(v)$, is the sum of its *in-degree* and *out-degree*.

An *undirected graph* or simply *graph*, denoted by $G = (V, E)$, consists of a set of nodes V and a set of unordered pairs of nodes, known as *edges*, $E \subseteq V \times V$. To better distinguish a graph from a digraph we may denote it by G^u throughout this thesis. The edge between nodes u and v is noted as $uv \in E$. Notably, we adopt the same notation for both arcs and edges, allowing the context to clarify the intended reference. Since the edges in G^u are undirected, an edge between nodes u and v can be referred to interchangeably as uv or vu . The *neighborhood* of a node $v \in V$, denoted by $N_G(v)$, is the subset of nodes $\{u \in V : uv \in E\}$. We may simply write $N(v)$ if the graph is clear from the context. The *degree* of a node $v \in V$, denoted by $\deg(v)$, is the number of edges incident to v , i.e. $\deg(v) = |N(v)|$.

Occasionally, we may have to consider the underlying undirected graph G^u of a directed graph G . The underlying graph G^u has the same set of vertices as G , and there is an edge uv in G^u if there exists an arc uv or vu in G .

A *path* in a digraph (resp. graph) is a sequence of nodes $(v_1, v_2, \dots, v_n)$ in which each consecutive pair (v_i, v_{i+1}) is connected by an arc (resp. edge) $v_i v_{i+1}$, and each node v_i for all $1 \leq i \leq n$ is distinct. A *cycle* is a path in which the first and last nodes are also connected by an arc (resp. edge).

Weighted Graph A *weighted digraph* (resp. *graph*) is a digraph $G = (V, A)$ (resp. graph $G^u = (V, E)$) with a weight function $w : A \rightarrow \mathbb{R}_+$ (resp. $w : E \rightarrow \mathbb{R}_+$), which assigns a real value to each arc (resp. edge) in G . In the context of telecommunication networks, weights often correspond to latency, bandwidth, or other network-specific attributes.

Shortest Path In a weighted digraph (resp. graph), the *shortest path* between two nodes u and v is the path with the minimum sum of arc (resp. edge) weights among all possible paths connecting u and v . The length of the shortest path is denoted by $\text{dist}(u, v)$. Notice that multiple shortest paths can exist between nodes u and v .

Clique A *clique* in a graph $G^u = (V, E)$ is a set of nodes $C \subseteq V$ such that for every pair of nodes $u, v \in C$, $uv \in E$. A *maximum clique* is a clique with the largest possible number of nodes.

Node Order A node order π in a graph $G^u = (V, E)$ is a sequence of nodes $v_1, v_2, \dots, v_n$, where $n = |V|$, and its inverse π^{-1} assigns a unique integer value, called *rank* $\pi^{-1}(v) \in \{1, 2, \dots, n\}$ to each node $v \in V$.

Perfect Elimination Order A node order $\pi = v_1, v_2, \dots, v_n$ of a graph G^u is a *perfect elimination order* if for every $i = 1, \dots, n$, v_i and the nodes $v_j \in N(v_i)$ with $j > i$ form a clique.

Chordal Graph A *chordal graph* is a graph in which every cycle of at least four nodes has a chord, which is an edge that is not part of the cycle but connects two nodes of the cycle.

A graph is chordal if and only if it has a *perfect elimination order*. The *perfect elimination order* is not unique in a chordal graph [28].

Chordal Super Graph Given a node order $\pi = v_1, v_2, \dots, v_n$ of a graph G^u , the *chordal super graph*, denoted by G_π^* , is the graph obtained by eliminating all the nodes of G^u in sequence according to π as follows. Each time a node v_i is eliminated, edges are inserted between each pair of its remaining neighbors, then all its incident edges to v_i are removed. By F_{v_i} we denote the set of edges inserted with the elimination of v_i . The original graph G^u with all additional edges inserted during the removal of all nodes $F = \bigcup_i F_{v_i}$ is the *chordal super graph* G_π^* .

Elimination Tree The *elimination tree* of a graph $G^u = (V, E)$ with respect to an elimination ordering $\pi = v_1, v_2, \dots, v_n$ is a directed tree, denoted by $T = (V, E_T)$, derived from the graph G^u . The nodes in T are the same as in G^u , and the arcs in T are determined as follows. For every node v_i in V , we define its parent node in T as the first node v_j appearing after v_i in the elimination order π such that the edge $v_i v_j$ exists in the graph $G_{v_i}^*$ (which is the graph after the elimination of nodes up to v_i according to π). If such a node v_j exists, then we add an arc from v_i to v_j in T . The node with no parent is the root of the elimination tree. Figure 2.1 shows an example of a graph and its corresponding elimination tree.

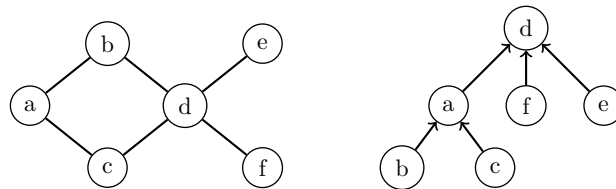


Figure 2.1: Example of elimination tree (right) obtained from a graph (left) with respect to the following elimination order : b, c, f, a, e, d .

Tree Depth The *tree depth* of a graph G^u is the minimum depth of the *elimination trees* over all elimination orders.

Tree Decomposition A *tree decomposition* of a graph $G^u = (V, E)$ is a pair (T, X) where $T = (I, F)$ is a tree and $X = \{X_i | i \in I\}$ is a family of subsets of V (also known as bags), such that:

1. The union of all bags equals the node set of G : $\bigcup X_i = V$.
2. For every edge $uv \in E$, there exists a bag X_i such that $u, v \in X_i$.
3. For all $i, j, k \in I$, if j lies on the unique path between i and k in T , then $X_i \cap X_k \subseteq X_j$.

The width of a tree decomposition is the size of its largest bag X_i minus one.

Tree-width The *tree-width* of G^u , denoted by $\text{tw}(G^u)$, is the minimum width over all possible tree decompositions of G^u . A graph with small *tree-width* exhibits a tree-like structure, which often allows for more efficient algorithms for solving various problems. In particular, a tree (or a forest) has a tree-width of 1.

2 Traffic Engineering

2.1 Network

A *network* is represented as a bidirected graph $G = (V, A)$, where V is the set of nodes and A is the set of arcs. Each node in V represents a device or a switch in the network, while each arc in A represents a link between two nodes. For each arc $uv \in A$, the *capacity* $c_{uv} \in \mathbb{R}_+$ refers to the maximum rate at which data can be transmitted through that arc. A *metric* $w \in \mathbb{Z}_+^{|A|}$, also known as administrative weight, is a positive integer assigned to each arc in the network. Metrics play a central role in determining the routing paths for data transmission, as shortest path routing protocols use these values to calculate and select the shortest paths in G with respect to the metrics.

Demand We denote by K the set of demands (also referred to as *traffic matrix*) to be routed in G . A demand $k \in K$ is defined by a triplet (s^k, t^k, d^k) where $s^k \in V$ is the source node of the demand, $t^k \in V$ is the target node, and $d^k \in \mathbb{R}_+$ is the volume of the demand.

Load The *load* refers to the amount of traffic that is being transmitted through a particular arc relative to its maximum capacity. It is an essential parameter for evaluating network performance, as high load can lead to congestion, increased latency, and reduced quality of service. Formally, we define the *load* of an arc $uv \in A$ induced by metric w , denoted by $\text{load}(uv, w)$, as the amount of the traffic flow f_{uv} traversing the arc uv over its capacity c_{uv} : $\text{load}(uv, w) = \frac{f_{uv}}{c_{uv}}$. The *load* is also often referred to as the *link utilization* in other studies.

Network Congestion *Network congestion* refers to the highest *load* among all the arcs in the network with respect to a metric w , which is an important network performance indicator. It can be formally defined as $\text{cong}(w) = \max_{uv \in A} \text{load}(uv, w)$. In some other studies, *network congestion* is also referred to as the *Maximum Link Utilization (MLU)*.

Delay *Delay*, also known as *latency*, is the amount of time it takes for a data packet to travel from its source to its destination through the network. *Delay* is a crucial performance indicator, as high latency can negatively impact user experience and the quality of service, particularly for real-time or delay-sensitive applications.

2.2 Routing Protocols

A *routing protocol* in telecommunication networks is a standardized set of algorithms and procedures employed by network devices, such as routers and switches, to determine the most efficient and reliable paths for data packet transmission. These protocols facilitate dynamic communication among network nodes, enabling the continuous exchange of topological information and adjusting to network changes, such as link failures and congestion.

OSPF Protocol The *Open Shortest Path First (OSPF) protocol* [72] is a widely-used link-state *routing protocol* in telecommunication networks, designed to efficiently and dynamically determine the routing paths for data packet transmission within an Autonomous System (AS). OSPF relies on Dijkstra's shortest path algorithm to calculate the routing paths based on metrics.

IS-IS Protocol The *Intermediate System-to-Intermediate System (IS-IS) protocol* [41] is another widely-used link-state *routing protocol* for data packet transmission within an Autonomous System (AS). Operating on the OSI (Open Systems Interconnection) model's network layer, IS-IS distinguishes itself from OSPF, through its adherence to the OSI addressing scheme and utilization of Network Service Access Points (NSAPs) for addressing. Additionally, IS-IS employs a more extensible structure with Type-Length-Value (TLV) encoding, allowing for greater adaptability and ease of integration with other protocols, as well as the seamless handling of both IPv4 and IPv6 routing. Subsequently, IS-IS also relies on Dijkstra's shortest path algorithm to calculate the routing paths.

ECMP *Equal-Cost Multi-Path (ECMP)* is a routing strategy employed in telecommunication networks to optimize the utilization of available network resources and enhance overall performance by distributing traffic across multiple equal-cost paths between a source and a destination. By leveraging parallel paths with the same cost metric, ECMP enables efficient load balancing. More precisely, the traffic volume that reaches a router is split equally among all links leaving that router and belonging to the shortest paths toward destination. Figure 2.2 shows an example of the ECMP rule.

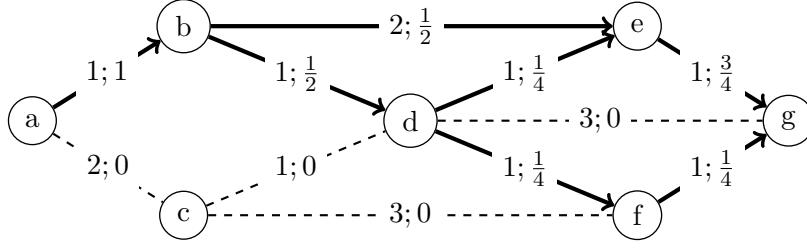


Figure 2.2: Illustration of a shortest path routing with the ECMP rule. In this figure, we assume unit capacities and suppose that a demand k with traffic volume $d^k = 1$ must be routed from $s^k = a$ to $t^k = g$. A label $w_{uv}; \text{load}(uv, w)$ is associated with each arc $uv \in A$.

3 Monte Carlo Based Approaches

The Monte Carlo Search algorithm is a stochastic approach that is widely used for solving difficult, multi-dimensional problems. By iteratively refining solutions through random sampling, the algorithm can often deliver high-quality approximations, even in the presence of uncertainty and noise. One of the Monte Carlo Search algorithm's most successful applications, Monte Carlo Tree Search (MCTS), has significantly advanced the state-of-the-art in fields such as game theory, reinforcement learning, and planning. Although the Monte Carlo Search algorithm has been successfully applied to many difficult problems, it has not yet been widely used for optimizing telecommunication networks.

In this thesis, we focus on discussing three Monte Carlo based algorithms: UCT (Upper Confidence Trees) [64], Nested Monte Carlo Search (NMCS) [19], and Nested Rollout Policy Adaptation (NRPA) [85]. These algorithms have shown remarkable performance in solving difficult optimization problems. Although there are numerous other Monte Carlo based algorithms, we chose these three due to their representativeness and high level of performance. We will present some preliminaries of these algorithms before we introduce them in detail in the following sections.

3.1 Preliminaries

Before presenting MCTS, it is important to first introduce the concept of Markov Decision Process (MDP) [80]. This is because MDP provides the foundational model for sequential decision-making in probabilistic environments, and MCTS builds on this model to optimize decision-making in complex scenarios.

Markov Decision Process is a model for scenarios where an agent, through probabilistic transitions, interacts with an environment to maximize its rewards over sequential decisions. A critical aspect of MDP is the Markov property, which states that the future state depends only on the current state and action, not on the sequence of states and actions that preceded it. This memoryless property simplifies the decision-making process as the agent does not need to consider the entire history.

An MDP is defined by:

- **A state space $\mathcal{S}$** , which is a collection of all possible states.

- **An action space** $\mathcal{A}(s)$, which is a collection of all possible agent actions for every state $s \in \mathcal{S}$.
- **A transition probability function** $\mathcal{P}(s'|s, a)$, which defines the probability of transitioning from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}(s)$.
- **A reward function** $\mathcal{R}(s, a)$, which defines the reward received after taking action $a \in \mathcal{A}(s)$ from state $s \in \mathcal{S}$.

The primary objective of an MDP is to derive an optimal decision policy π , which is a map from states $\mathcal{S}$ to actions $\mathcal{A}(s)$ that yields the highest expected reward. There are two key value functions which help evaluate how good a state or an action is:

- **State value function** $V(s)$, which denotes the empirical average reward when playing any action in state $s \in \mathcal{S}$.
- **State-action value function** $Q(s, a)$, which denotes the empirical average reward of playing action $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$.

An MDP is considered undiscounted if the agent aims to maximize the total reward it receives over the entire decision-making process, with no diminishing value placed on future rewards. In a discounted MDP, future rewards are discounted by a factor $\gamma \in [0, 1)$, which means they are considered less valuable than immediate rewards.

MCTS is under the assumption of an undiscounted MDP, although it can be adapted for discounted scenarios. An advantage of MCTS is its ability to provide reasonably good solutions for difficult problems without necessitating domain-specific knowledge. Using both the Monte Carlo method and tree search techniques, MCTS efficiently explores the vast space of potential actions and states to identify optimal or near-optimal decisions.

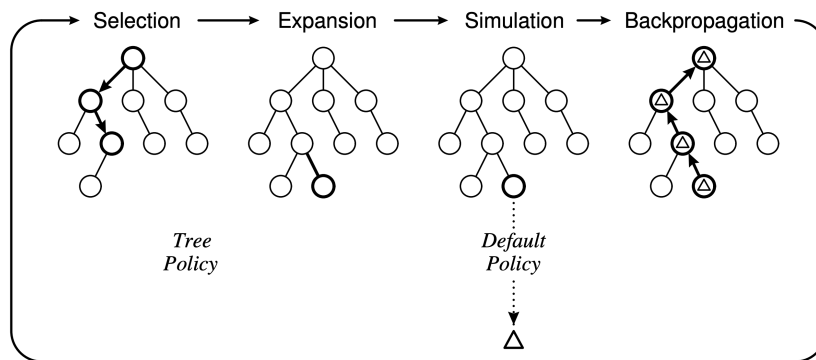


Figure 2.3: Example of an iteration of MCTS, from [16].

MCTS operates by iteratively building a search tree that deepens with each iteration. During each cycle, the algorithm proceeds through four main stages: selection, expansion, simulation, and backpropagation.

- **Selection:** Starting from the root, the algorithm traverses the tree by selecting the most promising child nodes according to a policy. The selection terminates when a leaf node or a terminal state is reached or the child node is not represented in the memory yet.
- **Expansion:** Upon reaching a leaf node, a new child node is added to expand the search tree based on available actions. The new node is then added to the search tree in memory.
- **Simulation:** A complete random simulation is executed from the newly expanded node until a terminal state is reached, and a score of the simulation is obtained. It is also referred to as a playout or rollout.
- **Backpropagation:** The score of the simulation is used to update the information of all nodes traversed in that iteration, refining their estimates of expected reward.

Figure 2.3 shows an example of an iteration of MCTS.

In MCTS, a node is said to be expanded when it has been visited at least once and some, but not necessarily all, of its children (representing possible subsequent states or actions) have been added to the search tree. A node is considered fully expanded when all of its children have been added to the search tree. A node is said to be terminal when it represents a state from which no further actions can be taken, which is also a state where the decision process ends. A node is said to be a leaf if it is a node without children, which is typically a state added to the tree but has not been expanded yet.

3.2 UCT

The first approach we explored is UCT (Upper Confidence Trees), which uses bandit ideas to guide Monte Carlo planning [64]. It follows the pattern of the basic MCTS algorithm: selection, expansion, simulation and back-propagation. UCT balances exploration and exploitation by selecting nodes in the search tree based on the sum of their average reward and an exploration term derived from the upper confidence bound. UCT is primarily applied in games such as chess, go [16], and others. It has become the foundation for many other MCTS algorithms, such as PUCT [84]. A variation of PUCT was used in the AlphaGo and AlphaZero algorithms [92], which made significant breakthroughs in the fields of game-playing and artificial intelligence.

The UCT first initializes the MCTS and root node, then iterates through the maximum number of iterations provided. It then selects a child node, expands unexplored actions, simulates the results of the actions, and back-propagates the results to update the tree. Given a state s , playouts are completed in a certain amount of steps and statistics about the states and the actions are collected. For each state $s \in \mathcal{S}$, the action a_s is chosen such that the upper bound of the score is maximized:

$$a_s = \arg \max_{a \in \mathcal{A}(s)} \left(\bar{Q}(s, a) + \tau \sqrt{\frac{\ln(Nb(s))}{Nb(s, a)}} \right) \quad (2.1)$$

where $\bar{Q}(s, a)$ is the estimated state-action value function for action a at state s , $Nb(s)$ is the number of times state s was visited, $Nb(s, a)$ is the number of times action a was selected at state s , τ is a constant value which controls the trade-off between exploration and exploitation. If more than one action has the same value, the tie is usually broken randomly.

Algorithm 1: UCT Algorithm

Input: State s , maximum iterations N , Monte Carlo Search Tree T

Output: The best action a for the given state s

Function UCT(s, N, T):

```

    root  $\leftarrow$  node  $s$  in  $T$ 
    for  $i \leftarrow 1$  to  $N$  do
        node  $\leftarrow$  root
        while node is fully expanded and not a terminal node do
            node  $\leftarrow$  SelectChild(node)
        end
        if node is not fully expanded then
             $a \leftarrow$  an unexplored action at node
             $s_{next} \leftarrow$  simulate action  $a$  from node
            add  $s_{next}$  as a new child node of node in  $T$ 
            node  $\leftarrow s_{next}$ 
        end
        get score from a random simulation from node
        Backpropagate(node, score)
    end
    return BestAction(root)

```

The UCT approach tries to find a balance between the exploitation and the exploration, corresponding the two terms in Equation 2.1. When no playout for action a is performed on state s , the exploration term would be $+\infty$, which forces the algorithm to prefer the unvisited actions on each state. This also guarantees that every action on state s is visited at least once before making any decision.

Algorithm 1 shows the UCT algorithm for state s . To start the UCT algorithm, we start with an empty Monte Carlo Search Tree T with a root node $root$. After executing the UCT algorithm in Algorithm 1, we return the best action a for $root$. Then after taking the action a , we execute the UCT algorithm again to find the best action for the next state. This process is repeated until we reach the terminal state.

3.3 NMCS

Another variant of MCTS algorithm we consider is Nested Monte Carlo Search (NMCS) [24], which uses a series of nested random simulations to estimate the value of a state.

NMCS is an algorithm that works well for puzzles. It biases its playouts using lower level playouts. At level zero, NMCS adopts a uniform random playout policy. Online learning of playout strategies combined with NMCS has given good results on optimization problems [83]. Other applications of NMCS include Single Player General Game Playing [70], Coding The-

ory [62], Cooperative Pathfinding [14], Software testing and heuristic Model-Checking [78], the Pancake problem and Games [24].

In NMCS, we often use “level” to denote the depth of nesting in simulations. For instance, a level 1 simulation involves executing a Monte Carlo simulation within another, while a level 2 simulation goes one layer deeper. At each level of the search, NMCS selects an action that maximizes the score returned by the next-level search. When the search reaches the maximum depth, it stops and returns the current state’s score based on a random simulation. By nesting the evaluation function inside another evaluation function, the ability of the traditional Monte Carlo is greatly improved.

However, this approach also has some significant disadvantages. For example, it is more sensible to the size of the search space. As the search depth increases, the number of nested simulations grows exponentially, which can lead to poor performance for problems with large search spaces. Another disadvantage is that NMCS heavily relies on random simulations, which can be sensitive or unstable.

Algorithm 2 shows the NMCS algorithm.

Algorithm 2: NMCS Algorithm

Input: State s , maximum depth D
Output: The best action a for the given state s
Function NMCS(s, D):

```

    if  $D = 0$  then
         $score \leftarrow$  perform a random simulation from current state
        return  $score$ 
    else
         $bestAction \leftarrow \emptyset$ 
         $bestScore \leftarrow -\infty$ 
        foreach action  $a$  in  $\mathcal{A}(s)$  do
             $s' \leftarrow$  execute action  $a$  from state  $s$ 
             $score \leftarrow$  NMCS( $s', D - 1$ )
            if  $score > bestScore$  then
                 $bestScore \leftarrow score$ 
                 $bestAction \leftarrow a$ 
            end
        end
        return  $bestScore$ 
    end
end

```

3.4 NRPA

Online learning of a playout policy in the context of nested searches has been further developed for puzzles and optimization with Nested Rollout Policy Adaptation (NRPA) [85]. NRPA has found new world records in Morpion Solitaire and crosswords puzzles. Edelkamp et al. have

applied the NRPA algorithm to multiple problems. They have optimized the algorithm for the Traveling Salesman with Time Windows (TSPTW) problem [26, 36]. Other applications deal with 3D Packing with Object Orientation [38], the physical traveling salesman problem [39], the Multiple Sequence Alignment problem [40], Logistics [37, 22], Graph Coloring [23] and Inverse Folding [21]. The principle of NRPA is to adapt the playout policy so as to learn the best sequence of moves found so far at each level.

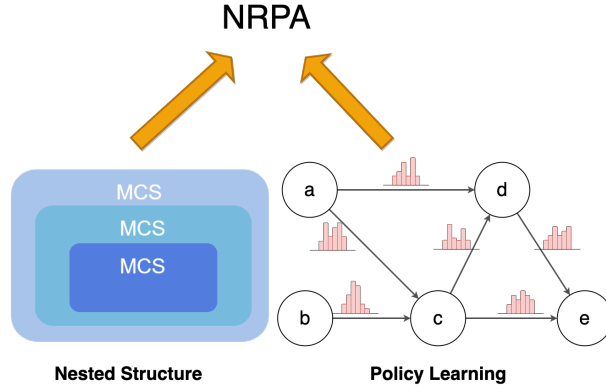


Figure 2.4: Illustration of the NRPA algorithm

The main idea behind NRPA is to learn a playout policy that guides the nested rollout sequences. The policy used in NRPA is a probability distribution over the available actions for each state. NRPA starts with an uniform policy, and iteratively refines the policy based on the outcomes of the simulations. Figure 2.4 shows an illustration of the NRPA algorithm.

The NRPA can be decomposed into three principal functions: the playout function, the adapt function and the NRPA function. The Algorithms 3, 4 and 5 show the three functions respectively. The NRPA uses a domain specific code $code(s, a)$ for the action a at state s in the representation of the policy, where many actions may share the same code, and actions with different codes are searched separately. For each nesting level, NRPA recursively calls to the lower level, searching to improve its current best score. When it succeeds, the best score of the corresponding state $score(s)$ is updated, and the current action sequence is recorded as the best sequence.

Algorithm 3: The playout algorithm

Input: State s , policy p

Output: The score of the state, the corresponding sequence of actions

Function $playout(s, p)$:

$sequence \leftarrow []$

while s is not terminal **do**

$z \leftarrow \sum_{a_i \in \mathcal{A}(s)} e^{p(a_i)}$

Draw action a from $a_i \in \mathcal{A}(s)$ with probability $\frac{1}{z} e^{p(a_i)}$

$s \leftarrow play(s, a)$

append a to $sequence$

end

return ($score(s)$, $sequence$)

Algorithm 4: The adapt algorithm

Input: Policy p , Sequence of actions $sequence$, learning rate α **Output:** updated policy pol **Function** $adapt(p, sequence, \alpha)$:

```

     $pol \leftarrow p$ 
     $s \leftarrow root$ 
    for  $a$  in  $sequence$  do
         $z \leftarrow \sum_{a_i \in \mathcal{A}(s)} e^{p[code(s, a_i)]}$ 
         $\forall a_i \in \mathcal{A}(s), pol[code(s, a_i)] -= \alpha * \frac{1}{z} e^{p[code(s, a_i)]}$ 
         $pol[code(s, a)] += \alpha$ 
         $s \leftarrow play(s, a)$ 
    end
    return  $pol$ 
end

```

Algorithm 5: The NRPA algorithm

Input: Level ℓ , number of iterations N , initial policy p , learning rate α **Output:** best score $bestScore$, best sequence $bestSequence$ **Function** $NRPA(\ell, N, p, \alpha)$:

```

    if  $\ell == 0$  then
        return  $playout(root, p)$ 
    end
    else
         $bestScore \leftarrow -\infty$ 
        for  $N$  iterations do
             $(result, new) \leftarrow NRPA(\ell - 1, N, p, \alpha)$ 
            if  $result \geq bestScore$  then
                 $bestScore \leftarrow result$ 
                 $bestSequence \leftarrow new$ 
            end
             $p \leftarrow adapt(p, bestSequence, \alpha)$ 
        end
        return  $(bestScore, bestSequence)$ 
    end
end

```

NRPA has many advantages, which includes a more efficient exploration of the search space, the ability to focus on the more promising part of the search space, and a wide range of applicable domains, etc. It also performs a limited number of simulations compared to other MCTS methods, which makes it efficient for large instances.

4 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning that focuses on training agents to make decisions to achieve a goal by interacting with the environment. The roots of RL can be traced back to the early 20th century, with the study of animal learning, behavior, and optimal control theory. The central idea behind RL is the concept of learning from trial and error, which is inspired by the way humans and animals learn to make decisions.

The field of RL gained significant traction in the 1980s and 1990s, with the development of key algorithms such as Temporal Difference (TD) learning [97], Q-learning [101], and State-Action-Reward-State-Action (SARSA) [86]. These algorithms laid the foundation for modern reinforcement learning research.

In recent years, RL has achieved remarkable success in various domains, due the integration of deep neural networks, leading to the emergence of Deep Reinforcement Learning (DRL), which also showcases the potential of reinforcement learning in solving difficult, real-world problems and indicate the possibility of even more significant breakthroughs in the future.

In RL, the interactions between the agent and the environment are also modeled using MDP (see section 3.1). The objective of the agent in step t is to maximize the expected cumulative reward:

$$G_t = \mathcal{R}_{t+1} + \gamma \mathcal{R}_{t+2} + \gamma^2 \mathcal{R}_{t+3} + \dots = \sum_{k=0}^{+\infty} \gamma^k \mathcal{R}_{t+k+1}$$

where $\gamma \in [0, 1]$ is a discount factor, which determines the relative importance of the future rewards, and $\mathcal{R}_t$ is the immediate reward of step t .

4.1 Q-learning

Q-learning is a model-free, off-policy reinforcement learning algorithm that aims to learn the optimal state-action value function, $Q^*(s, a)$, which represents the expected return for taking action $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$ and then following the optimal policy. The optimal policy is the one that maximizes the expected return from any state.

The Q-learning algorithm uses a table to store the Q-function values for each state-action pair, denoted as $Q(s, a)$. The algorithm starts with an arbitrary initialization of the Q-table and then iteratively updates the values using the Bellman equation. The update rule for Q-learning is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\mathcal{R}(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

The Q-learning algorithm is shown in Algorithm 6. Note that in the context of reinforcement learning, an episode is a single run where the agent starts from the initial state and interacts with the environment until a terminal state or condition is met.

Algorithm 6: Q-learning

```

Initialize  $Q(s, a)$  arbitrarily for all state-action pairs  $(s, a)$ 
for each episode do
    Initialize the current state  $s$ 
    while  $s$  is not a terminal state do
        Choose an action  $a$  for the current state  $s$  using policy  $\pi$  (e.g.,  $\epsilon$ -greedy policy)
        Take action  $a$  and observe the next state  $s'$  and reward  $\mathcal{R}(s, a)$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha [\mathcal{R}(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
    end
end

```

4.2 SARSA

SARSA (State-Action-Reward-State-Action) is a model-free, on-policy reinforcement learning algorithm that learns the state-action value function, $Q^\pi(s, a)$, for the policy π it follows. The algorithm aims to find a policy that maximizes the expected return from any state by iteratively updating the values for each state-action pair. The SARSA algorithm is closely related to Q-learning, but differs in its update rule, which is based on the action-value of the next state-action pair following the same policy π . The update rule for SARSA is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha [\mathcal{R}(s, a) + \gamma Q(s', a') - Q(s, a)]$$

where $a' \in \mathcal{A}(s')$ is the action chosen in state $s' \in \mathcal{S}$ according to the policy π . The SARSA algorithm is shown in Algorithm 7.

Algorithm 7: SARSA

```

Initialize  $Q(s, a)$  arbitrarily for all state-action pairs  $(s, a)$ 
for each episode do
    Initialize the current state  $s$ 
    Choose an action  $a$  for the current state  $s$  using policy  $\pi$  (e.g.,  $\epsilon$ -greedy policy)
    while  $s$  is not a terminal state do
        Take action  $a$  and observe the next state  $s'$  and reward  $\mathcal{R}(s, a)$ 
        Choose  $a'$  for  $s'$  using policy  $\pi$ 
         $Q(s, a) \leftarrow Q(s, a) + \alpha [\mathcal{R}(s, a) + \gamma Q(s', a') - Q(s, a)]$ 
         $s \leftarrow s'$ 
         $a \leftarrow a'$ 
    end
end

```

4.3 Eligibility Trace

The eligibility trace is a memory mechanism that helps the algorithm to assign credit or blame to state-action pairs that have been recently visited. It essentially keeps a trace, or memory

of recently visited state-action pairs, assigning them credit or blame for outcomes. By doing so, it allows the algorithm to update values for state-action pairs not only based on immediate outcomes but also considering their recent history. This acts as a bridge between immediate and delayed reward learning.

The eligibility trace for a state-action pair (s, a) at step t is denoted as $e_t(s, a)$, and it is updated at each step according to the following rule:

$$e_t(s, a) \leftarrow \gamma \lambda e_{t-1}(s, a) + \mathbb{1}(S_t = s, A_t = a)$$

where $\lambda \in [0, 1]$ is the trace-decay parameter, which determines how quickly the eligibility trace decays over time, and $\mathbb{1}(\cdot)$ is an indicator function, which equals 1 if the condition inside the parentheses is true, and 0 otherwise.

4.4 SARSA(λ)

SARSA(λ) is an extension of the SARSA algorithm that introduces eligibility traces, which allows the algorithm to update the state-action value function for multiple state-action pairs in a single step, making it more efficient in learning. The SARSA(λ) algorithm updates the state-action value function using the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$$

where

$$\delta = \mathcal{R}(s, a) + \gamma Q(s', a') - Q(s, a)$$

Algorithm 8 shows the pseudocode for SARSA(λ).

Algorithm 8: SARSA(λ)

```

Initialize  $Q(s, a)$  arbitrarily and  $e(s, a) = 0$  for all state-action pairs  $(s, a)$ 
for each episode do
    Initialize the current state  $s$ 
    Choose an action  $a$  for the current state  $s$  using the policy  $\pi$  (e.g.,  $\epsilon$ -greedy policy)
    while  $s$  is not a terminal state do
        Take action  $a$  and observe the next state  $s'$  and reward  $\mathcal{R}(s, a)$ 
        Choose the next action  $a'$  for  $s'$  using policy  $\pi$ 
         $e(s, a) \leftarrow \gamma \lambda e(s, a) + 1$ 
         $\delta = \mathcal{R}(s, a) + \gamma Q(s', a') - Q(s, a)$ 
        for all state-action pairs  $(s, a)$  do
             $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$ 
             $e(s, a) \leftarrow \gamma \lambda e(s, a)$ 
        end
        Set  $s \leftarrow s'$  and  $a \leftarrow a'$ 
    end
end

```

5 Dataset

In this thesis, we consider the classical dataset SNDlib [73] which contains real-world networks and traffic matrices of various sizes. Figure 2.5 shows some examples of SNDLib instances.

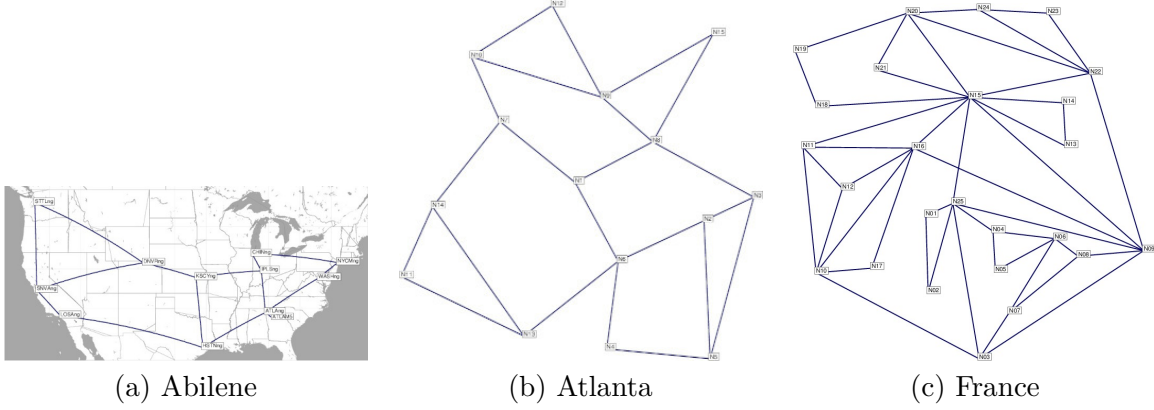


Figure 2.5: Examples of SNDlib instances

Some instances of Topology Zoo [63] are also used in this thesis, which is also commonly used in various studies in the domain. The Topology Zoo contains hundreds of network topologies, but often of small sizes, not more than hundreds of nodes. Figure 2.6 shows two instances in Topology Zoo.

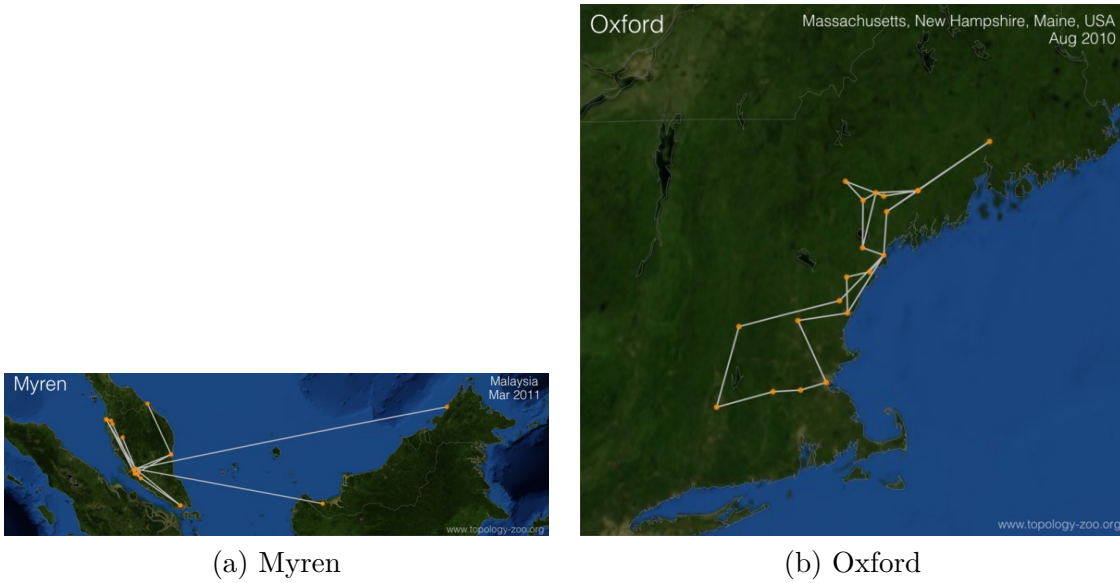


Figure 2.6: Examples of Topology Zoo instances

Since both datasets contain only small instances usually having less than 100 nodes, which is much smaller than the real-world networks, we also generated some large-scale networks using topology generators.

Over the last few decades, many topology generators have been proposed [102, 81, 69]. The first family of topology generators we used is the Random graph and Waxman graph, which

is commonly used in many previous researches [51, 17, 82]. For Random graph, the nodes are generated in a unit square and the arcs are generated according to a constant probability. As for the Waxman graphs, the nodes are generated using the same method as the Random graph, while the probability of having an arc between two nodes u and v is :

$$p(u, v) = \alpha e^{\frac{-D(u, v)}{\beta D_{\max}}}$$

where $D(u, v)$ is the Euclidean distance between nodes u and v , $D_{\max}$ represents the maximum Euclidean distance between any two nodes, α and β control the density of the graph. The capacities of all arcs are set to 1000.

We also used a topology generator *IGen* [81], which has a hierarchical network model, capable of generating large-scale two-level networks with a backbone and access topologies. Figure 2.7 shows an instance of *IGen*.

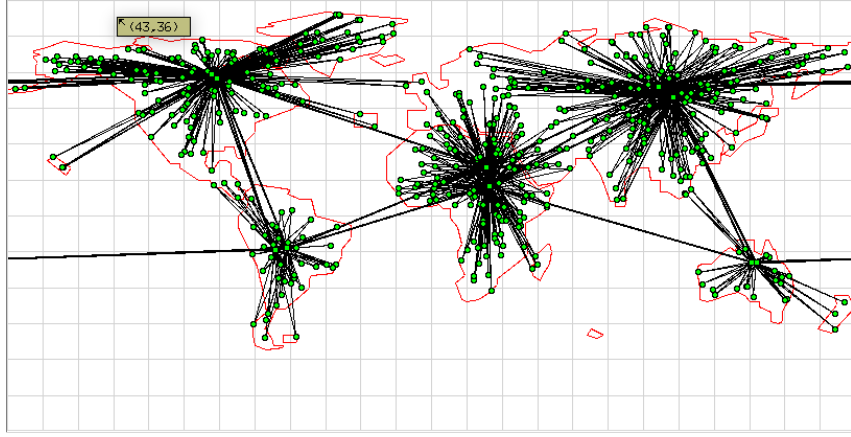


Figure 2.7: An example of *IGen* instance

The fourth topology generator is one of the Scale-Free network models, *Inet-3.0* [102]. In [43], the authors show that the distribution of node degree of the Internet topology follows a power law. The scale-free networks were proposed by [6] as models which follow the power law distributions in the connectivity of the network nodes. *Inet* is thus one of the topology generators that produces synthetic networks that more accurately follow the degree distributions of the Internet. Figure 2.8 shows an example of an *Inet* instance which is used in this thesis.

For all topology generator models, we create the demands set K as follows. We insert a demand k in K between each pair of network nodes with a fixed probability. Then, for each $k \in K$, the volume d^k is generated according to following formula:

$$d^k = \alpha S_{s^k} T_{t^k} C_{(s^k, t^k)} e^{\frac{-D(s^k, t^k)}{2D_{\max}}}$$

where $S_u, T_u \in [0, 1]$ are two random numbers associated to node u , $C_{(s^k, t^k)} \in [0, 1]$ is a random number for couple (s^k, t^k) , $D(s^k, t^k)$ is the Euclidean distance between nodes s^k and t^k , and $D_{\max}$ is the maximum Euclidean distance between any two nodes.

Table 2.1 shows information on the networks we used in our experiments.

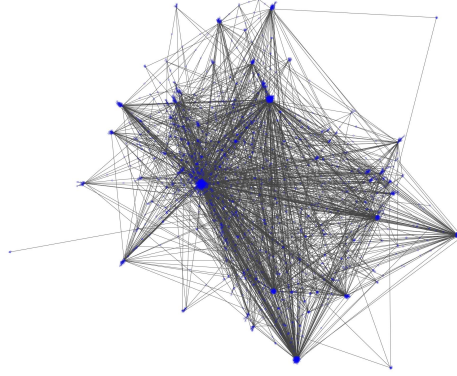


Figure 2.8: An example of Inet instance with 3040 nodes and 9856 arcs

Table 2.1: Table summarizing the different instances used in our experiments. The columns are : network name (“name”), number of nodes (“ $|V|$ ”), number of arcs (“ $|A|$ ”), number of demands (“ $|K|$ ”), total volume of the demands (“ $\sum d^k$ ”) and the maximum degree (“max degree”).

name	$ V $	$ A $	$ K $	$\sum d^k$	max degree
abilene	12	30	132	3000002	4
atlanta	15	44	210	136726	4
newyork	16	98	240	1774	11
france	25	90	300	99830	10
norway	27	102	702	5348	6
nobel-us	14	42	91	5420	4
nobel-ger	17	52	121	660	6
nobel-eu	28	82	378	1898	5
brain	161	332	14311	12.3e9	74
rand50	50	132	2450	81419	12
rand100	100	278	9900	269535	14
wax50	50	142	2450	85150	12
wax100	100	284	9900	331386	16
Myren	37	78	1332	44880	16
Oxford	20	52	303	13123	5
SwitchL3	42	126	1353	35753	10
igen50	50	202	1221	34517	21
igen100	100	400	2399	83248	35
igen500	500	2018	12565	377072	109
igen1000	1000	4050	14908	487523	105
igen2000	2000	8056	14918	488331	194
igen5000	5000	20058	19863	656416	545
inet3040	3040	9586	4540	144103	685
inet4000	4000	13288	7920	266207	856
inet5000	5000	17380	12717	414545	1025

5.1 Topological Properties

In this section, we present a comparative analysis of the topological properties between two large-scale network topology generators that we used, IGen [81] and Inet [102]. For the sake of clarity and relevance, we will limit our discussion to these generators alone and exclude the small real-world network families and Random/Waxman graphs from our analysis, as they do not meet the scale requirements necessary for our investigation.

Network Size IGen offers flexibility in generating instances of varying sizes, making it suitable for modeling telecommunication networks with diverse scales and complexities. On the other hand, Inet is designed to generate networks with a minimum of 3037 nodes, corresponding to the number of Autonomous Systems (ASs) on the Internet in November 1997. This constraint ensures that Inet-generated networks focus on larger-scale networks which are more similar to the structure of the Internet, reflecting the growth and expansion of the Internet over time.

Degree Distribution The node degree distributions in the networks generated by IGen and Inet are based on distinct underlying principles, reflecting the specific characteristics of the networks they are designed to model.

IGen relies on a hierarchical structure to create networks, representing the organization of telecommunication networks into Points of Presence (PoPs). This hierarchy is divided into backbone nodes, which interconnect nodes in different PoPs, and access nodes, which are only connected to routers within the same PoP. The degree distribution in IGen-generated networks is therefore determined by the structure of the PoPs and the heuristics employed to generate the backbone topology. This approach ensures that the generated networks exhibit realistic PoP structures consistent with operational practice in telecommunication networks.

On the other hand, Inet’s node degree distribution is driven by the power-law relationships observed in the topology of the Internet. The frequency-degree power law $f(d) \propto d^O$ was initially discovered by [43], which means that the frequency of nodes with degree d is proportional to d^O , where O is a constant. While it exhibits certain limitations, Inet focuses on the complementary cumulative distribution function $\bar{F}(d) = \sum_{i>d} f(i)$, which follows the power law $\bar{F}(d) \propto d^\alpha$, where α is a variable whose value varies over time. This relationship is characterized by a relatively constant fraction of low-degree nodes and an exponentially increasing degree for the top nodes. Inet-3.0 employs the following exponential growth law:

$$\bar{F}(d) = e^c d^{at+b}$$

where c , a , and b are constants, and t represents the number of months since November 1997. This approach captures the observed trends in the degree distribution of Autonomous Systems on the Internet, particularly for the top nodes.

The different mechanisms employed by IGen and Inet yield divergent node degree distributions. Figure 2.9 presents the histograms of node degree for two instances generated by IGen and Inet, each consisting of 5000 nodes. A distinct contrast between the two network families

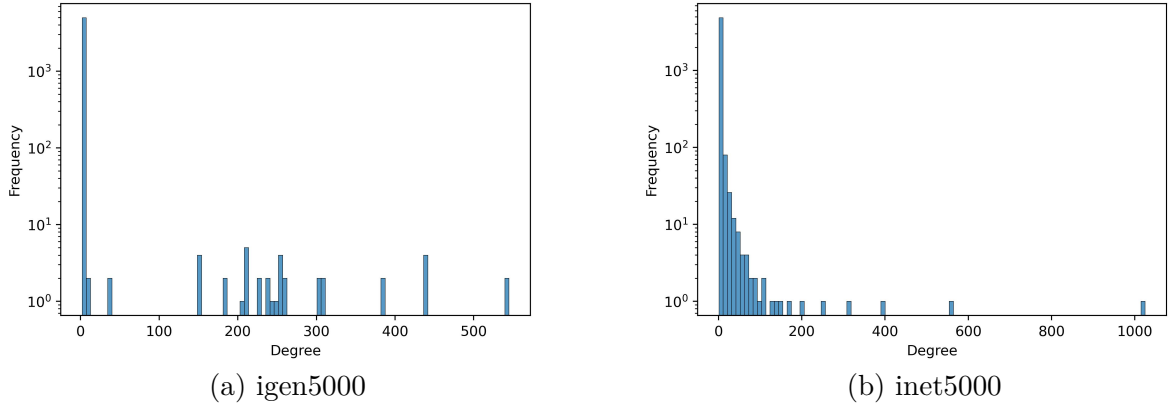


Figure 2.9: Degree distribution of IGen and Inet instances

is evident: The majority of nodes in the IGen-generated instance (igen5000) possess a very low degree, while the node degree distribution in the Inet-generated instance (inet5000) follows more closely to an approximate power-law relationship. The quantity of high-degree nodes is rather small, making them nearly imperceptible in Figure 2.9 (b).

To more effectively illustrate the disparities in degree distribution, we utilized Sankey diagrams, which are shown in Figure 2.10. Sankey diagrams are a type of flow diagram that effectively visualizes the quantitative relationships between different entities. In our context, we employ Sankey diagrams to display the connections between nodes with respect to their degrees in the generated network instances. In these diagrams, the left side represents the out-degree of a node, while the right side corresponds to the in-degree. Given that the networks are bidirected, the resulting Sankey diagrams exhibit complete symmetry in this case.

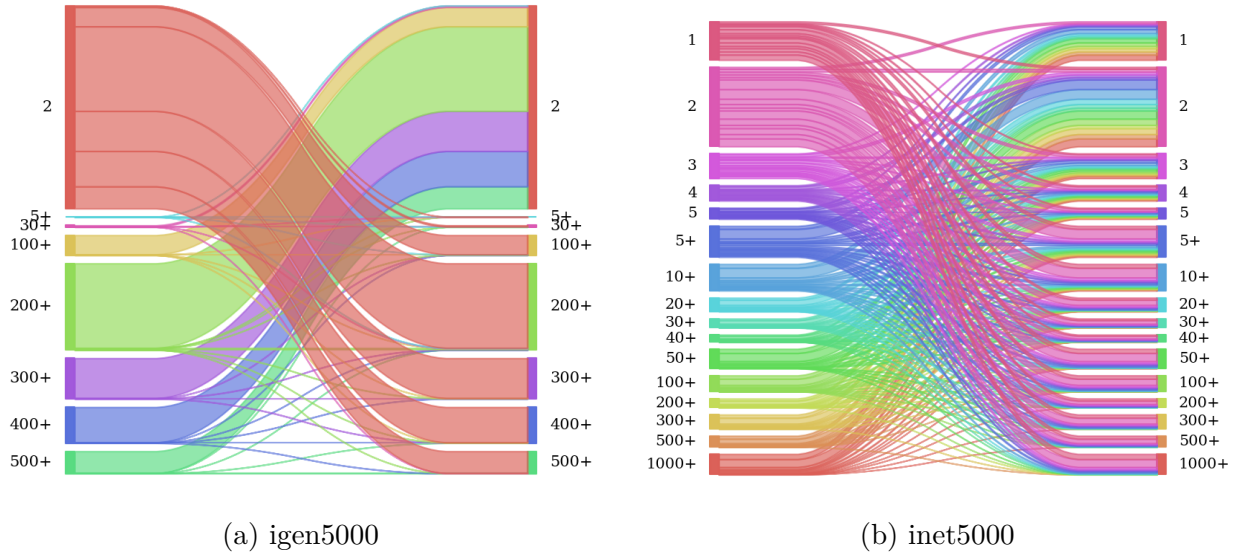


Figure 2.10: Degree sankey diagrams of IGen and Inet instances

The Sankey diagram effectively illustrates the connectivity of the network, as each area on

one side of the diagram signifies the total number of arcs connected to nodes with a specific degree. This visualization technique is particularly advantageous for illustrating highly centralized networks. For instance, the left diagram in Figure 2.10, which represents the igen5000 instance, reveals that the majority of nodes with a degree of 2 are connected to nodes with a degree greater than 100. This pattern indicates a distinct two-level network architecture, consisting of high-degree backbone nodes and access nodes, which predominantly have a degree of 2. In contrast, the right diagram, representing the inet5000 instance, exhibits a smoother transition from lower to higher degrees, with nodes displaying a more evenly distributed connectivity. Nodes of various degrees are interconnected, fostering a more diverse and less hierarchical network structure.

Betweenness Centrality Betweenness centrality is an important concept in network analysis that helps us understand the relative importance of nodes within a network. It is a measure that quantifies the extent to which a particular node acts as an intermediary or bridge along the shortest paths between other nodes in the network. In other words, betweenness centrality captures how frequently a node appears on the shortest paths connecting other node pairs. Consequently, nodes with high betweenness centrality are likely to play crucial roles in facilitating communication or information flow throughout the network.

The betweenness centrality of a node is calculated by determining the fraction of all shortest paths in the network that pass through that node. More formally, the betweenness centrality $C_B(v)$ of node v is represented as:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths between nodes s and t , and $\sigma_{st}(v)$ is the number of shortest paths between nodes s and t that pass through node v .

Figure 2.11 shows the distribution of betweenness centrality for igen5000 and inet5000. Because most of the nodes have an extremely small betweenness centrality for both instances (less than 0.1), we use logarithmic y-axis.

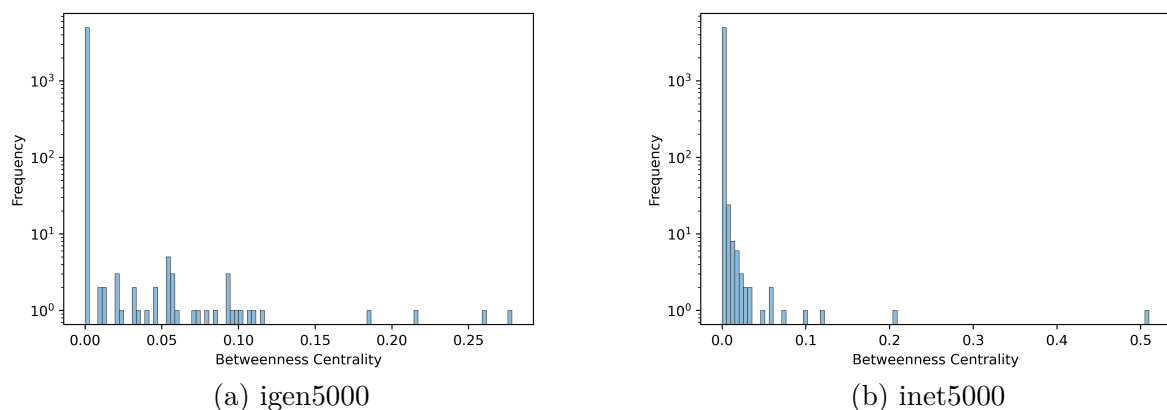


Figure 2.11: Betweenness centrality distribution of IGen and Inet instances

According to the result, in the IGen instance, the majority of nodes exhibit a betweenness centrality of 0, with a small number of nodes having higher betweenness centrality values (ranging from 0.02 to 0.28). This distribution reflects the hierarchical structure inherent in IGen-generated networks, where the backbone nodes have significantly higher betweenness centrality values due to their role in interconnecting different PoPs. In contrast, the access nodes, which are connected only to routers within the same PoP, have a betweenness centrality of 0, indicating their limited influence on the overall network connectivity.

The Inet instance, on the other hand, presents a more evenly distributed betweenness centrality. A large portion of nodes have a betweenness centrality of 0 or 0.02. The remaining nodes possess betweenness centrality values ranging from 0.04 to 0.52, albeit in lower frequencies. This distribution can be described using the power-law relationships that are utilized to generate Inet networks, resulting in a more diverse connectivity pattern among nodes with varying degrees. This leads to a smoother transition between different betweenness centrality values and a less pronounced hierarchical structure compared to IGen-generated networks.

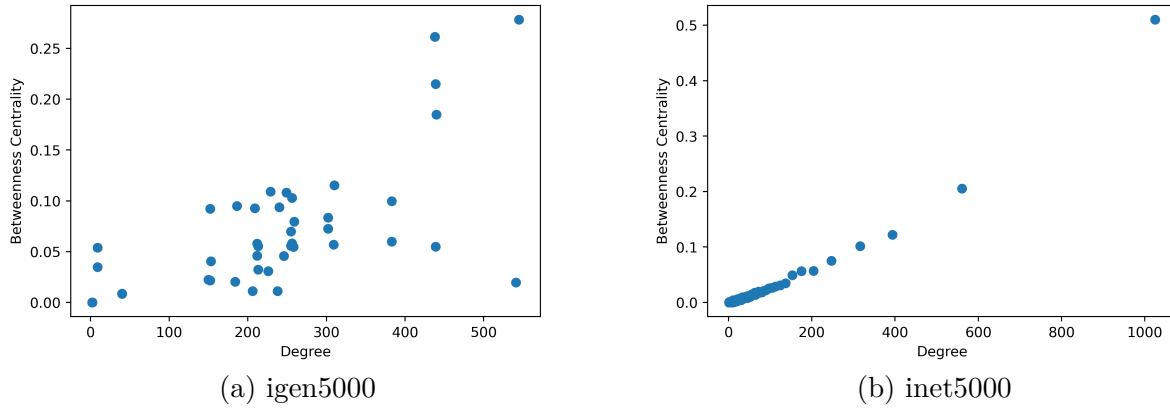


Figure 2.12: Correlation between betweenness centrality and degree of IGen and Inet instances

Figure 2.12 illustrates the relationship between betweenness centrality and node degree for the instances igen5000 and inet5000. An obvious observation for inet5000 is the presence of a strong, nearly linear correlation between node degree and betweenness centrality. In contrast, the igen5000 instance does not exhibit a clear correlation between these two network properties. However, it is discernible that a substantial number of high-degree nodes possess low betweenness centrality values.

The contrasting patterns observed in the correlation between betweenness centrality and node degree can be explained by the different design principles underlying the two topology generators. In Inet-generated networks, which are characterized by power-law relationships, the nodes with higher degrees tend to have more connections to other nodes, thus leading to a higher betweenness centrality. This results in the observed linear relationship between node degree and betweenness centrality in inet5000.

On the other hand, IGen-generated networks exhibit a distinct hierarchical structure, with backbone nodes interconnecting different PoPs and access nodes primarily having connections

within the same PoP. In this case, high-degree backbone nodes may exhibit relatively low betweenness centrality if their connections are predominantly within their respective PoPs, as they do not play a significant role in connecting different parts of the network. This leads to a lack of clear correlation between node degree and betweenness centrality in the igen5000 instance, as remarked in Figure 2.12 (a).

Diameter The diameter of a graph is the maximum number of edges in a shortest path from one node to another, which can be used as an indicator of the overall extent and connectivity of the graph. In the context of telecommunication networks, a smaller diameter is generally desirable, as it suggests that the network can relay information more rapidly and with fewer hops.

instance	diameter
igen200	5
igen500	5
igen1000	6
igen2000	7
igen5000	6
inet3040	9
inet4000	9
inet5000	8

Table 2.2: Diameter of the instances

Table 2.2 shows the diameter of the instances generated by IGen and Inet. It is evident that the diameters of IGen-generated instances are consistently lower compared to those of Inet-generated instances. This difference suggests that IGen-generated networks tend to have smaller maximum shortest paths between node pairs, possibly due to the hierarchical network structure and the presence of high-degree backbone nodes that facilitate connections across the network.

In contrast, Inet-generated networks exhibit larger diameters, indicating longer maximum shortest paths between nodes. This can be attributed to the power-law node degree distribution, which results in a more diverse and less hierarchical network structure, with nodes displaying a more evenly distributed connectivity.

Chapter 3

Weight Setting Problem with Monte Carlo Approaches

Objectifs

This chapter focuses on the application of Monte Carlo based algorithms to solve the WEIGHT SETTING problem. The chapter provides a modeling approach based on Monte Carlo algorithms. The chapter concludes with a discussion of the experiments and comparison of the results obtained for the WEIGHT SETTING problem. The content of this chapter is based on [31].

Contents

1	Introduction	32
2	Weight Setting Problem	32
2.1	Problem Formulation	32
2.2	State of the Art	34
3	Modeling Based on Monte Carlo Algorithm	34
3.1	Modeling	35
3.2	Variant of the NRPA	35
3.3	Restart	36
3.4	Force Exploration	36
3.5	Unique Metric	36
3.6	Increasing Metric Space	37
4	Experiments and Results	37
4.1	Comparison of the Monte Carlo Algorithms	37
4.2	Impact of the Metric Space	39
4.3	Comparison with the State of the Art Approach	40
4.4	Random Dense Networks	43
4.5	Increasing Metric Space	44
5	Conclusion	45

1 Introduction

In this chapter, we undertake the investigation of the WEIGHT SETTING problem by employing Monte Carlo Search methods in the context of traffic engineering for telecommunication networks of different scales. Despite the presence of various heuristics and exact methods, the WEIGHT SETTING problem continues to be a difficult challenge in optimizing network metrics on a large scale. We examine both the *splittable* and *unsplittable* versions of the problem. In the former version, each demand can be routed along several shortest paths, while the latter only allows one unique shortest path. Additionally, we consider a version of the problem with further constraints of delay.

We propose to use a Monte Carlo Search approach in order to get algorithms that (i) achieve a better scalability and (ii) can easily be extended to integrate operational constraints (unique shortest paths, delay, ...).

2 Weight Setting Problem

One of the most complicated problems faced by network administrators is the determination of efficient routing policies that induce high levels of QoS on operational networks while being robust to the inherent variation of traffic over time and network failures. In spite of the promises of the Multi-Protocol Label Switching (MPLS) [68] forwarding scheme, most nowadays networks heavily rely on the computation of shortest paths (Open Shortest Path First (OSPF) [72], Intermediate System to Intermediate System (IS-IS) [41], Segment Routing (SR) [46] or Flex-Algo [79]) to determine those routing policies.

In these routing protocols, network managers exert control over data flow by assigning administrative weights to the links within the network. Consequently, every demand is routed from its origin to its destination along the shortest paths determined by these weights. Although this method offers the advantage of simplified management, it lacks precise control over the elected routing paths, as modifications can only be made indirectly by altering the weights. As a result, network managers face the challenge of identifying a weight set that induces routing paths capable of minimizing load while maintaining a high level of Quality of Service (QoS) within operational networks. Unfortunately, solving this problem is computationally challenging.

In this chapter, we are interested in one of the network optimization problems related to this issue: Given a network and a set of demands, the task is to find a set of weights such that the demands routed along the induced shortest paths generate a minimum congestion *i.e.*, the maximum ratio value of the total traffic going through an arc over the arc's capacity is minimized.

2.1 Problem Formulation

We consider a network, which can also be represented as a bidirected graph where, for any arc uv , the reverse arc vu is also present. Given a network $G = (V, A)$, every node $v \in V$ corresponds to a router while an arc uv corresponds to a link between routers u and v . Every

arc uv is associated with a capacity denoted by c_{uv} . Let K denote a set of demands to be routed in G . Each demand $k \in K$ is defined by a pair of nodes s^k and t^k representing the source and the target of k , a traffic volume d^k to be routed from s^k to t^k . Such a demand k will be denoted by the triplet (s^k, t^k, d^k) . Given a metric $w \in \mathbb{Z}_+^{|A|}$, each demand $k \in K$ is routed along the shortest paths between s^k and t^k . If there are more than one shortest path joining the extremities of k , the traffic volume d^k is splitted evenly among those paths according to the so-called ECMP (Equal-Cost Multi-Path) rule. More precisely, the traffic volume that reaches a node $v \in V$ must be split equally among all arcs leaving v and belonging to the shortest paths toward destination t^k . We then define the *load* of an arc uv induced by w , denoted by $load(uv, w)$, as the amount of traffic traversing the arc uv over its capacity. Formally, the load on arc uv can be represented as $load(uv, w) = \frac{f_{uv}}{c_{uv}}$, where f_{uv} is the actual traffic flow on arc uv , and c_{uv} is the capacity of the arc uv . The network congestion $cong(w)$ of a given metric w , also known as the maximum link utility (MLU), is defined by $\max_{uv \in A} load(uv, w)$, that is the maximum load over all arcs.

We are now in position to define the optimization problem studied in this chapter. The WEIGHT SETTING problem, also known as the MINIMUM CONGESTION SHORTEST PATH ROUTING (MIN-CON-SPR) problem in some other researches [10], is to find a metric $w \in \mathbb{Z}_+^{|A|}$ and the routing paths induced by these weights such that the network congestion $cong(w)$ is minimum. The WEIGHT SETTING problem can be defined formally as follows:

WEIGHT SETTING

Input: A network $G = (V, A)$, where each arc uv has a capacity $c_{uv} \geq 0$ and a set of demands K defined for each $k \in K$ by the triplet (s^k, t^k, d^k) .

Output: A metric $w \in \mathbb{Z}_+^{|A|}$ of minimum congestion $cong(w)$.

In this chapter, we also consider the WEIGHT SETTING problem with some or all of the following additional constraints

Unicity : This constraint requires that each demand is routed along a unique shortest path.

Delay : This constraint requires that the routing paths have length at most the length of a shortest (s^k, t^k) -path (in terms of number of arcs) plus a constant $c \in \mathbb{N}$.

From an operational point of view, the unicity constraint is sometimes required by the network manager to monitor the flow circulating in the system more easily. We refer to the problem with unicity constraint as the *unsplittable* version of the problem, and to the version without unicity constraint as the *splittable* version. In addition to minimizing the congestion, the delay constraint ensures a certain level of QoS regarding the latency of answering the requests made by the clients.

Figure 3.1 illustrates an instance of optimization for the WEIGHT SETTING problem. In this example, we assume unit capacities for the arcs and that two demands with unit traffic volume $d^k = 1$ must be routed from node a to e and from node b to e , respectively. Each arc $uv \in A$ is associated with a label containing the weight and volume, $w_{uv}; load(uv, w)$. The network on the left demonstrates a set of weights that result in higher congestion values on the network, as

both demands select the shortest path according to the weights and pass through arc ce . By adjusting the weights on arcs ce and de , the shortest path for demand d^{ae} no longer traverses arc ce , leading to a lower congestion value for the network displayed on the right side.

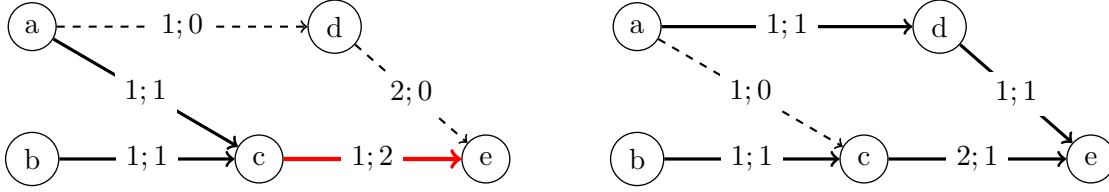


Figure 3.1: Instance of the WEIGHT SETTING problem. In this figure, we assume unit capacities and suppose that two demands with unit traffic volume $d^k = 1$ must be routed from node a to e and from node b to e , respectively. A label with the weight and the volume $w_{uv}; load(uv, w)$ is associated with each arc $uv \in A$.

2.2 State of the Art

Several authors proposed to solve this problem using integer programming models and meta-heuristics methods, the reader is referred to [13] for a complete overview of these approaches. Regarding the splittable variant of the problem, Fortz and Thorup [49] showed that it is NP-hard to approximate within a factor $\frac{3}{2} - \epsilon$ for all $\epsilon > 0$.

Hence, to cope with the hardness of this problem, many different meta-heuristics approaches were investigated. Fortz and Thorup [49] first proposed a local search algorithm to solve the splittable variant which was later implemented in the TOTEM library [66] and called IGP-WO. This approach was further extended to compute robust solutions against single link failures [50] or in the context of oblivious routing [4]. Genetic algorithms were also proposed to solve this problem [42, 17]. The reader is referred to the surveys [48, 3] for more details and references about the existing meta-heuristics approaches proposed to solve this splittable variant. Most of the previous meta-heuristics were tested on networks of small/moderate size and do not consider the unsplittable case or QoS constraints such as the delay.

Regarding the unsplittable case, Bley [11] showed that this variant is NP-hard even on bidirected cycles and not $\mathcal{O}(|V|^{1-\epsilon})$ -approximable unless $P = NP$ in the general case. In [12], the author proposed an exact algorithm using a two-phase approach: the problem is decomposed into a master problem that aims at finding an optimal shortest path routing, and a client problem which consists in finding a compatible set of weights for those shortest paths.

3 Modeling Based on Monte Carlo Algorithm

We detail in this section how the Monte Carlo Search methods can be used to minimize the congestion of the WEIGHT SETTING problem in different cases: with or without the previous additional constraints. The details of the Monte Carlo Search methods are presented in Chapter 2.

3.1 Modeling

To model the WEIGHT SETTING problem with Monte Carlo Search algorithms, we suppose that a solution for an instance of WEIGHT SETTING is represented by a point (i.e. the metric $w = \langle w_1, w_2, \dots, w_{|A|} \rangle$) in the discrete space $[1, 65535]^{|A|}$. To reduce the search space, we can set the value space of the metric as a subspace W of the original space $[1, 65535]$.

For each playout, the metric of the network is assigned and the objective function is evaluated. In our case, the maximum congestion value in the network, which is also the MLU, is used as the score, noted as $\text{cong}(w)$. For the case with additional constraints, after obtaining the congestion of the network, an additional bias will be added in the case that the constraints are not fully satisfied, which will encourage the algorithm to explore the solutions with smaller congestion value which satisfies all the constraints. The final score for state s is then $Q_s = \text{cong}(w) + \text{cost}(\text{unchecked constraints})$. More specifically, every unsatisfied constraint leads to a penalty of $1e5$, which is far greater than the congestion value. This can encourage the search algorithm to find valid solutions first.

Furthermore, we assume that the arcs of a network have a default order, and the metric values corresponding to them are assigned sequentially. The state s corresponds to the arc of the network, and an action a is a choice of metric values for an arc. In this case, the total number of states is in fact the number of arcs of the network, and every assigned metric value leads to the same next state: the next arc in the predefined order. For NRPA, the domain-specific code is uniquely determined by the index of the arc to which the metric is currently to be assigned. Because of the different learning mechanism, the order of the arcs has no effect on NRPA, but may lead to different results for UCT and NMCS. In this work, we do not study the effect of arc order on UCT and NMCS, but focus on the performance comparison of each algorithm for instances of different size.

3.2 Variant of the NRPA

In order to improve the stability of the NRPA algorithm, a stabilized version of NRPA is proposed in [25] to encourage exploration before the adaptation of the policy. During the level 0 of NRPA, instead of running a single playout and using its result as the score, multiple playouts will be performed and only the best result will be used as the score. It improves the average scores for many problems because of the stability of the playouts. But the disadvantage is that this variant may take several times more execution time to finish, which makes it more time consuming. For brevity, we denote the stabilized NRPA with m playouts as $\text{NRPA}(m)$.

There exists some other variants for the NRPA, for example, Beam NRPA [27] is an extension of the NRPA algorithm that incorporates a beam search to improve the performance of the search process. By keeping several best search policies, it achieves more robust results. But it bears the same disadvantage as the stabilized NRPA, which is also more time consuming. For brevity, in our study, we consider only stabilized NRPA.

3.3 Restart

One of the most significant challenges associated with Monte Carlo based approaches is that it tends to become entrapped in local minima, thus inhibiting exploration of better solutions. In our study, we discovered that, in general, given an equivalent total execution duration, performing several independent executions of Monte Carlo Search leads to more consistent and better results compared to a singular but extended execution.

In the context of addressing smaller instances, which often have less than several hundreds of nodes, the difference between multiple smaller executions and an extended execution is relatively negligible. But as soon as we deal with larger instances, the strategy involving multiple executions shows a significant improvement in comparison to its counterpart. This observation underscores the importance of incorporating restarts within the process. Chapter 5 will further introduce a more advanced technique to address the problem of restarting by warm-starting the Monte Carlo Search with optimal stopping theory. For the moment, we are satisfied with the naive approach of restarting after a small execution of the Monte Carlo search, and restarting the search from scratch.

3.4 Force Exploration

In our research, we have observed that the NRPA algorithm displays a preference towards exploitation over exploration when applied to small and medium-sized networks. As a result, the same metric is often repeatedly obtained without any new metrics being explored.

To mitigate this tendency, we propose two techniques. Firstly, we suggest employing a hash-table to store previously explored metrics and their associated scores. This approach can significantly reduce the computational overhead incurred by recalculating congestion. Secondly, we introduce a novel *force_exploration* mechanism that operates independently of the NRPA algorithm. This mechanism works as follows: first, all previously explored metrics w are recorded along with their respective hash codes. When performing a new playout, we check hash codes to see if the metric has already been explored. If yes, a random metric value is assigned to a random arc in the network until a new metric that has not been explored before is generated. This simple yet effective approach maximizes exploration while preserving the original mechanics of the NRPA algorithm. Our results demonstrate that the use of the *force_exploration* technique can greatly improve the performance of the NRPA and Stabilized NRPA algorithms.

3.5 Unique Metric

For some networks, it is difficult to find routing metrics that satisfy all constraints, especially unique path constraints for large instances. In such cases, using a unique metric for each arc can greatly increase the proportion of results that satisfy the constraints. However, this limits the number of metrics to be greater than or equal to the number of arcs. This will in many cases increase the proportion of valid solutions, *i.e.* solutions that satisfy the constraints. But as we will show later, in the absence of constraints, this restriction reduces the quality of the solution.

3.6 Increasing Metric Space

When dealing with large-scale instances, the size of the search space plays a critical role in the optimization process. In our case, when the number of arcs in an instance is fixed, reducing the metric space can lessen the optimization complexity. However, our experiments have shown that a smaller metric space may limit the optimization’s ability by ignoring many possible solutions. To address this issue, we propose an approach to increase the metric space during the process of the optimization.

Our procedure involves executing a single simulation using Monte Carlo Search with a metric space $W = \{1\}$, where an equal metric value which is equal to 1 is assigned to all arcs. Following the simulation, we double the metric space while maintaining the original search policy and perform a standard NRPA. After completing the NRPA, we double the metric space once again and repeat the process. This iterative approach allows us to incrementally expand the metric space and explore a broader range of solutions.

In our approach to expanding the metric space, we prioritize refining the previous search space rather than merely extending it. Specifically, when the previous search space is $W = \{1, 2\}$, we prefer to insert values into the search space, such as $W = \{0.5, 1, 1.5, 2\}$, rather than simply appending values such as $W = \{1, 2, 3, 4\}$. Our experience has shown that inserting new values between existing metrics can better preserve the relationships between arcs that the policy previously learned, rather than solely expanding the search space.

As the metric values used in our algorithm are restricted to integers, using decimals as metric values is not possible. To address this issue, we adopt a strategy where every metric value in the current search space W_{cur} is obtained by doubling the corresponding metric value in the previous search space W_{prev} . For instance, if the previous metric space is $W_{prev} = \{1, 2\}$, the new metric space would be $W_{cur} = \{1, 2, 3, 4\}$, where the previous metric value 1 maps to the new metric value 2, the previous value 2 maps to the new value 4, and the newly added metric value 0.5 and 1.5 map to 1 and 3. This approach allows us to maintain the policy for integer metric values while increasing the size of the metric space.

4 Experiments and Results

The algorithms are implemented in C++ and the experiments are done on a server with 64-core Intel(R) Xeon(R) Gold 5218 CPU of 2.30GHz with 125 GB of memory. Only one core is used during the experiments.

4.1 Comparison of the Monte Carlo Algorithms

In this part, we compare the performance between the Monte Carlo approaches that we proposed. With both unique path and delay constraints applied, all the approaches are evaluated during a limited execution time.

Table 3.1 presents the average score obtained by each approach based on five executions of the SNDLib instances. For NRPA and Stabilized NRPA, only *force_exploration* is used. The

Table 3.1: Comparison of the best scores of Monte Carlo Search algorithms in limited runtime with all constraints. Multiple: multiple short executions in limited time. Single: a single long execution. Format: average score of 5 executions (number of executions in which no valid solution was found). “-”: no valid solution is found in all executions.

name	runtime (min)	W	UCT		NMCS	NRPA		NRPA(10)
			multiple	single		multiple	single	
abilene	10	50	65.158	87.108	60.905	60.905	60.905	60.905
atlanta	10	50	2.9608	4.3968	2.3626	2.318	2.318	2.318
newyork	30	100	0.1255(1)	0.172(4)	0.0978	0.0636	0.062	0.0622
france	30	100	3.983	-	3.20836	2.892	2.92	2.9268
norway	60	150	-	-	0.508(2)	0.3054	0.295	0.3042
nobel-us	10	50	29.48	35.04	25.68	25.2	25.2	25.2
nobel-ger	10	100	4.44	6.06	4.4	4.4	4.4	4.4
nobel-eu	30	100	12.08	14.6(3)	10.94	10.7	10.88	10.74
brain	60	50	1.002	1.0513	0.9848	0.974	0.961	0.98

stabilized NRPA approach with 10 playouts is denoted as *NRPA*(10). Since both UCT and NRPA can execute multiple runs of short duration or a single long run within a limited period, we report both results for comparison purposes. We note that Stabilized NRPA requires more time to complete each instance, and NMCS exhibits exponential time complexity for larger instances; therefore, the multi-execution version of these approaches was omitted from our analysis.

Our experimental results have demonstrated that executing multiple different runs, *i.e.* Monte Carlo Search with restart, is generally more effective than executing a single long execution. The use of multiple runs allows the optimization policy to restart and avoid getting stuck at a local minimum, which can occur when executing only one long run. This phenomenon is particularly apparent in the case of UCT, as it tends to produce results that are much worse than the lower bound, leaving ample room for improvement. Although the difference is less pronounced for NRPA, it becomes more significant when solving large instances with hundreds or thousands of nodes. Therefore, executing multiple runs can be a valuable strategy to improve the overall performance of the optimization process.

The results presented in Table 3.1 clearly demonstrate that NRPA and its variants outperform the other two Monte Carlo Search methods. While UCT still has difficulties finding valid solution for several instances (*e.g.* france, norway), all executions of NRPA and Stabilized NRPA can easily find valid solutions which are also close to the lower bound (see Table 3.2 for the lower bound calculated with linear programming for SNDLib instances). Furthermore, the number of playouts required for NRPA is independent of the network size or the size of the metric space, making it easier to scale the algorithm to larger networks and search spaces. While NMCS performs well with smaller networks and search spaces, it does not scale well for larger instances in SNDLib. Consequently, in subsequent experiments, we will exclusively utilize NRPA-based algorithms.

To investigate the effect of various techniques for NRPA, we conducted experiments on several SNDlib instances, as shown in Figure 3.2. The distribution of the scores across the different configurations indicates the efficacy of each technique. A higher concentration of scores around

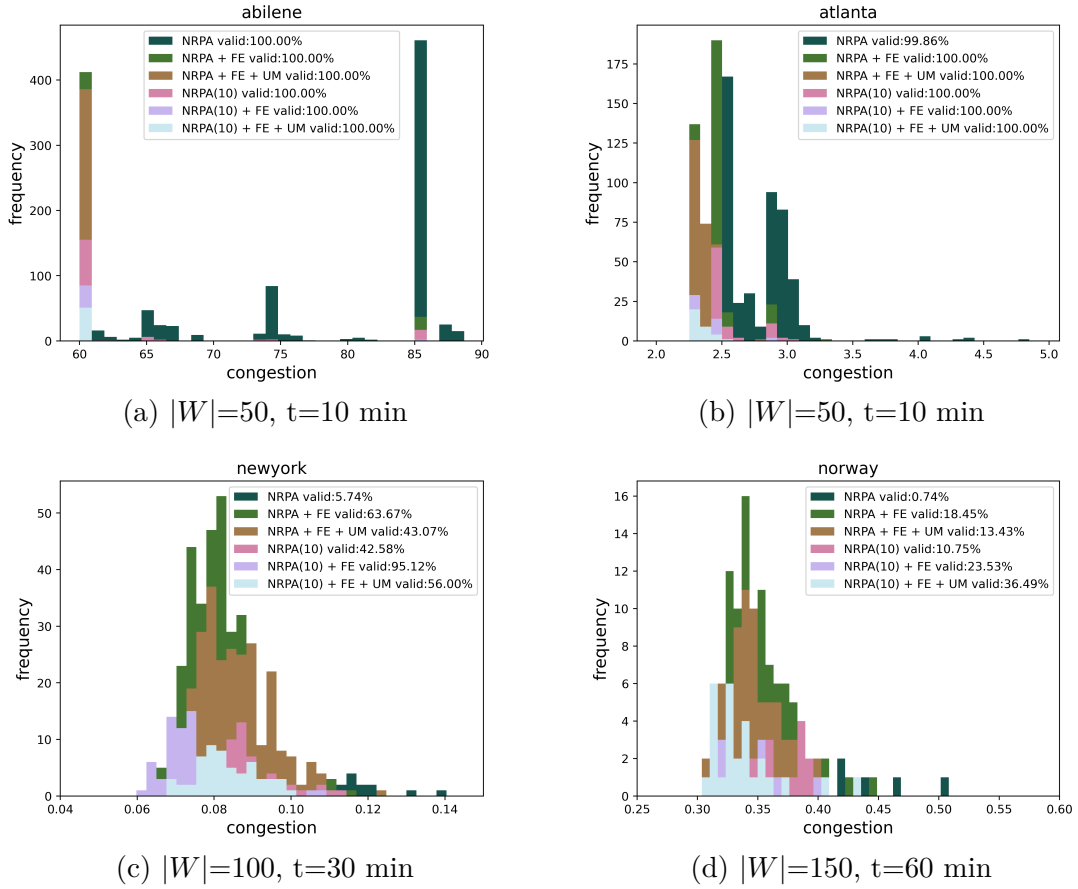


Figure 3.2: Distribution of the congestion values with all constraints on SNDlib instances

low values signifies a greater likelihood of obtaining lower congestion values and thus, better performance.

The results clearly demonstrate the positive impact of *force_exploration* on both NRPA and stabilized NRPA. The use of stabilized NRPA also significantly increases the ratio of valid solutions. However, the effect of the unique metrics constraint is not as evident, and it can sometimes make it more challenging to obtain better results.

4.2 Impact of the Metric Space

Although in many previous studies of heuristics, the metric space is typically considered a continuous set of integers, our research has shown that both the size and values of the metric space have a significant impact on the algorithm's performance.

For instance, compared to a continuous set of integers or the set of prime numbers, a set of randomly generated numbers of the same size usually yields a much higher rate of valid solutions. Figure 3.3 provides an example of the distribution of scores obtained with different metric spaces of the same size. In this experiment, the random numbers were uniformly generated in the range of $[1, 65535]$ and remained constant throughout the experiment. $W = \text{random}(100)$ means that the metric space is formed by 100 random distinct values in $[1, 65535]$. $W = \text{prime}(100)$ means

that the first 100 prime numbers are used as the values of the metric space. While prime numbers can not improve the ratio of the valid solutions, the optimization benefits more from random number metric space.

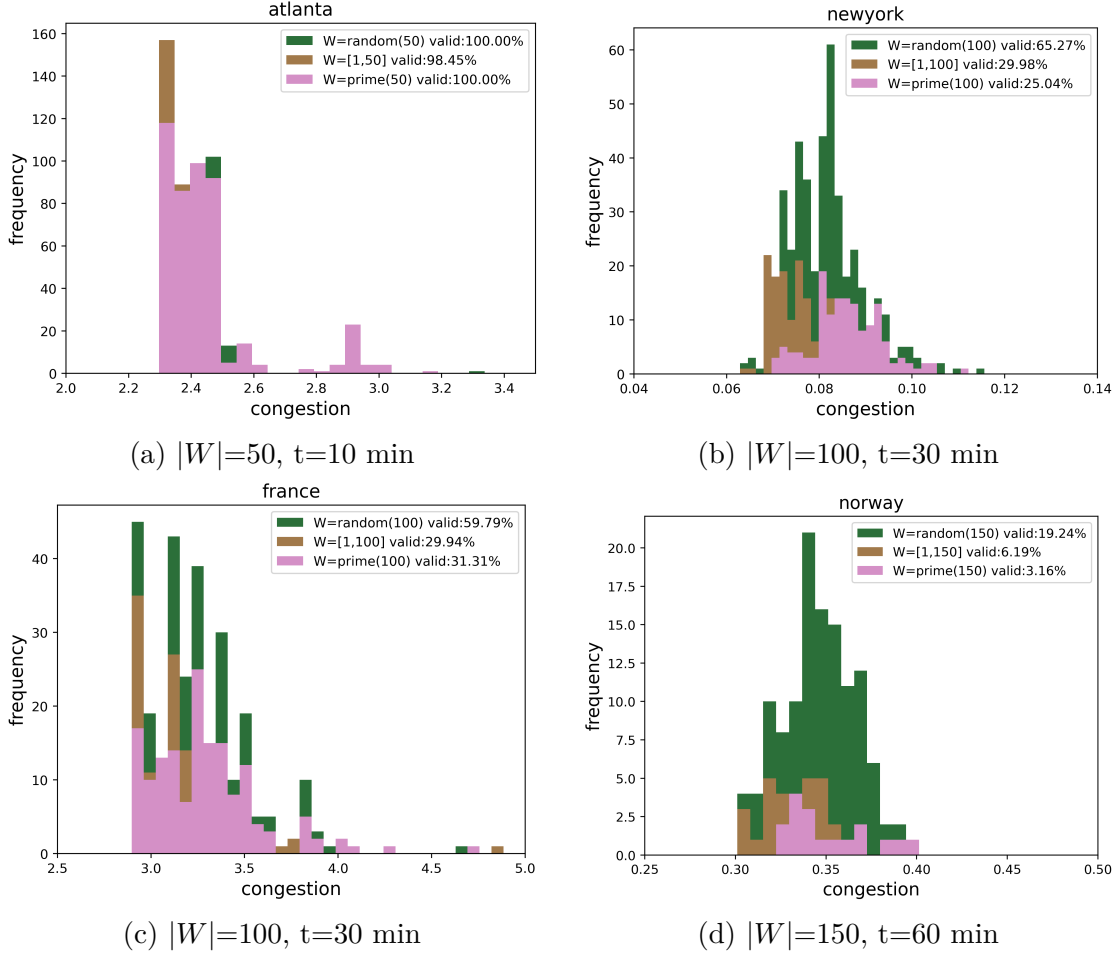


Figure 3.3: Distribution of the congestion values with different metric spaces of the same size using NRPA with *force_exploration*

Our research has also revealed that for most networks, a smaller metric space can help the algorithm converge better when no constraints are applied. This is because the algorithm can better explore the search space. Figure 3.4 illustrates the convergence of different metric space sizes. However, when unique path and delay constraints are applied, a small metric space can make it more challenging to find a metric that induces a valid solution. In such cases, increasing the search space can significantly increase the chances of obtaining a solution that satisfies all constraints. Figure 3.5 depicts the impact of the metric space size on the percentage of valid solutions.

4.3 Comparison with the State of the Art Approach

In this test, we compare the congestion value computed by our algorithm after a fixed amount of time with the congestion values obtained via other common approaches. The first approach for

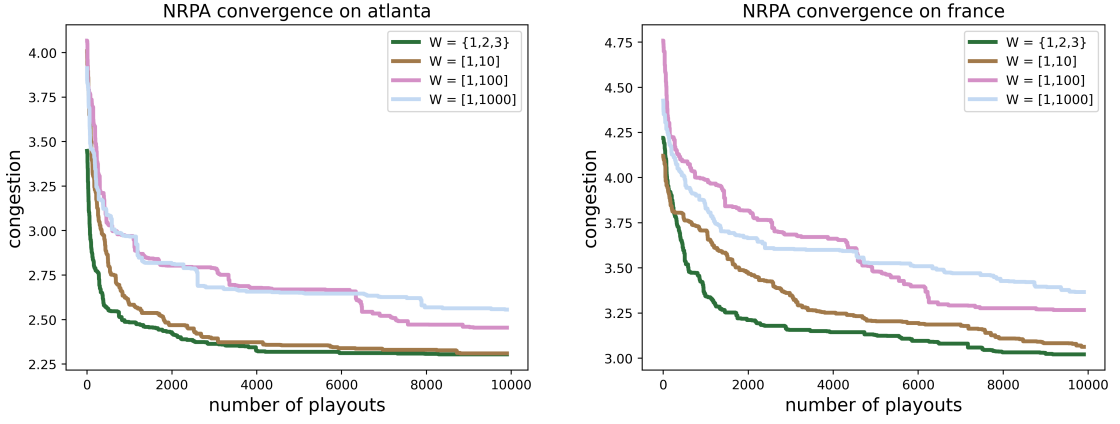


Figure 3.4: Convergence of NRPA without constraints. The score is averaged on 10 runs.

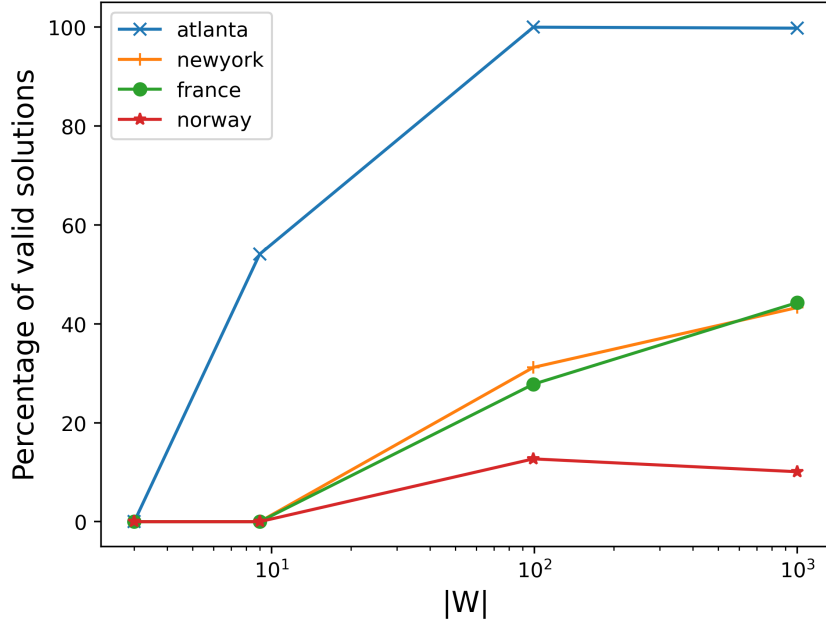


Figure 3.5: Size of the metric space has a positive impact on the valid solution ratio when constraints are applied. The result is averaged on 1000 runs.

the problem, which is also the most basic approach is the UnitOSPF, which assigns all arcs the same unit metric value. InvCapOSPF is another approach recommended by Cisco, which sets the metric inversely proportional to the arc's capacity. However, in many networks the capacity on all arcs is the same, so in many cases this method will give the same results as UnitOSPF. We also compare our algorithm with the local search IGP-WO implemented in [66] and based on [49]. We slightly modify the objective function of IGP-WO to minimize the congestion and, when considering the unicity constraint, add a high penalty for solutions that violate that constraint. We did not further modify the algorithm to take into account the delay constraint as this would require deeper modifications and understanding of the implementation of IGP-WO.

In order to evaluate the quality of the heuristic solutions, we compare them with the optimal

value obtained using the compact formulation of the MAXIMUM CONCURRENT FLOW (MCF) problem. The MCF problem seeks to maximize the throughput over multiple demands in a network, subject to the capacity constraints [90]. It is not hard to see that any optimal solution for MCF is a lower bound for the WEIGHT SETTING problem (denoted LPLB for Linear Programming Lower Bound). Indeed, a solution of a MCF instance defines routing paths for each demand that are not constrained to follow the ECMP rule or being induced by shortest paths w.r.t some metric.

Table 3.2: Maximum congestion value of state-of-the-art heuristics and NRPA. For each constraint configuration (“Without constraints”, “Unicity” and “All” i.e. both “Unicity” and “Delay”) and each heuristic (if available for the given configuration), we show the congestion induced by the computed weights. This value is in bold if it is the best one among those returned by the other heuristics (w.r.t the configuration). In addition, a value is followed by * if equal to the lower bound LPLB (which is reported in the last column). Finally, an entry $c : i$ indicates the minimum value i of c for which we were able to find a solution when considering all constraints.

name	Without constraints				Unicity		All	LPLB
	Unit OSPF	InvCap OSPF	LS	NRPA	LS	NRPA	NRPA	
abilene	187.55	89.48	60.42	60.412	60.41	60.41	60.90	60.411
atlanta	3.26	3.37	2.22	2.22	2.29	2.29	2.32	2.18
newyork	0.076	0.076	0.051	0.053	0.062	0.065	0.064	0.045
france	4.12	4.12	2.53	2.56	2.88	2.88	2.89	2.41
norway	0.42	0.42	0.28	0.29	0.29	0.30	0.31	0.27
nobel-us	37.15	37.15	24.4	24.7	24.7	24.7	25.2	24.2
nobel-ger	5.54	5.54	3.9	3.89	4.4	4.4	4.4	3.87
nobel-eu	13.31	13.31	10.68	10.67*	10.7	10.7	10.7	10.67
brain	1.415	1.415	0.962	0.903*	0.972	0.972	0.974	0.903
rand50a	7.9	7.9	5.55	5.77	5.84	5.92	5.96(c:2)	5.55
rand50b	2.88*	2.88*	2.88*	2.88*	-	2.88	2.88(c:3)	2.88
rand100a	15.71	15.71	10.42	9.59	-	10.35	10.76(c:4)	9.35
rand100b	4.15	4.15	4.38	3.85	-	6.06	5.94(c:5)	3.76
wax50a	6.46	6.46	4.63	4.66	4.665	4.67	4.71(c:2)	4.59
wax50b	2.279*	2.279*	2.284	2.279*	-	2.279	2.279(c:3)	2.279
wax100a	17.46	17.46	15.049	15.048	-	15.049	15.049(c:4)	15.048
wax100b	5.51	5.51	4.14	4.04	-	5.86	5.91(c:5)	3.44

Table 3.2 shows the comparison of the results. For local search (LS), we use the open sourced version which is called ICP-WO. To keep the results consistent across configurations, for NRPA, the metric space size and computation time of each instance are consistent with those shown in Table 3.1 for the constrained cases. For all generated instances, the metric space size is 250, and the computation time is 30 minutes for instances of 50 nodes, 60 minutes for instances of 100 nodes. However when no constraints are applied, the metric space for all instances is set to $\{1, 2, 3\}$ for better performance. The scores are averaged on 5 executions.

For Random graphs and Waxman graphs, the applied delay constraint is often too restrictive so that no valid solutions can be found. In this chapter, we propose an automatic delay relaxation mechanism that allows the NRPA algorithm to relax the delay constraint when it is difficult to

find a valid solution.

In the initial state, the delay constraint is actually defined as the length of the shortest path (in terms of arcs) plus one for each demand. We relax the constraint to shortest path plus c , where $c \in \mathbb{N}_+$. Assuming that the computation has a target time or total number of runs T , we divide all the computations into different phases. At each stage, we count the total number of runned playouts, the number of valid solutions and the number of solutions that violate the delay constraint. After $T \times (1 - \frac{1}{2^c})$ iterations, if the percentage of valid solutions is less than 5% and the percentage of results that violate the delay constraint is greater than 10%, we move to the next stage, where all statistics are recalculated and $c = c + 1$, until c reaches a maximum provided value c_{max} .

The results show that NRPA performs very well for all three different constraint configurations on all sizes of networks and is very close to the lower bound. Compared to local search, our algorithm gives better results in most cases. At the same time, only very little computational time and resources are used. The different runs can be computed in parallel, which substantially improves the running time of NRPA.

The proposed automatic delay relaxation mechanism is not the best way to solve the problem of minimizing the congestion of the network while keeping the delay minimized. Nevertheless, we obtained rather encouraging results that show the strong adaptability of our approach and a promising potential for solving even more difficult variants of the problem (single-link failure, oblivious routing, capacity planning, ...).

4.4 Random Dense Networks

For dense networks, the number of potential routing paths increases rapidly, which makes the problem even harder to solve in particular with the unicity constraint, as observed in [12]. We show that NRPA can easily scale up to networks with a large number of nodes and arcs. So we generated ten random graphs of different sizes with the same generation mechanism as described in Section 5, and the traffic between two nodes is generated with a probability of 0.1. The largest network contains 1000 nodes and 99450 arcs. The problem we consider in this section is without constraint, since both the unique shortest path and delay constraints are difficult to satisfy for instances of large sizes.

Figure 3.6 shows the scores of the Random graphs without constraints. The metric space is set to $\{1, 2, 3\}$ for both local search and NRPA algorithms, and the scores of NRPA are averaged on 10 executions.

Regarding the LPLB bound, with a compact formulation, we were not able to compute the lower bounds for instances larger than 100 nodes because the random graphs are more dense and thus the problem is too large to be solved. In future experiments, using a non-compact path formulation of the MCF may greatly improve the chances of obtaining lower bounds for larger networks.

The results show that on large networks, local search does not provide acceptable results within an execution time of 30 minutes. On the contrary, even on networks with thousands of nodes, our method still gives reasonable results in a very short time.

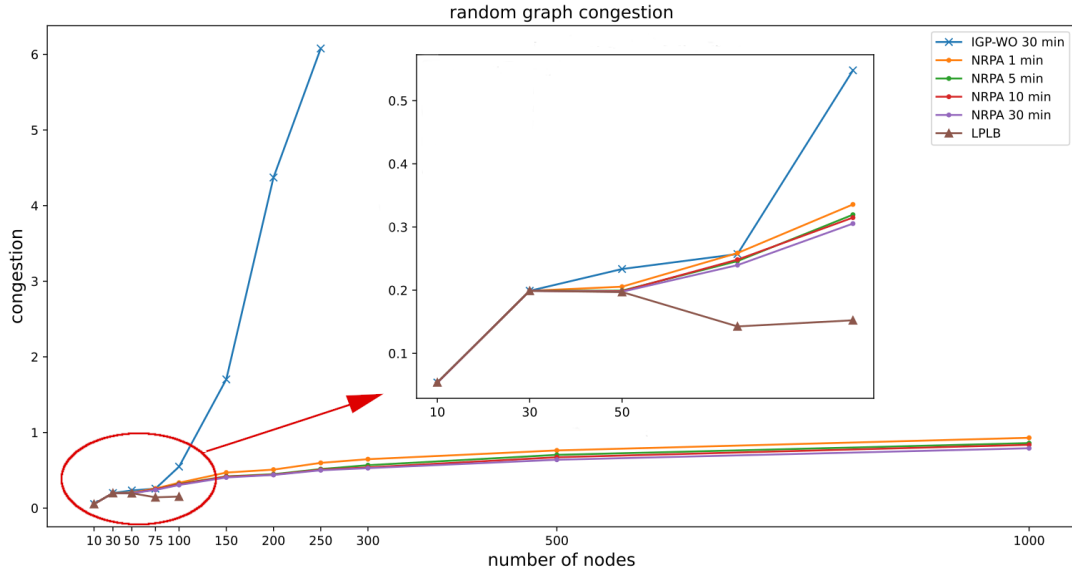


Figure 3.6: Congestion with respect to the number of the nodes

4.5 Increasing Metric Space

While the proposed method achieves state-of-the-art results in addressing the problem, it does so at the expense of exploring only a limited metric space. It is essential to acknowledge that a large search space can potentially hinder the searching process, making it more computationally difficult and time-consuming. Conversely, a small search space restricts the range of possible outcomes, which may lead to suboptimal results due to the limited exploration of alternative solutions.

We demonstrate here the experimental results of increasing metric space while exploring with NRPA. The metric space is increased two times during each execution if it is within the time limit, *i.e.*, the maximum metric space is then $\{1, 2, 3, 4\}$. To compare with our proposition, we use a fixed metric space of $\{1, 2, 3, 4\}$. Figure 3.7 shows the results on two families of networks: IGen and Inet. The algorithms are executed 5 times for each instance. The execution time differs according to the size of the instance: 10 minutes for Igen500 and Igen100, 60 minutes for all other instances. Because of the increasing difficulty of satisfying the constraints, we only consider here the unsplittable version of the problem.

The findings evidently highlight that the strategy of increasing the metric space considerably enhances the performance of NRPA on the IGen instances. This underscores the importance of examining a larger metric space for the discovery of more effective solutions. However, the influence on the Inet dataset is less conclusive, which could potentially be attributed to the distinct underlying architectures of the network topology.

Further investigation is conducted to examine the inefficiency of this technique on the inet5000 instance. Figure 3.8 presents a comparison of the convergence among diverse metric spaces for the inet5000 instance with different numbers of demands generated using the same technique. The final result is the average of five independent executions. It can be observed that the metric space $\{1, 2, 3, 4\}$ consistently outperforms other metric spaces for this instance, even

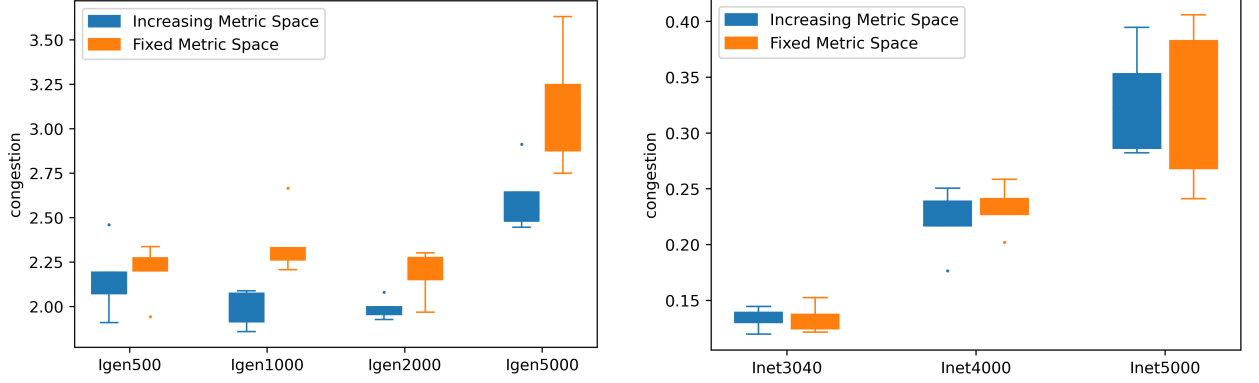


Figure 3.7: Comparison of congestion between increasing and fixed metric space for IGen and Inet instances

with a reduced number of demands, whereas both $\{1, 2\}$ and $[1, 8]$ contribute to a decline in performance. This observation further highlights the limitations of expanding the metric space. When the increased metric space performs significantly worse, simply enlarging the metric space from a smaller one does not lead to improvement.

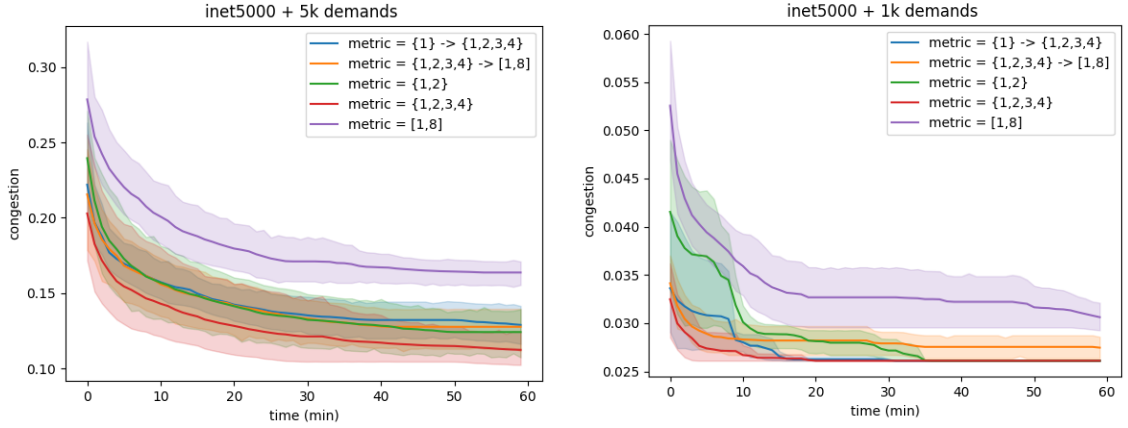


Figure 3.8: Comparison of convergence between different metric spaces for inet5000. The result is the average convergence of 5 independent executions.

These results highlight the importance of understanding the relationship between metric space size and algorithm performance in different contexts. While increasing metric space has proven beneficial for the IGen dataset, further investigation is necessary to determine its impact on other datasets of different characteristics of typology, such as Inet, and to identify the specific factors influencing the observed results.

5 Conclusion

In this chapter, we applied for the first time the Monte Carlo Search approach in the context of setting efficient weights in IP networks, in particular for the WEIGHT SETTING problem. The principle of the Monte Carlo Search algorithm is to learn a policy online on each instance using

nested levels of best solutions.

We compare several Monte Carlo Search methods and propose the most appropriate to the target problem. Experiments show that our approach is comparable with the existing ones for instances from the literature. Nevertheless, for networks of large size, our approach outperforms the local search heuristics and gives results close to the lower bound. At the same time, this approach can be easily extended for problems with additional constraints and is not sensitive to the size of the network or the size of the search space, in particular the number of available weights, giving it a large range of applications. Furthermore, for the unsplittable case, this method may provide optimal solutions (or close to the optimal) for instances where exact approaches fail, especially for dense graphs [12].

For some instances where it is not possible to find a way to satisfy all the constraints, we also propose a mechanism for automatically relaxing the constraints. Another algorithm specifically aimed at optimizing congestion with the lowest possible delay constraint could be the direction of subsequent research.

Chapter 4

Hierarchical Shortest Path Routing

Objectifs

In this chapter, we are interested in the problem of efficiently simulating shortest path routing protocols in large-scale telecommunication networks, which is crucial for solving traffic engineering related optimization problems that often involve the computation of shortest paths as a subroutine. We address this problem by proposing the Hierarchical Shortest Path Routing (HSPR) algorithm, which is an adaptation of the Customizable Contraction Hierarchy (CCH) into the context of network routing protocols. Our experiments show that HSPR significantly reduces the simulation time of routing protocols such as OSPF and IS-IS with the Equal-Cost Multi-Path (ECMP) rule, in comparison to a direct application of the classical Dijkstra algorithm. Finally, we provide an application of HSPR as the shortest path computation subroutine of our NRPA algorithm for the WEIGHT SETTING problem introduced in Chapter 3.

Contents

1	Introduction	48
2	Customizable Contraction Hierarchy	49
2.1	Preliminaries	49
2.2	Phase 1: Preprocessing	51
2.3	Phase 2: Customization	53
2.4	Phase 3: Query	54
3	Extending CCH for Shortest Path Routing Computation	54
3.1	Exploiting Telecommunication Networks Properties	54
3.2	Multiple shortest paths computation & ECMP rule	55
3.3	Further improvements	56
4	Experiments and Results	57
4.1	Elimination Tree Depth	57
4.2	Congestion Calculation Time	58
5	Application to Weight Setting Problem	60
5.1	Experimental results	60
6	Conclusion	63

1 Introduction

The question of quickly computing the shortest paths in a graph is crucial when it comes to simulating telecommunication network protocols and evaluating the Key Performance Indicator (KPI) of a routing policy, especially in large scale networks. We observed during the experiments of the Chapter 3 that, the simulation of shortest paths is the main bottleneck for solving the so-called WEIGHT SETTING problem [51, 42, 75]. This is because the simulation is repeatedly performed to evaluate routing metrics and obtain a congestion value for the heuristics. The existing literature primarily relies on Dijkstra’s algorithm to evaluate those metrics, which can be a computationally intensive task, particularly in large real-world networks. Efficient updating procedures exist to avoid running Dijkstra’s algorithm from scratch in the case that only a few metrics have changed between two consecutive calls [18]. However, we cannot assume this in most meta-heuristics as two consecutive evaluated solutions may have completely different metrics. In the past decades, several graph pre-processing methods have been proposed to substantially improve the running time of shortest paths computation for route planning on large scale road networks [55, 87, 8]. Contraction hierarchies [53] proposed in 2012 explored the inherent hierarchical structure of road networks. Shortcut edges are added, and some nodes are removed to facilitate the query of the shortest paths. Some other improvements [33, 34] were also proposed, to separate the time-expensive preprocessing into a slow metric-independent preprocessing phase and a fast metric-dependent customization phase, improving the overall speed of route planning.

We propose in this chapter the Hierarchical Shortest Path Routing (HSPR) algorithm to quickly simulate shortest path routing protocols. It is a refined version of the Customizable Contraction Hierarchy (CCH) algorithm [34], specifically tailored for use in telecommunications networks and includes routing rules such as Equal-Cost Multi-Path (ECMP). It is worth mentioning that our approach is generic and can be used as a basic building block for efficiently simulating protocols based on the computation of shortest paths such as Segment Routing (SR). Our experiments establish that the simulation time of shortest path routing protocols is significantly reduced when using our proposed algorithm, HSPR, in comparison to the classical approach with Dijkstra’s algorithm.

To further validate the efficiency of HSPR, we applied it to compute the score function in the Monte Carlo Search algorithm presented in Chapter 2 to solve the classical traffic engineering problem WEIGHT SETTING problem defined in Chapter 3. Given a network and a set of demands, the task is to compute a set of arc weights such that each demand is routed along the shortest paths induced by these weights while minimizing the maximum congested arc (*i.e.*, minimizing the highest ratio value of traffic traversing an arc over its capacity). Our results indicate that HSPR substantially improves the quality of the solutions returned by the Monte Carlo Search algorithm in terms of congestion value. More generally, this suggests that HSPR is likely to improve the efficiency of any algorithm that requires an efficient evaluation of a routing shortest paths policy.

The remainder of the chapter is organized as follows. Section 2 is devoted to the basic

definitions and description of the Customizable Contraction Hierarchy (CCH) algorithm. We introduce and describe the HSPR algorithm in Section 3. Then we present experiments to show the efficiency of our method in calculating shortest paths for large networks and different topologies in Section 4. In Section 5, we propose an application of HSPR in the context of solving the routing optimization problem WEIGHT SETTING problem with the NRPA algorithm presented in previous chapter; our experiments provide further evidence that HSPR can clearly improve the performance of meta-heuristics. Conclusions and future directions are given in Section 6.

2 Customizable Contraction Hierarchy

The Customizable Contraction Hierarchy (CCH) is one of the state-of-the-art methods for shortest path computation in road networks. First proposed by [34] as an improvement of the original algorithm Contraction Hierarchy [53], CCH has achieved great results in large road networks with constantly changing traffic metrics. CCH has three phases: metric-independent preprocessing, metric-related customization, and query. The preprocessing phase, which is only performed once per graph, analyses the topology of the graph and creates additional links between nodes, which are also called shortcuts. Every time the metrics of the network change, the customization phase updates the graph with the new metrics and makes necessary adjustments to the shortcuts and the super graph generated in the preprocessing phase. Finally, the query phase calculates the path and unpacks the shortcuts to determine the shortest path in the original network.

2.1 Preliminaries

We define here some important notions for the understanding of the CCH algorithm.

Upward directed graph Given a graph $G^u = (V, E)$, the upward directed graph, denoted as $G_\pi^\wedge = (V, A_\pi^\wedge)$, is a directed graph that is derived from the original graph G^u using a topological ordering of its nodes π . The upward directed graph contains only “upward” arcs, which are the arcs that connect nodes with lower ranks to nodes with higher ranks according to the given order π , which also guarantees that all upward directed graphs are acyclic.

Formally, let $n = |V|$ and $\pi : v_1, v_2, \dots, v_n$ be a node order of the graph G^u . The set of upward arc $A_\pi^\wedge \subseteq E$, which is also referred to as upward directed edges in the literatures, is defined as $A_\pi^\wedge = \{uv | uv \in E \text{ and } \pi^{-1}(u) < \pi^{-1}(v)\}$.

In other words, an arc uv belongs to the set of upward arcs $A_\pi^\wedge$ if and only if the rank of node u is strictly smaller than the rank of node v , i.e., $\pi^{-1}(u) < \pi^{-1}(v)$.

In our case, we only utilize the upward directed graph derived from the chordal super graph G_π^* of the graph G^u . For simplicity, we refer the upward directed graph as the upward directed graph of the chordal super graph G_π^* of the graph G^u .

Triangle A triangle $\{u, v, w\}$ is a set of 3 adjacent nodes.

Lower triangle A triangle $\{u, v, w\}$ is a lower triangle of uv if it is formed by nodes $u, v, w \in V$ such that $\pi^{-1}(w) < \pi^{-1}(u)$ and $\pi^{-1}(w) < \pi^{-1}(v)$. In this case, the node w has a lower rank than both u and v according to the order π .

Upper triangle A triangle $\{u, v, w\}$ is an upper triangle of uv if it is formed by nodes $u, v, w \in V$ such that $\pi^{-1}(w) > \pi^{-1}(u)$ and $\pi^{-1}(w) > \pi^{-1}(v)$. In this case, the node w has a higher rank than both u and v according to the order π .

Intermediate triangle A triangle $\{u, v, w\}$ is an intermediate triangle of uv if it is formed by nodes $u, v, w \in V$ such that $\pi^{-1}(u) < \pi^{-1}(w) < \pi^{-1}(v)$ or $\pi^{-1}(v) < \pi^{-1}(w) < \pi^{-1}(u)$. In this case, the node w has a rank between the ranks of u and v according to the order π .

Figure 4.1 shows an example of a triangle.

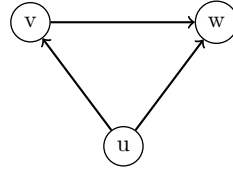


Figure 4.1: Example of a triangle $\{u, v, w\}$. This triangle is a lower triangle of the arc vw , an intermediate triangle of the arc uw , and an upper triangle of the arc uv .

Up-down path The up-down path P in an upward directed graph $G_\pi^\wedge$ is a path between two nodes $s, t \in V$ that satisfies the following conditions:

- The path starts at node s and ends at node t .
- The path initially follows an upward or non-decreasing sequence of node ranks, i.e., the rank of each node in the path is greater than or equal to the rank of its predecessor. This part of the path is called the “upward” segment.
- After reaching a highest-ranked node, the path switches to a downward or non-increasing sequence of node ranks, i.e., the rank of each node in the path is less than or equal to the rank of its predecessor. This part of the path is called the “downward” segment.

It is essentially a combination of an upward path P_u and a downward path P_d . Formally, an up-down path P between nodes s and t can be represented as a sequence of nodes $v_1, v_2, \dots, v_k$ such that $v_1 = s$, $v_k = t$, and there exists an index i ($1 \leq i \leq k$) for which:

1. $\pi(v_j) \leq \pi(v_{j+1})$ for all $1 \leq j < i$ (upward segment), and
2. $\pi(v_j) \geq \pi(v_{j+1})$ for all $i \leq j < k$ (downward segment).

2.2 Phase 1: Preprocessing

The first step in the preprocessing phase is to create a metric-independent elimination order $\pi : \{v_1, v_2, \dots, v_n\}$ on the nodes of an unweighted graph G^u , which can be computed offline. This is an essential aspect of the preprocessing, and we often use *nested dissection* orders (ND-orders) as the elimination order. ND-orders are a technique for ordering nodes in a graph that has been proposed in previous research [9].

To compute an ND-order π for a graph G^u , the algorithm follows a recursive approach. At each step, it uses a minimum balanced separator S to divide the graph into two parts A and B . A balanced separator of a graph G^u is a set of vertices whose removal splits the graph into two disconnected subgraphs of roughly the same size, and a minimum balanced separator is such a separator S with the smallest possible size. The nodes in S are then assigned to positions $\pi(n - |S| + 1), \dots, \pi(n)$ in the order, where n represents the total number of nodes in G^u . The algorithm proceeds to calculate ND-orders π_A and π_B for the subgraphs induced by A and B recursively and assigns the nodes to positions $\pi(1), \dots, \pi(|A|)$ and $\pi(|A| + 1), \dots, \pi(|A| + |B|)$, respectively. The recursion terminates when the subgraphs become empty.

The main challenge in computing ND-orders is finding good balanced graph separators. A good ND-order tends to result in a shallow elimination tree because it minimizes the number of children for each node, keeping the tree balanced and limiting its depth. However, finding the elimination tree of a graph with minimum tree depth is NP-hard [76]. Many heuristics have been proposed to tackle this challenge, such as NDMetis [60], Inertial Flow [88], and FlowCutter [56], among which FlowCutter is widely used as a base algorithm. We will discuss more in depth about the heuristics for ND-orders.

Given a metric-independent order π , we can compute the Contraction Hierarchy for graph G^u . The nodes are contracted iteratively according to the reverse order of π , which means the least important nodes are extracted first during this process. For each contraction, the selected node $v_i \in V$ is temporarily removed from the graph, the generated graph is noted as $G_{v_i} = G_{v_{i-1}} \setminus \{v_i\}$. As a result, all paths incident to node v_i need to be redirected to its neighbors. Thus, following each contraction, additional edges, referred to as shortcuts, are added between its neighbors $u, w \in V \setminus \{v\}$ if they do not already exist. The contractions are performed on all the nodes in the graph. After this process, the resulting graph, which is the original graph G augmented by all the shortcuts, is referred to as a chordal super graph $G_\pi^* = \bigcup_i G_{\pi,i}$ or a Contraction Hierarchy, where $G_{\pi,i}$ denotes the core graph, which can be obtained by contracting nodes $\pi(1) \dots \pi(i-1)$ from the original graph G^u . After replacing every edge by an arc directed to the node of higher rank, we obtain an upward directed graph $G_\pi^\wedge$ of the chordal super graph G_π^* . From $G_\pi^\wedge$, we can generate the elimination tree, which is a tree directed towards the root. Every parent of a node is its lowest-ranked higher neighbor in the chordal super graph G_π^* .

Figure 4.2 shows an example of preprocessing on an input graph as Figure 4.2a. Figure 4.2b shows the result of applying the balanced separator recursively, from which we can obtain a ND-order. Then, the nodes are contracted according to the order, which are shown in each node in Figure 4.2c, and shortcuts are added, shown as the dotted lines. Figure 4.2d shows the corresponding elimination tree.

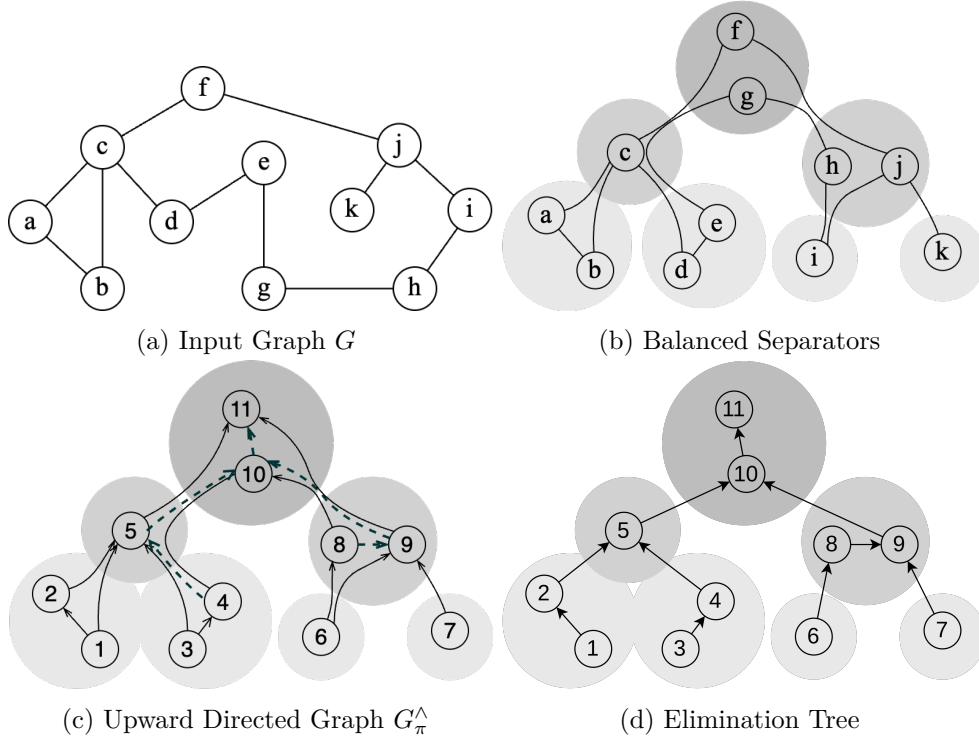


Figure 4.2: Example of CCH preprocessing with an example graph

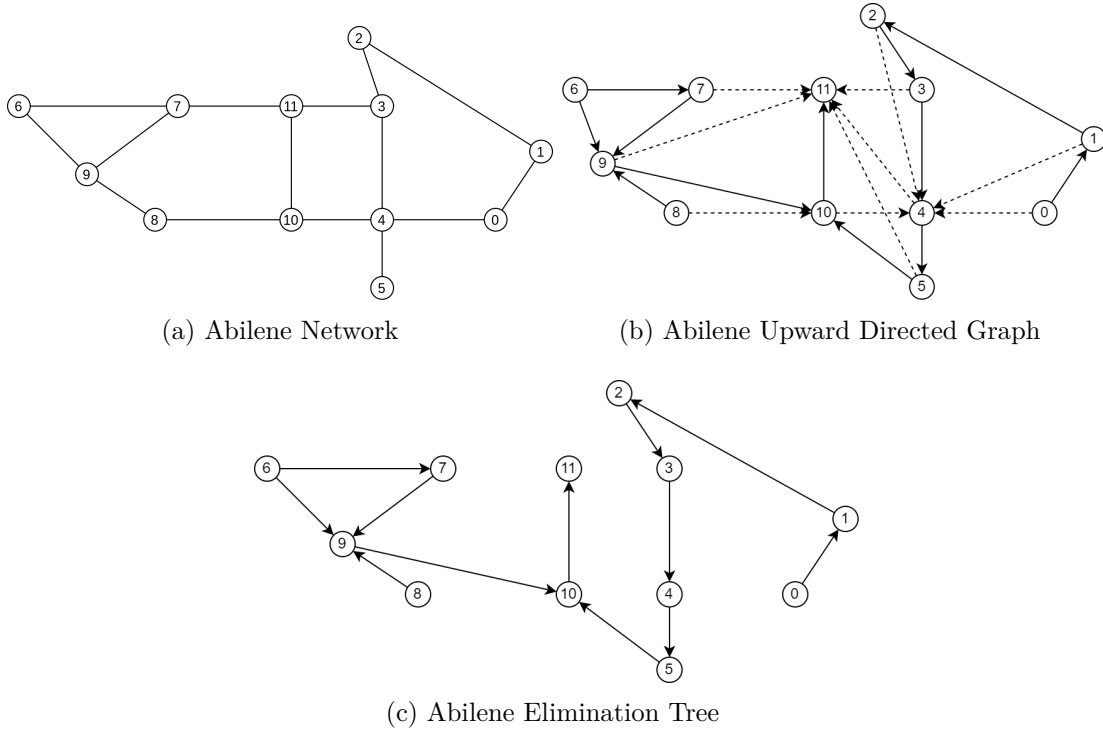


Figure 4.3: Example of CCH preprocessing with a real telecommunication network Abilene

Figure 4.3 shows another example of the network Abilene from dataset SNDLib [73] with an elimination order, the corresponding upward directed graph $G_{\pi}^{\wedge}$ that contains all the shortcuts generated by CCH, and the elimination tree.

2.3 Phase 2: Customization

The customization phase plays a vital role in adapting the algorithm to changes in the network's edge weights, facilitating efficient calculation for new weights. We refer to $dist_{UD}(s, t)$ as the distance for the shortest path between s and t in $G_\pi^\wedge$ when only up-down paths are allowed. We also note $dist_A(s, t)$ as the distance of the shortest path from s to t in G_π^* , which can also be treated as an arbitrary path in $G_\pi^\wedge$, and $dist_I(s, t)$ as the distance of the shortest path between s and t in G^u .

In the customization phase, the metrics of all arcs in graph $G_\pi^\wedge$ are calculated according to the metrics of the input graph G^u in order to guarantee that $dist_{UD}(s, t)$ is the same as $dist_I(s, t)$, thus also guaranteeing the correctness of the CCH algorithm. In other words, this means that to guarantee the correctness of the CCH, $dist_I(s, t) = dist_A(s, t) = dist_{UD}(s, t)$ must hold for all nodes s and t . A metric that satisfies $dist_I(s, t) = dist_A(s, t)$ is said to *respect* the input weights, and a metric is called *customized* if additionally $dist_A(s, t) = dist_{UD}(s, t)$ holds.

A respecting metric can be constructed trivially by assigning the metrics of all arcs of $G_\pi^\wedge$ which also exist in G^u as their input weight, and all additional arcs as $+\infty$. But constructing a customized metric is more complicated, which consists of two main steps: basic customization and perfect customization.

The goal of the basic customization step is to compute a customized metric m_C given a respecting metric. This is done by iterating over all arcs $uv \in G_\pi^\wedge$ ordered increasingly by the rank of u in a bottom-up fashion. For each arc uv , the algorithm enumerates all lower triangles $\{u, v, w\}$ of uv and checks whether $m_C(u, w) + m_C(w, v)$ is smaller than $m_C(u, v)$. If so, the algorithm decreases $m_C(u, v)$ so that both paths have equal lengths. Formally, it performs the operation $m_C(u, v) \leftarrow \min\{m_C(u, v), m_C(u, w) + m_C(w, v)\}$ for every arc uv .

The key observation of the basic customization algorithm is that because we only work with the lower triangles, the rank of w must be strictly smaller than the ranks of u and v . Therefore, when the algorithm considers the arc uv , the weights $m_C(u, w)$ and $m_C(w, v)$ are guaranteed to remain unchanged until the termination of the basic customization.

Afterward, the perfect customization step computes the perfect metric m_P given a customized metric m_C . The perfect metric is a customized metric where the weight of every arc uv in $G_\pi^\wedge$ is equal to the shortest path distance $dist_I(u, v)$. It starts with a customized metric m_C . It first copies all values of m_C into m_P . The algorithm then iterates over all arcs uv in decreasing order of the rank of u in a top-down fashion. For every arc, it enumerates all intermediate and upper triangles $\{u, v, w\}$ and checks whether the path over w is shorter, adjusting the value of $m_P(u, v)$ accordingly. It performs the operation $m_P(u, v) \leftarrow \min\{m_P(u, v), m_P(u, w) + m_P(w, v)\}$. After all arcs have been processed, m_P is the perfect metric, corresponding to the shortest uv -path distance for every arc uv . While both basic and perfect customization steps ensure that $dist_A(s, t) = dist_{UD}(s, t)$, the perfect customization step optimizes the metric to achieve a more optimal solution in terms of arc representation and efficiency.

For bidirected input graphs, two metrics are assigned per arc in the upward directed graph $G_\pi^\wedge$, each representing one direction, and customization is performed accordingly.

2.4 Phase 3: Query

The query algorithm of CCH is based on the elimination tree introduced by [34]. The goal is to find the shortest up-down path P in $G_\pi^\wedge$, which can then be unpacked into a shortest path in the original graph G .

The shortest path in $G_\pi^\wedge$ can be found with the help of two tentative distance arrays $dist_f(v)$ and $dist_b(v)$, which are called forward and backward tentative distances, respectively. We can employ a bidirectional Dijkstra's algorithm, *i.e.*, we run an instance of Dijkstra's algorithm from s and another one from t on the graph $G_\pi^\wedge$. We note x as the lowest common ancestor of the nodes s and t in the elimination tree, and r as the root node of the elimination tree. By iterating over all nodes y in the elimination tree which is on the path from s to x , from t to x , and from x to r , the algorithm relaxes forward and backward arcs and computes the node z that minimizes the sum of forward and backward tentative distances $dist_f(z) + dist_b(z)$. Knowing z allows the computation of the shortest path distance as well as the sequence of arcs that comprise the shortest path.

After determining the shortest up-down path, the algorithm needs to unpack the path to obtain the sequence of edges that form a shortest path in the original graph. Let $(p_1, \dots, p_k)$ be the up-down path found by the query. The algorithm iterates over the up-down path and checks for the existence of a lower triangle $\{p_i, p_{i+1}, x\}$ for each arc $p_i p_{i+1}$ with $m(p_i, p_{i+1}) = m(x, p_i) + m(x, p_{i+1})$. If such a triangle exists, the node x is inserted between p_i and p_{i+1} in the path. This process is repeated until all lower triangles are resolved and the final shortest path edge sequence is obtained.

It is proven that the ancestors of a node in the elimination tree are the nodes of the search space [9], which also means that each search requires relaxing arcs up to the common ancestor in the elimination tree, and that the running time of the exploration is only bounded by $\mathcal{O}(td^2)$, where td is the tree depth of the elimination tree [56]. A smaller tree depth leads to smaller search space, thus faster query time.

3 Extending CCH for Shortest Path Routing Computation

In this section, we propose Hierarchical Shortest Path Routing (HSPR), an adapted version of CCH for shortest path calculation in telecommunication networks, which extracts information about the topology during the preprocessing stage, allowing us to benefit from it for faster acquisition of shortest paths after each metric change.

3.1 Exploiting Telecommunication Networks Properties

The original CCH algorithm was primarily designed for use in real-world road networks. However, the topological characteristics of telecommunication networks differ significantly from those of road networks. For instance, large-scale telecommunication networks exhibit an approximate power-law relationship between the maximum node degree and the size of the network [43], whereas such a relationship is not commonly observed in road networks. Additionally, large-

scale telecommunication networks typically contain nodes with thousands of degrees, which is not possible in road networks.

This particular structure makes it easier to conceptualize alternative methods to generate elimination orders. We explored some simple heuristics that can benefit from extremely centralized structures, such as ordering by node degree or node centrality. The betweenness centrality is a measure of centrality based on shortest paths on a graph. The betweenness centrality of a node v is defined as

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from s to t , and $\sigma_{st}(v)$ is the number of shortest paths from s to t passing through v .

We also tried several more modern heuristics to generate the order, such as Inertial Flow[88], FlowCutter[95] and ExTREEm[98]. Inertial flow is a simple algorithm for calculating the minimum cut of a graph based on geometric properties. FlowCutter and ExTREEm are two more sophisticated heuristics, both using the original FlowCutter[56] algorithm. They won the second and the first place respectively in the PACE 2020 heuristic tree-depth implementation challenge. They both incorporated multiple heuristics to guarantee the performance on a large range of networks. Given that the preprocessing can be executed only once, it is practical to select the best elimination order among all approaches for subsequent queries.

Because the depth of a node in an elimination tree is the search space of the CCH query, the average depth of all nodes in the elimination tree can be a good indicator of the general performance of the query. Therefore, in our study, our goal is to find the elimination order that minimizes the average depth of all nodes, instead of focusing only on the minimum depth of the tree as in the original CCH algorithm, which aims to speed up only the worst query time.

The queries between these two types of networks are also different. Traffic engineering for telecommunication networks, such as the WEIGHT SETTING problem, usually already has a given traffic matrix, while this is not the case for typical road networks. Due to this difference, we use a many-to-one version of CCH where the intermediate computation results of the query phase can still be used within the same target node, which, according to our experiments, makes the query much faster.

3.2 Multiple shortest paths computation & ECMP rule

In order to correctly simulate shortest path protocols with ECMP, we had to extend CCH in the following ways

Multiple shortest paths Unlike the original CCH, the calculation of congestion using ECMP requires knowledge of all shortest paths instead of just the shortest distances or one of the shortest paths. As a result, all possible shortest paths need to be obtained during the query process, which necessitates the preservation of all shortcuts and the unpacking of these shortcuts during the query process. To accomplish this, we store all intermediate nodes in the query phase, and each time a new sub-path is added to the result, we must verify that no duplicate values

are present.

When unpacking the shortcuts during the queries, it is necessary to develop every sub-path and verify every triangle in order to correctly obtain all shortest paths between node pairs. Although these modifications result in increased computation during the query phase, which slows the execution, it is a necessary approach to guarantee the correctness of the short paths and it is still significantly faster than using Dijkstra's algorithm.

ECMP rule During the query phase, after all shortest paths P have been unpacked, we split the traffic volume d^k of a demand $k \in K$ according to the ECMP rule as follows. We begin with the source s^k of the demand, adding all the nodes in P to a priority queue. We set the residual degree of all the nodes as their corresponding in-degree. The priority value of a node is then equal to its residual degree. We set a so-called split factor value for every arc and node except s^k to 0. The split factor of s^k is set to 1.

ECMP is calculated in an iterative manner, popping nodes one by one from the priority queue. The node v with the smallest residual degree is popped first, then the residual degrees of all the out-neighbors are subtracted by one. The computation ends when the queue is empty. Let $0 \leq \alpha_v \leq 1$ be the split factor of v divided by its out-degree. The split factor of each out-arc (resp. out-neighbor u) of v belonging to P is then equal to α_v (resp. the split factor of u plus α_v). Finally, to determine the percent of traffic volume d^k that traverses an arc uv , we simply multiply d^k by the split factor of uv .

A recursive version of this algorithm was also implemented and tested but it was much less efficient than the above iterative version.

3.3 Further improvements

Caching packed paths To accelerate CCH, we add a cache to the path unpacking process. For a given set of metrics, many demands could traverse the same path while the unpacked shortest path remains the same. By saving the already packed paths to the cache, it significantly speeds up the process of shortcut unpacking, which is often the most time-consuming part for large instances. The cache is stored in a hash map, and is cleared after each modification of the metrics and regenerated during the computation.

Early stop In many heuristics in shortest path routing optimization, such as Local Search [51] or Monte Carlo Search, many simulations are performed to compare their corresponding congestion value in the network. In this case, one only needs to calculate the exact congestion value when necessary. Very often, when the newly found congestion value is already higher than the best known value, there is no need to continue the rest of the computation. Therefore, by ranking traffic in advance based on demand volume, we can stop the calculation when the current maximum congestion value is already greater than the current best value. This technique is useful for situations with a large amount of demands relative to the size of the network. However, in some cases during our experiments, large instances which have thousands of nodes are often accompanied by a relatively small number of demands, making this technique not suitable for

every situation. Despite this, early stop still shows its advantage for small and medium-sized instances which are less than a thousand of nodes.

Other heuristics for generating elimination tree We also explored various methods to reduce the elimination tree depth for telecommunication networks. We evaluated some state-of-the-art approaches, and proposed several simple heuristics as well. One of the heuristics, based on node degree, demonstrated remarkable results. It effectively reduced the elimination tree depth and outperformed some of the more complex methods. Our findings suggest that even simple heuristics can be highly effective in reducing the elimination tree depth for telecommunication networks, particularly when considering their special structure. This approach can serve as an alternative to the case where the more complicated approaches are not available.

4 Experiments and Results

In this section, we show the performance of HSPR on different families of network topologies of different sizes. The HSPR was implemented in C++ based on the open source implementation of CCH. The experiments were performed on a server with 64-core Intel(R) Xeon(R) Gold 5218 CPU with 125 GB of RAM. Only one CPU core was used for each execution.

4.1 Elimination Tree Depth

We tried different heuristics to generate the elimination tree and try to minimize the average tree depth of all nodes in the network.

The first heuristic is degree based. Nodes are ranked according to their degree, and this rank is the elimination order, on which we generate the elimination tree. We also used another heuristic according to betweenness centrality. The elimination tree of betweenness centrality is generated using the same way as the degree based elimination tree, *i.e.*, nodes are ranked according to the betweenness centrality.

Three other heuristics are also tested: Inertial Flow[88], FlowCutter[95] and ExTREEm[98]. Table 4.1 shows the average tree depth of all nodes in elimination trees using different heuristics. For Inertial Flow, FlowCutter and ExTREEm, we used the open source version, all executed 1 minute for each instance. For some technical issues, we don't have results on several instances using FlowCutter.

We can see that FlowCutter and ExTREEm perform best among all methods, outperforming other heuristics on almost all the instances. Due to the specific structure of high-degree nodes often found in telecommunication networks, degree-based heuristics also have encouraging results. Considering the simplicity of the node-degree-based approach, it is remarkable that it can generate such good results for tested networks.

In the subsequent experiments, we will use the results generated by ExTREEm for the best overall performance.

Table 4.1: Average Tree Depth of all nodes using different heuristics after 1 minute of execution.
 -: no solution found

instance	degree	BC	IF	FC	Ex
france	4.24	4.4	5.16	3.76	4.08
norway	8.15	8.26	6.37	5.96	5.93
brain	3.37	3.37	3.83	-	3.03
rand50	7.2	6.92	10.06	-	6.06
rand100	14.15	15.73	17.32	-	11.15
wax50	9.46	10.74	11.2	-	7.42
wax100	15.18	16.94	22.03	-	11.5
igen100	4.51	5.79	5.2	5.24	5.1
igen500	7.12	9.548	13.548	7.388	7.77
igen1000	13.98	15.295	13.884	11.559	11.4
igen2000	14.69	16.9635	12.7715	12.309	12.505
igen5000	14.554	16.3856	12.6388	11.798	12.088
inet3040	44.16	48.1	438.682	41.96	46.64
inet4000	62.19	68.56	574	56.23	56.96
inet5000	71.98	79.02	556	71.47	66.9

4.2 Congestion Calculation Time

Table 4.2 shows the execution time of various versions of HSPR, compared with Dijkstra. Congestion is computed with 10,000 different metrics for every network of SNDLib and Random/Waxman, 1000 metrics for igen100 and igen500, and 100 metrics for all other networks.

From the result we can find that HSPR with cache significantly outperforms the original version of HSPR and Dijkstra. If we also apply early stop, execution time on networks with relatively small traffic matrices are greatly improved, which is usually small instances in our case.

We also show the ratio of the execution time of the Dijkstra algorithm and HSPR in Figure 4.4, using the best version that we proposed on each network family. The number of calculations for each network is the same as in Table 4.2.

We can easily find that HSPR works remarkably well on all instances, even better on larger networks. Even in the worst case, HSPR can be about twice as fast as Dijkstra.

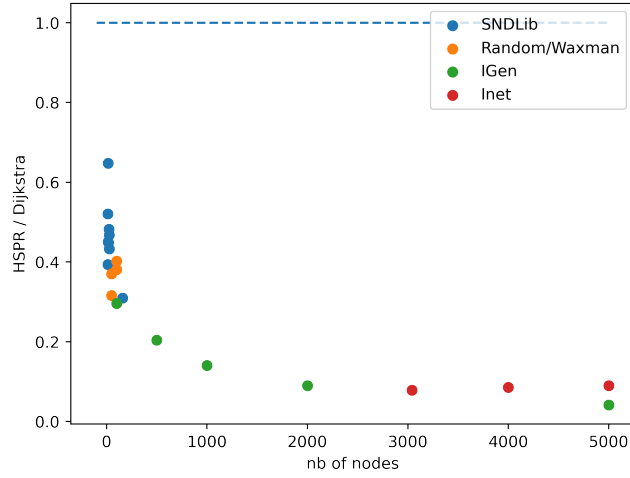


Figure 4.4: Comparison of execution time between HSPR and Dijkstra on all families of networks. The ordinate represents the ratio of execution time between HSPR and Dijkstra. Each point is an instance.

Table 4.2: Execution time of different versions of HSPR. The three tested versions are : HSPR with only multiple shortest paths computation (“HSPRv1”), HSPRv1+cache (“HSPRv2”) and HSPRv2+early stop (“HSPRv3”)

name	nb cal.	Dijkstra	HSPRv1	HSPRv2	HSPRv3
abilene	10,000	0.462s	0.244s	0.18s	0.182s
atlanta	10,000	0.781s	0.386s	0.388s	0.352s
newyork	10,000	1.116s	1.033s	0.731s	0.723s
france	10,000	1.526s	1.100s	0.819s	0.736s
brain	10,000	56.6s	20.636s	17.597s	17.521s
norway	10,000	3.55s	2.403s	1.598s	1.538s
nobel-us	10,000	0.382s	0.28s	0.259s	0.199s
nobel-eu	10,000	2.097s	1.404s	1.005s	0.94s
nobel-ger	10,000	0.571s	0.329s	0.273s	0.267s
rand50	10,000	18.22s	9.81	5.82	5.76
rand100	10,000	82.7s	69.27	32.77	31.51
wax50	10,000	18.17s	12.26	6.51	6.72
wax100	10,000	84.99	81.01	33.69	34.20
igen100	1,000	2.5s	1.5s	1.5s	0.74s
igen500	1,000	37.79s	16.3s	7.7s	8.1s
igen1000	100	13.34s	4.78s	1.88s	2.4s
igen2000	100	50.96s	12.09s	4.57s	6.51s
igen5000	100	337.63s	27.59s	14.1s	27.68s
inet3040	100	97s	13.67s	7.59s	10.54s
inet4000	100	192.41s	31.31s	16.43s	22.69s
inet5000	100	339.32	63.08s	30.57s	41.62s

5 Application to Weight Setting Problem

Monte Carlo Search has been applied to telecommunications network traffic engineering, which we have presented in Chapter 3. The details of NRPA and the modeling can be found in the same chapter as well.

In this section, we will use NRPA combined with HSPR and Dijkstra for solving the WEIGHT SETTING problem while comparing the congestion obtained with both approaches.

5.1 Experimental results

NRPA was implemented in C++ and the experiment environment is the same as in Section 4.

5.1.1 Convergence

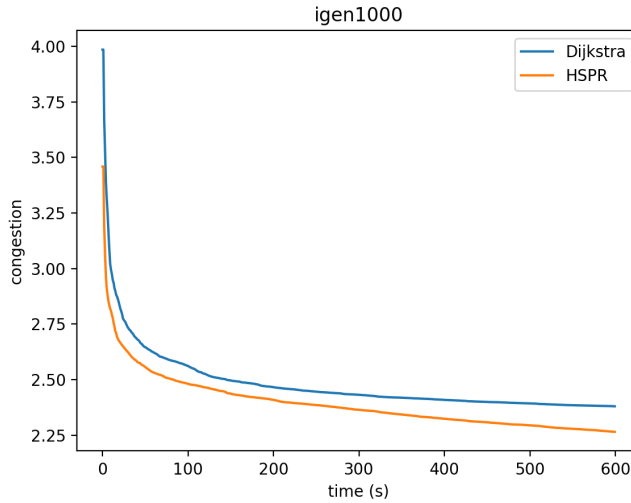


Figure 4.5: NRPA Convergence on Igen1000

Figures 4.5 and 4.6 show two examples of convergence on two large networks within the first 10 minutes of execution. They both show that the fast execution time of the shortest path computation can be translated into the fast convergence of NRPA. This is very intuitive because faster simulations mean that more iterations can be performed in the same amount of time, so NRPA can learn faster and get better results.

However, since the convergence rate of NRPA is logarithmic, even multiple times faster simulation time does not imply the same amount of improvement in congestion. Nevertheless, we still observe a very significant improvement in the congestion values.

5.1.2 Congestion value

We applied HSPR to NRPA and tested its effect on the congestion values obtained from NRPA. Metric space for all networks is set to $\{1, 2, 3, 4\}$. We used NRPA with level 2 and 50 iterations.

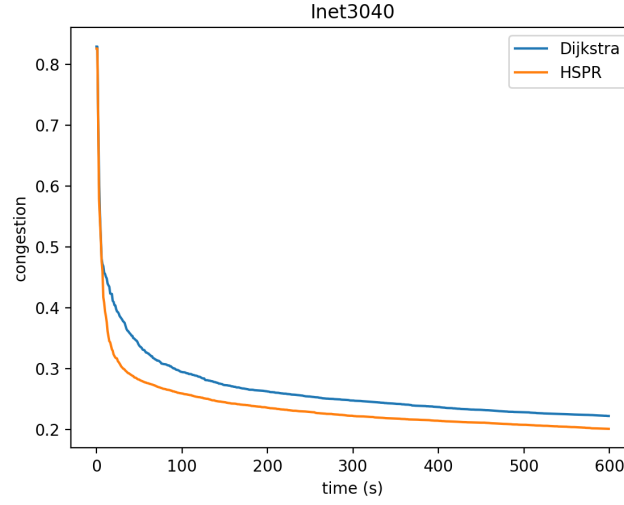


Figure 4.6: NRPA Convergence on Inet3040

During the execution, if one NRPA execution is finished, another NRPA is immediately started, until timeout. The congestion is thus the best congestion among all executions.

Table 4.3: Congestion value using level 2, iteration 100 NRPA with *force_exploration*. The results are normalized according to the lower bound obtained with linear programming.

name	Unit-OSPF	Dijkstra NRPA	HSPR NRPA
abilene	310.456	100.002	100.002
atlanta	149.85	101.63	101.63
newyork	168.895	120.89	120.66
france	170.95	106.85	105.89
norway	155.56	109.37	108.81
nobel-us	153.50	100.10	100.02
nobel-ger	143.16	100.41	100.39
nobel-eu	124.74	100.03	100.02
brain	156.70	100	100
rand50	142.33	104.46	102.34
rand100	168.03	106.75	104.45
wax50	140.75	101.90	101.42
wax100	116.04	100.05	100.01

Table 4.3 presents a comparison of the congestion values obtained by Dijkstra NRPA, HSPR NRPA, and Unit OSPF on SNDLib and Random/Waxman graphs. The results of Unit OSPF are obtained by setting all the metrics to 1, which serves as a baseline for the optimization. The congestion values are normalized according to the lower bound calculated by linear programming and averaged over 10 independent executions of 5 minutes.

The table shows that in all cases, the HSPR NRPA outperforms, or at least performs as well as, the Dijkstra NRPA. However, the improvement is insignificant for these small instances, where quasi-optimal results can be obtained quickly. Therefore, after five minutes of computation, there is limited room for congestion optimization, and the difference in results between

the two algorithms is subtle. However, for relatively large instances, the difference in results is much more significant.

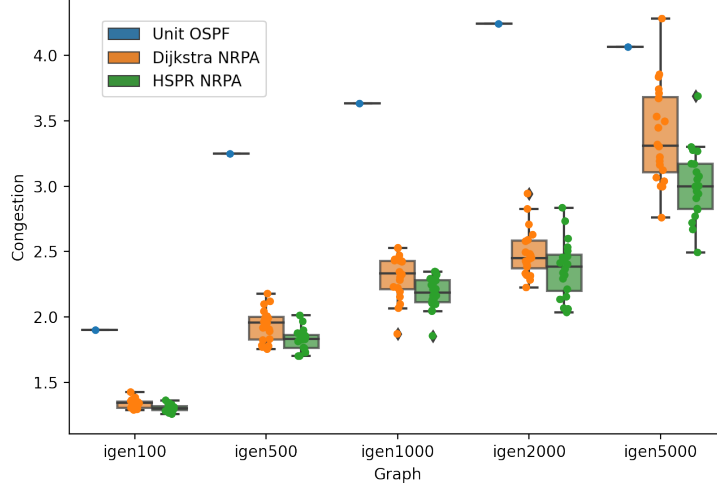


Figure 4.7: Comparison of congestion value between Unit OSPF, Dijkstra based NRPA and HSPR based NRPA on Igen instances.

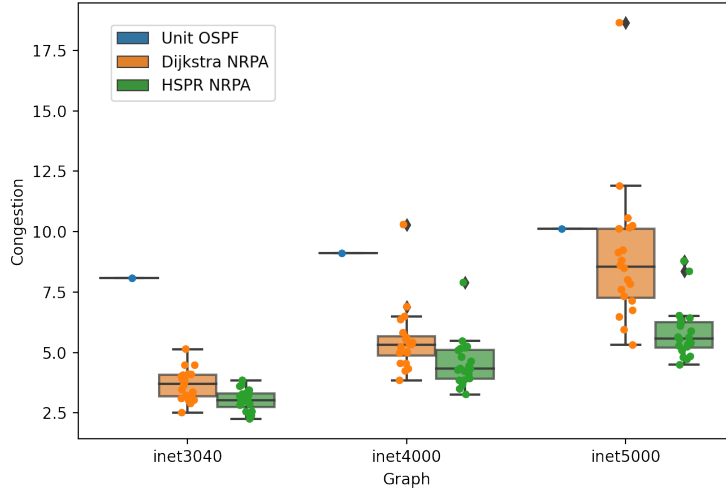


Figure 4.8: Comparison of congestion value between Unit OSPF, Dijkstra based NRPA and HSPR based NRPA on Inet instances.

Figure 4.7 and 4.8 show the comparison of the congestion values of Dijkstra NRPA and HSPR NRPA on 20 executions. Each execution is run for 2 hours, except for the instances igen100 and igen500 which is 30 minutes. Since the instances are large, the results are normalized according to the lower bound calculated by Fleischer's approximation scheme [47] with $\epsilon = 0.1$ if the input network has less than 3000 nodes and $\epsilon = 0.3$ otherwise.

We can clearly note that the HSPR NRPA is significantly better than the Dijkstra NRPA,

and it has a smaller variance of the congestion value. For large networks of 4000 or 5000 nodes, Dijkstra NRPA can sometimes have congestion worse than Unit OSPF even after 2 hours of execution, while HSPR NRPA can constantly perform better than Unit OSPF, and is always closer to the lower bound.

6 Conclusion

In this chapter, we introduced the Hierarchical Shortest Path Routing (HSPR) algorithm based on an optimized adaptation of the Customizable Contraction Hierarchy algorithm [34]. Our experiments show that HSPR is by orders of magnitude faster than Dijkstra for all the tested networks and especially for large scale networks. Furthermore, preliminary numerical results indicate that this improvement can be beneficial for algorithms that need to quickly evaluate shortest path routing solutions.

A natural extension of this work would be to apply HSPR for other shortest path routing optimization problems, such as those related to Segment Routing. In particular, the use of HSPR would clearly be beneficial when optimizing both the OSPF metrics and the segments as proposed in [75]. Finally, our algorithm can speed up the calculation substantially for obtaining solutions that are not only optimal in the nominal case, but also robust to multiple links/nodes failures.

Chapter 5

Warm Starting NRPA

Objectifs

In this chapter, we propose Meta-NRPA, which combines optimal stopping theory with NRPA for warm-starting and significantly improves the performance of NRPA. We also present several exploratory techniques for NRPA which enable it to perform better exploration. We establish this for three notoriously difficult problems ranging from telecommunication, transportation and coding theory, namely WEIGHT SETTING problem, Traveling Salesman Problem with Time Windows and Snake in the Box. We improve the lower bounds of the Snake in the Box problem for multiple dimensions. Meta-NRPA also achieved significantly better results on the WEIGHT SETTING problem. The content of this chapter is based on [32].

Contents

1	Introduction	66
2	The Secretary Problem	66
3	Warm Starting NRPA - Meta-NRPA	68
3.1	Exploratory NRPA	69
4	Experiments and Results	70
4.1	Weight Setting Problem	70
4.2	Traveling Salesman Problem with Time Windows	72
4.3	Snake in the Box	74
5	Conclusion	75

1 Introduction

As presented in Chapter 3, search algorithms can be used to solve many difficult combinatorial optimization problems. Monte Carlo Search algorithms rely on randomness to discover good sequences of decisions for difficult problems. Following their success in games [30, 64, 15, 91, 93, 94], they were applied with success to multiple combinatorial optimization problems [19, 85]. They work particularly well when combined with machine learning.

In this chapter, we propose new general techniques to Monte Carlo Search algorithms that improve the algorithms for multiple applications. Using these techniques we get better results than the previous ones for three difficult combinatorial optimization problems from varied fields, namely telecommunication, transportation and coding theory.

The problems we address in this chapter were already successfully addressed using Nested Rollout Policy Adaptation (NRPA) [85]. They are the WEIGHT SETTING problem from Chapter 3, the Traveling Salesman Problem with Time Windows [36] and the Snake-in-the-Box problem [35]. For these three problems we improve the results compared to standard NRPA. In particular for the Snake-in-the-Box problem, we provide new lower bounds for several dimensions.

Our contributions deal with the initialization of NRPA using the optimal stopping theory and with better exploration avoiding already scored sequences of decisions. More precisely, during the experiment with NRPA, we observed that whenever NRPA gets unsatisfactory solutions from the beginning, it is highly unlikely that it will find significantly better solutions in subsequent iterations.

To remedy this problem, we cast the initialization step of NRPA as an instance of the secretary problem, a well-known optimal stopping problem: a decision maker (DM) wants to recruit a secretary for a job position, n candidate secretaries are thus interviewed one after the other in a random order, which can be ranked among the examined candidates. The DM can decide whether to terminate the recruitment process by accepting the last interviewed candidate. The decision of the DM about recruiting a candidate needs to be just after the interview of the candidate and it is irrevocable. In addition, the DM has no knowledge of the quality of the upcoming candidates. The goal is to maximize the probability of selecting the best candidate. In our case, we are interested in a variant of this problem, where candidates are NRPA runs and we aim at minimizing the expected rank of the chosen candidate.

To our knowledge, this is the first time that the optimal stopping theory is used in the context of Monte Carlo search.

2 The Secretary Problem

The secretary problem, which is also known as the marriage problem, the sultan's dowry problem, or the best choice problem, is a classical problem in optimal stopping theory. It was first introduced in the early '60s, but there is some obscurity in its origin. It is commonly agreed that [67] firstly proposed a solution in a scientific journal. Later, more variations of the problem are proposed [54].

The classic secretary problem models an idealized situation in which an individual must choose the best candidate out of a pool of applicants, given that you can only evaluate one candidate at a time and must make an immediate decision to either accept or reject the candidate after each evaluation. Once a candidate is rejected, you cannot go back and reconsider them. The objective is to maximize the probability of choosing the best candidate among all the applicants.

The description of the problem is as follows: a decision maker (DM) wants to recruit a secretary for a job position, n candidate secretaries are thus interviewed one after the other in a random order, which can be ranked among the examined candidates. We assume that the ranks are distinct and randomly ordered. Without loss of generality, let's assume the best candidate has a rank of 1, the second-best candidate has a rank of 2, and so on. The DM can decide whether to terminate the recruitment process by accepting the last interviewed candidate. The decision of the DM about recruiting a candidate needs to be just after the interview of the candidate and it is irrevocable. In addition, the DM has no knowledge of the quality of the upcoming candidates.

The classical secretary problem attempts to maximize the probability of selecting the best candidate. Let X_i represent the rank of the i -th candidate in the sequence, where $X_i \in \{1, 2, \dots, n\}$ and $X_i \neq X_j$ for every $i \neq j$. The best candidate is the one with rank 1. The problem seeks an optimal strategy that maximizes the probability of selecting this candidate.

In our case, we don't focus on maximizing the probability of selecting the best candidate. Instead, we are interested in a variant of this problem, which aims to minimize the expected rank of the chosen candidate. In [29], it is showed that the optimal stopping rule leads to a optimal expected rank of $\prod_{j=1}^n (\frac{j+2}{j})^{1/j+1} \cong 3.8695$ for $n \rightarrow +\infty$, which can be determined by dynamic programming. In practice, this is hard to implement for large n . In [65], the authors proposed much simpler stopping rules that perform almost equally well in minimizing the sum of the expected rankings of the chosen items when one or more candidates are desired.



Figure 5.1: The illustration of the secretary problem

When only one candidate is needed, with stopping rule $R_i \leq \frac{ic}{n+1-i}$ for a given constant c , where R_i is the relative rank of the i th item and n the total number of items, we can have an asymptotic expected absolute rank of 3.928 for the chosen candidate. When multiple candidates are to be retained, we are interested in the case where a given percentage of candidates are desired. For a fixed given percentage α , $0 < \alpha < 1$, the item j is kept if it satisfies the condition $R_i \leq \lceil \alpha i \rceil$, which means we keep only candidates with a relative rank inferior or equal to the upper value of the round of αi . With this simple rule, we have $\frac{S_n}{nL_n} \rightarrow \frac{\alpha}{2}$ a.s. as $n \rightarrow +\infty$, where L_n is the number of candidates retained, S_n the sum of the absolute ranks of the candidates retained.

3 Warm Starting NRPA - Meta-NRPA

Inspired by the secretary problem, we hereby present Meta-NRPA, which warm-starts the NRPA with optimal stopping theory.

When dealing with difficult combinatorial optimization problems with large instances, sometimes the variance between executions of NRPA can be very important. After being stuck in the local minimum, it is difficult to escape to a better solution. If NRPA is optimized towards a better local minimum from the beginning, we can easily get better results at the end of the execution. On the contrary, if it gets unsatisfactory results from the beginning, there is only a small probability that it will get better results in subsequent executions than in other executions (see for instance Figure 5.3).

From this observation, we have the idea of warm-starting the executions of NRPA by selecting those that lead to better results and discarding those that do not. The choice is made according to the performance of a small fraction of the NRPA runs at the beginning. If it satisfies the criteria of the optimal stopping, the NRPA is continued to its full execution, otherwise, the current execution is abandoned and a new NRPA is executed.

When there is not enough time, only several complete level ℓ NRPA executions can be performed, and the optimal stopping rule with only one chosen candidate is used. In practice, we execute NRPA with level $\ell - 1$ and compare its score with existing scores. If the relative rank R_i of the i -th NRPA satisfies $R_i \leq \frac{ic}{n+1-i}$, we continue the current NRPA with level ℓ ; otherwise, it is abandoned and another NRPA with level $\ell - 1$ is performed and compared, until the timeout. The algorithm is shown in Algorithm 9. The function *ContinueNrpa* executes a normal NRPA, but with the best score and the best sequence of previous NRPA run of level $\ell - 1$.

Algorithm 9: Meta-NRPA with one item

Function MetaNrpa(*level*):

```

    scoreList  $\leftarrow$  empty ordered list
     $i \leftarrow 0$ 
    while not timeout do
        weights  $\leftarrow$  InitPolicy()
        score, seq  $\leftarrow$  NRPA(level - 1, weights)
         $R_i \leftarrow$  scoreList.insert(score)
         $i+ = 1$ 
         $n \leftarrow$  NbExecutionEstimation()
        if  $R_i \leq \frac{ic}{n+1-i}$  then
            score  $\leftarrow$  ContinueNrpa(level, weights)
            scoreList.insert(score)
             $i+ = 1$ 
        end
    end
    return min(scoreList)

```

When many executions of the NRPA are possible within the time limit, we tend to select a

Algorithm 10: Meta-NRPA with $\alpha\%$ items

```

Function MetaNrpa(level):
  scoreList  $\leftarrow$  empty ordered list
  i  $\leftarrow$  0
  while not timeout do
    weights  $\leftarrow$  InitPolicy()
    score, seq  $\leftarrow$  NRPA(level - 1, weights)
     $R_i \leftarrow \text{scoreList.insert}(\text{score})$ 
    i + = 1
    if  $R_i \leq \lceil \alpha i \rceil$  then
      score, seq  $\leftarrow$  ContinueNrpa(level, weights)
      scoreList.insert(score)
      i + = 1
    end
  end
  return min(scoreList)

```

certain percentage of executions to continue, as this will allow the NRPA to be optimized more deeply in a variety of different directions. The algorithm is shown in Algorithm 10. Figure 5.2 illustrates the process of this version of Meta-NRPA with level 2.

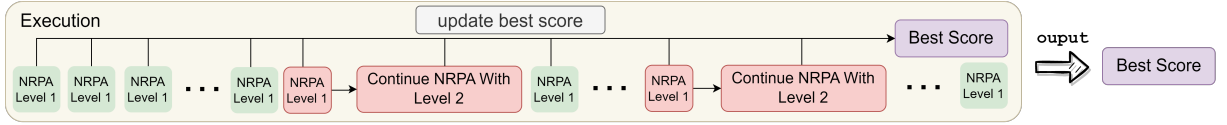


Figure 5.2: Meta-NRPA. We execute NRPA with level $\ell - 1$ and compare its score with existing scores. If the relative rank R_i of the i -th NRPA satisfies a certain condition, we continue the current NRPA with level ℓ ; otherwise, it's abandoned and the next NRPA with level $\ell - 1$ is performed and compared, until the timeout. The final score of Meta-NRPA is the best score among all executions of NRPA.

Note that the assumption we apply here is that NRPA that performs well at the beginning will also provide better results in the later optimizations. This is not always true, but statistically speaking, this assumption holds most of the time. And the larger the fraction of NRPA we run, the more likely this assumption will hold.

Using a larger fraction of the NRPA strengthens this assumption, but it also takes more time, so we get fewer candidates in a fixed period and therefore risk losing the opportunity to get better candidates. In fact, running only the inferior level of NRPA in our tests has roughly shown a strong correlation with the final NRPA results and has been able to obtain satisfactory results.

3.1 Exploratory NRPA

Since NRPA tends to converge quickly and has difficulty getting away from local minima, we also use several techniques to help NRPA do more exploration. These techniques do not work for every problem, but in our tests, they help a lot for one or many problems.

3.1.1 Force Exploration

force_exploration was introduced in Chapter 3 and greatly improves the performance of NRPA by performing random selection when an already discovered solution is encountered. This simple mechanism encourages exploration while maintaining the original structure and strategy learning of NRPA.

3.1.2 ϵ -greedy NRPA

When the NRPA is executed after some time, the NRPA may converge to some specific solution and keep exploiting the same solution, which does not help to escape the local minima. Inspired by the ϵ -greedy agent in multi-armed bandit problem [96], we propose here ϵ -greedy NRPA. ϵ -greedy NRPA uses the same idea as a ϵ -greedy agent: each choice is made with uniform probability for a probability of ϵ , while following the learned policy of NRPA with a probability of $1 - \epsilon$. It allows NRPA to keep exploring at any stage of the execution, giving better performance for long-running executions.

3.1.3 Dynamic Learning Rate

Without finding a better solution, NRPA continues to enforce the policy on the best-known sequence at the current level, which can easily lead to a rapid convergence to a sub-optimal value. As a result, the exploration of other options decreases, which makes it more difficult to find better results.

To address this problem, we propose a dynamic learning rate that reduces the learning rate of the NRPA if it has difficulty finding a better solution. In this way, NRPA still converges quickly towards the best-known solution at the beginning but slows down when another better solution is not found, thus encouraging exploration. The modified learning rate α_{new} is defined as:

$$\alpha_{new} = \frac{\alpha}{N_{stagnant}}$$

where α is the default learning rate, $N_{stagnant}$ is the number of times that the playout result is not better than the best result at the current level.

4 Experiments and Results

In this section, we test the efficiency of our approach on the following three notoriously NP-hard optimization problems. The experiments were performed on a server with 64-core Intel(R) Xeon(R) Gold 5218 CPU with 125 GB of RAM. Only one CPU core was used for each execution.

4.1 Weight Setting Problem

The WEIGHT SETTING problem is a renowned problem for its complexity, especially for real telecommunication networks. Chapter 3 demonstrates in detail the modeling of this problem

with various Monte Carlo Search algorithms. In this section, we use the same modeling with NRPA and meta-NRPA, and compare their efficiency in solving this problem. We define the score used in NRPA to evaluate a solution $w \in \mathbb{Z}_+^{|A|}$, where $|A|$ is the number of arcs in the network, as the congestion value induced by w , that is:

$$\text{score}(w) = \text{cong}(w)$$

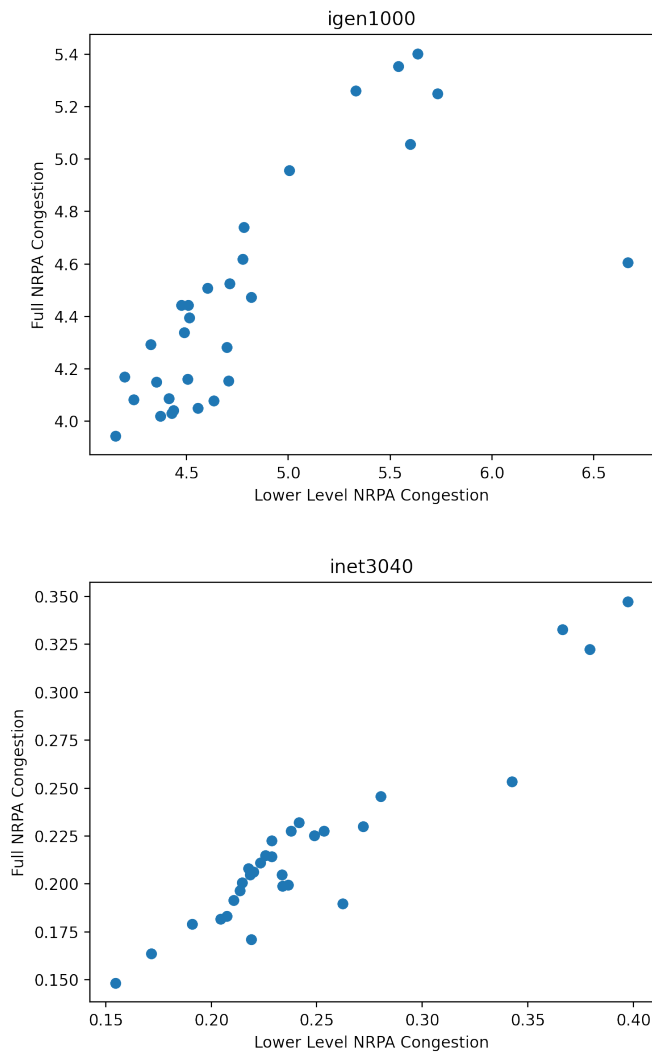


Figure 5.3: Correlation between the congestion values of the lower level NRPA and the full NRPA of 30 independent executions on two generated telecommunication networks igen1000 and inet3040. The full NRPA execution is the continuation of the lower level NRPA.

Figure 5.3 shows an example of the correlation of the congestion values between the 30 independent runs of a small fraction of the NRPA executed in Meta-NRPA and the congestion value of the full NRPA execution. This confirms the earlier assumption that, in general, NRPAs that perform better at the beginning provide better results when they are fully executed.

Figure 5.4 shows the comparison between the congestion values obtained with NRPA and

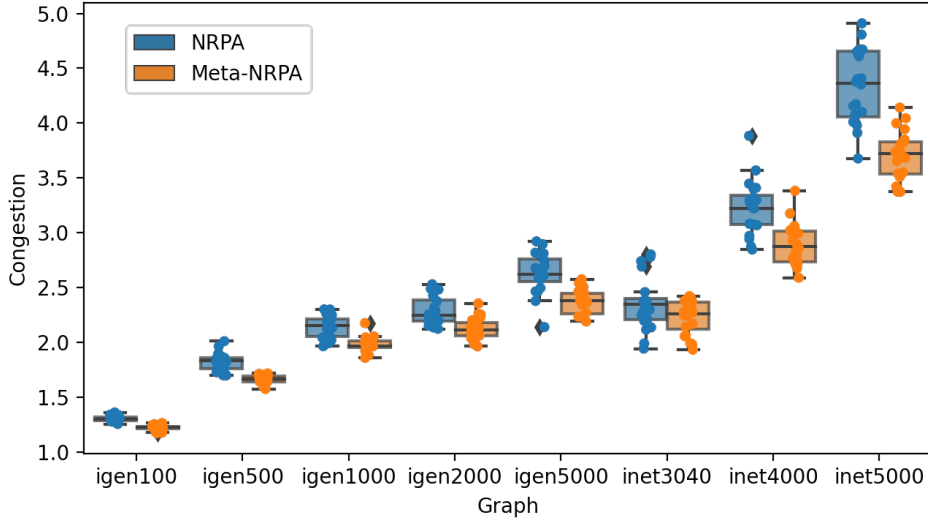


Figure 5.4: Congestion value comparison of 20 independent executions of NRPA and Meta-NRPA on large generated telecommunication network instances

Meta-NRPA with one item. Our implementation of NRPA is similar as in Chapter 3 except that we use the Customizable Contraction Hierarchy (CCH) algorithm introduced in Chapter 4, instead of Dijkstra, for improving the computation time of the shortest paths and, hence, the congestion value.

Since each playout takes a long time, we use NRPA with a level of 2 and 50 iterations. Each method executes 20 independent runs on each instance, the results are normalized according to the lower bound calculated by Fleischer’s approximation scheme with $\epsilon = 0.1$ [47]. Instances having more than 400 nodes are executed for 2 hours, others for 30 minutes. Due to the large size of the instances, the NRPA already has difficulty converging, *i.e.*, much more time is needed for convergence than the time we executed. For this reason, we do not use ϵ -greedy or dynamic learning rate, as these are exploratory techniques which help slow the process of convergence for better solutions.

The results clearly show that Meta-NRPA can help to reduce the congestion values on all instances and can lead to a reduction in the variance between executions.

4.2 Traveling Salesman Problem with Time Windows

In the Traveling Salesman Problem (TSP), a set of n cities (one of which is the depot) and their pairwise distances are given. The task is to find the shortest route that starts and ends at the depot and visits each city only once. In the Traveling Salesman Problem with Time Windows (TSPTW), in addition to the conditions of TSP, each city must be visited and left within a given time interval, which is much more difficult TSP.

The TSPTW can be formulated as follows. A graph $G^u = (V, E)$ represents the cities and the corresponding transportation network, where the nodes $V = \{v_1, v_2, \dots, v_{|V|}\}$ corresponds to the cities, among which the node v_1 corresponds to the depot, and $E = V \times V$ represents the edges between nodes. The distance between two cities v_i and v_j is also the cost function $c(v_i v_j)$.

We aim to minimize the total cost function:

$$\text{cost}(P) = \sum_{k=1}^{|V|} c(p_k p_{k+1})$$

where $P = (p_1, \dots, p_{|V|+1})$, $(p_1, \dots, p_{|V|})$ is a permutation of $\{v_1, v_2, \dots, v_{|V|}\}$ and $p_{|V|+1} = p_1 = v_1$.

Apart from the distance cost, a time window is also added as a constraint. For every city v_i , we have a time interval $[e_i, \ell_i]$, which means it must be visited before ℓ_i , but if it is visited before e_i , we must wait until e_i to depart. We can set every violation of the constraint as a penalty of one million in the score. So the final score of path P is:

$$\text{score}(P) = -\text{cost}(P) - 10^6 \times \Omega(P)$$

where $\Omega(P)$ is the number of violated constraints.

NRPA-based algorithms have been used on this problem and have achieved good results [20].

We test our algorithms on rc204.1, which is the most difficult instance in the Solomon-Potwin-Bengio TSPTW benchmark. We use NRPA of level 4 and 100 iterations. The inverse distance between every two cities is used to initialize the NRPA policy, which means cities closer to the current cities are more likely to be chosen as the next stop. In our test, this bias in policy shows better performance than the commonly used uniform default policy. The learning rate of NRPA α is set to 0.01 to avoid premature convergence.

Table 5.1: Average score of 30 independent executions of Exploratory NRPA with TSPTW on rc204.1 instance. NRPAv1: NRPA + force exploration, NRPAv2: NRPA + force exploration + greedy, NRPAv3: NRPA + force exploration + dynamic alpha, NRPAv4: NRPA + force exploration + dynamic alpha + greedy

T (min)	NRPA	NRPAv1	NRPAv2	NRPAv3	NRPAv4
1	-30914.67	-10913.89	-910.06	-893.14	-891.30
5	-30914.45	-7580.00	-901.19	-893.11	-888.92
10	-30914.45	-7579.77	-897.30	-893.11	-887.54
30	-30914.12	-7579.48	-892.33	-893.09	-886.80

Table 5.1 shows the comparison of different versions of exploratory NRPA. Each result is the average score of 30 independent executions. We can see the original NRPA struggles to make sure every solution satisfies the constraints, while *force_exploration* greatly improved the performance of NRPA. With ϵ -greedy and dynamic learning rate, NRPA achieves much better performance.

Table 5.2 shows the corresponding Meta-NRPA of every version of exploratory NRPA shown in Table 5.1. Because NRPA takes a relatively long time on each execution, we use Meta-NRPA with 30% items. We can see that for every version, Meta-NRPA significantly improves the score. Moreover, meta-NRPA with all the exploratory techniques achieves the best result among all variants.

Table 5.2: Average score of 30 independent executions of Meta-NRPA with TSPTW on rc204.1 instance

T (min)	Meta-NRPA	Meta-NRPAv1	Meta-NRPAv2	Meta-NRPAv3	Meta-NRPAv4
1	-4241.87	-908.41	-903.19	-892.37	-888.03
5	-901.01	-903.69	-895.45	-887.53	-884.73
10	-896.53	-898.84	-892.93	-884.71	-883.00
30	-890.84	-890.49	-887.15	-881.68	-881.22

4.3 Snake in the Box

The Snake in the Box problem is that of finding the longest induced path in an n -dimensional hypercube, which is a special case of the induced subgraph isomorphism problem. The path starts at one node of the hypercube and moves as far as possible along the edge to a neighbor node. Each time it moves one step to a new node, the previous node, and its neighboring nodes become unavailable and can never be traveled to. The detailed description of the problem can be found in [1]. The papers [77], [74], [104] and [61] respectively gave the records on this problem on dimensions 7, 8, 9 and 10, and [2] has the latest record on dimensions 11, 12, and 13. The detail of these records is shown in Table 5.4 along with our best result.

In this problem, we define the score used in NRPA to evaluate a path P as the number of edges of P :

$$\text{score}(P) = |P|$$

Since, by definition, whenever a node is visited all its neighbors are unavailable, it is preferable to visit nodes with fewer available neighboring nodes to increase the chance of finding a long path. A prior is thus added to prefer the nodes with fewer available neighbors.

Table 5.3: Average score of 20 independent executions of 10 minutes for the Snake-in-the-Box problem. NRPAv1: NRPA + force exploration, NRPAv2: NRPA + force exploration + greedy, NRPAv3: NRPA + force exploration + dynamic alpha, NRPAv4: NRPA + force exploration + dynamic alpha + greedy

d	NRPA	NRPAv1	NRPAv2	NRPAv3	NRPAv4	Meta-NRPAv2
8	95.55	97.00	97.00	97.00	97.00	97.00
9	175.55	185.30	186.40	182.40	184.90	186.55
10	333.60	350.50	357.25	352.50	355.90	358.65
11	612.75	628.05	668.05	647.75	597.00	668.35

Table 5.3 compares the performance of NRPA. Each result is the average score of 20 independent runs of 10 minutes. We also used the best sequence of the lower dimension as primers for the NRPA. Each playout will first apply the primer, and try to extend the sequence. With *force_exploration*, NRPA has the possibility of changing the primer after finding that the current sequence has already been discovered. We use NRPA with level 4, 100 iterations, Meta-NRPA with 10% items, and 5% for ϵ -greedy, 0.01 for learning rate α .

For this problem, NRPA with ϵ -greedy and dynamic learning rate have a similar effect of improving the score, but ϵ -greedy performs significantly better than dynamic learning rate.

Using all two techniques leads to worse performance than using only ϵ -greedy as it would cause over-exploration. From the result, we can see that NRPA with *force_exploration*, ϵ -greedy, and Meta-NRPA has the best performance among all variants.

We also used another slightly different approach to try to achieve a new record for this problem. At each execution, the current record in each dimension is used as a primer. We only used Meta-NRPA with *force_exploration* and did not use ϵ -greedy or dynamic learning rate.

As a result of the *force_exploration*, Meta-NRPA will apply the best sequence before changing one random choice in the sequence to another random valid value and following the policy for the rest of the choices. We run 20 executions in parallel over 30 minutes. Each time we have a new record, it is used as a new primer, and a new execution is performed. We kept the experiment going until we could not get better results.

Table 5.4: Comparison with known lower bounds on the Snake-in-the-Box

Dimension	Meta-NRPA with force exploration	Best Known Score
7	50	50
8	98	98
9	190	190
10	373	370
11	721	712
12	1383	1373
13	2709	2687

Table 5.4 shows the comparison of our result of dimension 7 to 13 and the best known record for these dimensions [77], [74], [104] and [61]. New records are achieved for dimension 10 to 13, showing the advantages of Meta-NRPA.

5 Conclusion

In this chapter, we introduced the Meta-NRPA algorithm along with several exploratory techniques that substantially improve the performance of NRPA. Specifically, by utilizing optimal stopping theory and encouraging exploration while keeping the learning mechanism of NRPA, we achieved better results for three challenging problems from various fields.

Our Meta-NRPA approach improved the lower bounds for the Snake-in-the-Box problem and significantly enhanced the performance of NRPA on the WEIGHT SETTING problem. The techniques we proposed represent general improvements to NRPA and could be applied to many other difficult combinatorial optimization problems.

Chapter 6

Waypoint Optimization Problem

Objectifs

This chapter is dedicated to the WAYPOINT OPTIMIZATION problem, which arises in the context of the Segment Routing Traffic Engineering (SR-TE). We propose different approaches based on Reinforcement Learning and Local Search and compare them with state-of-the-art methods. Additionally, this chapter discusses some essential techniques for reducing the search space for the same problem and their practical applications. We also briefly discuss the joint optimization of the WEIGHT SETTING problem and the WAYPOINT OPTIMIZATION problem.

Contents

1	Introduction	78
2	Waypoint Optimization Problem	78
	2.1 Introduction	78
	2.2 State of the Art	80
	2.3 Problem Formulation	81
3	Our Contributions	82
	3.1 Reinforcement Learning	82
	3.2 Local Search: GreedierWPO	84
	3.3 Reduction of Search Space	86
	3.4 Joint Optimization	89
4	Experiments and Results	90
	4.1 Comparison of Reinforcement Learning Approaches	91
	4.2 GreedierWPO	93
	4.3 Comparison Between Approaches	97
	4.4 Comparison with Other Approaches	99
	4.5 Joint Optimization	100
5	Conclusion	102

1 Introduction

With the ever-increasing demand for high-quality and efficient communication services, modern telecommunication networks must be able to adapt and optimize their performance in real-time. Segment Routing (SR) is a promising technology that enables flexible and efficient traffic engineering by encoding routing instructions directly in the packet header. The WAYPOINT OPTIMIZATION (WPO) problem arises in the context of Segment Routing Traffic Engineering (SR-TE), and has gained significant attention in recent years due to its potential to further optimize network performance.

The WPO problem is a central problem in Traffic Engineering (TE), because it allows network operators to get more fine-grained control of the individual demand and optimize network performance. By placing waypoints strategically, it is possible to minimize network congestion, improve resource utilization, and enhance the overall quality of service provided to end-users. As telecommunication networks continue to evolve and grow in complexity, developing efficient solutions to the WPO problem becomes increasingly critical to maintain high-quality communication services and meet the ever-growing demands of users.

This chapter is dedicated to the WPO problem and explores various approaches to tackle it. We propose novel methods based on Reinforcement Learning and Local Search, comparing their performance with state-of-the-art techniques. The chapter also discusses essential techniques for reducing the search space for the WPO problem and their practical applications, enabling more efficient solutions.

In this introductory section, we provide the background and motivation for addressing the WPO problem. Section 2 delves into the WPO problem, presenting the state-of-the-art and providing a formal problem formulation. Sections 3.1 and 3.2 introduce our proposed reinforcement learning and local search-based approaches, respectively. In Section 3.3, we discuss techniques for reducing the search space to enhance the efficiency of the algorithms. Section 3.4 briefly discusses the joint optimization of the WEIGHT SETTING problem and the WPO problem. Section 4 presents experimental results and a comparison with existing methods. Finally, Section 5 concludes the chapter, summarizing the findings and outlining future research directions.

2 Waypoint Optimization Problem

2.1 Introduction

SR is a relatively new approach to network routing that offers a lot of flexibility and scalability [7, 45]. With traditional routing, packets are sent through the network by following a path determined by each router, in what is called hop-by-hop forwarding. But with SR, a path is predetermined by using a set of labels, which is also called segments, that specify the route that packets should take through the network. The labels are stored as a stack in the packets, and are popped out each time the packets reach the target router or link.

One of the great advantages of SR is that it provides more control in choosing the routing paths compared to routing protocols solely based on shortest path routing. Another technique

called Multiprotocol Label Switching (MPLS) [44] also allows the packets to route over non shortest paths, but it is difficult to scale [105], contrary to the SR.

There are two primary flavors of SR: Segment Routing over MPLS (SR-MPLS) [7] and Segment Routing over IPv6 (SRv6) [45]. While the specific implementation details of these two flavors are important for understanding their operation in real-world networks, our primary focus lies in the algorithmic abstraction to address the optimization problem in SR.

The SR-TE addresses the optimization challenges encountered when implementing SR to enhance network performance, reliability, and resource utilization. SR-TE focuses on determining the most efficient data transmission paths within a network using SR, while adhering to various constraints, including latency, link utilization, and load balancing. To steer the traffic in the network, an ordered list of waypoints are distributed to each demand. Thus, the waypoint list plays an important role in SR-TE.

One specific optimization problem that arises in this context is the WPO problem. In the literature, this problem is also often referred to as the SEGMENT ROUTING TRAFFIC ENGINEERING PROBLEM (SRTEP) [57]. In this thesis, we will use the term WPO to refer to this problem. We refer to waypoints as the points of detour for demands in SR. The goal of WPO is to pinpoint an optimal list of waypoints for routing demands across the network, aiming to minimize congestion and ensure a high level of Quality of Service (QoS) simultaneously. To achieve this goal, WPO algorithms must consider various network factors to calculate the most efficient routing paths for every demand. By optimizing waypoint selection, WPO aims to enhance the overall network performance by balancing load distribution, reducing congestion, and improving resource utilization.

Figure 6.1 shows an example of the SR for reducing network congestion. This example shows the flexibility of the SR by manipulating the route of single demand, providing us a great tool for network traffic optimization.

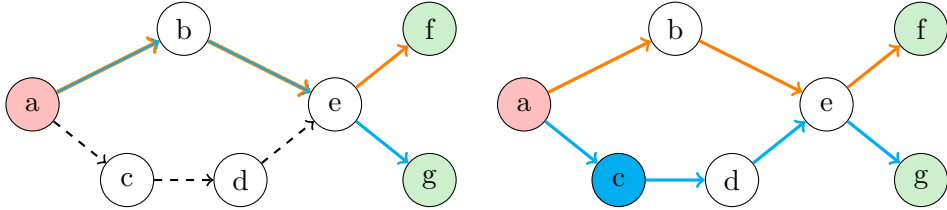


Figure 6.1: Illustration of the WAYPOINT OPTIMIZATION problem. In this figure, we assume unit capacities and unit metric for network arcs. We also suppose that two demands with unit demand volume $d^k = 1$ must be routed from node a to f and from node a to g , respectively. The pink, green and cyan nodes represent the source, target and waypoint, respectively. On the left, without SR, both demands pass through arc ab and be , causing a higher congestion value. While on the right, using SR, by adding node c as the waypoint for demand from a to g , the demands are separated evenly between node a and e , thus leading to a smaller congestion value.

2.2 State of the Art

The WPO problem, which is also referred to as the General Segment Routing Problem in [58], was proved NP-hard by a reduction from the well known partition problem [58].

The number of waypoints plays an important role in the WPO problem. The larger the number of waypoints, the larger the search space becomes, making it more difficult to find a good solution. It was found in [100] from experimental results that one waypoint per demand is sufficient for good performance of congestion minimization.

Various solution methods, such as linear programming, mixed-integer linear programming, and heuristic/meta-heuristic algorithms, can be employed to solve this problem. A recent survey [103] on SR-TE can be found to introduce various methods to solve this problem, as well as other important problems in the domain. In [57], the authors used Mixed Integer Programming (MIP), Local Search, and Constraint Programming to solve the WPO problem, which is referred to as SRTEP. In [5], the authors studied in depth the SR-TE, especially on the algorithmic aspect using Linear Programming (LP).

While existing works have shown promise in solving instances of a certain size, they are often not suitable for solving large instances. These large instances can have hundreds or even thousands of nodes, which make the WPO problem computationally challenging to solve. Many heuristics have been proposed to solve the problem at scale, and most of them use local search. In [52], [71], and [75], the authors proposed their versions of local search, with different focuses that we will briefly present here.

In [52], SR-TE at an extremely small timescale is investigated, addressing a crucial need for modern wide area networks. Existing approaches based on integer linear programming and constraint programming have not thoroughly examined the speed at which these methods can react to unforeseen traffic changes and failures. The proposed solution employs a local search-based SR-TE technique to minimize network congestion. The local search moves are designed for efficient neighborhood exploration, eliminating moves that do not contribute to congestion reduction. Additionally, the authors employ a stochastic heuristic algorithm to simplify neighborhood exploration, focusing on areas most likely to improve the current solution. Experimental results reveal that the proposed solution demonstrates high performance in the context of time-constrained optimization.

In [71], the authors propose Integer Linear Programming (ILP) models and heuristics to assess the TE performance of SR-based packet networks. Their findings indicate that the standard SR behavior, which leverages Equal Cost Multiple Paths (ECMP), can result in increased network resource utilization compared to cases where ECMP is not utilized. This study also presents a local search-based heuristic solution for minimizing congestion. A heuristic algorithm encodes a list of all possible paths between all source and destination pairs as a segment list, with the corresponding waypoints which satisfy the single-path constraint. The local search algorithm then iterates each link to find an alternative segment list to decrease the congestion. The algorithm is done in an iterative way. The results show that the proposed local search algorithm can efficiently reduce the network congestion in sub-second time.

In [75], the authors address the advantages of jointly optimizing link weights and waypoints

Algorithm 11: GreedyWPO [75]

```

input: network instance  $G = (V, A)$ , a set of demands  $K$ 
output: waypoint setting  $\mathcal{W}$ 
Let  $\mathcal{W}$  be an empty set of waypoints
Let  $cong_{min}$  be the network congestion with respect to  $G$ ,  $K$ , and  $\mathcal{W}$ 
for each demand  $k \in K$  in the descending order of  $d^k$  do
    for each node  $v \in V$  do
         $\omega_k \leftarrow v$  //  $\omega_k$  is the waypoint for demand  $k$ 
        Let  $cong$  be the network congestion under  $\mathcal{W} \cup \{\omega_k\}$ 
        if  $cong < cong_{min}$  then
             $\omega_k^* \leftarrow \omega_k$ 
             $cong_{min} \leftarrow cong$ 
        end
     $\mathcal{W} \leftarrow \mathcal{W} \cup \{\omega_k^*\}$ 
end

```

for traffic engineering, examining the topic both analytically and empirically. The authors propose a heuristic algorithm that extends a local search algorithm. This extension is achieved by combining the local search algorithm with a greedy WPO algorithm (GreedyWPO). The algorithm GreedyWPO for WPO is shown in Algorithm 11.

In [59], a method called CG4SR, leveraging column generation and a dynamic program for solving the pricing problem were proposed. This method offers solutions with gap guarantees, better scalability on large networks, outperforming multiple existing techniques based on Linear Programming or Local Search.

2.3 Problem Formulation

In this chapter, we focus on the WPO problem. The problem aims to optimize the choice of waypoints for each demand to minimize the congestion, *i.e.*, MLU, in the network. Before formally defining the problem, we need to introduce some preliminary notions for the understanding of the problem.

Segment A segment is a label that is added to each package of the demand. There are two types of segments: node segment and adjacency segment, where a node segment represents a router in the network, and an adjacency segment represents a link. In the context of this thesis, we only consider the node segment.

SR-path A SR-path is defined as an ordered sequence $\langle x_1, x_2, \dots, x_n \rangle$, where each x_i represents a segment. A SR-path for a demand $k \in K$ starts at the source s^k and ends at the target t^k . Because of the ECMP, a SR-path can represent a set of paths in the original network.

Waypoint A waypoint, also known as a middle point or midpoint, refers to a specific node or set of nodes that a packet must traverse along its SR-path. Formally, a waypoint for demand

$k \in K$ can be denoted by $\omega_k^i \in V$, where i is an index in the ordered sequence of waypoints in an SR-path. When constructing a SR-path, the waypoints are strategically placed in the sequence of segments, ensuring that the packet passes through these designated nodes.

With these notions introduced, we can now formally define the WPO problem.

Given a bidirected graph $G = (V, A)$ representing a network topology, where V is the set of nodes and A is the set of arcs. Let K denote a set of demands to be routed in G . Each demand $k \in K$ is defined by a pair of nodes $s^k \in V$ and $t^k \in V$ representing the source and the target of k , a demand volume d^k to be routed from s^k to t^k . Such a demand k is denoted by the triplet (s^k, t^k, d^k) . Let $w \in \mathbb{Z}_+^{|A|}$ be a given metric for graph G , where each arc $uv \in A$ has a weight w_{uv} . The goal of the WPO problem is to determine a set of waypoints $\mathcal{W}_k = \{\omega_k^1, \omega_k^2, \dots, \omega_k^{m^k}\}$ for each demand k in the network, where m^k is the number of waypoints for demand k , such that the network congestion is minimized. Network congestion is defined as the Maximum Link Utilization (MLU) in the network, denoted by $cong$.

- For each demand $k \in K$, $\mathcal{W}_k$ is a subset of all possible nodes in the network, i.e., $\mathcal{W}_k \subseteq V$.
- For each demand between s^k and t^k , an SR-path $\mathcal{P} = \langle x_1, x_2, \dots, x_n \rangle$ is generated such that it includes all waypoints in $\mathcal{W}_k$ in the given order.
- Minimize the network congestion $cong$, while adhering to given routing policies and network constraints.
- The SR-path $\mathcal{P}$ must be loop-free, which ensures the packet reaches the destination without traversing any cycles in the network.

In this chapter, we focus on a special case of the WPO problem that allows for at most one waypoint per demand. Specifically, for each demand $k \in K$, at most one waypoint is allowed. This restriction greatly simplifies the optimization process and makes it easier to deal with large-scale network instances. For simplicity, we employ in the following the notation $\omega_k \in V$ as the only waypoint for demand $k \in K$. We define the problem as follows.

WAYPOINT OPTIMIZATION

Input: A network $G = (V, A)$, where each arc uv has a capacity $c_{uv} \geq 0$, a set of demands K defined for each $k \in K$ by the triplet (s^k, t^k, d^k) , and a metric $w \in \mathbb{Z}_+^{|A|}$.

Output: At most one waypoint $\omega_k \in V$ for each demand $k \in K$ of minimum congestion $cong$.

In our case, we consider the metric w to be the unit metric, i.e., $w_{uv} = 1$ for all $uv \in A$.

3 Our Contributions

3.1 Reinforcement Learning

In this section, we explore the application of classic Reinforcement Learning to address the WPO problem. The objective is to investigate the potential of these well-established algorithms in providing efficient solutions for the problem at hand. We focus on three prominent reinforcement

learning algorithms: Q-learning, SARSA, and SARSA(λ), each offering unique characteristics and learning dynamics.

Reinforcement Learning has been successful in solving various optimization problems by allowing agents to learn from their interactions with the environment. In the context of the WPO problem, we model the network as the environment and formulate the learning process as a Markov Decision Process (MDP). The reinforcement learning agent, representing a network controller, is responsible for selecting waypoints and constructing SR-paths that minimize network congestion.

We begin by discussing the modeling of the WPO problem with three chosen reinforcement learning algorithms, then detailing the necessary modifications and considerations required. We also present different variations of the modeling. The details of the chosen algorithms are presented in Chapter 2.

To model the WPO problem as a reinforcement learning problem, we will first define the essential components of a reinforcement learning framework: states, actions, rewards, and the policy.

States In our case, to make state space manageable, especially for large-scale network instances, we only use the current demand index as the state. This means that the agent will only have access to the current demand, and will not be able to observe the network topology or the current waypoint settings. To represent the end of the decision-making process, we introduce a terminal state *terminal*. More formally, $\mathcal{S} = \{1, \dots, |K|\} \cup \{terminal\}$.

Actions An action $\mathcal{A}(s)$ in this context corresponds to selecting a waypoint $\omega_k \in V$ for each demand $k \in K$ in the network $G = (V, A)$, where state $s \in \mathcal{S}$ corresponds to a demand $k \in K$.

Rewards The reward function, $\mathcal{R}(s, a)$, is designed to encourage the reduction of network congestion. There are two ways to model the rewards.

Negative Maximum Link Utilization: As network congestion is defined as the Maximum Link Utilization (*cong*) in the network, we can use the negative of this value as the reward, i.e., $\mathcal{R}(s, a) = -cong(s)$, where $cong(s)$ represents the congestion incurred from applying the waypoints for demands starting from the first one and proceeding up to the demand corresponding to state s . Lower values of $cong(s)$ correspond to higher rewards, encouraging the agent to minimize congestion.

Congestion Reduction: Alternatively, we can use the reduction in congestion as the reward, i.e., $\mathcal{R}(s, a) = cong(s') - cong(s)$, where s' is the previous state of s . In this case, a greater reduction in the congestion value leads to higher rewards, which encourages the agent to take actions that result in the largest possible decrease in congestion.

Both approaches aim to minimize network congestion, but the choice of reward function may influence the learning dynamics and convergence properties of the reinforcement learning algorithm. In our case, the Congestion Reduction leads to much better learning dynamics.

In addition to the essential components discussed earlier, we found that two modeling approaches: incrementally adding demands along with their waypoints, and routing all demands

without waypoints at once and then learning to add, remove, or modify the waypoints can lead to drastically different results. We will further explore this finding in the subsequent experimental section.

In summary, the reinforcement learning agent will be trained to learn an optimal policy that selects the best waypoint for each demand in the network, aiming to minimize network congestion. Algorithm 12 serves as an example of describing the general procedure to solve the WPO problem using Q-learning. Other algorithms can be readily adapted to this problem by modifying components such as the reward function as needed.

Algorithm 12: Q-learning for WPO Problem

input: network instance $G = (V, A)$, a set of demands K , a demand order π
output: waypoint setting $\mathcal{W}$
 Let $\mathcal{W}$ be an empty set
 Initialize $Q(k, \omega_k)$ arbitrarily for all state-action pairs (k, ω_k)
 $cong_{min} \leftarrow +\infty$
for each episode **do**
 Clear all demands in the network G
 Initialize k with the first demand in π
 while k is not the terminal state **do**
 Choose a waypoint ω_k for demand k with ε -greedy policy
 Route the demand k along with waypoint ω_k , and observe the reward $\mathcal{R}(k, \omega_k)$
 and the next demand k'
 $Q(k, \omega_k) \leftarrow Q(k, \omega_k) + \alpha [\mathcal{R}(k, \omega_k) + \gamma \max_{\omega'_k} Q(k', \omega'_k) - Q(k, \omega_k)]$
 $k \leftarrow k'$
 end
 Let $cong$ be the current network congestion
 if $cong < cong_{min}$ **then**
 let $\mathcal{W}$ be the current waypoint setting
 $cong_{min} \leftarrow cong$
end

3.2 Local Search: GreedierWPO

In this section, we will explore an alternative approach to solving the WPO problem using local search. Local search methods are a class of optimization algorithms that operate iteratively, refining a candidate solution by exploring its local neighborhood in the search space. These methods have proven to be efficient for solving a variety of difficult combinatorial optimization problems, and their simplicity and scalability make them an attractive choice for tackling the WPO problem.

The general idea behind local search is to start with an initial solution and iteratively improve it by making small adjustments or “move” within the solution’s local neighborhood. The neighborhood of a solution is defined by the set of solutions that can be reached by applying a single move operation. The choice of move operations and neighborhood structures is critical for the efficiency of the local search algorithm. In the context of WPO problem, we will need

to carefully design the move operations to ensure that they enable efficient exploration of the search space.

Our proposed algorithm is based on the works in [75], which proposed a simple yet effective local search algorithm. We introduced several important changes to make the original algorithm more efficient and can scale to large instances. Our version, which is called GreedierWPO, is shown in Algorithm 13.

Algorithm 13: GreedierWPO

```

input: network instance  $G = (V, A)$ , a set of demands  $K$ 
output: waypoint setting  $\mathcal{W}$ 
Let  $\mathcal{W}$  be an empty set of waypoints
 $cong_{prev} \leftarrow +\infty$ 
Let  $cong$  be the network congestion with respect to  $G$ ,  $K$ , and  $\mathcal{W}$ 
while  $cong \neq cong_{prev}$  do
     $cong_{prev} \leftarrow cong$ 
    Let  $a_{critic}$  be the most loaded arc under  $\mathcal{W}$ 
    Let  $K_{critic}$  be the set of demands passing through  $a_{critic}$ 
    Let  $\pi \leftarrow PerturbedOrder(K_{critic})$ 
    for each demand  $k \in K_{critic}$  in order  $\pi$  do
        Let  $\omega_k$  be the waypoint for demand  $k$  in  $\mathcal{W}$ 
        if  $\omega_k$  exists then
             $\mathcal{W}' \leftarrow \mathcal{W} \setminus \{\omega_k\}$ 
        else
             $\mathcal{W}' \leftarrow \mathcal{W}$ 
        Let  $V_k \leftarrow GetWaypointCandidates(k)$ 
        for each node  $v \in V_k$  do
             $\omega'_k \leftarrow v$ 
            Let  $cong'$  be the current network congestion under  $\mathcal{W}' \cup \{\omega'_k\}$ 
            if  $cong' < cong$  then
                 $\omega_k^* \leftarrow \omega'_k$ 
                 $cong \leftarrow cong'$ 
                Let  $a'_{critic}$  be the most loaded arc under  $\mathcal{W}' \cup \{\omega'_k\}$ 
                if  $a'_{critic} \neq a_{critic}$  then
                     $\mathcal{W} \leftarrow \mathcal{W}' \cup \{\omega_k^*\}$ 
                    break both for loop
            end
        if  $\omega_k^*$  exists then
             $\mathcal{W} \leftarrow \mathcal{W}' \cup \{\omega_k^*\}$ 
    end
end

```

The primary alteration introduced in the GreedierWPO algorithm, compared to the original GreedyWPO, revolves around the implementation of a while loop that continually adjusts the waypoint selection process until there is no change in the network congestion between iterations. This iterative approach is designed to enhance the algorithm's performance by repeatedly fine-tuning the waypoint settings, which ultimately leads to a more evenly distributed network traffic.

In the original GreedyWPO algorithm, the demands are processed in descending order based

Algorithm 14: PerturbedOrder

input: a set of demands K_{critic}
output: demand order π
Let L be an empty list
for each demand $k \in K_{critic}$ **do**
 | Let $L_k \leftarrow \mathcal{N}(d^k, d^k/5)$
end
Let π be the list of indices of descending order of L

on their volume, and for each demand, the nodes are assessed as potential waypoints to minimize the network congestion. However, the GreedierWPO algorithm pinpoints the most heavily charged arc in the network (*i.e.*, the arc with the highest congestion value) and works specifically with the demands that traverse it.

Furthermore, GreedierWPO, when using the descending order based on demand volume size, becomes a deterministic algorithm, yielding a single, fixed solution for each instance. To address this limitation, we incorporate a degree of variability into the order by using a slightly disrupted order. The idea is to maintain the overall demand order while allowing for variations in each iteration: demands with larger volumes have a higher likelihood of being prioritized. This approach transforms GreedierWPO into a randomized algorithm, which increases its capability of escaping local minima and finding better solutions. More specifically, we generate a random number for each demand, derived from a normal distribution: $\mathcal{N}(d^k, d^k/5)$, where d^k is the volume of the demand k , which influences the random number. The variance of the Gaussian distribution is experimentally determined. The order is then established based on these generated random numbers. Algorithm 14 shows the algorithm to generate the disrupted order. It is important to note that this is merely a basic heuristic for generating such an order. Other methods can also be effective, such as implementing a permutation amongst neighboring nodes within the ranking at a predetermined probability.

We also propose to use only a subset of the nodes $V_k \subseteq V$ as waypoint candidates for each demand k to improve performance. The function *GetWaypointCandidates*(k) is used to retrieve V_k in Algorithm 13. The detailed approaches to propose the waypoint candidates are introduced in the next section.

Another time-saving technique we adopted is to exit the nested for loops when the most heavily loaded arc in the network is altered. This feature enables the algorithm to quickly adapt to the new bottleneck and further optimize the waypoint settings, leading to quicker convergence towards an ideal solution.

3.3 Reduction of Search Space

The complexity of the WPO is inherently high due to the combinatorial nature of the problem, as it involves the selection of waypoints from the entire set of nodes in the network for each demand. As the network size and the traffic grows, the search space increases exponentially, leading to a substantial computational burden when attempting to find an optimal or near-

optimal solution. This requires the development of techniques to reduce the search space, which would accelerate the optimization process.

A reduction in the search space is crucial in our study. The most obvious reason is that it avoids the computational burden associated with evaluating an excessive number of potential waypoint combinations. This efficiency improvement is of great importance, especially when real-time or near-real-time network optimization is desired. Moreover, a smaller search space facilitates a more in-depth examination of the remaining potential solutions. In turn, this can lead to better network performance, characterized by minimized congestion.

Some approaches have been proposed to solve this issue. In [99], the authors propose selecting only a few crucial nodes as waypoint candidates for all demands. Some graph theory concepts, for example, node degree, shortest path centrality, and group shortest path centrality, are employed to determine these important waypoint candidates. The results indicate that by selecting a small percentage of highly central nodes, the method achieves favorable outcomes while significantly reducing runtime complexity. In [89], the authors also deal with similar problems. It proposes limiting the number of waypoint candidates using a so-called bounded stretch constraint, limiting the number of candidates for intermediate nodes based on their distance from the source and destination nodes. This constraint is relative to the shortest path between a source-destination pair, and aims to strike a balance between computation time and near-optimal performance. By applying this constraint, the so-called Segment Routed Traffic Engineering with Bounded Stretch (SRTE-BS) method, based on linear programming, ensures that the length of a SR path does not exceed a specified constant value, or stretch, relative to the shortest path.

Despite the proposed algorithms in the literature for solving the WPO problem and its variants, there remains a noticeable gap with regard to the systematic evaluation of the impact of search space reduction. In this section, we aim to address this gap by presenting a detailed analysis of the influence of search space reduction on the WPO. We also propose to use tree decomposition to reduce the search space. This approach will generate a flexible number of waypoint candidates that take into account both the topology of the network and the demands, which means for demands whose source node and target node are close to each other, we would have less waypoint candidates, and vice versa. Such property would ensure that the search space is reduced while the quality of the solution is not compromised.

The utilization of tree decomposition to propose waypoint candidates involves marking bags in the tree that contain the source and the target and identifying bags that are part of the path between them as waypoint candidates. We illustrate this approach with examples and introduce a formal definition of the process.

Example 1 Consider a network featuring a demand k from node c to node h , as illustrated in Figure 6.2 on the left. We note the set of all bags containing the source node as X^s , and bags containing the target node as X^t . Then, we identify the set of bags in the paths in T joining the bags in X^s and X^t , marked as $\mathcal{P}(X^s, X^t)$. Then, all the nodes in bags $\mathcal{P}(X^s, X^t)$ can be considered as waypoint candidates. In this instance, X^s is the set of abc , cde and bce , and X^t contains the bag egh . As a result, $\mathcal{P}(X^s, X^t)$, which is the set of beg , constitutes the shortest

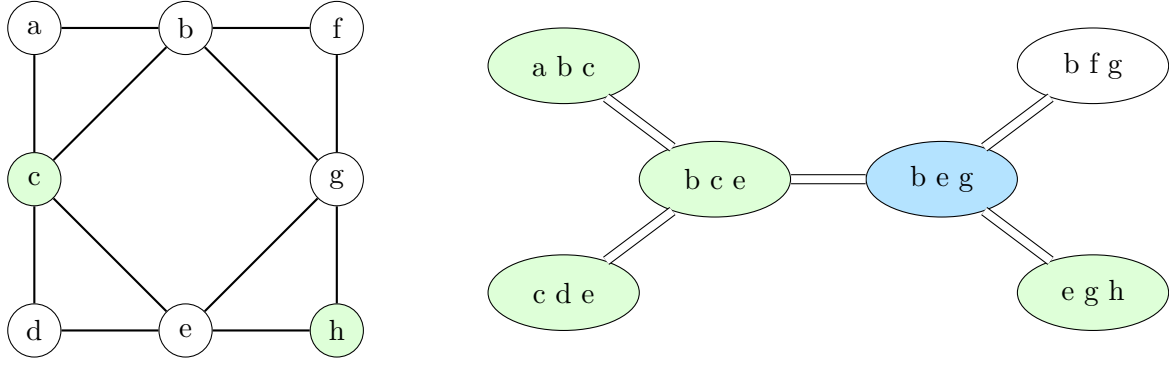


Figure 6.2: Illustration of a tree decomposition T (right) for a network G (left). The left figure is the original network, where the green nodes are the source and the target nodes of a demand $k = \{(c, h, 1)\}$. The figure on the right shows an example of tree decomposition, where the green bags are the bags containing source node c or target node h , and the blue bags are the bags on the path in T between green bags.

path between the X^s and X^t , as depicted on the right side of Figure 6.2. Nodes $\{b, e, g\}$ are thus selected as waypoint candidates V_k for demand k . In particular scenarios where the source and target are located either within the same bag or in adjacent bags in the tree, we use the nodes in the intersection of X^s and X^t as waypoint candidates.

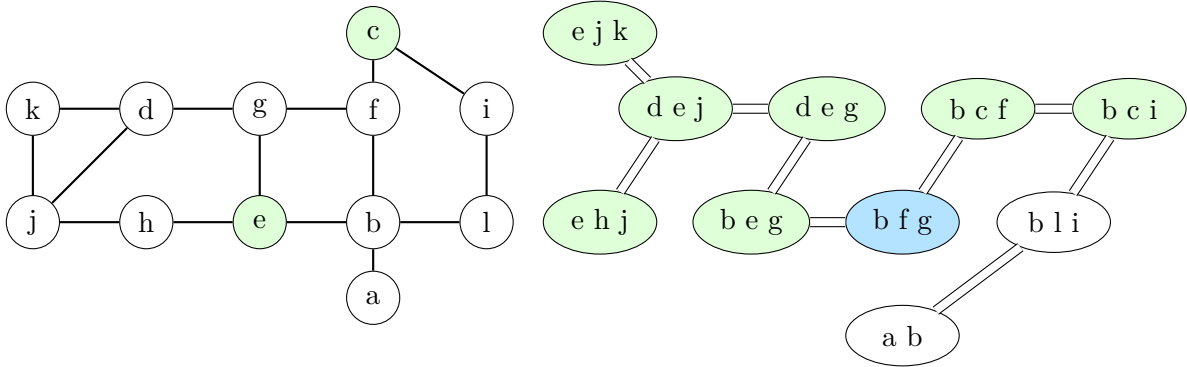


Figure 6.3: Abilene instance (left) with an associated tree decomposition (right)

Example 2 A limitation of utilizing a smaller set of waypoint candidates is that it could restrict the quality of solution. For instance, consider a real-world telecommunication network, Abilene, depicted in Figure 6.3, with a set of demands K . We here focus on a specific demand k , where nodes c and e represent the source and the target, respectively. In this instance, the path between X^s and X^t , noted as $\mathcal{P}(X^s, X^t)$, comprises bag bfg . Consequently, nodes $\{b, f, g\}$ are selected as waypoint candidates. In the Abilene network, however, one of the most frequently congested arcs is arc cf , primarily caused by the demand k . With such a restricted set of waypoint candidates, as initially proposed, optimizing this high congestion is impossible, as the demand k must traverse arc cf in every scenario. The optimal solution for demand k in this case involves passing through nodes i, l, b , and e , necessitating the inclusion of either node i or l as a waypoint.

This issue can be addressed by employing alternative metric on the network's arcs, which can be resolved through the methodologies proposed in Chapter 3. We propose another possible solution to address this concern: the selection of all nodes in bags containing the source and the target, *i.e.*, X^s and X^t , as well as all bags constituting the path between X^s and X^t in the decomposition tree, *i.e.*, $\mathcal{P}(X^s, X^t)$. This approach enables the consideration of all neighbors of the source and the target nodes, in addition to the nodes situated between them. As a result, the search space for waypoints is expanded, thereby increasing the chance of finding good solutions. In the case of Abilene for demand k , the proposed waypoint candidates are thus $\{d, j, h, g, f, k, b, i\}$

Now we formally describe our approach. Let $G = (V, A)$ be a bidirected graph representing a network topology with a set of nodes V and a set of arcs A . Given a demand k with source s^k and target t^k , we aim to reduce the number of waypoint candidates V_k in the WPO problem using tree decomposition. We note the undirected version of G as G^u .

First, perform a tree decomposition of graph G^u to obtain a pair (T, X) , where X is the set of tree bags and T is the tree whose nodes are subsets $X_i \subseteq V$. Next, identify the set of tree bags X^s, X^t that contain the source node s^k and the target node t^k , respectively. Afterward, find the path $\mathcal{P}(X^s, X^t) = (X_{s_1}, X_{s_2}, \dots, X_{s_n})$ in the tree T between X^s and X^t .

Finally, for the case where we only want the smallest number of candidates, which we note as TD_s , define the set of waypoint candidates V_k as follows:

$$V_k = \begin{cases} \{v \mid \exists X_i \in \mathcal{P}(X^s, X^t) : v \in X_i\} & \text{if } \mathcal{P}(X^s, X^t) \neq \emptyset \\ \{v \mid \exists X_i \in X^s : v \in X_i\} \cap \{v \mid \exists X_i \in X^t : v \in X_i\} & \text{otherwise} \end{cases}$$

Note that sometimes $\mathcal{P}(X^s, X^t)$ could be empty because the X^s and X^t are adjacent or even overlapping in the tree. In this case, we would select the intersection of nodes in X^s and X^t as waypoint candidates.

And for the alternative variant which proposes a larger set of candidates, which is noted as TD_l , the waypoint candidates V_k are defined as follows:

$$V_k = \{v \mid \exists X_i \in X^s \cup \mathcal{P}(X^s, X^t) \cup X^t : v \in X_i\}$$

The tree decomposition-based approach for reducing the search space of waypoint candidates offers a tradeoff between (i) having few nodes as candidates, as proposed by [99], and (ii) considering every node as a candidate. The number of candidates and their selection are inherently dependent on both the network topology and the specific demand under consideration. This results in a more adaptive set of candidates for different demands.

3.4 Joint Optimization

In this section, we address the joint optimization of two critical problems in telecommunication traffic engineering: the WEIGHT SETTING problem and the WPO problem.

Here we briefly recall the WEIGHT SETTING problem introduced in Chapter 3. The WEIGHT SETTING problem aims to find optimal weights for the arcs in a network topology such that the

network congestion is minimized. The objective of this problem is to distribute demands across the network in the most efficient manner possible. The problem has two main variants: splittable and unsplittable, which differ in how demands are routed along the shortest paths.

While both the WEIGHT SETTING problem and the WPO problem have been studied extensively in the literature, there is a lack of research focusing on their joint optimization. Indeed, optimizing two problems at the same time can overcome the potential limitation of optimizing only one of them. The primary objective of this section is to explore the joint optimization of the WEIGHT SETTING problem and WPO problem.

There are few studies related to the joint optimization problem. In [75], the author introduces the term “optimality gap” to formally quantify the effectiveness of joint optimization as compared to optimizing link weights and waypoints separately. The paper proves that this joint approach can reduce the network congestion by a factor bounded between $\Omega(n \log n)$ and $\mathcal{O}(n \log n)$, where n is the number of network nodes. Additionally, this study offers a polynomial-time approximation algorithm specifically for the WEIGHT SETTING problem and a local search based algorithm for the WPO problem. The joint optimization algorithm proposed in the study is a sequential combination of the two algorithms.

We employ the same idea here. Our method combines two existing heuristics proposed in this thesis: Nested Rollout Policy Adaptation (NRPA) for the WEIGHT SETTING problem and GreedierWPO for the WPO problem. Our approach is straightforward: we first run NRPA for a fixed execution time to solve the WEIGHT SETTING problem. Once the link weights are set, we proceed to run GreedierWPO to determine the optimal waypoints for routing the demands. The two heuristics are run sequentially, with the output of NRPA serving as an input for GreedierWPO. Algorithm 15 shows our proposed heuristic for the joint optimization. While our heuristic is naive, it serves as a proof-of-concept that joint optimization of these two problems is not only feasible but also efficient. Preliminary results indicate that this simple combination can yield solutions that are competitive, if not better, to existing state-of-the-art methods in terms of minimizing network congestion.

Algorithm 15: Naive-JOINT

input: network instance $G = (V, A)$, a set of demands K

output: weights w and waypoints $\mathcal{W}$

Run NRPA to obtain a metric $w \in \mathbb{Z}_+^{|A|}$

Run GreedierWPO using w to obtain waypoint setting $\mathcal{W}$;

4 Experiments and Results

We establish in this section the experiments and the results. The algorithms of reinforcement learning and local search are implemented in Python, while the network simulation, including the calculation of the shortest path and the congestion is implemented in C++. The experiments were performed on a server 64-core Intel(R) Xeon(R) Gold 5218 CPU of 2.30GHz with 125 GB of memory. Only one CPU core was used for each execution.

During the experiments, we separately studied the reinforcement learning and local search approaches. For reinforcement learning, we mainly focus on the performance of Q-learning and SARSA, and the impact of different demand orders and modeling approaches. For local search, we compared the performance of GreedierWPO and the impact of different demand orders and the reduction of search space.

4.1 Comparison of Reinforcement Learning Approaches

Here we compare the performance of our proposed algorithms, including three reinforcement learning methods, GreedierWPO, and the efficiency of the proposed techniques.

4.1.1 Demand Order

For reinforcement learning algorithms, the demand order could significantly influence the quality of the solution. We experimented with several orders: the increasing volume order, where demands are sequenced according to their respective volumes in an increasing fashion; the decreasing volume order, which is essentially the inverse of the increasing volume order; the fixed random order, wherein a randomly generated order is utilized across all executions; and the random order, which uses a different random order for each algorithmic execution. To maintain experimental fairness, each random order was employed by all algorithms, leading to the generation of ten unique random orders over ten independent executions.

Figure 6.4 and 6.5 highlight the comparison of different orders using Q-learning and SARSA, respectively. Each result represents ten independent executions over a duration of 10 seconds each. The congestion is normalized according to the lower bound. From the result, no distinct advantage is discernible for any specific order across all instances. The results fluctuate significantly depending on the specific instances. Despite the lack of a clearly superior order, the results of Q-learning and SARSA exhibit certain common characteristics: a specific order that performs well in a given instance with Q-learning often performs comparably well with SARSA, and vice versa.

While the demand order can greatly influence the outcomes of reinforcement learning, there is no universally superior order across all instances. To address this, a potential solution for a specific instance is to explore multiple orders and select the one that yields the best performance. In our experiment with reinforcement learning approaches, we adopt a fixed random order for the sake of convenience.

4.1.2 Incremental addition of demands with waypoints

The two approaches for modeling the problem introduced previously: incrementally adding demands with waypoints, and adding all demands beforehand without waypoints and then adding waypoints, represent different strategies that are discussed in this part.

Incremental addition of demands with waypoints could be seen as a more realistic scenario for the online version of the problem, *i.e.*, the algorithm optimizes as new demands are generated.

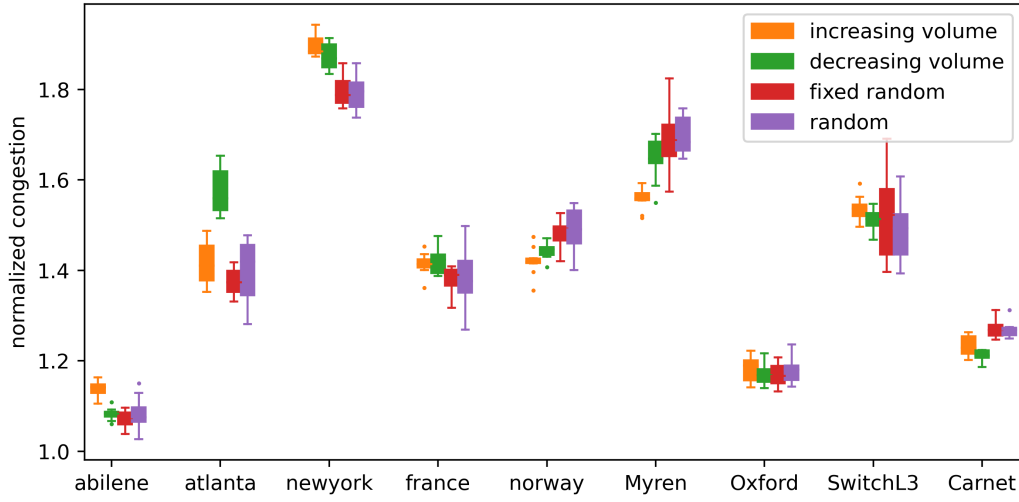


Figure 6.4: Comparison of normalized congestion with different demand order using Q-learning. Each result represents 10 independent executions of 10 seconds.

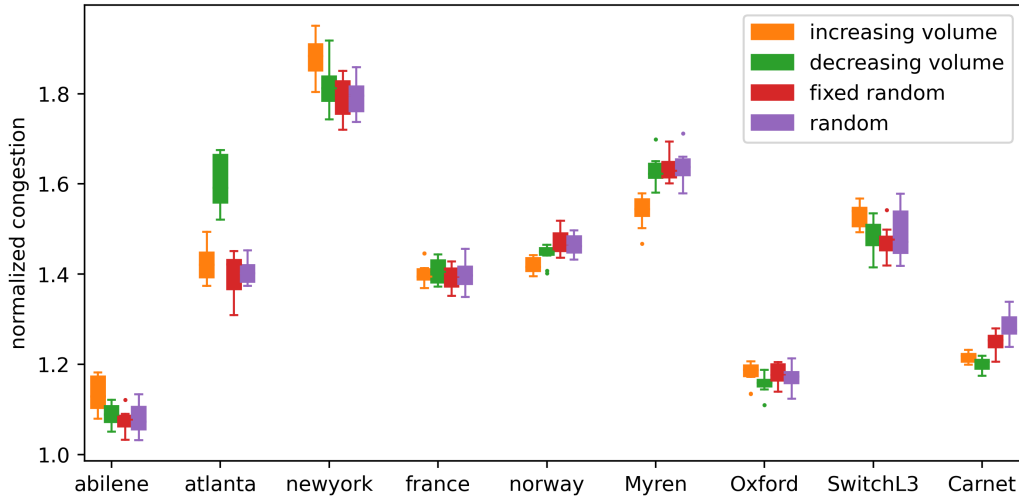


Figure 6.5: Comparison of normalized congestion with different demand orders using SARSA. Each result represents 10 independent executions of 10 seconds.

However, because of the lack of sufficient information of future demands, this approach might lead to suboptimal solutions for the case that the traffic matrix is known.

Figure 6.6 shows the comparison of the incremental modeling and non-incremental modeling using Q-learning and SARSA. The figure show that for certain instances, such as Abilene, Myren, SwitchL3, and Carnet, the strategy of incrementally adding demands results in significantly improved results, while for other instances such as Atlanta, New York, and Oxford, the non-incremental approach proves to be more effective.

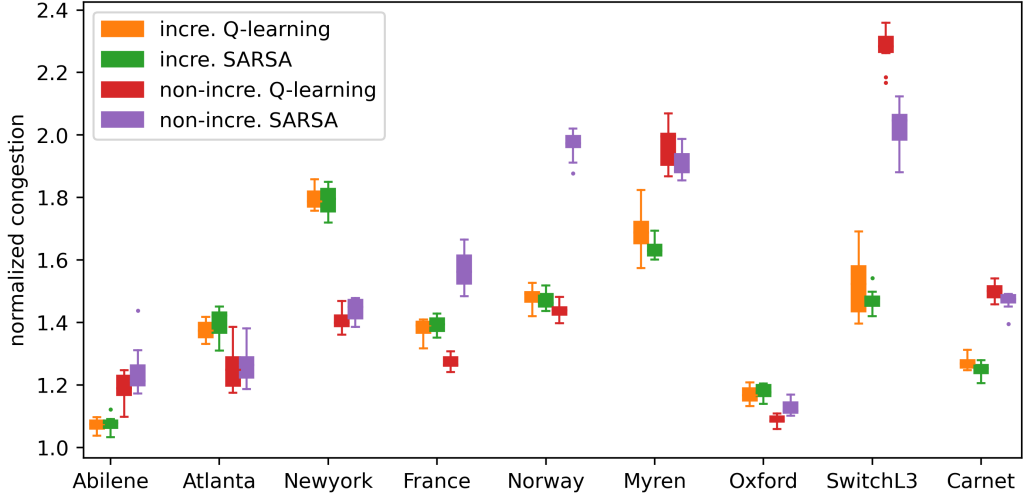


Figure 6.6: Comparison of normalized congestion between incremental demands and non-incremental demands modeling. Each result represents 10 independent executions of 10 seconds.

The reason behind this result is still unclear to us. Potential contributing factors could be the underlying topological structure or the demands distributions.

Given this situation, the most pragmatic approach would be the adoption of a problem-specific strategy, similar to the demand order, rather than pursuing a universal solution. This would create a decision mechanism that selects the most appropriate strategy based on the characteristics of each instance. For simplicity, in our following experiments, we will use incremental modeling for all instances.

4.2 GreedierWPO

4.2.1 Demand Order

We begin by investigating the effect of demand order on the outcome of our local search algorithm. In the default setting, the algorithm sorts the demands in descending order according to volume, which results in deterministic outcomes. To enrich the exploration capability of our algorithm, we have introduced a disrupted order as an example to introduce a slight randomness in the order. In fact, there could be other methods working well for the same purpose.

For example, we also implement a variant method from strict descending order by incorporating a 20% chance of swapping adjacent demands. To evaluate the efficacy of these approaches, we compare the randomized descending order method with a fully random order wherein the demand sequence is generated again before each local search iteration.

The result of these different orderings are compared in Figure 6.7. For each algorithm variant, we perform 10 independent executions on each instance.

The result clearly illustrates that while the descending order approach outperforms the random order in large instances like IGen, its lack of diversity limits its performance in smaller

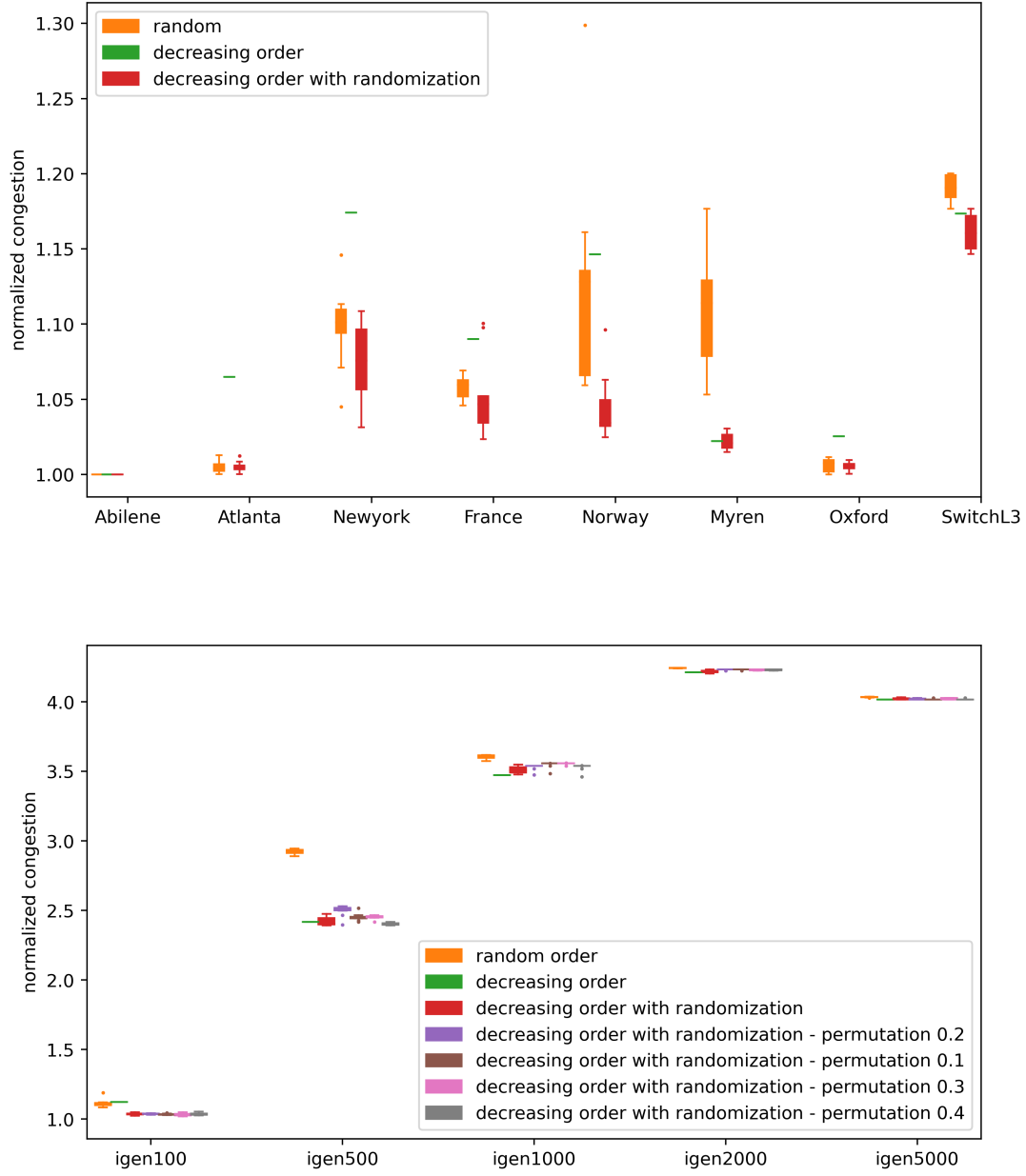


Figure 6.7: Comparison of congestion with different demand orders using GreedierWPO for SNDlib, IGen and Inet instances. Each result represents 10 independent executions of, respectively, 0.1 second and 100 seconds.

instances and leads to identical results across multiple executions. In contrast, the randomized descending order yields better performance across almost all instances, thus showcasing the benefits of introducing an element of randomness to the original descending order approach.

4.2.2 Search Space

In this section, we examine various techniques to reduce the search space. Figure 6.8 presents a comparison of candidate numbers. When fewer candidates are used, these are selected randomly from the nodes of the network.

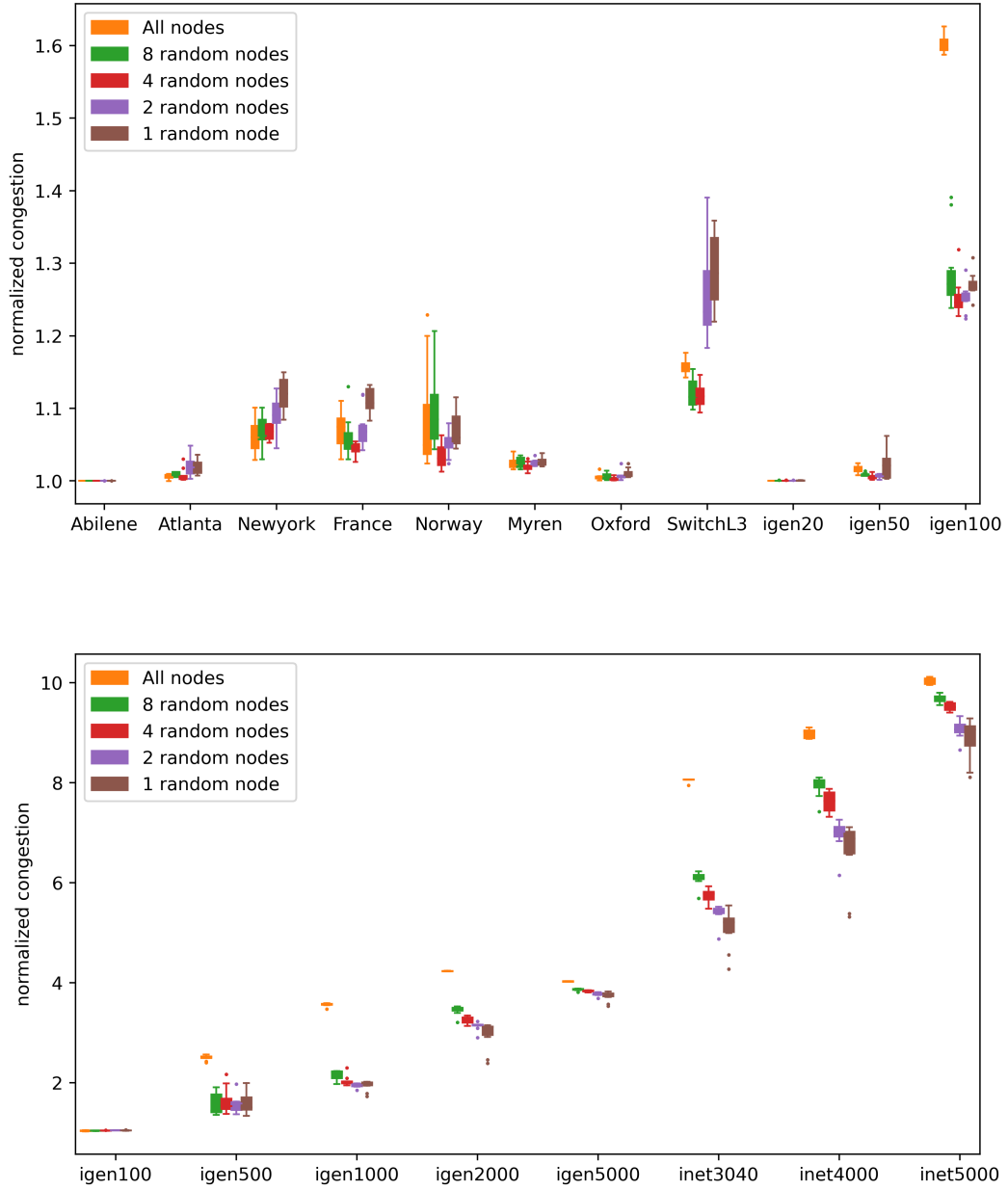


Figure 6.8: Comparison of normalized congestion using different number of waypoint candidates on SNDlib and IGen instances. Each result represents 10 independent execution of, respectively, 0.1 second and 100 seconds.

The results show that small instances do not indicate a clear preference for a specific can-

didate number. However, for larger instances, there is an apparent advantage in limiting the number of candidates. This is primarily caused by the reduced complexity of the search space, which simplifies the search process and allows for more rapid iteration.

Figures 6.9 and 6.10 further illustrate the convergence of the GreedierWPO algorithm. Each color's largest point corresponds to the scenario wherein all nodes are considered as candidates, whereas smaller points within the same color series respectively represent cases where 8, 4, 2, and 1 randomly chosen nodes are considered. Each data point corresponds to the average congestion over 10 independent executions, normalized by the lower bound.

It can be observed that more iterations typically yield better performance. However, given the same number of iterations, limiting the number of candidates can compromise solution quality when compared to scenarios with more candidates. Yet, in larger instances, fewer candidates often lead to more iterations, thereby enhancing the results. Thus, selecting the candidate number represents a trade-off between solution quality and iteration speed.

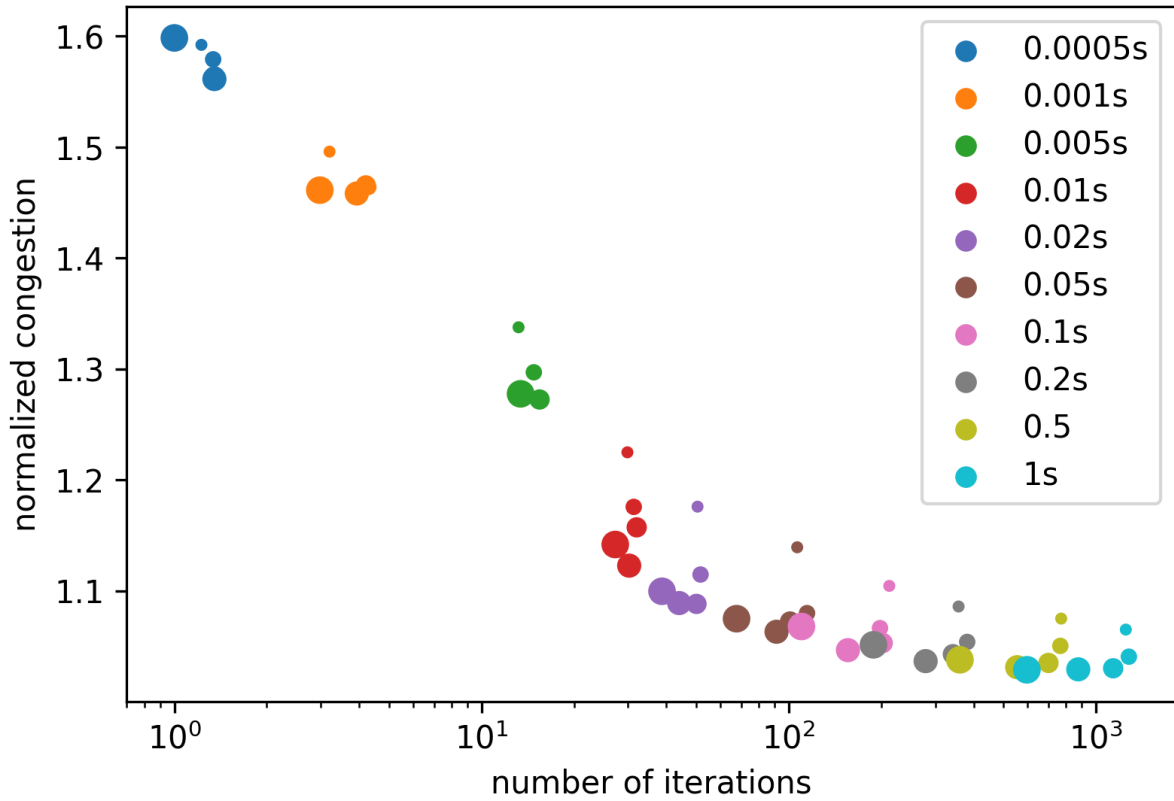


Figure 6.9: Convergence of GreedierWPO on france. The largest point of each color signifies the case with all nodes as candidates. The smaller points represent the cases with 8, 4, 2, and 1 nodes as candidates.

As a result, for smaller instances, we choose 4 nodes, which generally perform well across all cases. Conversely, for larger instances, we utilize a single candidate node. Figures 6.11 and 6.12 compare different techniques for proposing waypoint candidates. For smaller instances, we examined the 4 nodes with the highest degree or betweenness centrality, 4 random nodes, and

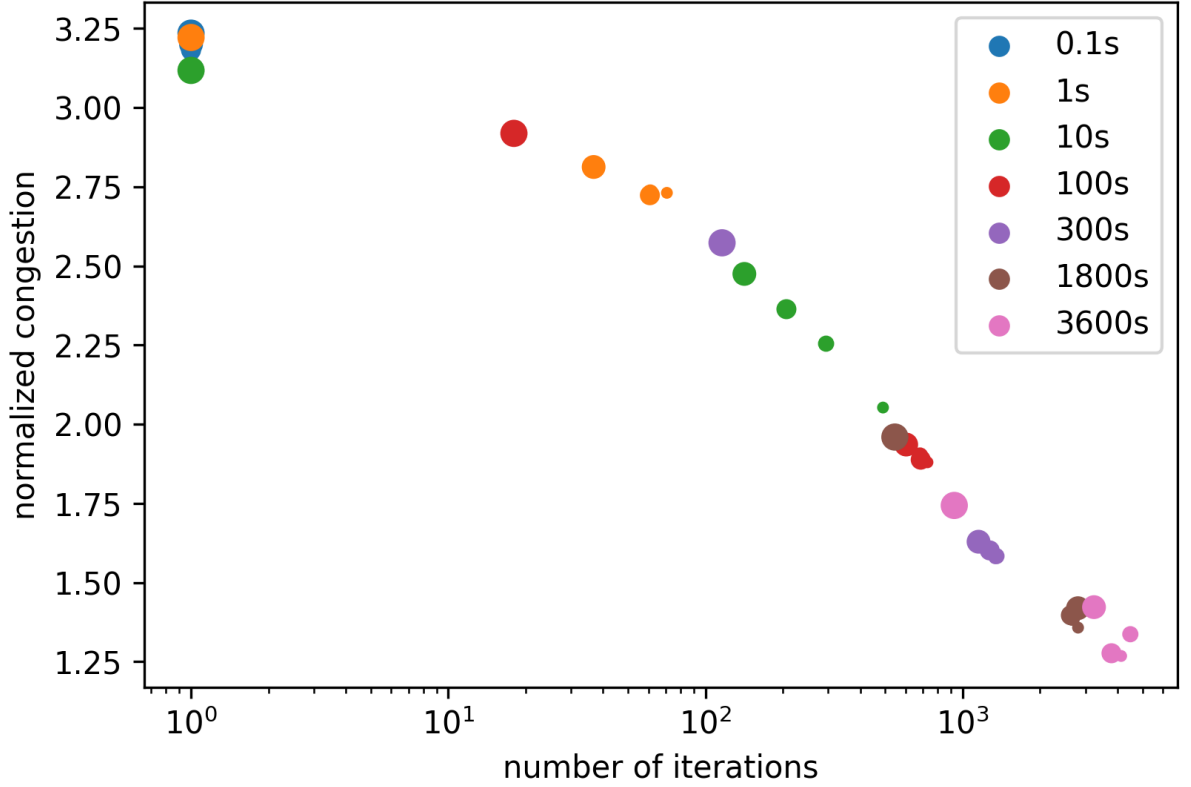


Figure 6.10: Convergence of GreedierWPO on igen500. The largest point of each color signifies the case with all nodes as candidates. The smaller points represent the cases with 8, 4, 2, and 1 nodes as candidates.

4 random nodes selected via our proposed tree decomposition methods: TD_l and TD_s . For larger instances, we contrasted a single random node against a node selected randomly using tree decomposition.

Results show that random node selection is the best approach for smaller instances. In the case of large instances, a random node chosen via tree decomposition TD_l is considered as the superior choice.

4.3 Comparison Between Approaches

Figure 6.13 shows the comparison between our proposed approaches, which includes three reinforcement learning approaches and GreedierWPO. For the reinforcement learning methods, we employ a fixed random order along with incremental demands modeling. Each result represents 10 independent executions with a duration of ten seconds each.

The results clearly demonstrate that our proposed local search algorithm, GreedierWPO, considerably surpasses every reinforcement learning algorithm in performance across all tested instances, coming remarkably close to the lower bound in nearly every case. Despite having a more intricate learning mechanism, SARSA(λ) does not exhibit better performance compared to the other methods.

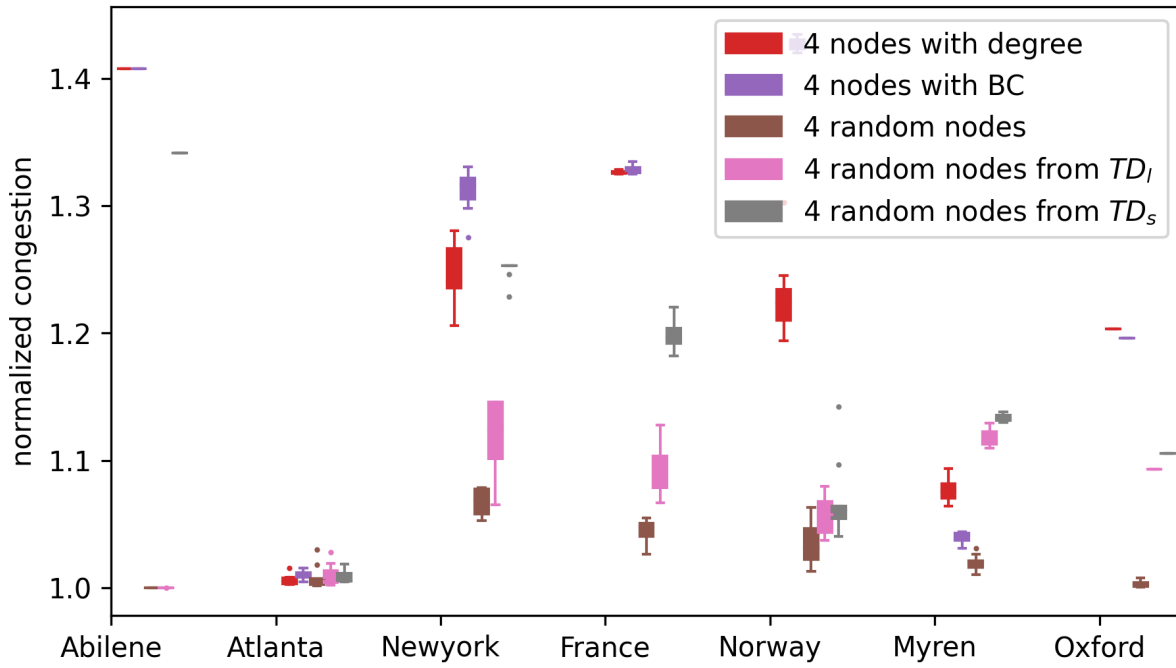


Figure 6.11: Comparison of normalized congestion using GreedierWPO with different demand candidates. Each result represents 10 independent executions of 0.1 second. The number of candidates is 4.

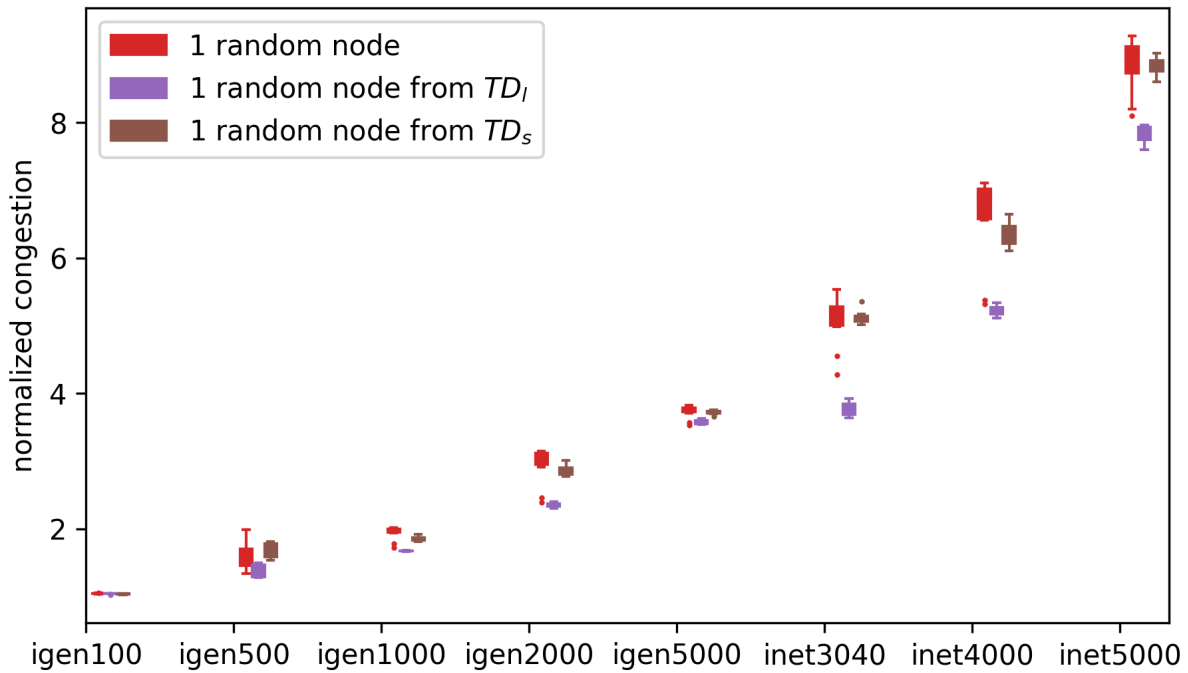


Figure 6.12: Comparison of normalized congestion using GreedierWPO with different demand candidate on large instances. Each result represents 10 independent executions of 100 seconds. The number of candidates is 1.

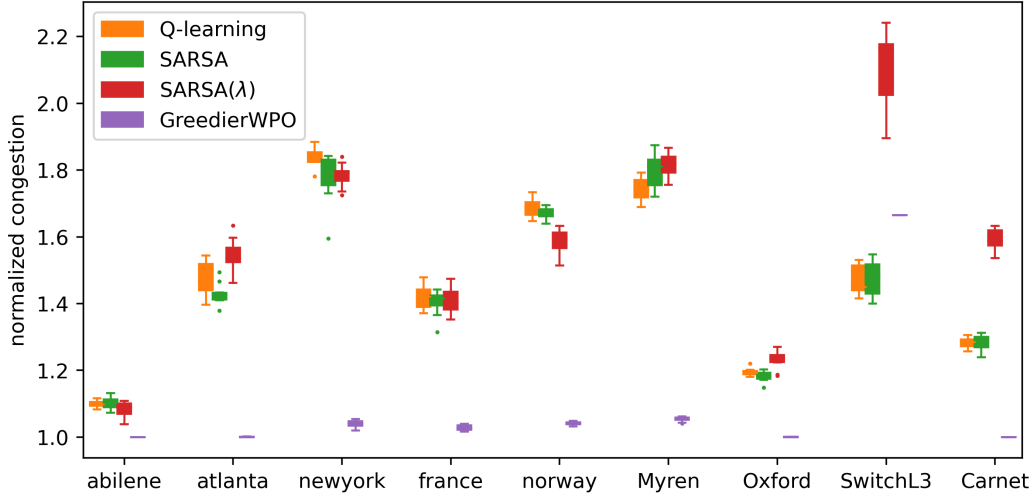


Figure 6.13: Comparison of normalized congestion between proposed approaches. Each result represents 10 independent executions of 10 seconds

The significantly better performance of GreedierWPO can be attributed to the inherent characteristics of the algorithm. Local search algorithms are better at navigating vast state spaces by implementing incremental improvements to an existing solution. This capability facilitates an effective exploration of the solution space, enabling a faster convergence towards near-optimal solutions. Additionally, GreedierWPO cyclically optimizes demands, allowing for the re-optimization of demands after applying future demands with corresponding waypoints. In a certain sense, this implies that GreedierWPO maintains a higher level of foresight regarding future demands during the optimization. Lastly, GreedierWPO has a considerably lower computational complexity relative to reinforcement learning methods, contributing to its better performance.

4.4 Comparison with Other Approaches

Table 6.1 presents a comparison between our proposed GreedierWPO and the original local search algorithm, GreedyWPO. The GreedierWPO algorithm shows better performance across all tested instances within the same execution time.

Table 6.1: Comparison of congestion between GreedierWPO and GreedyWPO. The result for GreedierWPO is the average value of 20 independent executions. lb: lower bound of the instance

instance	GreedierWPO	GreedyWPO	time (s)	lb
abilene	60.414	108.06	0.024	60.411
atlanta	2.207	3.2045	0.0432	2.1795
newyork	0.0478	0.06626	0.05	0.0445
france	2.524	3.52	0.11	2.41
norway	0.2792	0.33425	0.2411	0.27

In Table 6.2, we compare GreedierWPO with the CG4SR algorithm introduced in [59]. This algorithm uses a mathematical programming approach that leverages on column generation. Originally, this algorithm solves the WPO problem by considering the SR-paths made of nodes and/or arcs. In our implementation of CG4SR, the algorithm considers SR-paths formed by nodes exclusively, so as our GreedierWPO algorithm. Since CG4SR is not an anytime algorithm, we aim to match the execution times for GreedierWPO. For particular instances, such as france and igen500, which have extended execution times, we choose shorter execution periods of 1 or 100 seconds.

Table 6.2: Comparison of congestion between GreedierWPO and CG4SR. The result for GreedierWPO is the average value of 20 independent executions.

x: not enough memory for CG4SR

instance	GreedierWPO	GreedierWPO time (s)	CG4SR	CG4SR time (s)	lower bound
abilene	60.41248	0.07	62.0858	0.07	60.411
atlanta	2.189	0.74	2.180	0.74	2.1795
newyork	0.0471	0.21	0.054	0.21	0.0445
france	2.4826	1	2.497	753.78	2.41
norway	0.2763	0.16	0.319625	0.16	0.27
Myren	0.3347	1.3	0.33135	1.3	0.331325
Oxford	0.1897	0.23	0.1896	0.23	0.1895
SwitchL3	0.3523	0.76	0.3272	0.76	0.3272
igen100	0.448	2.69	0.6196	2.69	0.42748
igen500	1.64579	100	3.59432	277.16	1.19493
igen1000	1.733	100	3.118	2429.92	1.03585
igen2000	2.4378	100	x	x	1.03684
igen5000	4.268	100	x	x	1.19137
inet3040	0.2531	100	x	x	0.0672
inet4000	0.4011	100	x	x	0.0768
inet5000	0.644	100	x	x	0.0821

For smaller instances, and within the same execution time, GreedierWPO outperforms CG4SR in some cases. Even in scenarios where GreedierWPO does not surpass CG4SR, the results are generally comparable. In three out of the four cases where CG4SR shows better performance to GreedierWPO, the disparity in congestion does not exceed 1% of the respective lower bound. Significantly, GreedierWPO consistently outperforms CG4SR in all large instances. We encounter difficulties when executing CG4SR for instances larger than 2000 nodes due to its substantial memory requirements. In contrast, GreedierWPO, requiring minimal memory, efficiently solves instances involving thousands of nodes.

4.5 Joint Optimization

We also compared the results of the joint optimization versus optimizing the metrics or the waypoint individually. Figure 6.14 presents the comparative performance of Naive-JOINT, the proposed joint optimization method against Nested Rollout Policy Adaptation (NRPA) and GreedierWPO on IGen and Inet instances. NRPA is proposed in Chapter 3 to solve the WEIGHT

SETTING problem at scale, while GreedierWPO is our proposed method for the WPO problem. The result clearly indicates that for smaller instances, like igen100, etc., GreedierWPO outperforms the others. However, as the instance size increases, both the NRPA and Naive-JOINT improved the results compared to GreedierWPO. And our proposed method, Naive-JOINT demonstrated the best performance among the three on large instances. This empirically shows the advantages of joint optimization compared to solving two problems separately.

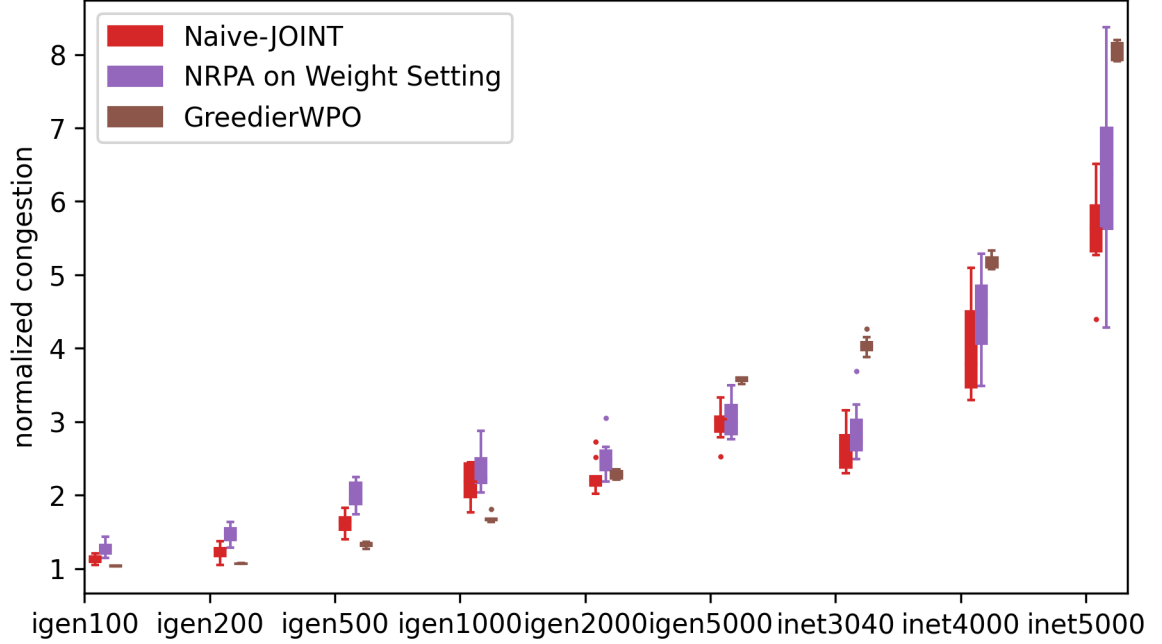


Figure 6.14: Comparison of normalized congestion between joint optimization, WEIGHT SETTING and WPO problems. Each optimization is run in 200 seconds, with 10 independent executions. The congestion value is normalized according to the lower bound. For Naive-JOINT, both WEIGHT SETTING and WPO were given a duration of 100 seconds each for execution.

We also compared our approach with the joint optimization approach JOINT-HEUR proposed by [75]. Table 6.3 shows the comparison between our proposed method and JOINT-HEUR on small instances of SNDLib. The table clearly shows that Naive-JOINT demonstrates better results for every instance even with much less execution time.

Table 6.3: Comparison of the congestion value between Naive-JOINT and JOINT-HEUR. Each result is the average of 5 independent executions.

instance	Naive-JOINT	Naive-JOINT time (s)	JOINT-HEUR	JOINT-HEUR time (s)
abilene	60.4323	2	62.344	9.895
atlanta	2.2098	2	2.2579	22.231
france	2.5855	20	2.7238	849.61
newyork	0.0561	20	0.083	952.938
norway	0.2956	20	0.342	2520.53

As for the large instances generated by IGen or Inet, after an execution period exceeding one hour, JOINT-HEUR still failed to give results for even the smallest instance in this test: igen100. It seems that the bottleneck of JOINT-HEUR is the WEIGHT SETTING heuristics. This observation again underscores the efficacy and the ability to solve the problem at scale of our proposed methods.

5 Conclusion

In conclusion, our local search algorithm, GreedierWPO, introduces important improvements to existing solutions, demonstrating better performance in certain scenarios and comparable outcomes in others. Importantly, GreedierWPO outperforms existing methods such as CG4SR when handling large instances, offering a more efficient, memory-friendly alternative. We also found that the number of candidates selected represents a pivotal trade-off point. While more iterations typically result in better performance, restricting the number of candidates can greatly affect solution quality. Joint optimization of the WEIGHT SETTING problem and the WPO problem is also briefly discussed in this chapter, our proposed heuristics also showed better performance on large instances than the state-of-the-art approach.

Future research can further refine GreedierWPO's efficiency and applicability to even larger-scale instances. Our work opens up opportunities for exploring various methods to further reduce search space and enhance the quality of solutions, thus driving progress in the optimization of SR-TE problems.

Chapter 7

Conclusion and perspectives

1 Conclusion

The primary objective of this thesis was to address key challenges in telecommunication traffic engineering, specifically focusing on the `WEIGHT SETTING` problem and the `WAYPOINT OPTIMIZATION` problem. The aim was to improve the performance and scalability of large-scale telecommunication networks by optimizing network congestion. To facilitate algorithmic solutions, these real-world engineering problems were abstracted into graph problems.

The third chapter was dedicated to the application of Monte Carlo methods to the `WEIGHT SETTING` problem. Three different approaches were assessed, and techniques were proposed to improve their performance, especially for large-scale instances.

The fourth chapter addressed a significant computational bottleneck: the calculation of the shortest path and congestion in large, dynamic networks. An algorithm based on Customizable Contraction Hierarchy (CCH) was introduced, which not only improved the computational efficiency but also yielded better results when combined with the Monte Carlo methods from the third chapter.

In the fifth chapter, the Nested Rollout Policy Adaptation (NRPA) was revisited and improved upon by introducing the Meta-NRPA algorithm. By incorporating optimal stopping theory, the algorithm was able to escape local minima more effectively. This led to better performance not just for the `WEIGHT SETTING` problem but also for other difficult combinatorial optimization problems.

Another significant contribution in the sixth chapter was the development of the Greedier-WPO algorithm for the `WAYPOINT OPTIMIZATION` problem, which demonstrated better performance in large instances and outperformed existing state-of-the-art methods like CG4SR. The algorithm also highlighted the importance of the trade-off between the number of candidates selected and the quality of the solution, providing a new direction for future research.

All algorithms proposed were rigorously tested on both real and synthetic datasets. The results confirmed their scalability and effectiveness. Moreover, these algorithms are versatile and can be adapted to problems with additional constraints. They are not sensitive to the size

of the network or the search space, making them highly applicable in a variety of settings.

The thesis also laid the groundwork for future research. For example, optimizing congestion with minimal delay is identified as a promising area for subsequent work. Additionally, the algorithms developed here, e.g. HSPR, could be applied to other shortest path routing problems.

In summary, this thesis has successfully developed and validated multiple heuristic algorithms for solving difficult problems in telecommunication traffic engineering. The algorithms developed are not only effective and scalable but also versatile, capable of being adapted to a wide range of related problems. As networks continue to grow in size and complexity, the algorithms and insights presented here provide a foundation for more robust, efficient, and adaptive network management solutions.

2 Perspectives

The contributions of this work represent only a fraction of the research directions regarding combinatorial optimization problems arising in the field of telecommunication. The algorithms and methodologies developed here have broader implications, and their adaptability suggests several promising directions for future research:

- **Network Robustness and Resilience:** The current work has primarily focused on optimizing network performance in normal conditions. Future work could extend this to scenarios involving network failures, aiming to develop algorithms that are not just efficient but also resilient.
- **Real-Time Network Optimization:** The algorithms developed in this thesis are primarily designed for static or quasi-static network conditions. However, telecommunication networks are inherently dynamic. Future work could focus on extending these algorithms for real-time optimization, adapting to network changes on-the-fly. This could involve leveraging streaming analytics and real-time data feeds to make immediate adjustments to network weights or routes.
- **Joint Optimization:** The thesis has proposed a naive heuristic for the joint optimization of the WEIGHT SETTING problem and WAYPOINT OPTIMIZATION problem. A natural extension would be to investigate more sophisticated algorithms that can better perform joint optimization. This could lead to more cohesive and efficient network configurations, as the two problems are intrinsically linked in real-world scenarios.
- **Quality-of-Service (QoS)-Constrained Optimization:** The primary focus of this thesis has been on optimizing network congestion. However, telecommunication networks often have to meet various QoS requirements, such as latency, packet loss, and jitter. We only considered some of the constraints in Chapter 3 for the WEIGHT SETTING problem. Future work could extend the current algorithms to operate under these additional QoS constraints, providing a more comprehensive solution for network optimization.

- **Multi-Objective Network Optimization:** While the current work has primarily focused on congestion, there are other objectives like energy efficiency and fault tolerance that are equally important. Future research could explore algorithms that simultaneously optimize multiple objectives, providing a balanced and more holistic network management approach.

Résumé Français

1 Introduction

1.1 Context et Problématique

Au cours des dernières années, l'industrie des télécommunications a connu un changement de paradigme significatif dû à l'émergence des technologies de réseau de nouvelle génération. Ces avancées révolutionnaires offrent des vitesses de transfert de données accrues, une latence réduite et une capacité améliorée, modifiant fondamentalement le fonctionnement et la gestion des réseaux de télécommunications.

Un développement majeur dans ce domaine est l'incorporation de la mise en réseau définie par logiciel (SDN) et la virtualisation de réseau. Cette approche innovante découple le matériel physique de la couche logicielle qui le contrôle, permettant la coexistence de multiples réseaux virtuels sur la même infrastructure physique. Cela permet non seulement une plus grande flexibilité et évolutivité, mais facilite également une utilisation plus efficace des ressources, une mise en réseau et une maintenance simplifiées, et une meilleure tolérance aux pannes. De plus, des algorithmes d'optimisation peuvent être déployés et exécutés directement dans les contrôleurs de réseau (contrôleur SDN), une rupture nette avec les méthodes traditionnelles.

Ces nouveaux paradigmes ont conduit à une architecture de réseau plus centralisée, engendrant de nouveaux défis en matière de routage du trafic, d'équilibrage de charge et d'efficacité énergétique. Ces défis sont particulièrement prononcés dans le domaine de la virtualisation et de l'optimisation des réseaux, car le maintien de la qualité de service (QoS) devient de plus en plus crucial face au volume croissant du trafic généré par le streaming haute définition, le jeu en nuage et d'autres applications gourmandes en bande passante.

Un autre développement important est le Segment Routing (SR), qui est apparu comme une solution clé pour simplifier les opérations de réseau et améliorer l'évolutivité dans l'industrie des télécommunications. C'est un mécanisme de calcul de chemin et d'ingénierie du trafic basé sur la source qui fonctionne au sein de la couche de réseau, utilisant les protocoles de routage existants (par exemple, OSPF, IS-IS, etc.) pour établir des chemins de transmission efficaces et flexibles.

Le SR peut être particulièrement bénéfique dans les réseaux de télécommunication à grande échelle, où il peut optimiser les flux de trafic, améliorer les performances du réseau et réduire la complexité opérationnelle. En permettant un contrôle plus granulaire sur le routage et la

transmission des paquets, le SR permet aux opérateurs de réseau d’optimiser les ressources du réseau, de minimiser la congestion et d’assurer une meilleure Qualité de Service (QoS) pour les utilisateurs finaux.

Pour garantir que la QoS soit maintenue dans les réseaux de télécommunication modernes, il est impératif de développer des algorithmes efficaces capables de s’adapter à la taille du réseau et de répondre aux exigences spécifiques de la virtualisation. Une voie prometteuse pour la conception d’algorithmes d’optimisation pour les réseaux virtualisés est l’intégration de l’apprentissage automatique (ML) et de l’optimisation combinatoire (CO). Cette combinaison offre le potentiel de créer des algorithmes capables de s’adapter à la nature dynamique des réseaux virtualisés en tirant parti des capacités des réseaux modernes, ouvrant ainsi un nouveau domaine de recherche avec d’importantes opportunités d’exploration et d’innovation.

Dans cette thèse, nous entreprenons une exploration des algorithmes heuristiques pour aborder certains des problèmes d’ingénierie du trafic les plus notablement difficiles, en particulier le Problème de Métriques et le problème d’Optimisation des Waypoint (WPO). Les algorithmes heuristiques sont particulièrement adaptés pour fournir des résultats de haute qualité dans un temps limité pour de grands réseaux de télécommunication, les rendant idéaux pour relever les défis posés par les technologies de réseau émergentes. Notre principal objectif est de minimiser la congestion du réseau de télécommunication tout en maintenant les paramètres requis de Qualité de Service (QoS), tels que le délai, dans un environnement de plus en plus virtualisé et dynamique.

En examinant le potentiel des algorithmes heuristiques dans le domaine de l’optimisation des réseaux de télécommunication, cette thèse aspire à contribuer au développement continu de solutions avancées pour gérer et optimiser des réseaux futurs. Grâce à des recherches et des expérimentations sur des réseaux synthétiques et réels, nous étudierons les avantages et les limites de diverses méthodes heuristiques et évaluerons leur adaptabilité à l’évolution rapide des réseaux de télécommunication. En outre, nous présenterons des propriétés clés pour l’optimisation des réseaux à grande échelle, ce qui permettra de mieux comprendre le processus d’optimisation pour de tels problèmes et d’offrir des connaissances précieuses aux opérateurs de réseaux et aux chercheurs.

En outre, certaines des méthodes présentées dans cette thèse peuvent être appliquées à un large éventail de problèmes, englobant non seulement des problèmes d’optimisation combinatoire difficiles tels que la simulation du trafic réseau, mais aussi des problèmes plus généraux tels que le problème du voyageur de commerce, le problème du snake in the box, et d’autres encore.

En résumé, cette thèse vise à découvrir le potentiel des méthodes heuristiques pour l’optimisation des réseaux de télécommunications, en aidant à relever les défis et à capitaliser sur les opportunités offertes par les technologies de réseaux de nouvelle génération. En étudiant des techniques heuristiques innovantes, nous nous efforçons de faire progresser l’état de l’art en matière d’optimisation des réseaux, contribuant ainsi à l’amélioration des performances, de l’efficacité et de la fiabilité des réseaux de télécommunications au profit de tous les utilisateurs.

Il est important de souligner que cette thèse traite exclusivement de l’optimisation au niveau algorithmique et n’entre pas dans les détails de la structure du réseau et de la conception et de

la mise en œuvre du protocole. Par conséquent, dans notre travail, la structure du réseau, les protocoles et les aspects connexes ont été abstraits dans des modèles abstraits et des algorithmes à des fins d'optimisation algorithmique.

1.2 Objective et Méthodologie

L'objectif principal de cette thèse est de développer des algorithmes heuristiques pour aborder certains problèmes d'ingénierie du trafic en télécommunication, dans le but d'assurer la performance des réseaux de télécommunication à grande échelle et de permettre la scalabilité des solutions. Pour se concentrer uniquement sur leurs aspects algorithmiques, nous avons simplifié et modélisé les problèmes d'ingénierie du trafic sous forme de problèmes de graphe. Les problèmes d'ingénierie du trafic pris en compte sont le Problème de Métriques et le Problème d'Optimisation des Waypoint (WPO). Notre principal intérêt réside dans l'optimisation de la congestion du réseau, qui, dans notre contexte, représente le rapport entre le trafic traversant l'arc du réseau et la capacité de cet arc. En dirigeant adroitement le trafic au sein du réseau à l'aide d'algorithmes d'optimisation, nous pouvons atténuer la congestion et ainsi offrir une meilleure qualité de service (QoS).

Les algorithmes proposés dans cette thèse sont principalement basés sur des heuristiques, englobant des algorithmes basés sur Monte Carlo, des techniques de recherche locale, et certains algorithmes d'apprentissage par renforcement. Ces algorithmes sont spécifiquement conçus pour traiter le problème d'ingénierie du trafic à grande échelle et sont testés sur un ensemble d'instances, incluant à la fois des données réelles de réseau de télécommunication et des ensembles de données synthétiques. Pour démontrer l'efficacité des algorithmes proposés, des résultats empiriques sur des instances de différentes tailles sont présentés, corroborant leur performance et mettant en évidence leur potentiel à contribuer de manière significative au domaine de l'optimisation des réseaux de télécommunication.

De plus, afin de renforcer la fiabilité de nos conclusions, nous réalisons des analyses comparatives entre les algorithmes heuristiques proposés et les techniques d'optimisation existantes, en mettant en évidence les avantages de nos méthodes en termes d'efficacité computationnelle, d'adaptabilité et de scalabilité. En examinant les performances et l'applicabilité des algorithmes heuristiques, cette thèse vise à faire progresser l'état de l'art en matière d'optimisation des réseaux de télécommunication et à contribuer au développement de solutions plus efficaces, robustes et évolutives pour les problèmes d'ingénierie du trafic.

1.3 Préliminaires de Graph

Un *graphe dirigé* ou *digraphe*, noté $G = (V, A)$, est constitué d'un ensemble de nœuds V et d'un ensemble de paires ordonnées de nœuds, appelées *arcs*, $A \subseteq V \times V$. Soit $|V|$ le nombre de nœuds et $|A|$ le nombre d'arcs. Pour un arc $uv \in A$, nous disons que uv est dirigé de u vers v . Un *graphe bidirectionnel* est un *graphe dirigé* où, pour chaque arc $uv \in A$ dans le graphe, il existe également un arc $vu \in A$. Le *in-degree* d'un nœud $v \in V$ est le nombre d'arcs dirigés vers le nœud v . Le *out-degree* d'un nœud $v \in V$ est le nombre d'arcs provenant du nœud v . Et le *degree*

d'un nœud $v \in V$, dénoté par $\deg(v)$, est la somme de son *in-degree* et de son *out-degree*.

Un *graphe non dirigé* ou simplement *graphe*, noté $G = (V, E)$, est constitué d'un ensemble de nœuds V et d'un ensemble de paires non ordonnées de nœuds, appelées *arêtes*, $E \subseteq V \times V$. Pour mieux distinguer un graphe d'un digraphe, nous le désignerons par G^u tout au long de cette thèse. L'arête entre les nœuds u et v est notée $uv \in E$. Notamment, nous adoptons la même notation pour les arcs et les arêtes, ce qui permet au contexte de clarifier la référence voulue. Comme les arêtes de G^u ne sont pas dirigées, une arête entre les nœuds u et v peut être désignée indifféremment par uv ou vu . Le voisinage d'un nœud $v \in V$, noté $N_G(v)$, est le sous-ensemble des nœuds $\{u \in V : uv \in E\}$. Nous pouvons simplement écrire $N(v)$ si le graphe est clair dans le contexte. Le *degré* d'un nœud $v \in V$, noté $\deg(v)$, est le nombre d'arêtes incidentes à v , c'est-à-dire $\deg(v) = |N(v)|$.

Il peut arriver que l'on doive considérer le graphe non orienté sous-jacent G^u d'un graphe orienté G . Le graphe sous-jacent G^u a le même ensemble de sommets que G , et il y a une arête uv dans G^u s'il existe un arc uv ou vu dans G .

Un *chemin* dans un digraphe (resp. graphe) est une séquence de nœuds $(v_1, v_2, \dots, v_n)$ dans laquelle chaque paire consécutive (v_i, v_{i+1}) est connectée par un arc (resp. une arête) $v_i v_{i+1}$, et chaque nœud v_i pour tout $1 \leq i \leq n$ est distinct. Un *cycle* est un chemin dans lequel le premier et le dernier nœud sont également reliés par un arc (resp. une arête).

Graphe Pondéré Un *digraphe* (resp. *graphe*) *pondéré* est un digraphe $G = (V, A)$ (resp. graphe $G^u = (V, E)$) avec une fonction de poids $w : A \rightarrow \mathbb{R}_+$ (resp. $w : E \rightarrow \mathbb{R}_+$), qui attribue une valeur réelle à chaque arc (resp. arête) dans G . Dans le contexte des réseaux de télécommunications, les poids correspondent souvent à la latence, à la largeur de bande ou à d'autres attributs propres au réseau.

Plus Court Chemin Dans un digraphe (resp. graphe) pondéré, le *plus court chemin* entre deux nœuds u et v est le chemin dont la somme des poids des arcs (resp. arêtes) est la plus faible parmi tous les chemins possibles reliant u et v . La longueur du plus court chemin est notée $\text{dist}(u, v)$. Notez que plusieurs plus courts chemins peuvent exister entre les nœuds u et v .

Clique Une *clique* dans un graphe $G^u = (V, E)$ est un ensemble de nœuds $C \subseteq V$ tel que pour chaque paire de nœuds $u, v \in C$, $uv \in E$. Une *clique maximale* est une clique avec le plus grand nombre possible de nœuds.

Ordre des Nœuds Un ordre des nœuds π dans un graphe $G^u = (V, E)$ est une séquence de nœuds $v_1, v_2, \dots, v_n$, où $n = |V|$, et son inverse π^{-1} attribue une valeur entière unique, appelée *rank* $\pi^{-1}(v) \in \{1, 2, \dots, n\}$ à chaque nœud $v \in V$.

Ordre d'Élimination Parfaite Un ordre des nœuds $\pi = v_1, v_2, \dots, v_n$ d'un graphe G^u est un *ordre d'élimination parfaite* si pour chaque $i = 1, \dots, n$, v_i et les nœuds $v_j \in N(v_i)$ avec $j > i$ forment une clique.

Graphe Cordal Un *graphe cordal* est un graphe non orienté dans lequel chaque cycle d'au moins quatre nœuds possède une corde, c'est-à-dire une arête qui ne fait pas partie du cycle mais qui relie deux nœuds du cycle.

Un graphe est cordal si et seulement s'il possède un *ordre d'élimination parfait*. L'*ordre d'élimination parfait* n'est pas unique dans un graphe cordal [28].

Supergraphe Cordal Etant donné un ordre de nœuds $\pi = v_1, v_2, \dots, v_n$ d'un graphe G^u , le *supergraphe cordal*, noté G_π^* , est le graphe obtenu en éliminant tous les nœuds de G^u dans l'ordre selon π de la manière suivante. Chaque fois qu'un nœud v_i est éliminé, des arêtes sont insérées entre chaque paire de ses voisins restants, puis toutes ses arêtes incidentes vers v_i sont supprimées. Par F_{v_i} nous désignons l'ensemble des arêtes insérées lors de l'élimination de v_i . Le graphe original G^u avec toutes les arêtes supplémentaires insérées lors de l'élimination de tous les nœuds $F = \bigcup_i F_{v_i}$ est le *supergraphe cordal* G_π^* .

Arbre d'Élimination L'arbre d'élimination d'un graphe $G^u = (V, E)$ par rapport à un ordre d'élimination $\pi = v_1, v_2, \dots, v_n$ est un arbre orienté, désigné par $T = (V, E_T)$, dérivé du graphe G^u . Les nœuds de T sont les mêmes que dans G^u , et les arcs de T sont déterminés comme suit. Pour chaque nœud v_i dans V , nous définissons son nœud parent dans T comme le premier nœud v_j apparaissant après v_i dans l'ordre d'élimination π de sorte que l'arête $v_i v_j$ existe dans le graphe $G_{v_i}^*$ (qui est le graphe après l'élimination des nœuds jusqu'à v_i selon π). Si un tel nœud v_j existe, alors nous ajoutons un arc de v_i à v_j dans T . Le nœud sans parent est la racine de l'arbre d'élimination. La figure 8.1 montre un exemple de graphe et l'arbre d'élimination correspondant.

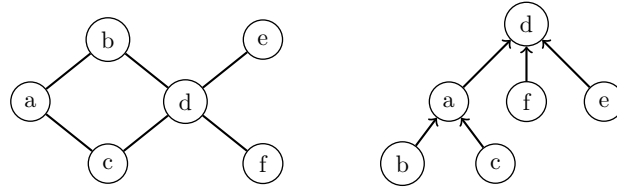


Figure 8.1: Exemple d'arbre d'élimination (à droite) obtenu à partir d'un graphe (à gauche) en respectant l'ordre d'élimination suivant : b, c, f, a, e, d .

Profondeur de l'Arbre La *profondeur de l'arbre* d'un graphe G^u est la profondeur minimale des arbres d'élimination sur tous les ordres d'élimination.

Décomposition Arborescente Une *décomposition arborescente* d'un graphe $G^u = (V, E)$ est une paire (T, X) où $T = (I, F)$ est un arbre et $X = \{X_i | i \in I\}$ est une famille de sous-ensembles de V (également connus sous le nom de sacs), telle que :

- L'union de tous les sacs est égale à l'ensemble des nœuds de G : $\bigcup X_i = V$.
- Pour chaque arête $uv \in E$, il existe un sac X_i tel que $u, v \in X_i$.

- Pour tous les $i, j, k \in I$, si j se trouve sur l'unique chemin entre i et k dans T , alors $X_i \cap X_k \subseteq X_j$.

La largeur arborescente (tree-width) d'une décomposition arborescente est la taille de son plus grand sac X_i moins un.

Largeur Arborescente La *largeur arborescente* de G^u , désignée par $\text{tw}(G^u)$, est la largeur minimale de toutes les décompositions arborescentes possibles de G^u . Un graphe avec une petite *tree-width* présente une structure arborescente, ce qui permet souvent d'obtenir des algorithmes plus efficaces pour résoudre divers problèmes. En particulier, un arbre (ou une forêt) a une largeur de 1.

1.4 Préliminaires de l'Ingénierie du Trafic

1.4.1 Réseau

Un *réseau* est représenté par un graphe bidirectionnel $G = (V, A)$, où V est l'ensemble des nœuds et A l'ensemble des arcs. Chaque nœud de V représente un appareil ou un commutateur dans le réseau, tandis que chaque arc de A représente un lien entre deux nœuds. Pour chaque arc $uv \in A$, la *capacité* $c_{uv} \in \mathbb{R}_+$ désigne le débit maximal auquel les données peuvent être transmises par cet arc. Une *métrique* $w \in \mathbb{Z}_+^{|A|}$, également connue sous le nom de poids administratif, est un nombre entier positif attribué à chaque arc du réseau. Les métriques jouent un rôle central dans la détermination des chemins de routage pour la transmission de données, car les protocoles de routage par le plus court chemin utilisent ces valeurs pour calculer et sélectionner les chemins les plus courts dans G en fonction des métriques.

Demande Nous désignons par K l'ensemble des demandes (également appelé *matrice de trafic*) à acheminer dans G . Une demande $k \in K$ est définie par un triplet (s^k, t^k, d^k) où $s^k \in V$ est le nœud source de la demande, $t^k \in V$ est le nœud cible, et $d^k \in \mathbb{R}_+$ est le volume de la demande.

Charge La *charge* fait référence à la quantité de trafic qui est transmise par un arc particulier par rapport à sa capacité maximale. Il s'agit d'un paramètre essentiel pour évaluer les performances d'un réseau, car une charge élevée peut entraîner une congestion, une augmentation de la latence et une réduction de la qualité de service. Formellement, nous définissons la *charge* d'un arc $uv \in A$ induit par la métrique w , dénotée par $\text{load}(uv, w)$, comme la quantité du flux de trafic f_{uv} traversant l'arc uv par rapport à sa capacité c_{uv} : $\text{load}(uv, w) = \frac{f_{uv}}{c_{uv}}$. La *charge* est aussi souvent appelée *utilisation du lien* dans d'autres études.

Congestion du Réseau La *congestion du réseau* désigne la *charge* la plus élevée parmi tous les arcs du réseau en ce qui concerne une métrique w , qui est un indicateur important de la performance du réseau. Il peut être formellement défini comme $\text{cong}(w) = \max_{uv \in A} \text{load}(uv, w)$.

Dans d'autres études, la *congestion du réseau* est également appelé *utilisation maximale de l'arc* (*MLU*).

Delay *Delay*, également connu sous le nom de *latence*, est le temps nécessaire à un paquet de données pour voyager de sa source à sa destination à travers le réseau. *Delay* est une mesure de performance cruciale, car une latence élevée peut avoir un impact négatif sur l'expérience de l'utilisateur et la qualité du service, en particulier pour les applications en temps réel ou sensibles aux délais.

1.4.2 Protocole de Routage

Un *protocole de routage* dans les réseaux de télécommunications est un ensemble normalisé d'algorithmes et de procédures employés par les dispositifs de réseau, tels que les routeurs et les commutateurs, pour déterminer les chemins les plus efficaces et les plus fiables pour la transmission des paquets de données. Ces protocoles facilitent la communication dynamique entre les nœuds du réseau, en permettant l'échange continu d'informations topologiques et en s'adaptant aux changements du réseau, tels que les pannes d'arc et la congestion.

Protocole OSPF Le protocole *Open Shortest Path First (OSPF)* [72] est un *protocole de routage* largement utilisé dans les réseaux de télécommunication, conçu pour faciliter la détermination efficace et dynamique des itinéraires optimaux pour la transmission de paquets de données au sein d'un système autonome (AS). OSPF s'appuie sur l'algorithme du plus court chemin de Dijkstra pour calculer les chemins de routage sur la base de métriques.

Protocole IS-IS Le protocole *Intermediate System-to-Intermediate System (IS-IS)* [41] est un autre *protocole de routage* largement utilisé pour la transmission de paquets de données au sein d'un système autonome (AS). Fonctionnant sur la couche réseau du modèle OSI (Open Systems Interconnection), IS-IS se distingue d'OSPF, un protocole basé sur IP, par son adhésion au schéma d'adressage OSI et l'utilisation de points d'accès au service réseau (NSAP) pour l'adressage. En outre, IS-IS utilise une structure plus extensible avec un encodage de type-longueur-valeur (TLV), ce qui permet une plus grande adaptabilité et une plus grande facilité d'intégration avec d'autres protocoles, ainsi que la gestion transparente du routage IPv4 et IPv6. Par la suite, IS-IS applique également l'algorithme du plus court chemin de Dijkstra pour calculer les chemins les plus favorables sur la base des métriques d'arc, ce qui favorise la fiabilité du réseau, la convergence rapide et l'utilisation efficace des ressources.

ECMP *Equal-Cost Multi-Path (ECMP)* est une stratégie de routage employée dans les réseaux de télécommunication pour optimiser l'utilisation des ressources réseau disponibles et améliorer la performance globale en distribuant le trafic sur plusieurs chemins à coût égal entre une source et une destination. En exploitant des chemins parallèles avec la même métrique de coût, l'ECMP permet un équilibrage efficace de la congestion, améliorant la fiabilité du réseau et la résilience contre la congestion et les défaillances d'arc. La figure 8.2 présente un exemple de règle ECMP.

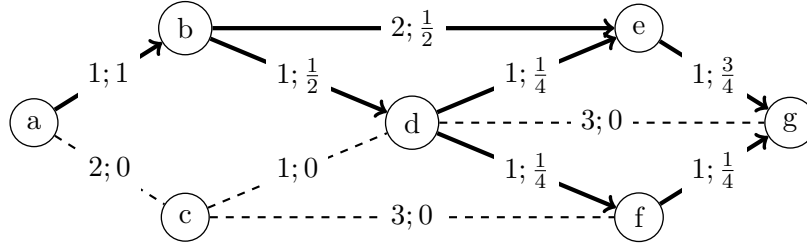


Figure 8.2: Illustration d'un routage par le plus court chemin avec la règle ECMP. Dans cette figure, nous supposons que les capacités sont unitaires et qu'une demande k avec un volume de trafic $d^k = 1$ doit être acheminée de $s^k = a$ à $t^k = g$. Une étiquette $w_{uv}; load(uv, w)$ est associée à chaque arc $uv \in A$.

1.5 Dataset

Dans cette thèse, nous avons examiné l'ensemble de données populaire SNDLib [73] des études précédentes, qui contient des graphes du monde réel de différentes tailles. La Figure 8.3 montre quelques exemples d'instances SNDLib.

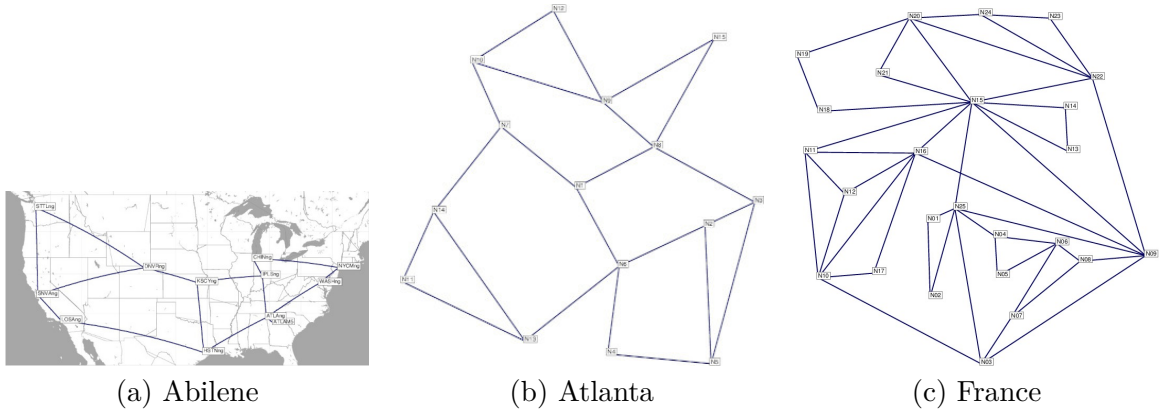


Figure 8.3: Exemples d'instances SNDLib

Certaines instances de Topology Zoo [63] sont également utilisées dans cette thèse. Elles sont couramment utilisées dans diverses études du domaine. Topology Zoo contient des centaines de topologies réseau, mais souvent de petites tailles, généralement pas plus de quelques centaines de nœuds. La Figure 8.4 montre deux instances du Topology Zoo.

Au cours des dernières décennies, de nombreux générateurs de graphes ont été proposés [102, 81, 69]. La première famille de générateurs de graphes que nous avons utilisée est celle des graphes Random et des graphes Waxman, couramment utilisée dans de nombreuses recherches précédentes [51, 17, 82]. Pour les graphes Random, les nœuds sont générés dans un carré unitaire et les arcs sont générés selon une probabilité constante. Quant aux graphes Waxman, la probabilité d'avoir un arc entre deux nœuds est :

$$p(u, v) = \alpha e^{\frac{-d(u, v)}{\beta d_{\max}}}$$

où $d(u, v)$ est la distance euclidienne entre les nœuds u et v , $d_{\max}$ représente la distance

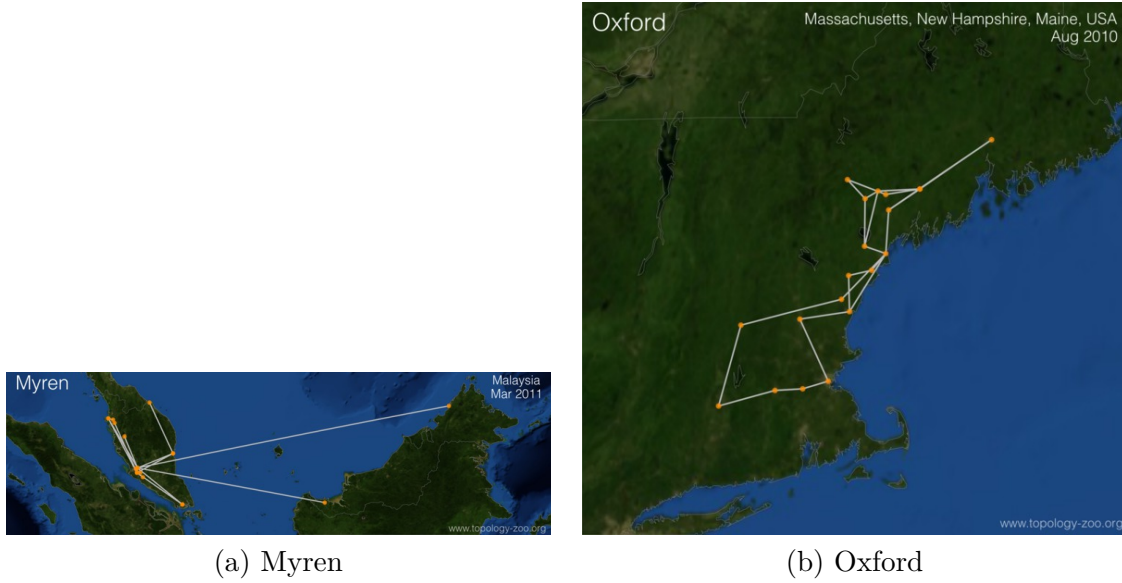


Figure 8.4: Exemples d'instances de Topology Zoo

euclidienne maximale entre deux nœuds quelconques, α et β contrôlent la densité du graphe. Les capacités de tous les arcs sont fixées à 1000.

Nous avons également utilisé le générateur de graphes *IGen* [81], qui possède un modèle de réseau hiérarchique, capable de générer des réseaux à deux niveaux de grande échelle avec une topologie d'infrastructure et une topologie de nœuds d'accès. La Figure 8.5 montre une instance d'*IGen*.

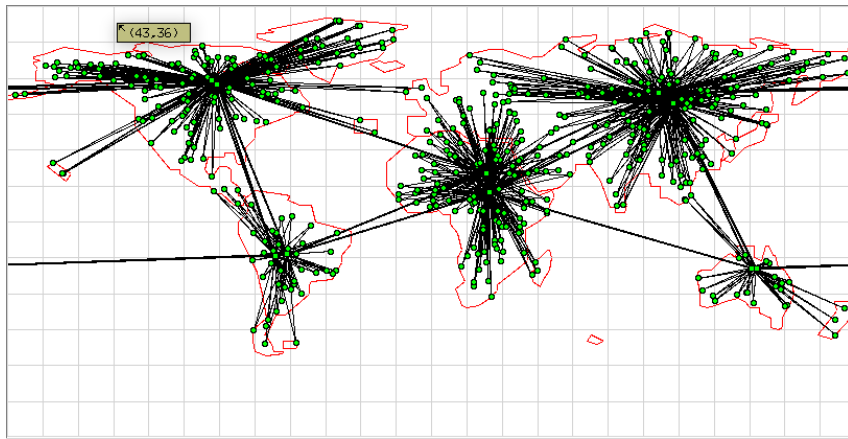


Figure 8.5: Un exemple d'instance IGen

Le quatrième générateur de graphes est l'un des modèles de réseau sans échelle, *Inet-3.0* [102]. Des études ont montré que pour de nombreux réseaux, la distribution du degré des nœuds suit une loi de puissance [43]. Et les réseaux sans échelle ont été proposés par [6] comme des modèles qui suivent les distributions de loi de puissance dans la connectivité des nœuds du réseau. *Inet* est donc l'un des générateurs de topologie qui produit des réseaux synthétiques suivant de manière plus précise les distributions de degrés d'Internet. La Figure 8.6 montre un exemple d'instance

Inet avec 3040 nœuds qui est utilisée dans cette thèse.

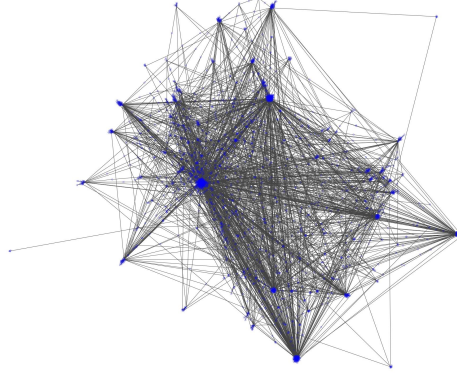


Figure 8.6: Un exemple d'instance Inet

Pour tous les modèles de générateurs de graphes, nous générons du trafic entre deux nœuds avec une probabilité fixe, et le volume de la demande d^k pour la demande k entre les nœuds s^k et t^k est :

$$d^k = \alpha S_{s^k} T_{t^k} C_{(s^k, t^k)} e^{\frac{-d(s^k, t^k)}{2d_{\max}}}$$

où $S_u, T_u \in [0, 1]$ sont deux nombres aléatoires pour le nœud u , et $C_{(s^k, t^k)} \in [0, 1]$ est un nombre aléatoire pour le couple (s^k, t^k) , $d(s^k, t^k)$ est la distance euclidienne entre les nœuds s^k et t^k , $d_{\max}$ est la distance euclidienne maximale entre deux nœuds quelconques.

La Table 8.1 montre les informations des graphes que nous avons utilisés dans nos expériences.

Table 8.1: Tableau résumant les différentes instances utilisées dans nos expériences. Les colonnes sont : le nom du réseau (“name”), le nombre de noeuds (“ $|V|$ ”), le nombre d’arcs (“ $|A|$ ”), le nombre de demandes (“ $|K|$ ”), le volume total des demandes (“ $\sum d^k$ ”) et le degré maximum (“max degree”).

nom	$ V $	$ A $	$ K $	$\sum d^k$	degré maximal
abilene	12	30	132	3000002	4
atlanta	15	44	210	136726	4
newyork	16	98	240	1774	11
france	25	90	300	99830	10
norway	27	102	702	5348	6
nobel-us	14	42	91	5420	4
nobel-ger	17	52	121	660	6
nobel-eu	28	82	378	1898	5
brain	161	332	14311	12.3e9	74
rand50	50	132	2450	81419	12
rand100	100	278	9900	269535	14
wax50	50	142	2450	85150	12
wax100	100	284	9900	331386	16
Myren	37	78	1332	44880	16
Oxford	20	52	303	13123	5
SwitchL3	42	126	1353	35753	10
igen50	50	202	1221	34517	21
igen100	100	400	2399	83248	35
igen500	500	2018	12565	377072	109
igen1000	1000	4050	14908	487523	105
igen2000	2000	8056	14918	488331	194
igen5000	5000	20058	19863	656416	545
inet3040	3040	9586	4540	144103	685
inet4000	4000	13288	7920	266207	856
inet5000	5000	17380	12717	414545	1025

2 Problème de Métriques

2.1 Introduction

L’un des problèmes les plus difficiles auxquels sont confrontés les administrateurs de réseau est la détermination de politiques de routage efficaces qui induisent un niveau élevé de QoS sur les réseaux opérationnels tout en étant robustes aux variations inhérentes du trafic dans le temps et aux défaillances du réseau. Malgré les promesses du protocole de transfert Multi-Protocol Label Switching (MPLS) [68], la plupart des réseaux actuels reposent fortement sur le calcul des chemins les plus courts (Open Shortest Path First (OSPF) [72], Intermediate System to Intermediate System (IS-IS) [41], Segment Routing (SR) [46] ou Flex-Algo [79]) pour déterminer ces politiques de routage.

Dans ces protocoles de routage, les gestionnaires de réseau contrôlent le flux de données en attribuant des poids administratifs aux liens du réseau. Par conséquent, chaque paquet est acheminé de son origine à sa destination le long des chemins les plus courts déterminés par

ces poids. Bien que cette méthode offre l'avantage d'une gestion simplifiée, elle manque de contrôle précis sur les chemins de routage élus, car les modifications ne peuvent être apportées qu'indirectement en modifiant les poids. En conséquence, les gestionnaires de réseau sont confrontés au défi d'identifier un ensemble de métriques qui induit des chemins de routage capables de minimiser la congestion tout en maintenant un niveau élevé de Qualité de Service (QoS) au sein des réseaux opérationnels. Malheureusement, résoudre ce problème est computationnellement exigeant.

Dans cette thèse, nous nous intéressons à l'un des problèmes d'optimisation de réseau liés à cette question : Étant donné un graphe bidirigé et un ensemble de demandes, la tâche consiste à trouver un ensemble de poids de telle sorte que les demandes acheminées le long des chemins les plus courts induits génèrent une congestion minimale, *i.e.*, la valeur ratio maximale du trafic total passant par un arc par rapport à la capacité de cet arc est minimisée.

2.2 Définition du Problème

Nous considérons un graphe bidirigé qui est un digraphe où, pour tout arc uv , l'arc inverse vu est également présent. Étant donné un graphe bidirigé $G = (V, A)$, chaque sommet $v \in V$ correspond à un routeur tandis qu'un arc uv correspond à un lien entre les routeurs u et v . À chaque arc uv est associée une capacité notée c_{uv} . Soit K un ensemble de demandes ou de commodités à router dans G . Chaque demande $k \in K$ est définie par une paire de sommets s^k et t^k représentant la source et le target de k , un volume de trafic d^k à router de s^k à t^k . Une telle demande k sera notée par le triplet (s^k, t^k, d^k) .

Étant donnée une métrique $w \in \mathbb{Z}_+^{|A|}$, chaque demande $k \in K$ est routée le long des chemins les plus courts entre s^k et t^k . S'il existe plusieurs chemins les plus courts reliant les extrémités de k , le volume de trafic d^k est réparti uniformément entre ces chemins selon la règle ECMP. Plus précisément, le volume de trafic qui atteint un nœud $v \in V$ doit être réparti équitablement entre tous les arcs quittant v et appartenant aux chemins les plus courts vers la destination t^k . Nous définissons alors la *charge* d'un arc uv induite par w , notée $load(uv, w)$, comme la quantité de trafic traversant l'arc uv par rapport à sa capacité. La congestion du réseau $cong(w)$ d'une métrique donnée w , également connue sous le nom d'utilisation maximale des arcs (MLU), est définie par $\max_{uv \in A} load(uv, w)$, soit la charge maximale sur tous les arcs.

Nous sommes maintenant en mesure de définir le problème d'optimisation étudié dans ce chapitre. Le problème de Métriques, également connu sous le nom de MINIMUM CONGESTION SHORTEST PATH ROUTING (MIN-CON-SPR) dans d'autres recherches [10], consiste à trouver une métrique $w \in \mathbb{Z}_+^{|A|}$ et les chemins de routage induits par ces poids de telle sorte que la congestion du réseau $cong(w)$ soit minimale. Le problème de Métriques peut être défini formellement comme suit :

PROBLÈME DE MÉTRIQUES

Entrée : Un graphe bidirigé $G = (V, A)$, où chaque arc uv a une capacité $c_{uv} \geq 0$ et un ensemble de commodités K défini pour chaque $k \in K$ par le triplet (s^k, t^k, d^k) .

Sortie : Une métrique $w \in \mathbb{Z}_+^{|A|}$ de congestion minimale $cong(w)$.

Dans cette thèse, nous considérons également le problème de Métriques avec certaines ou

toutes les contraintes supplémentaires suivantes :

Unicité : Dans cette contrainte, nous exigeons que chaque demande soit routée le long d'un chemin le plus court unique.

Délai : Cette contrainte exige que les chemins de routage aient une longueur au plus égale à la longueur d'un chemin le plus court (s^k, t^k) (en termes de nombre d'arcs) plus une constante $c \in \mathbb{N}_+$.

D'un point de vue opérationnel, la contrainte d'unicité est parfois requise par le gestionnaire de réseau pour surveiller plus facilement le flux circulant dans le système. Nous appelons le problème avec contrainte d'unicité la version *unsplittable* du problème, et la version sans contrainte d'unicité la version *splittable*. En plus de minimiser la congestion, la contrainte de délai garantit un certain niveau de QoS en ce qui concerne la latence des réponses aux demandes faites par les clients.

La figure 8.7 illustre un exemple d'optimisation pour le Problème de Métriques. Dans cet exemple, nous supposons des capacités unitaires pour les arcs et que deux demandes avec un volume de trafic unitaire $d^k = 1$ doivent être acheminées du nœud a au nœud e et du nœud b au nœud e , respectivement. Chaque arc $uv \in A$ est associé à une étiquette contenant le poids et le volume, $w_{uv}; load(uv, w)$. Le réseau à gauche montre un ensemble de poids qui résulte en une valeur de congestion plus élevée sur le réseau, car les deux demandes sélectionnent le chemin le plus court selon les poids et passent par l'arc ce . En ajustant les poids sur les arcs ce et de , le chemin le plus court pour la demande d^{ae} ne traverse plus l'arc ce , conduisant à une valeur de congestion plus basse pour le réseau affiché sur la droite.

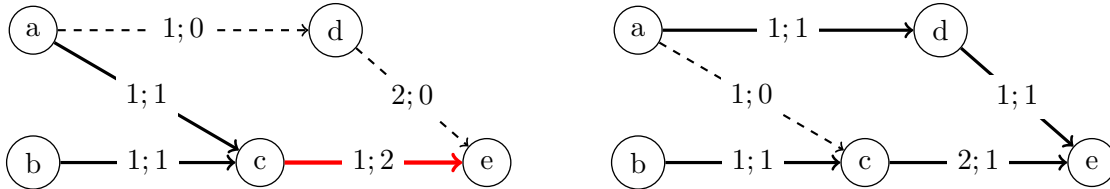


Figure 8.7: Illustration du Problème de Métriques. Dans cette figure, nous supposons que les capacités sont unitaires et que deux demandes avec un volume de trafic unitaire $d^k = 1$ doivent être acheminées du nœud a à e et du nœud b à e , respectivement. Une étiquette avec le poids et le volume $w_{uv}; load(uv, w)$ est associée à chaque arc $uv \in A$.

3 Problème d'Optimisation des Waypoints

3.1 Introduction

Avec la demande croissante de services de communication de haute qualité et efficaces, les réseaux de télécommunication modernes doivent être capables de s'adapter et d'optimiser leurs performances en temps réel. Le Segment Routing (SR) est une technologie prometteuse qui permet un ingénierie de trafic flexible et efficace en encodant directement les instructions de routage dans l'en-tête du paquet. Ces dernières années, le problème d'Optimisation des Waypoints (WPO)

dans le contexte de l'Ingénierie de Trafic Segment Routing (SR-TE) a suscité une attention considérable en raison de son potentiel à optimiser davantage les performances du réseau.

Dans cette thèse, nous proposons de nouvelles méthodes basées sur l'Apprentissage par Renforcement et la Recherche Locale, en comparant leurs performances avec les techniques de pointe. On discute également des techniques essentielles pour réduire l'espace de recherche pour le problème de WPO et de leurs applications pratiques, permettant des solutions plus efficaces et performantes.

Le problème de WPO est essentiel à résoudre car il permet aux opérateurs de réseau d'imposer des politiques de routage spécifiques et d'optimiser les performances du réseau, en garantissant que les flux de trafic traversent les nœuds désignés du réseau. En sélectionnant stratégiquement les waypoints, il est possible de minimiser la congestion du réseau, d'améliorer l'utilisation des ressources et d'améliorer la qualité globale du service fourni aux utilisateurs finaux. À mesure que les réseaux de télécommunication continuent d'évoluer et de gagner en complexité, le développement de solutions efficaces et performantes pour le Problème de WPO devient de plus en plus crucial pour maintenir des services de communication de haute qualité et répondre aux demandes sans cesse croissantes des utilisateurs.

3.2 Segment Routing

Le Segment Routing (SR) est une approche relativement nouvelle du routage de réseau qui offre beaucoup de flexibilité et de scalabilité. Avec le routage traditionnel, les paquets sont envoyés à travers le réseau en suivant un chemin déterminé par chaque routeur, dans ce qu'on appelle le transfert saut par saut (hop-by-hop forwarding). Mais avec le SR, un chemin est prédéterminé en utilisant un ensemble d'étiquettes, également appelées segments, qui spécifient la route que les paquets doivent prendre à travers le réseau. Les étiquettes sont stockées sous forme de pile dans les paquets et sont retirées à chaque fois que les paquets atteignent le routeur ou le lien target.

L'un des grands avantages du SR est qu'il offre un moyen flexible pour les paquets d'exploiter des chemins qui ne sont pas forcément les plus courts dans les réseaux de télécommunication. Une autre technique, le Multiprotocol Label Switching (MPLS) [44], permet également aux paquets de passer par des chemins qui ne sont pas forcément les plus courts, mais elle doit maintenir un état supplémentaire qui est difficile à scaler [105]. Mais le SR ne nécessite de maintenir que les étiquettes comme état, ce qui le rend plus facile à scaler.

Il existe deux principales variantes du segment routing : le Segment Routing sur MPLS (SR-MPLS) [7] et le Segment Routing sur IPv6 (SRv6) [45]. Bien que les détails spécifiques de la mise en œuvre de ces deux variantes soient importants pour comprendre leur fonctionnement dans les réseaux réels, notre attention principale se porte sur l'abstraction algorithmique pour traiter le problème de SR-TE.

Le problème de SR-TE traite des défis d'optimisation rencontrés lors de la mise en œuvre du SR pour améliorer les performances, la fiabilité et l'utilisation des ressources du réseau. Le SR-TE se concentre sur la détermination des chemins de transmission de données les plus efficaces au sein d'un réseau en utilisant le SR, tout en respectant diverses contraintes, notamment la

latence, l'utilisation des liens et la répartition des charges. Pour diriger le trafic dans le réseau, une liste ordonnée de segments est distribuée à chaque demande. Ainsi, les listes de segments jouent un rôle important dans le SR-TE.

Un défi spécifique dans ce contexte est le problème de WPO. L'objectif du WPO est d'identifier la liste optimale de waypoints pour router les demandes à travers le réseau, dans le but de minimiser la congestion et d'assurer simultanément un niveau élevé de Qualité de Service (QoS).

Pour atteindre cet objectif, les algorithmes WPO doivent prendre en compte divers facteurs du réseau pour calculer les chemins de routage les plus efficaces pour chaque demande de trafic. En optimisant efficacement la sélection des waypoints, le WPO vise à améliorer les performances globales du réseau en équilibrant la distribution des charges, en réduisant la congestion et en améliorant l'utilisation des ressources.

La Figure 8.8 montre un exemple du segment routing pour réduire la congestion du réseau. Dans cet exemple, nous supposons des capacités unitaires et une métrique unitaire pour les arcs du réseau. Nous supposons également que deux demandes avec un volume de trafic unitaire $d^k = 1$ doivent être routées du nœud a au nœud f et du nœud a au nœud g , respectivement. Les nœuds rose, vert et cyan représentent respectivement la source, la target et le waypoint. À gauche, sans segment routing, les deux demandes passent par les arcs ab et be , provoquant une valeur de congestion plus élevée. Tandis qu'à droite, en utilisant le segment routing et en ajoutant le nœud c comme waypoint pour la demande de a à g , le trafic est réparti uniformément entre le nœud a et e , conduisant ainsi à une valeur de congestion plus faible. Cet exemple démontre la flexibilité du segment routing en manipulant la route d'une seule demande, nous fournissant un excellent outil pour l'optimisation du trafic réseau.

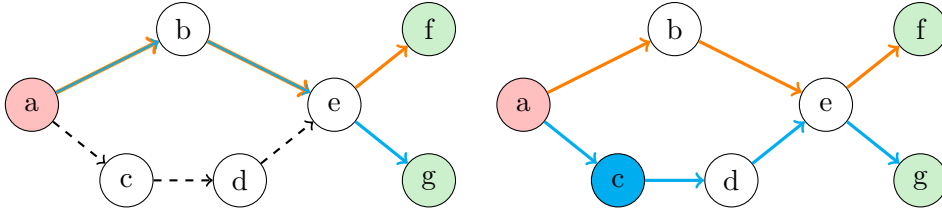


Figure 8.8: Illustration du problème d'optimisation des waypoints. Dans cette figure, nous supposons que les capacités et la métrique des arcs du réseau sont unitaires. Nous supposons également que deux demandes avec un volume de trafic unitaire $d^k = 1$ doivent être acheminées du nœud a vers f et du nœud a vers g , respectivement. Les nœuds rose, vert et cyan représentent respectivement la source, la target et le waypoint.

Dans cette thèse, nous nous concentrons sur le problème de WPO. Le problème vise à optimiser le choix des waypoints pour chaque demande afin de minimiser la congestion dans le réseau en optimisant les waypoints de la demande. Pour définir formellement le problème, nous devons introduire quelques notions essentielles qui seront utiles pour comprendre le problème.

Segment Un segment est une instruction codée dans l'en-tête du paquet qui représente une sous-structure topologique au sein du réseau, qui peut représenter un sous-chemin dans le réseau. Il peut être désigné par S_i , où i est un index dans la séquence ordonnée des segments d'un chemin

SR. Il existe de nombreux types de segments SR, les deux les plus utilisés étant les segments d'adjacence et les segments de préfixe.

SR-path Un SR-path est défini comme une séquence ordonnée $\langle x_1, x_2, \dots, x_n \rangle$, où chaque x_i représente un segment. Un chemin SR pour une demande $k \in K$ commence à la source s^k et se termine à la cible t^k . En raison de l'ECMP, un chemin SR peut représenter un ensemble de chemins dans le réseau original.

Waypoint Un waypoint désigne un nœud spécifique ou un ensemble de nœuds qu'un paquet doit traverser le long de sa trajectoire SR. Formellement, un waypoint pour une demande $k \in K$ peut être désigné par ω_k^i , où i est un index dans la séquence ordonnée des waypoints d'un chemin SR. Lors de la construction d'un chemin SR, les waypoints sont placés de manière stratégique dans la séquence des segments, de sorte que le paquet passe par ces nœuds désignés.

Ces notions étant introduites, nous pouvons maintenant définir formellement le problème de WPO.

Etant donné un graphe bidirectionnel $G = (V, A)$ représentant une topologie de réseau, où V est l'ensemble des nœuds et A l'ensemble des arcs, notons $|V|$ le nombre de nœuds et $|A|$ le nombre d'arcs. Soit K un ensemble de demandes à acheminer dans G . Chaque demande $k \in K$ est définie par une paire de nœuds $s^k \in V$ et $t^k \in V$ représentant la source et la cible de k , un volume de demande d^k à acheminer de s^k à t^k . Une telle demande k sera désignée par le triplet (s^k, t^k, d^k) . L'objectif du problème WPO est de déterminer un ensemble de waypoints $\mathcal{W}_k = \{\omega_k^1, \omega_k^2, \dots, \omega_k^{m^k}\}$ pour chaque demande k dans le réseau, où m^k est le nombre de waypoints pour la demande k , de telle sorte que la congestion du réseau soit minimisée. La congestion du réseau est définie comme l'utilisation maximale de l'arc (MLU) dans le réseau, désignée par $cong$.

Le problème d'optimisation des waypoints vise à trouver l'ensemble optimal de waypoints $\mathcal{W}$ qui satisfait aux conditions suivantes :

- Pour chaque demande $k \in K$, $\mathcal{W}_k$ est un sous-ensemble de tous les nœuds possibles du réseau, i.e., $\mathcal{W}_k \subseteq V$.
- Pour chaque demande de trafic entre s^k et t^k , un SR-chemin $\mathcal{P} = \langle x_1, x_2, \dots, x_n \rangle$ est généré de manière à inclure tous les waypoints dans $\mathcal{W}_k$ dans l'ordre donné.
- Minimiser la congestion du réseau $cong$, tout en respectant les protocoles de routage données et les contraintes du réseau.
- Le SR-chemin $\mathcal{P}$ doit être sans boucle, garantissant que le paquet atteint la destination sans traverser de cycles dans le réseau.

Dans cette thèse, nous nous concentrons sur un cas particulier du problème de WPO qui permet au plus un waypoint par demande. Plus précisément, pour chaque demande k , au plus un waypoint est autorisé. Cette restriction simplifie grandement le processus d'optimisation et

le rend plus facile à gérer pour des instances de réseau à grande échelle. Par souci de simplicité, nous utilisons dans ce qui suit la notation $\omega_k \in V$ comme le seul waypoint pour la demande $k \in K$.

Nous définissons le problème comme suit.

OPTIMISATION DES WAYPOINTS

Input: Un réseau $G = (V, A)$, où chaque arc uv a une capacité $c_{uv} \geq 0$, un ensemble de demandes K défini pour chaque $k \in K$ par le triplet (s^k, t^k, d^k) , et une métrique $w \in \mathbb{Z}_+^{|A|}$.

Sortie: Un waypoint $\omega_k \in V$ pour chaque demande $k \in K$ de congestion minimale *cong*.

Dans notre cas, nous considérons que la métrique w est la métrique unitaire, i.e., $w_{uv} = 1$ pour tout $uv \in A$.

4 Nos Contributions

Dans cette thèse, nous présentons plusieurs heuristiques et intuitions empiriques pour améliorer la performance de l'algorithme visant à résoudre les problèmes d'ingénierie du trafic à grande échelle.

Le second chapitre de cette thèse se concentre sur les préliminaires nécessaires à la compréhension des problèmes traités et des principales approches utilisées dans notre travail, en particulier les approches basées sur Monte Carlo. Ce chapitre introduit les fondamentaux de la théorie des graphes, de l'ingénierie du trafic et des algorithmes Monte Carlo.

Le troisième chapitre se consacre à l'utilisation des approches Monte Carlo pour aborder le Problème de Métriques. Nous fournissons une introduction formelle au Problème de Métriques dans ce chapitre, accompagnée d'une explication détaillée de la manière dont ce problème est modélisé en utilisant les approches Monte Carlo. Nous évaluons trois différentes approches et proposons des techniques générales pour augmenter la performance de ces approches pour les grandes instances.

Dans le quatrième chapitre, nous traitons du goulet d'étranglement de la simulation pour le problème d'ingénierie du trafic, qui est le calcul du chemin le plus court et de la congestion pour de grands réseaux de télécommunication aux métriques constamment changeantes. Nous proposons un algorithme basé sur la Hiérarchie de Contraction Personnalisable (CCH) [34], qui montre une amélioration significative par rapport aux approches traditionnelles. Lorsqu'il est appliqué aux approches proposées dans le chapitre 3, le nouveau système produit des résultats nettement supérieurs. Cette approche peut également être utilisée dans d'autres algorithmes d'optimisation basés sur la simulation pour le Problème de Métriques, ainsi que dans d'autres simulations nécessitant des changements de métriques fréquents dans le réseau.

Dans le cinquième chapitre, nous présentons une amélioration de la meilleure approche Monte Carlo que nous avons testée dans le troisième chapitre : Nested Rollout Policy Adaptation (NRPA) [85]. Ce chapitre traite d'un problème critique fréquemment rencontré dans le problème d'optimisation : échapper aux minima locaux. Nous proposons l'utilisation de la théorie de l'arrêt optimal pour démarrer NRPA, en estimant la performance de l'exécution NRPA à un stade précoce et en éliminant les instances sous-performantes. L'algorithme résultant, nommé Meta-

NRPA, montre une performance exceptionnelle sur trois problèmes d'optimisation combinatoire, dont le Problème de Métriques.

Dans le sixième chapitre, nous nous concentrons sur le problème de WPO, qui se pose dans le contexte du SR. Nous présentons GreedierWPO, un algorithme de recherche locale qui surpasse les méthodes existantes dans le traitement de grandes instances. Ce chapitre explore également les techniques de réduction de l'espace de recherche, améliorant ainsi l'efficacité du processus de résolution. Nous avons également brièvement discuté des avantages de l'optimisation conjointe entre les deux problèmes : le problème de métriques et le problème de WPO. Notre travail ouvre la voie à de futures recherches sur l'optimisation à grande échelle du routage des segments de réseau.

List of Figures

2.1	Example of elimination tree obtained from a graph	9
2.2	Illustration of a shortest path routing with the ECMP rule	12
2.3	Example of an iteration of MCTS, from [16].	13
2.4	Illustration of the NRPA algorithm	17
2.5	Examples of SNDlib instances	22
2.6	Examples of Topology Zoo instances	22
2.7	An example of IGen instance	23
2.8	An example of Inet instance with 3040 nodes and 9856 arcs	24
2.9	Degree distribution of IGen and Inet instances	26
2.10	Degree sankey diagrams of IGen and Inet instances	26
2.11	Betweenness centrality distribution of IGen and Inet instances	27
2.12	Correlation between betweenness centrality and degree	28
3.1	Instance of the WEIGHT SETTING problem	34
3.2	Distribution of the congestion on SNDlib instances	39
3.3	Distribution of the congestion with different metric spaces	40
3.4	Convergence of NRPA without constraints	41
3.5	Impact of the metric space size	41
3.6	Congestion with respect to the number of the nodes	44
3.7	Comparison of congestion between increasing and fixed metric space	45
3.8	Comparison of convergence between different metric spaces	45
4.1	Example of a triangle	50
4.2	Example of CCH preprocessing with an example graph	52
4.3	Example of CCH preprocessing with Abilene	52
4.4	Comparison of execution time between HSPR and Dijkstra	59
4.5	NRPA Convergence on Igen1000	60
4.6	NRPA Convergence on Inet3040	61
4.7	Comparison of congestion value on Igen	62
4.8	Comparison of congestion value on Inet	62
5.1	The illustration of the secretary problem	67

5.2	Meta-NRPA demonstration	69
5.3	Correlation between the lower level NRPA and the full NRPA	71
5.4	Comparison of NRPA and Meta-NRPA on large generated instances	72
6.1	Illustration of the WAYPOINT OPTIMIZATION problem	79
6.2	Illustration of a tree decomposition for a network	88
6.3	Abilene instance with an associated tree decomposition	88
6.4	Comparison of different demand order using Q-learning	92
6.5	Comparison of different demand order using SARSA	92
6.6	Comparison of incremental demands and non-incremental demands modeling	93
6.7	Comparison of different demand orders using GreedierWPO	94
6.8	Comparison of different number of waypoint candidates	95
6.9	Convergence of GreedierWPO on france	96
6.10	Convergence of GreedierWPO on igen500	97
6.11	Comparison of GreedierWPO with different demand candidates	98
6.12	Comparison of GreedierWPO with different demand candidate on large instances	98
6.13	Comparison of all proposed approaches on WPO	99
6.14	Comparison between joint optimization, WEIGHT SETTING and WPO	101
8.1	Exemple d'arbre d'élimination obtenu à partir d'un graphe	110
8.2	Illustration d'un routage par le plus court chemin avec la règle ECMP	113
8.3	Exemples d'instances SNDlib	113
8.4	Exemples d'instances de Topology Zoo	114
8.5	Un exemple d'instance IGen	114
8.6	Un exemple d'instance Inet	115
8.7	Illustration du Problème de Métriques	118
8.8	Illustration du problème d'optimisation des waypoints	120

List of Tables

2.1	Different instances used in our experiments	24
2.2	Diameter of the instances	29
3.1	Comparison of Monte Carlo Search algorithms with all constraints	38
3.2	Maximum congestion value of the state-of-the-art heuristics and NRPA.	42
4.1	Average Tree Depth using different heuristics	58
4.2	Execution time of different heuristics	59
4.3	Congestion value comparison	61
5.1	Average score of Exploratory NRPA with TSPTW	73
5.2	Average score of Meta-NRPA with TSPTW	74
5.3	Average score for the Snake-in-the-Box problem	74
5.4	Comparison with known lower bounds on the Snake-in-the-Box	75
6.1	Comparison between GreedierWPO and GreedyWPO	99
6.2	Comparison between GreedierWPO and CG4SR	100
6.3	Comparison between Naive-JOINT and JOINT-HEUR	101
8.1	Informations sur les réseaux	116

Bibliography

- [1] Harvey L. Abbott and Meir Katchalski. “On the snake in the box problem”. *Journal of Combinatorial Theory, Series B* 45.1 (1988), pp. 13–24 (cit. on p. 74).
- [2] David Allison and Daniël Paulusma. “New Bounds for the Snake-in-the-Box Problem”. *ArXiv* abs/1603.05119 (2016) (cit. on p. 74).
- [3] Aysegül Altin, Bernard Fortz, Mikkel Thorup, and Hakan Ümit. “Intra-domain traffic engineering with shortest path routing protocols”. *Annals of Operations Research* 204.1 (2013), pp. 65–95 (cit. on p. 34).
- [4] Aysegül Altin, Bernard Fortz, and Hakan Ümit. “Oblivious OSPF routing with weight optimization under polyhedral demand uncertainty”. *Networks* 60.2 (2012), pp. 132–139 (cit. on p. 34).
- [5] François Aubry. “Models and Algorithms for Network Optimization with Segment Routing”. PhD thesis. Ecole Polytechnique de Louvain, 2020 (cit. on p. 80).
- [6] Albert-László Barabási and Réka Albert. “Emergence of Scaling in Random Networks”. *Science* 286.5439 (1999), pp. 509–512 (cit. on pp. 23, 114).
- [7] Ahmed Bashandy, Clarence Filsfils, Stefano Previdi, Bruno Decraene, Stephane Litkowski, and Rob Shakir. *Segment Routing with the MPLS Data Plane*. RFC 8660. 2019 (cit. on pp. 78, 79, 119).
- [8] Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. “Fast Routing in Road Networks with Transit Nodes”. *Science* 316.5824 (2007), pp. 566–566 (cit. on p. 48).
- [9] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. “Search-space size in contraction hierarchies”. *Theoretical Computer Science* 645 (2016), pp. 112–127 (cit. on pp. 51, 54).
- [10] Andreas Bley. “An Integer Programming Algorithm for Routing Optimization in IP Networks”. *Algorithmica* 60(1) (2008), pp. 21–45 (cit. on pp. 33, 117).
- [11] Andreas Bley. “Approximability of unsplittable shortest path routing problems”. *Networks* 54.1 (2009), pp. 23–46 (cit. on p. 34).
- [12] Andreas Bley. “An Integer Programming Algorithm for Routing Optimization in IP Networks”. *Algorithmica* 60.1 (2011), pp. 21–45 (cit. on pp. 34, 43, 46).
- [13] Andreas Bley, Bernard Fortz, Éric Gourdin, Kaj Holmberg, Olivier Klopfenstein, Michal Pióro, et al. “Optimization of OSPF Routing in IP Networks”. *Graphs and Algorithms in Communication Networks: Studies in Broadband, Optical, Wireless and Ad Hoc Networks*. Ed. by Arie Koster and Xavier Muñoz. EATCS Series. Springer, 2010, pp. 199–240 (cit. on p. 34).
- [14] Bruno Bouzy. “Monte-Carlo Fork Search for Cooperative Path-Finding”. *Computer Games Workshop at IJCAI*. 2013, pp. 1–15 (cit. on p. 16).
- [15] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, et al. “A Survey of Monte Carlo Tree Search Methods”. *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43 (cit. on p. 66).
- [16] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, et al. “A Survey of Monte Carlo Tree Search Methods”. *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43 (cit. on pp. 13, 14).
- [17] Luciana S. Buriol, Mauricio G. C. Resende, Celso C. Ribeiro, and Mikkel Thorup. “A hybrid genetic algorithm for the weight setting problem in OSPF/IS-IS routing”. *Networks* 46.1 (2005), pp. 36–56 (cit. on pp. 23, 34, 113).
- [18] Luciana S. Buriol, Mauricio G. C. Resende, and Mikkel Thorup. “Speeding Up Dynamic Shortest-Path Algorithms”. *INFORMS J. Comput.* 20.2 (2008), pp. 191–204 (cit. on p. 48).

- [19] Tristan Cazenave. “Nested Monte-Carlo Search”. *IJCAI*. Ed. by Craig Boutilier. 2009, pp. 456–461 (cit. on pp. [12](#), [66](#)).
- [20] Tristan Cazenave. “Generalized Nested Rollout Policy Adaptation”. *Monte Search at IJCAI*. 2020 (cit. on p. [73](#)).
- [21] Tristan Cazenave and Thomas Fournier. “Monte Carlo Inverse Folding”. *Monte Search at IJCAI*. 2020 (cit. on p. [17](#)).
- [22] Tristan Cazenave, Jean-Yves Lucas, Thomas Triboulet, and Hyoseok Kim. “Policy Adaptation for Vehicle Routing”. *Ai Communications* (2021) (cit. on p. [17](#)).
- [23] Tristan Cazenave, Benjamin Negrevergne, and Florian Sikora. “Monte Carlo Graph Coloring”. *Monte Search at IJCAI*. 2020 (cit. on p. [17](#)).
- [24] Tristan Cazenave, Abdallah Saffidine, Michael John Schofield, and Michael Thielscher. “Nested Monte Carlo Search for Two-Player Games”. *AAAI*. 2016, pp. 687–693 (cit. on pp. [15](#), [16](#)).
- [25] Tristan Cazenave, Jean-Baptiste Sevestre, and Matthieu Toulemon. “Stabilized Nested Rollout Policy Adaptation”. *Monte Search at IJCAI*. 2020 (cit. on p. [35](#)).
- [26] Tristan Cazenave and Fabien Teytaud. “Application of the Nested Rollout Policy Adaptation Algorithm to the Traveling Salesman Problem with Time Windows”. *Learning and Intelligent Optimization - 6th International Conference, LION 6*. 2012, pp. 42–54 (cit. on p. [17](#)).
- [27] Tristan Cazenave and Fabien Teytaud. “Beam Nested Rollout Policy Adaptation”. *Computer Games Workshop, ECAI 2012*. 2012, pp. 1–12 (cit. on p. [35](#)).
- [28] Leela Sunil Chandran, Louis Ibarra, Frank Ruskey, and Joe Sawada. “Generating and characterizing the perfect elimination orderings of a chordal graph”. *Theoretical Computer Science* 307.2 (2003), pp. 303–317 (cit. on pp. [9](#), [110](#)).
- [29] Yuan-Shih Chow, Sigeiti Moriguti, Herbert Robbins, and Stephen Mitchell Samuels. “Optimal Selection Based on Relative Rank (the “Secretary Problem”)”. *Israel Journal of Mathematics* 2.2 (1964), pp. 81–90 (cit. on p. [67](#)).
- [30] Rémi Coulom. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search”. *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. Donkers. Vol. 4630. Lecture Notes in Computer Science. Springer, 2006, pp. 72–83 (cit. on p. [66](#)).
- [31] Chen Dang, Cristina Bazgan, Tristan Cazenave, Morgan Chopin, and Pierre-Henri Wuillemin. “Monte Carlo Search Algorithms for Network Traffic Engineering”. *ECML PKDD 2021, Part IV*. Vol. 12978. LNCS. 2021, pp. 486–501 (cit. on p. [31](#)).
- [32] Chen Dang, Cristina Bazgan, Tristan Cazenave, Morgan Chopin, and Pierre-Henri Wuillemin. “Warm-Starting Nested Rollout Policy Adaptation with Optimal Stopping”. *Proceedings of the AAAI Conference on Artificial Intelligence* 37.10 (2023), pp. 12381–12389 (cit. on p. [65](#)).
- [33] Daniel Delling, Andrew Goldberg, Thomas Pajor, and Renato Werneck. “Customizable Route Planning”. *SEA’11*. Vol. 6630. LNCS. 2011, pp. 230–241 (cit. on p. [48](#)).
- [34] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. “Customizable Contraction Hierarchies”. *ACM J. Exp. Algorithmics* 21.1 (2016), 1.5:1–1.5:49 (cit. on pp. [5](#), [48](#), [49](#), [54](#), [63](#), [122](#)).
- [35] Stefan Edelkamp and Tristan Cazenave. “Improved diversity in nested rollout policy adaptation”. *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer. 2016, pp. 43–55 (cit. on p. [66](#)).
- [36] Stefan Edelkamp, Max Gath, Tristan Cazenave, and Fabien Teytaud. “Algorithm and knowledge engineering for the TSPTW problem”. *Computational Intelligence in Scheduling (SCIS), 2013 IEEE Symposium on*. IEEE. 2013, pp. 44–51 (cit. on pp. [17](#), [66](#)).
- [37] Stefan Edelkamp, Max Gath, Christoph Greulich, Malte Humann, Otthein Herzog, and Michael Lawo. “Monte-Carlo Tree Search for Logistics”. *Commercial Transport*. Springer International Publishing, 2016, pp. 427–440 (cit. on p. [17](#)).
- [38] Stefan Edelkamp, Max Gath, and Moritz Rohde. “Monte-Carlo Tree Search for 3D Packing with Object Orientation”. *KI 2014: Advances in Artificial Intelligence*. Springer International Publishing, 2014, pp. 285–296 (cit. on p. [17](#)).
- [39] Stefan Edelkamp and Christoph Greulich. “Solving physical traveling salesman problems with policy adaptation”. *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*. IEEE. 2014, pp. 1–8 (cit. on p. [17](#)).

-
- [40] Stefan Edelkamp and Zhihao Tang. “Monte-Carlo Tree Search for the Multiple Sequence Alignment Problem”. *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015*. AAAI Press, 2015, pp. 9–17 (cit. on p. 17).
 - [41] RFC Editor. *OSI IS-IS Intra-domain Routing Protocol*. RFC 1142. 1990 (cit. on pp. 11, 32, 112, 116).
 - [42] M. Ericsson, Mauricio G. C. Resende, and Panos M. Pardalos. “A Genetic Algorithm for the Weight Setting Problem in OSPF Routing”. *Journal of Combinatorial Optimization* 6.3 (2002), pp. 299–333 (cit. on pp. 34, 48).
 - [43] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. “On Power-law Relationships of the Internet Topology”. *SIGCOMM 1999*. ACM, 1999, pp. 251–262 (cit. on pp. 23, 25, 54, 114).
 - [44] Francois Le Faucheur. “IETF Multiprotocol Label Switching (MPLS) Architecture”. *1998 1st IEEE International Conference on ATM. ICATM’98*. IEEE. 1998 (cit. on pp. 79, 119).
 - [45] Clarence Filsfils, Pablo Camarillo, John Leddy, Daniel Voyer, Satoru Matsushima, and Zhenbin Li. *Segment Routing over IPv6 (SRv6) Network Programming*. RFC 8986. Internet Engineering Task Force (IETF), 2021 (cit. on pp. 78, 79, 119).
 - [46] Clarence Filsfils, Stefano Previdi, Les Ginsberg, Bruno Decraene, Stephane Litkowski, and Rob Shakir. *Segment Routing Architecture*. RFC 8402. 2018 (cit. on pp. 32, 116).
 - [47] Lisa Fleischer. “Approximating Fractional Multicommodity Flow Independent of the Number of Commodities”. *SIAM J. Discret. Math.* 13.4 (2000), pp. 505–520 (cit. on pp. 62, 72).
 - [48] Bernard Fortz. “Applications of meta-heuristics to traffic engineering in IP networks”. *International Transactions in Operational Research* 18.2 (2011), pp. 131–147 (cit. on p. 34).
 - [49] Bernard Fortz and Mikkel Thorup. “Increasing Internet Capacity Using Local Search”. *Computational Optimization and Applications* 29 (2000), pp. 13–48 (cit. on pp. 34, 41).
 - [50] Bernard Fortz and Mikkel Thorup. “Robust optimization of OSPF/IS-IS weights”. *Proceedings of the 1st International Network Optimization Conference (INOC 2003)*. 2003, pp. 225–230 (cit. on p. 34).
 - [51] Bernard Fortz and Mikkel Thorup. “Increasing Internet Capacity Using Local Search”. *Computational Optimization and Applications* 29.1 (2004), pp. 13–48 (cit. on pp. 23, 48, 56, 113).
 - [52] Steven Gay, Renaud Hartert, and Stefano Vissicchio. “Expect the unexpected: Sub-second optimization for segment routing”. *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9 (cit. on p. 80).
 - [53] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. “Exact Routing in Large Road Networks Using Contraction Hierarchies”. *Transportation Science* 46.3 (2012), pp. 388–404 (cit. on pp. 48, 49).
 - [54] John P. Gilbert and Frederick Mosteller. “Recognizing the Maximum of a Sequence”. *Journal of the American Statistical Association* 61.313 (1966), pp. 35–73. eprint: <https://www.tandfonline.com/doi/pdf/10.1080/01621459.1966.10502008> (cit. on p. 66).
 - [55] Andrew V. Goldberg and Chris Harrelson. “Computing the Shortest Path: A* Search Meets Graph Theory”. *SODA ’05*. 2005, pp. 156–165 (cit. on p. 48).
 - [56] Michael Hamann and Ben Strasser. “Graph Bisection with Pareto-Optimization”. *ACM J. Exp. Algorithms* 23, Article No.: 1.2 (2018), pp. 1–34 (cit. on pp. 51, 54, 55).
 - [57] Renaud Hartert. “Fast and scalable optimization for segment routing”. PhD thesis. Catholic University of Louvain, Louvain-la-Neuve, Belgium, 2018 (cit. on pp. 79, 80).
 - [58] Renaud Hartert, Pierre Schaus, Stefano Vissicchio, and Olivier Bonaventure. “Solving Segment Routing Problems with Hybrid Constraint Programming Techniques”. *Principles and Practice of Constraint Programming*. Ed. by Gilles Pesant. Cham: Springer International Publishing, 2015, pp. 592–608 (cit. on p. 80).
 - [59] Mathieu Jadin, Francois Aubry, Pierre Schaus, and Olivier Bonaventure. “CG4SR: Near Optimal Traffic Engineering for Segment Routing with Column Generation”. *IEEE International Conference on Computer Communications (INFOCOM 2019)*. IEEE, 2019, pp. 1333–1341 (cit. on pp. 81, 100).
 - [60] George Karypis and Vipin Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. *SIAM Journal on Scientific Computing* 20.1 (1998), pp. 359–392 (cit. on p. 51).
 - [61] David Kinny. “A New Approach to the Snake-In-The-Box Problem”. *Proceedings of the 20th European Conference on Artificial Intelligence. ECAI’12*. Montpellier, France: IOS Press, 2012, pp. 462–467 (cit. on pp. 74, 75).

- [62] David Kinny. “A New Approach to the Snake-In-The-Box Problem”. *ECAI 2012*. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, pp. 462–467 (cit. on p. 16).
- [63] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. “The Internet Topology Zoo”. *Selected Areas in Communications, IEEE Journal on* 29.9 (2011), pp. 1765–1775 (cit. on pp. 22, 113).
- [64] Levente Kocsis and Csaba Szepesvari. “Bandit based Monte-Carlo planning”. *17th European Conference on Machine Learning (ECML’06)*. Vol. 4212. LNCS. Springer, 2006, pp. 282–293 (cit. on pp. 12, 14, 66).
- [65] Abba Krieger and Ester Samuel-Cahn. “The secretary problem of minimizing the expected rank: a simple suboptimal approach with generalizations”. *Advances in Applied Probability* 41 (2009), pp. 1041–1058 (cit. on p. 67).
- [66] Guy Leduc, Henrik Abrahamsson, Simon Balon, Sandford Bessler, Maurizio D’Arienzo, Olivier Delcourt, et al. “An open source traffic engineering toolbox”. *Computer Communications* 29.5 (2006), pp. 593–610 (cit. on pp. 34, 41).
- [67] Dennis Victor Lindley. “Dynamic Programming and Decision Theory”. *Journal of the Royal Statistical Society Series C* 10.1 (1961), pp. 39–51 (cit. on p. 66).
- [68] Jim McManus, Joseph Malcolm, Michael D. O’Dell, Daniel O. Awduche, and Johnson Agogbua. *Requirements for Traffic Engineering Over MPLS*. Tech. rep. 2702. 1999. 29 pp. (cit. on pp. 32, 116).
- [69] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John W. Byers. “BRITE: An Approach to Universal Topology Generation”. *MASCOTS 2001*. IEEE Computer Society, 2001, p. 346 (cit. on pp. 22, 113).
- [70] Jean Méhat and Tristan Cazenave. “Combining UCT and Nested Monte Carlo Search for Single-Player General Game Playing”. *IEEE Transactions on Computational Intelligence and AI in Games* 2.4 (2010), pp. 271–277 (cit. on p. 15).
- [71] Eduardo Moreno, Alejandra Beghelli, and Filippo Cugini. “Traffic engineering in segment routing networks”. *Computer Networks* 114 (2017), pp. 23–31 (cit. on p. 80).
- [72] John Moy. *OSPF Version 2*. Tech. rep. 2328. 1998. 244 pp. (cit. on pp. 11, 32, 112, 116).
- [73] Sebastian Orłowski, Michal Pióro, Artur Tomaszewski, and Roland Wessäly. “SNDlib 1.0–Survivable Network Design Library”. *Networks* 55.3 (2010), pp. 276–286 (cit. on pp. 22, 52, 113).
- [74] Patric Östergård and Ville Pettersson. “Exhaustive Search for Snake-in-the-Box Codes”. *Graphs and Combinatorics* 31 (2014) (cit. on pp. 74, 75).
- [75] Mahmoud Parham, Thomas Fenz, Nikolaus Süß, Klaus-Tycho Foerster, and Stefan Schmid. “Traffic Engineering with Joint Link Weight and Segment Optimization”. *CoNEXT ’21*. ACM, 2021, pp. 313–327 (cit. on pp. 48, 63, 80, 81, 85, 90, 101).
- [76] Alex Pothén. *The Complexity of Optimal Elimination Trees*. Technical report. Pennsylvania State University, Department of Computer Science, 1988 (cit. on p. 51).
- [77] Walter D. Potter, Robert W. Robinson, John A Miller, Krys Kochut, and D. Z. Redys. “Using the Genetic Algorithm to Find Snake-in-the-Box Codes”. *Proceedings of the 7th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*. IEA/AIE ’94. Austin, Texas, USA: Gordon and Breach Science Publishers, Inc., 1994, pp. 421–426 (cit. on pp. 74, 75).
- [78] Simon M. Poulding and Robert Feldt. “Heuristic Model Checking using a Monte-Carlo Tree Search Algorithm”. *GECCO*. 2015, pp. 1359–1366 (cit. on p. 16).
- [79] Peter Psenak, Shraddha Hegde, Clarence Filsfil, Ketan Talaulikar, and Arkadiy Gulko. *IGP Flexible Algorithm*. Internet-Draft draft-ietf-lsr-flex-algo-20. Internet Engineering Task Force, 2022. 47 pp. (cit. on pp. 32, 116).
- [80] Martin L. Puterman. “Markov Decision Processes: Discrete Stochastic Dynamic Programming”. *Wiley Series in Probability and Statistics*. 1994 (cit. on p. 12).
- [81] Bruno Quoitin, Virginie Van den Schrieck, Pierre Francois, and Olivier Bonaventure. “IGen: Generation of router-level Internet topologies through network design heuristics”. *21st International Teletraffic Congress*. 2009, pp. 1–8 (cit. on pp. 22, 23, 25, 113, 114).
- [82] Roger Reis, Marcus Ritt, Luciana Buriol, and Mauricio Resende. “A biased random-key genetic algorithm for OSPF and DEFT routing to minimize network congestion”. *International Transactions in Operational Research* 18.3 (2011), pp. 401–423 (cit. on pp. 23, 113).

-
- [83] Arpad Rimmel, Fabien Teytaud, and Tristan Cazenave. “Optimization of the Nested Monte-Carlo Algorithm on the Traveling Salesman Problem with Time Windows”. *EvoApplications*. Vol. 6625. LNCS. Springer, 2011, pp. 501–510 (cit. on p. 15).
 - [84] Christopher Rosin. “Multi-armed Bandits with Episode Context”. *Annals of Mathematics and Artificial Intelligence* 61 (2010), pp. 203–230 (cit. on p. 14).
 - [85] Christopher D. Rosin. “Nested Rollout Policy Adaptation for Monte Carlo Tree Search”. *IJCAI*. 2011, pp. 649–654 (cit. on pp. 5, 12, 16, 66, 122).
 - [86] Gavin A Rummery and Mahesan Niranjan. “On-Line Q-Learning Using Connectionist Systems”. *Technical Report CUED/F-INFENG/TR 166* (1994) (cit. on p. 19).
 - [87] Peter Sanders and Dominik Schultes. “Engineering Highway Hierarchies”. *Algorithms - ESA 2006, 14th Annual European Symposium, Proceedings*. Vol. 4168. LNCS. 2006, pp. 804–816 (cit. on p. 48).
 - [88] Aaron Schild and Christian Sommer. “On Balanced Separators in Road Networks”. *SEA 2015*. Vol. 9125. LNCS. Springer, 2015, pp. 286–297 (cit. on pp. 51, 55, 57).
 - [89] Tossaphol Settawatcharawanit, Vorapong Suppakitpaisarn, Shigeki Yamada, and Yusheng Ji. “Segment Routed Traffic Engineering with Bounded Stretch in Software-Defined Networks”. *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. 2018, pp. 477–480 (cit. on p. 87).
 - [90] Farhad Shahrokhi and D. W. Matula. “The Maximum Concurrent Flow Problem”. *J. ACM* 37.2 (1990), pp. 318–334 (cit. on p. 42).
 - [91] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, et al. “Mastering the game of Go with deep neural networks and tree search”. *Nature* 529.7587 (2016), pp. 484–489 (cit. on p. 66).
 - [92] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, et al. “Mastering the game of Go with deep neural networks and tree search”. *Nature* 529 (2016), pp. 484–489 (cit. on p. 14).
 - [93] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”. *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815 (cit. on p. 66).
 - [94] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. *Science* 362.6419 (2018), pp. 1140–1144 (cit. on p. 66).
 - [95] Ben Strasser. “PACE Solver Description: Tree Depth with FlowCutter”. *IPEC*. 2020 (cit. on pp. 55, 57).
 - [96] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 70).
 - [97] Richard S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. *Mach. Learn.* 3.1 (1988), pp. 9–44 (cit. on p. 19).
 - [98] Sylwester Swat. “PACE Solver Description: Finding Elimination Trees Using ExTREEm - a Heuristic Solver for the Treedepth Decomposition Problem”. *IPEC*. 2020 (cit. on pp. 55, 57).
 - [99] George Trimponias, Yan Xiao, Hong Xu, Xiaorui Wu, and Yanhui Geng. “Centrality-based Middlepoint Selection for Traffic Engineering with Segment Routing”. *arXiv: Networking and Internet Architecture* (2017) (cit. on pp. 87, 89).
 - [100] George Trimponias, Yan Xiao, Hong Xu, Xiaorui Wu, and Yanhui Geng. “On Traffic Engineering with Segment Routing in SDN based WANS”. *CoRR* abs/1703.05907 (2017). arXiv: 1703.05907 (cit. on p. 80).
 - [101] Chris Watkins. “Learning from delayed rewards”. PhD thesis. King’s College, 1989 (cit. on p. 19).
 - [102] Jared Winick and Sugih Jamin. *Inet-3.0: Internet Topology Generator*. Tech. rep. UM-CSE-TR-456-02. EECS, University of Michigan, 2002 (cit. on pp. 22, 23, 25, 113, 114).
 - [103] Duo Wu and Lin Cui. “A comprehensive survey on Segment Routing Traffic Engineering”. *Digital Communications and Networks* (2022) (cit. on p. 80).
 - [104] Ed Wynn. “Constructing circuit codes by permuting initial sequences”. *arXiv* abs/1201.1647 (2012) (cit. on pp. 74, 75).
 - [105] Seisho Yasukawa, Adrian Farrel, and Olufemi Komolafe. “An Analysis of Scaling Issues in MPLS-TE Core Networks”. *RFC* 5439 (2009), pp. 1–45 (cit. on pp. 79, 119).

Cette thèse s'intéresse aux problèmes d'optimisation issus du domaine de l'ingénierie de trafic dans les réseaux de télécommunications. En particulier, nous avons étudié le problème de minimisation de la congestion dans les réseaux IP reposant sur les protocoles de routage au plus court chemin (OSPF, IS-IS) d'une part et le segment routing d'autre part. La principale contribution de la thèse est l'application de la méthode de recherche Monte Carlo Nested Rollout Policy Adaptation (NRPA) au problème d'optimisation des métriques (WEIGHT SETTING). L'approche proposée s'adapte bien à la taille de l'instance et peut être facilement étendue pour prendre en compte des contraintes ou des critères d'optimisation supplémentaires (délai bout en bout, unicité des chemins de routage, etc...).

En outre, cette thèse présente une version améliorée de NRPA, appelée Meta-NRPA, qui utilise la théorie de l'arrêt optimal pour mieux guider la recherche de NRPA. De plus, nous présentons plusieurs techniques exploratoires pour NRPA permettant une exploration plus efficace de l'espace des solutions. Nos expérimentations confirment l'amélioration significative des performances de NRPA.

Par ailleurs, cette thèse introduit l'algorithme Hierarchical Shortest Path Routing (HSPR), une version adaptée de l'algorithme Customizable Contraction Hierarchy (CCH). HSPR permet d'accélérer significativement la simulation des protocoles de routage au plus court chemin comparé aux méthodes traditionnelles basées sur l'algorithme de Dijkstra. Nous l'avons combiné avec l'algorithme Meta-NRPA pour résoudre le problème WEIGHT SETTING. Les résultats obtenus montrent une amélioration substantielle des performances de Meta-NRPA, surtout dans les réseaux de grande taille.

Enfin, nous étudions également plusieurs heuristiques pour résoudre le problème d'optimisation WAYPOINT OPTIMIZATION, lié au Segment Routing. Nous combinons une approche basée sur la recherche locale et la décomposition arborescente du réseau afin d'explorer plus efficacement l'espace des solutions. Nous proposons également d'autres approches basées sur l'apprentissage par renforcement et évaluons leur efficacité à travers une série d'expériences approfondies sur des données réelles et synthétiques.

MOTS CLÉS

Optimisation des Réseaux, Methodes de Monte Carlo, Ingénierie de Trafic, Segment Routing, Apprentissage par Renforcement

ABSTRACT

This thesis examines optimization problems that arise from telecommunications traffic engineering. In particular, we have studied the problem of congestion minimization in IP networks based on shortest path routing protocols (OSPF, IS-IS) on the one hand, and segment routing on the other.

The main contribution of this thesis is the application of the Monte Carlo Nested Rollout Policy Adaptation (NRPA) method for solving the metric optimization problem WEIGHT SETTING. The proposed approach adapts well to the size of the instance and can be readily expanded to consider additional constraints or optimization criteria (end-to-end delay, uniqueness of routing paths, etc...).

Furthermore, this thesis introduces an enhanced version of NRPA, named Meta-NRPA, which leverages the optimal stopping theory to better guide the search of NRPA. Additionally, we present several exploratory techniques for NRPA, enabling a more efficient exploration of the search space. Our experiments confirm a significant improvement in NRPA's performance.

Subsequently, this thesis presents the Hierarchical Shortest Path Routing (HSPR) algorithm, an adapted version of the Customizable Contraction Hierarchy (CCH) algorithm. HSPR allows to significantly accelerate the simulation of shortest path routing protocols compared to traditional methods based on the Dijkstra algorithm. We combined it with the Meta-NRPA algorithm to address the WEIGHT SETTING problem. The results demonstrate a substantial enhancement in Meta-NRPA's performance, especially in large networks.

Lastly, we also investigate several heuristics for solving the optimization problem WAYPOINT OPTIMIZATION related to Segment Routing. We combine a local search based approach with tree decomposition to explore more effectively the solution space. We also propose other methods based on reinforcement learning and assess their effectiveness through a series of in-depth experiments on real and synthetic data.

KEYWORDS

Network Optimization, Monte Carlo Methods, Traffic Engineering, Segment Routing, Reinforcement Learning