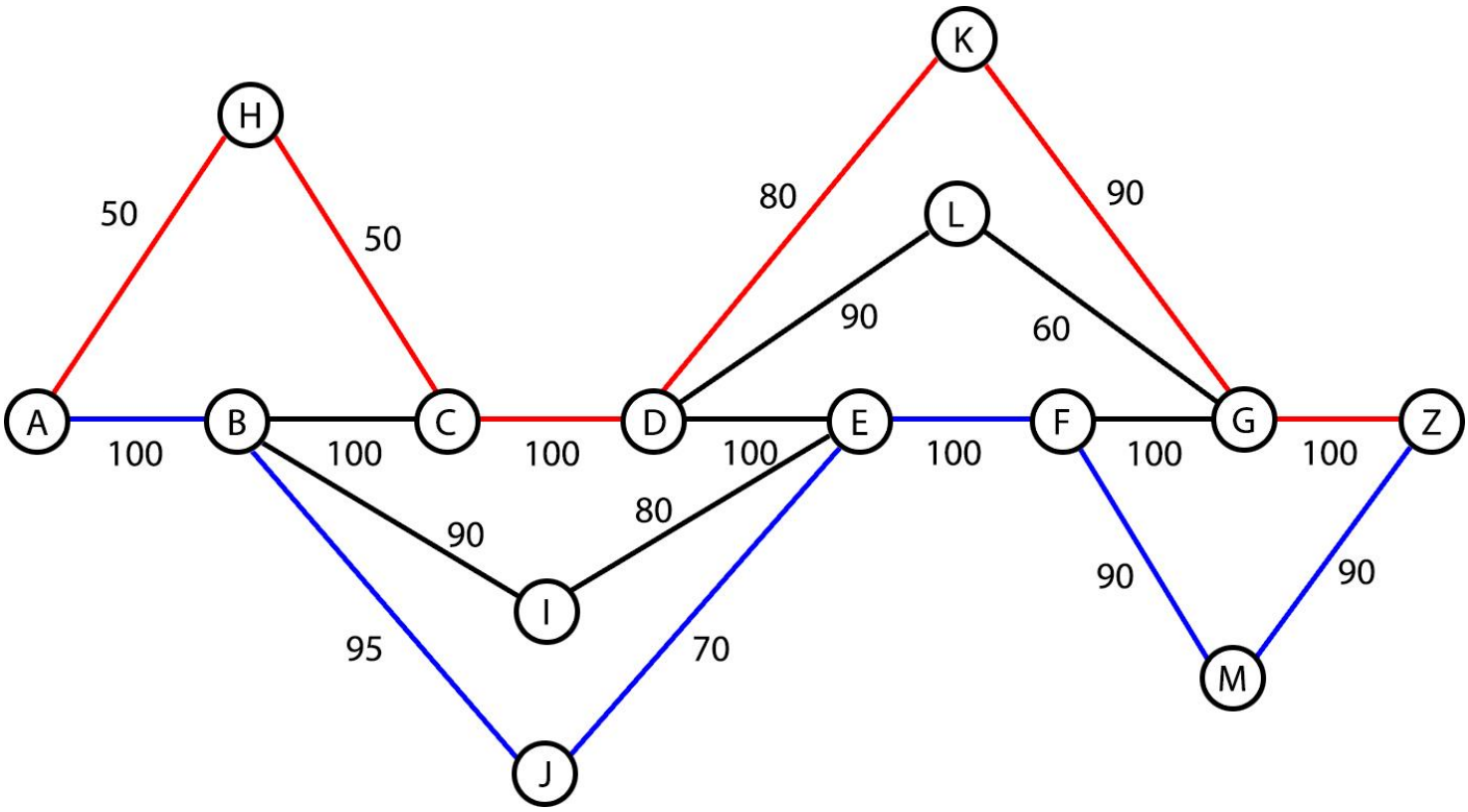


Fernando Pedro Ferreira da Cruz

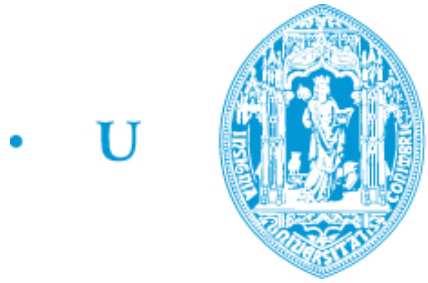
DISJOINT PATH PAIR CALCULATION, CONSIDERING BANDWIDTHS

Dissertation in Integrated Master in Electrical and Computer Engineering (Specialization in Telecommunications), supervised by Professor Teresa Martinez dos Santos Gomes and presented in the Department of Electrical and Computer Engineering

September 2014

· U C ·

UNIVERSIDADE DE COIMBRA



FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Disjoint path pair calculation, considering bandwidths

Fernando Pedro Ferreira da Cruz

Members of the Jury:

President: Professor José Manuel Fernandes Craveirinha

Supervisor: Professor Teresa Martinez dos Santos Gomes

Member: Professor Henrique José Almeida da Silva

4 September 2014

Agradecimentos

Agradeço...

...aos meus pais pelas oportunidades e pela educação que deram a mim e ao meu padrinho. À minha mãe, por me ter ensinado que o melhor prémio é a satisfação de um trabalho bem feito. Ao meu pai, por me ter ensinado jogos que me ajudaram a desenvolver o pensamento.

...aos meus amigos, pela alegria. Em particular à minha namorada, a minha Cristina, que me ajudou com revisões, carinho e paciência.

...ao meu padrinho, pelo incentivo e por reforçar a importância da família, a quem agradeço pelos exemplos e pelo calor.

...à Professora Doutora Teresa Gomes, por tudo o que se relaciona com este trabalho.

...aos meus professores da escola, da Universidade e da vida. Se não me conseguiram ensinar o que queriam, pelo menos ensinaram-me perseverança.

...aos meus colegas, pela companhia.

...a quem ainda não tiver agradecido e, de uma forma ou de outra, também tenha contribuído para quem sou, para o que tenho e para o que hei-de conquistar.

...aos meus cães, pela energia, e à minha cadeira, pelo suporte.

Acknowledgements

I thank...

...my parents for the opportunities and education that they gave to me and my godfather. To my mother, for teaching me that the best reward is the satisfaction of a job well done. To my father, for teaching me games that helped me develop thought.

...my friends, for the joy. In particular my girlfriend, my Cristina, who helped me with revisions, care and patience.

...my godfather, for the incentive and for stressing the importance of family, to whom I thank for the examples and for the warmth.

...Professor Teresa Gomes, for everything related to this work.

...my teachers from school, from the University, and from life. If you couldn't teach me what you wanted, at least you taught me perseverance.

...my colleagues, for the company.

...whoever I didn't thank yet and, in one way or another, also contributed to who I am, what I have and what I'll accomplish.

...my dogs, for the energy, and my chair, for the support.

Resumo

A dependência da sociedade moderna dos serviços de telecomunicações tem vindo a crescer nos últimos anos, assim como a responsabilidade de providenciar serviços com elevada qualidade. A garantia de fornecimento de um serviço de comunicações sem perda de conetividade é extremamente importante, pois qualquer interrupção nas comunicações é fortemente sentida pelos utilizadores. A proteção global de um caminho é uma forma simples de aumentar a resiliência de uma ligação ponto-a-ponto. A sua implementação requer o cálculo de pelo menos um par de caminhos disjuntos.

O cálculo de pares de caminhos disjuntos, com ou sem restrições, tem sido objeto de estudo por numerosos autores. Existem algoritmos exactos para a resolução de alguns destes problemas, contudo outros são de mais difícil resolução. A determinação de um par de caminhos disjuntos de custo aditivo mínimo pode ser resolvido de forma eficiente utilizando o algoritmo de Suurballe. Contudo, o problema da determinação de um par de caminhos de largura de banda total máxima é NP-Completo.

Com isto em mente, este trabalho foca-se em três problemas de encaminhamento disjunto. O primeiro é um problema de otimização lexicográfica para obter caminhos disjuntos de largura de banda máxima e, depois, maximizar a largura de banda do caminho mais largo do par. O segundo é o cálculo de um par de caminhos tal que a soma das larguras de banda é máxima para um dado par de nós. Finalmente, o terceiro é um problema que tem como objetivo encontrar um par de caminhos disjuntos que satisfaçam duas restrições de largura de banda diferentes.

Estes problemas de encaminhamento disjunto são formalizados como problemas de programação linear inteira (PLI) e é proposta uma heurística para cada um deles. O desempenho das heurísticas é analisado tendo em conta o tempo de CPU das heurísticas e o requerido pelo CPLEX; é também analisada a qualidade das soluções obtidas, utilizando como referência a solução ótima devolvida pelo CPLEX ao resolver a formulação PLI do problema correspondente.

Palavras-Chave: caminhos disjuntos, caminho mais largo, proteção, otimização lexicográfica, max-sum, encaminhamento com restrições

Abstract

Modern society's dependency on telecommunications services has been increasing throughout the years and so has the responsibility to provide high quality services. The guarantee that a communications service is provided without loss of connectivity is extremely important, because any interruption in communications is strongly felt by the users. Global path protection is a simple way to increase resilience in an end-to-end connection. Its implementation requires the calculation of at least a pair of disjoint paths.

The calculation of disjoint path pairs, with or without restrictions, has been the subject of study by many authors. There are exact algorithms that solve these problems, however others are harder to solve. The determination of a pair of disjoint paths of additive minimum cost can be solved efficiently using Suurballe's algorithm. However, the problem of determining a pair of paths with maximum total bandwidth is NP-Complete.

With this in mind, this work addresses three disjoint routing problems. The first is a lexicographic optimization problem for obtaining maximum bandwidth disjoint paths and, then, maximizing the widest path in the pair. The second is the calculation of a path pair such that the sum of the bandwidths is maximum for the given pair of nodes. Finally, the third is a problem that aims to find a pair of disjoint paths that satisfy two different bandwidth constraints.

These disjoint routing problems are formalized as integer linear programming (ILP) and an heuristic is presented for each one. The performance of these heuristics is analyzed taking into account the heuristic's and CPLEX's CPU time; it is also analyzed the quality of the obtained solutions using as reference the optimal solutions obtained by CPLEX when it solves the ILP formulation of the corresponding problem.

Keywords: disjoint paths, widest path, protection, lexicographic optimization, max-sum, constrained routing

"If at first the idea is not absurd then there is no hope for it."

Albert Einstein

Contents

List of Figures

List of Tables

Abbreviations

BFS	Breadth-First-Search
DPLD	Dual Path Label Dijkstra
DPLD-ML	Dual Path Label Dijkstra's Algorithm with Minimum Limits
DPLD-MS	Dual Path Label Dijkstra's Algorithm for Maximum Sum
HLO	Heuristic for widest edge-disjoint path pair Lexicographic Optimization
HML	Heuristic for edge-disjoint path pairs with Minimum Limits
HMS	Heuristic for Maximum bandwidth Sum
ILP	Integer Linear Programming
QoS	Quality-of-Service
SPDP-BG	Shortest Pair of Disjoint Paths with Bandwidth Guarantee
WEDLO	Widest Edge-disjoint Path Pair Lexicographic Optimization
WPDPC	Widest Pair of Disjoint Paths Coupled
WPDPD	Widest Pair of Disjoint Paths Decoupled

Chapter 1

Introduction

1.1 Motivation and Objectives

The set of users of telecommunication networks is expanding everyday. They range from single individuals, small groups and associations to massive corporations, governments and armed forces, each one with different needs and requirements. On a more local level, and just to name a few examples, there are schools, policemen, hospitals and other smaller medical centers, firemen, news stations, supermarkets. All of these rely on efficient communication to work properly, so a failure in the telecommunications sector can have an effect on education, security, health and other needs that are deemed basic nowadays. In order to satisfy this wide heterogeneous group of users, service providers are constantly creating new services and improving existing ones. And it is not just a matter of getting the user what he needs, users demand Quality-of-Service (QoS), which includes adequate levels of reliability and availability.

To guarantee these qualities, path protection may be employed. This is done by automatically switching to an operational path when the protected path has a fault. In $1 + 1$ protection, traffic is sent simultaneously and redundantly over both active and protection paths. In case of a fault in one of the paths, transmission will not fail, because there is still a connection between both ends. In $1 : 1$ protection, traffic is sent through the protection path only in the event of a fault in the active path. While this method makes a more efficient use of bandwidth, as the protection path may be used for other ends when it is not needed, there is a need for signaling overhead and restoration may take more time.

In both protection scenarios, the active and protection paths must be as disjoint as possible so as to minimize the risk of a fault affecting both paths and resulting in transmission failure. Because of this, algorithms for calculating paths and path pairs are at the center of QoS routing. This motivated the development of this work, which focus on disjoint routing as a way to increase network resilience. Three different problems are addressed:

- The lexicographic optimization of a widest edge-disjoint path pair, so that after maximizing the minimum bandwidth of the pair, the bandwidth of the widest path in the pair is maximized. The problem of maximizing the minimum bandwidth of disjoint paths already has an efficient answer, but the bandwidth of the widest path is still left to chance. This can be

of practical interest in communication networks such as for Multi-Protocol Label Switching based virtual private network services:
  - when using path protection, for ensuring the selected *protection path* has spare bandwidth to take into account traffic fluctuations that take place in the event of faults;
  - when using path protection, for ensuring the selected *active path* has spare bandwidth, to guarantee the desired QoS and that there is some extra available capacity to absorb traffic fluctuations.
- The maximization of the sum of the bandwidths of the pair. While it is not as protection oriented as the other two (though it can be used as partial protection), it is particularly useful for services that require large bandwidths, such as video conferencing, real-time data backup or telemedicine.
- The search for a path pair that satisfies two bandwidth constraints: X_1 and X_2 (with $X_1 > X_2$). The practical application is obvious, as there is a direct answer to the questions: Is there a path pair that meets the requirements? If so, which is it? This can be of practical use when there are specific requirements.

The objective is to find efficient algorithms or heuristics, that can find optimal solutions to these problems (or feasible solutions, in the case of the last problem).

1.2 Content

This work is organized as follows. Chapter 2 presents definitions and notation, as well as problems and algorithms relevant in the context of this thesis. In Chapter 3, the addressed problems are explained, defined and formalized. Chapter 4 contains some auxiliary algorithms that were important for this study. Chapter 5 shows the proposed heuristics that attempt to solve the problems described in Chapter 3. In Chapter 6 the performance of the aforementioned heuristics is analyzed. Chapter 7 concludes this work.

Chapter 2

Related Work

In this chapter, some basic notation and concepts are presented as well as an overview of the relevant algorithms that inspired and provided the foundation for this work.

Initially, the basic definitions and notation is presented, as they are going to be needed further ahead. Then, some problems and their respective solutions are described, in order of increasing complexity and closeness to the addressed problems.

2.1 Definitions and Notation

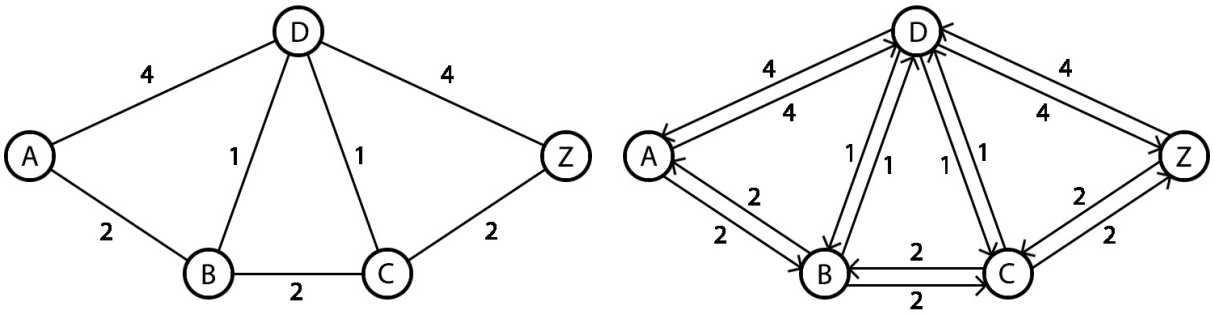
Networks, in their simplest form, can be seen as sets of nodes connected by links. In graph theory, a link is defined by the pair of nodes it connects. If these pairs are ordered, it means that links are represented as arcs and can only be used in one direction, from the tail node to the head node, the first and second elements of the pair, respectively. If the pairs are unordered, then the links are represented as edges and communication can be performed in both directions. Graphically, nodes are depicted as circles and arcs are shown as arrows, pointing from the tail node to the head node, while edges are drawn as straight lines.

A graph composed of nodes and arcs is said to be directed and a graph composed of nodes and edges is said to be undirected. A mixed graph can have both edges and arcs. An edge can be represented by two arcs with equal characteristics, but opposing directions (a symmetrical pair of arcs). An undirected graph can be turned into a directed graph if all edges suffer this transformation. This work is focused on undirected graphs, but one needs to consider their directed representation.

In Figure 2.1, two equivalent graphs are shown. The first is undirected (Figure 2.1(a)), while the second (Figure 2.1(b)) is the directed representation of the same graph, where the number adjacent to each edge/arc represents the edge/arc weight.

No parallel edges or arcs are taken into consideration. Two edges are parallel if they are defined by the same pair of nodes. The same applies to arcs, paying attention to the order of the nodes, which must be the same.

Let G_u be an undirected graph represented by (N,E), with $N = \{v_1, v_2, \ldots, v_n\}$ being the set of n nodes and $E = \{e_1, e_2, \ldots, e_m\}$ the set of m edges, where $e_x = (v_i, v_j)$ is an unordered pair of nodes, with $v_i, v_j \in N$. Similarly, let G be a directed graph denoted by (N,A), with $A = \{a_1, a_2, \ldots, a_m\}$ being the set of m arcs, where $a_x = (v_i, v_j)$, $v_i, v_j \in N$ (v_i is the tail node and v_j is the head node). A



(a) Undirected graph. (b) Directed graph.

Figure 2.1: Graph examples.

path p_{st} from a source node s to a target node t is an ordered sequence of alternated nodes and arcs or edges, starting with s and finishing with t ($s \neq t$), $\langle s = v'_1, a'_1, v'_2, \ldots, a'_{r-1}, v'_r = t \rangle$, where: $a'_k \in A$ for any $k = 1, \ldots, r-1$; $v'_k \in N$ for any $k = 1, \ldots, r$; and $a'_k = (v'_k, v'_{k+1})$ for any $k = 1, \ldots, r-1$.

A rooted tree is a graph composed solely by the set of arcs (or edges) that are part of the paths that connect one node, called root, to the remaining $n-1$ nodes. Citing Dijkstra in [7], *a tree is a graph with one and only one path between every two nodes*. This is true for undirected graphs, which are the type of graphs Dijkstra worked with in [7], but not for directed graphs, where a path may exist from s to t, yet not from t to s. For every arc $a'_k = (v'_k, v'_{k+1})$ that is part of a path, node v'_k is designated v'_{k+1}'s predecessor and v'_{k+1} is v'_k's successor.

Let p_{ij} be a path from node i to j; the concatenation of paths p_{ij} and p_{jl} is the path, $p_{ij} \diamond p_{jl}$, from i to l, which coincides with p_{ij} from i to j and with p_{jl} from j to l. Let p_{ij} be a path from i to j; the concatenation of paths p_{ij} and arc (j,k) is the path, $p_{ij} \diamond (j,k)$, from i to k, which coincides with p_{ij} from i to j and ends with the arc from j to k.

In this work, only loopless paths are relevant. A path is loopless if all of its nodes are different.

In a network, the topology of which is represented by a graph, edges and nodes may have one or more attributes, such as weights or other parameters. The calculation of a path between two nodes in a graph starts with the choice of the metric that will define it. In this work, two types of metrics will be considered: additive and concave. The additive weight of a path is equal to the sum of the individual weights of its elements, while its concave weight corresponds to the minimum weight among the weights of all elements that belong to it. For example, in telecommunications networking, delay may be seen as an additive weight as the delay of each link will contribute to the delay of the complete path. Throughout this study, node weights are not considered, so only link weights will affect path calculation. In addition, network graphs will be referred to as networks or graphs.

Telecommunication networks are the context of application of the algorithms studied in this work, on that account additive and concave weights will be denominated as costs and bandwidths (or capacities), respectively, for easier reference. The cost and bandwidth of an edge between nodes i and j are given by c_{ij} and b_{ij}, respectively, and both are assumed to be positive (unless

otherwise stated). Following this, the cost of a path p (additive metric) is given by:

$$c(p) = \sum_{(i,j)\in p} c_{ij} \tag{2.1}$$

And the bandwidth of path p (concave metric) is given by:

$$b(p) = \min_{(i,j)\in p} b_{ij} \tag{2.2}$$

Additionally, a path can also be evaluated in terms of hop-count, which is an additive weight and corresponds to the number of arcs (hops) in that path.

2.2 Shortest and Widest Paths

A path between two nodes that has a cost equal to the minimum additive cost among all paths between those two nodes is called a shortest path. Let P_{st} designate the set of (loopless) paths from s to t and A_p the set of edges of path p ($p \in P_{st}$). In [7], Dijkstra presents an algorithm that solves the problem of finding the shortest path between a source node s and a target node t that is given by:

$$p^* = \arg\min_{p\in P_{st}} c(p) \tag{2.3}$$

In [1], forward and reverse Dijkstra's algorithms are presented. The first is the original Dijkstra's algorithm presented in [7] that, in addition to solving the problem in Equation (2.3), can also be used to calculate the shortest path from a given node to every other node in the graph. In other words, the shortest path tree. If implemented using a binary heap, it has a running time of $O(|A|\log_2|N|)$ [1, page 116]. The second, the reverse Dijkstra's algorithm, does the opposite, meaning that it obtains the shortest path from every other node to a given node. While the solutions found by these algorithms may be equal in the case of undirected networks, they are not equal in the case of directed networks. Throughout this work, "Dijkstra's algorithm" refers to the one presented in [7], unless otherwise stated.

Ahuja et al. also describe in [1] the bidirectional Dijkstra's algorithm, which is an application of the forward and reverse Dijkstra's algorithms simultaneously. This is used to obtain a shortest path from a source node to a target node, instead of a complete tree like the previous two algorithms.

These algorithms have a wide range of applications, since many systems can be represented by networks. An example of a network can be a road network where nodes are cities and links are the roads that connect them. As roads may have different characteristics depending on the direction of travel (traffic, obstacles, speed limits, inclination, etc), this should be a directed network. In this case, the cost of an arc could be the time that takes to travel from the tail node to the head node and, in that case, the total cost of a path would be the sum of the arc costs that make the path.

There are many algorithms that can compute this kind of path. Among them, the most notable for this study, for the simplicity and computational efficiency, are the Dijkstra's algorithm and the

Breadth-First-Search algorithm (BFS). The latter can be applied to graphs with negative costs, but Dijkstra's algorithm can be adapted to these graphs [3], losing efficiency in nonnegative graphs.

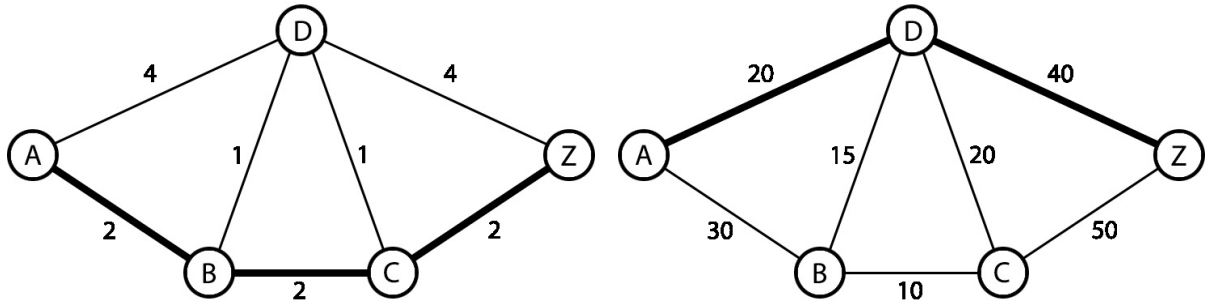
A path that has maximum bandwidth between two nodes is called a widest path. This problem can be formalized as follows:

$$p^* = \arg\max_{p \in P_{st}} b(p) \tag{2.4}$$

Everyday examples to illustrate this problem are fluids transport networks, like plumbing systems or blood vessels. In these cases, it may be more perceptible and easier to understand the impact that a bottleneck arc has on the capacity of the whole path. In the context of telecommunications, a good example is a packet transport network.

In [17], the author makes the observation that several shortest path algorithms can be adapted to compute widest paths. The problem of minimizing a sum is replaced by the maximization of a minimum value. An algorithm for the calculation of paths with maximum capacity for all node pairs was proposed in [8].

It is common to have multiple shortest or widest paths for a given pair of nodes. In those cases, additional criteria may be contemplated. For example, in routing, it is usually a good practice to minimize the used resources. Hence, out of all the widest paths between two nodes, the path with less hops may be preferable. In Figure 2.2, there is an example of a shortest path and a widest path, both with minimum hop-count as secondary criterion.



(a) Shortest path. (b) Widest path.

Figure 2.2: Graphs with additive (left) and concave (right) weights.

In Figure 2.2(a), the numbers next to the edges correspond to their cost. The edges with thicker lines are the ones that are part of a shortest path from node A to node Z, with cost equal to 6. In this small network, it is fairly easy to find another shortest path between those nodes. If instead of $A-B-C-Z$, the path $A-B-D-C-Z$ were to be chosen, the cost would be the same, but the hop-count would be higher.

In Figure 2.2(b), the values represent bandwidths. In this case, there are two widest paths between A and Z: $A-D-Z$ and $A-D-C-Z$. Both have bandwidth equal to 20, but the first one (the one in thick lines in the illustration) has fewer hops.

Hop-count is a simple criterion that is also an additive metric. If all links of a network have unitary costs, the shortest path between two nodes is also the path with less hops and its cost is equal to the number of hops.

2.3 Shortest-Widest and Widest-Shortest Paths

Even though the focus of this work, in terms of metrics, falls solely on costs and bandwidths, the calculation of an optimal path pair can be the subject of many different criteria. An algorithm for finding the shortest path with bandwidth guarantee can be found in [13, Algorithm 17.1, page 592]; this problem is sometimes referred to as the constrained shortest path problem. Wang and Crowcroft [22] proposed an algorithm for the calculation of the shortest path among all paths of maximum bandwidth and designated it shortest-widest path.

The approach considered in this work when multiple criteria are relevant corresponds to the calculation of an optimum path considering lexicographic multi-criteria. This means the shortest-widest path problem is given, according to the lexicographic method described in [12], by the lexicographic minimization of f_i (f_1 followed by f_2):

$$p^* = \arg\min_{p \in P_{st}} f_i, \quad i = 1,2 \tag{2.5}$$

where f_1 and f_2 are

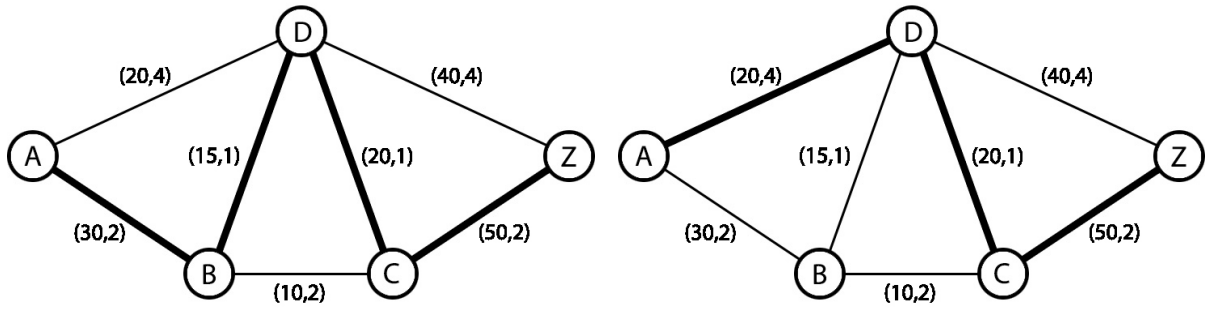
$$\begin{cases} f_1 = 1/b(p) \\ f_2 = c(p) \end{cases}, \text{ with } p \in P_{st}. \tag{2.6}$$

The resolution of Equation (2.5) is equivalent to selecting the path with minimum cost among all paths with the largest bandwidth: the shortest among the widest. The widest-shortest path problem is formalized in a similar way, but with

$$\begin{cases} f_1 = c(p) \\ f_2 = 1/b(p) \end{cases}, \text{ with } p \in P_{st}. \tag{2.7}$$

In this case, the minimization occurs first for the cost and then for the inverse of the path's bandwidth (maximizing $b(p)$), resulting in the selection of one of the shortest paths: the widest among the shortest.

Figure 2.3 exemplifies both shortest-widest and widest-shortest paths in the same graph. Values associated with the edges represent bandwidth and cost, in this order, and coincide with the values presented in Figure 2.2. This is important, because it shows how different paths can be depending on the chosen criteria. While there were multiple shortest paths in Figure 2.2(a) and multiple widest paths in Figure 2.2(b) and the decisive criterion was the hop-count, in Figure 2.3 the provided criteria leads to single solutions. In Figure 2.3(a), the widest-shortest path from A to Z is $A-B-D-C-Z$, with cost 6 and bandwidth 15. In Figure 2.2(a), the shortest path was $A-B-C-Z$,



(a) Widest-shortest path.

(b) Shortest-widest path.

Figure 2.3: Graphs with multiple weights (bandwidth, cost).

which in this case would be a non-optimal solution, because, despite the fact that it has cost 6, the bandwidth is just 10. In Figure 2.3(b), the shortest-widest path is presented for the same set of source and target nodes. Now, the path is $A-D-C-Z$, with bandwidth 20 and cost 7. This contrasts with the widest path obtained in Figure 2.2(b), because even though the bandwidth of the path $A-D-Z$ is equal to 20, the cost is higher ($8 > 7$).

Dijkstra's algorithm [7] can also be adapted to solve these problems as will be seen in Section 4.1.

2.4 Edge-Disjoint Path Pairs

In global path protection, the working path between a source and a target nodes is protected by a backup path, which ensures data transfer in the event of a fault resulting in the failure of the active path.

Assuming every edge has a risk of failure, the protection and active paths should be as disjoint as possible. A pair of disjoint paths between given source and target nodes is a set of two paths that do not share nodes or edges (except the source and target nodes). From here on, path pairs are assumed to be sets of two paths that have the same source and target nodes and, consequently, share these nodes. A path pair is edge-disjoint if no edges are shared between the two paths. Likewise, a pair is node-disjoint if the paths do not share nodes. It should also be noted that, since an edge is essentially defined by a pair of nodes, if two paths are node-disjoint, then they are also edge-disjoint. However, edge-disjoint paths may not be node-disjoint. Figure 2.4 shows a path pair ($A-D-Z$ in red and $A-B-D-C-Z$ in blue) that is edge-disjoint, but not node-disjoint, as they are joint at node D. This work focuses on edge-disjointness rather than node-disjointness and edge-disjoint path pairs may be referred to simply as disjoint path pairs.

The set of edge-disjoint path pairs from s to t is designated by $\bar{P}_{st} = \{(p,q) : A_p \cap A_q = \emptyset, p \neq q, p, q \in P_{st}\}$, with A_p and A_q being the set of arcs that make paths p and q, respectively.

A maximally edge-disjoint path pair (p,q) is a path pair that minimizes $A_p \cap A_q$. If there is at least one edge-disjoint pair between s and t, then the calculation of the maximally edge-disjoint

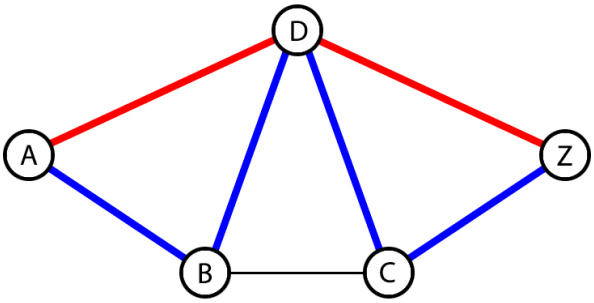


Figure 2.4: Edge-disjoint path pair that is not node-disjoint.

path pair should return a pair (p,q) with $A_p \cap A_q = \emptyset$ and, in this case, it is an edge-disjoint path pair. If no edge-disjoint pair exists, it might still be an option to use a maximally edge-disjoint pair, since it minimizes the use of shared resources and, therefore, the risk of failure of the pair.

The calculation of a set k ($k \geq 2$) of edge-disjoint paths with minimum additive cost (min-sum cost) was proposed by Suurballe [20]. In [21], the calculation of edge-disjoint path pairs from a source node to every other node, such that each pair has minimum additive cost, was proposed by Suurballe and Tarjan. Both algorithms can be used to solve the problem in Equation (2.8).

$$(p^*,q^*) = \arg \min_{(p,q)\in \bar{P}_{st}} [c(p)+c(q)] \tag{2.8}$$

In [2], the authors examined the complexity of different variants of the min-sum edge-disjoint paths problem and proposed heuristics to address them. After the minimization of the total cost of the pair, the problems Beshir and Kuipers [2] set out to answer had a secondary criteria such as the minimization of the shorter path (min-min), the minimization of the longer path (min-max), the limitation of the costs according to given values (bounded) and maximization of the minimum bandwidth of both paths (widest).

Edge-disjoint path pairs based on concave metrics will require further notation. So, given a pair (p,q), the minimum bandwidth among the pair is denoted

$$b_m(p,q) = \min[b(p),b(q)] \tag{2.9}$$

the bandwidth of the widest path of the pair is

$$b_M(p,q) = \max[b(p),b(q)], \text{ with } b_m(p,q) > 0 \tag{2.10}$$

and the sum of the bandwidths of the pair is

$$b_S(p,q) = b(p)+b(q) = b_m(p,q)+b_M(p,q), \text{ with } b_m(p,q) > 0 \tag{2.11}$$

The value of $b_m(p,q)$ is the protected path pair's bandwidth, assuming path protection is being used. In this context one may say $b_m(p,q)$ is the bandwidth of the path pair.

For network protection, multi-path and disjoint routing may require the determination of a set of edge-disjoint paths that maximize the sum of the bandwidth of a set of edge-disjoint paths. Shen et al. [18] tackled two inter-related problems formulated as questions (not as optimization problems):

1. Is there an edge-disjoint path pair (p,q) with bandwidth sum large enough to satisfy a bandwidth guarantee?
2. Is there an edge-disjoint path pair (p,q) such that each of the paths in the pair satisfies a (different) specific bandwidth guarantee?

The first problem was designated Widest Pair of Disjoint Paths Coupled (WPDPC) and the second Widest Pair of Disjoint Paths Decoupled (WPDPD) in [18].

The authors proved that the two problems are NP-Complete and proposed integer linear programming (ILP) formulations for solving them; additionally, they also proposed two heuristics.

The ILP formulation of the first problem maximizes $b_S(p,q)$ and its solution allows to answer the first question. The second ILP formulation seeks to maximize the bandwidth of both paths with the constraint that each satisfies a (different) specific bandwidth guarantee. The solution to this ILP allows to solve the second problem.

The authors evaluate the performance of their proposed heuristics with the optimal solution of the ILP formulations. In the case of the maximization of $b_S(p,q)$, they conclude that the proposed heuristic presents about 1/3 optimal solutions. In the case of the second ILP formulation, the result analysis in [18] is inaccurate, because the underlying optimization corresponding to the second problem is a bicriteria optimization problem with constraints. In fact, in table II of [18], the heuristic presents solutions which weakly dominate the optimal solutions of the ILP formulation. This is because, for one path, the heuristic and the ILP formulation obtain the same bandwidth, but, for the other, the heuristic finds a larger bandwidth than the ILP solution (while both satisfy the constraints of WPDPD).

An heuristic algorithm for finding a path pair that maximizes the sum of the bandwidths of a node-disjoint pair is also presented in [6]. An interesting related problem is considered in [10]: how to find a pair of disjoint paths between a source and a target node such that the sum of the bandwidth in the path pair is larger than a given bandwidth guaranteed value, while minimizing the total additive cost of the path pair. They designated this problem as finding the shortest pair of disjoint paths with bandwidth guarantee (SPDP-BG). This problem was shown to be NP-Complete and an heuristic for solving it was also proposed in [10].

Bhandari [3] discussed possible implementations of Suurballe's algorithm [21] for calculating an edge-disjoint path pair of min-sum cost. The underlying algorithm in this approach was based on Dijkstra's or BFS. Moreover, the author proposed an alternative algorithm (Bhandari's algorithm), which can be more efficient than Suurballe's algorithm for calculating an edge-disjoint path pair. This problem can be solved in polynomial time; however, if additional restrictions are added, the problem may become NP-Complete [2].

Since Bhandari's algorithm is essentially the calculation of the best path in the original and transformed graphs (see Section 2.4.3), it can be adapted to calculate widest edge-disjoint path pairs. The widest edge-disjoint path pair problem consists in obtaining a path pair that maximizes the bandwidth of the narrowest path of the pair. The adaptation of Bhandari's algorithm to this problem maximizes the minimum bandwidth of the pair, b_m:

$$(p^*, q^*) = \arg \max_{(p,q) \in \bar{P}_{st}} b_m(p,q) \tag{2.12}$$

Solving the problem given in Equation (2.12) does not ensure the maximization of $b_M(p,q)$ nor $b_S(p,q)$.

2.4.1 Suurballe's Algorithm

Suurballe and Tarjan's algorithm, presented in [21], starts by using Dijkstra's algorithm to obtain the shortest path tree with the root being the provided source node s. Then, G suffers a transformation. Let G' be the modified graph in which every arc a_{ij} has the reduced cost

$$c'_{ij} = c_{ij} + \pi(i) - \pi(j) \tag{2.13}$$

where $\pi(x)$ is the cost of the computed path from source node s to node x.

This way, the cost of every arc that belongs to the tree is 0, because $c_{ij} = \pi(j) - \pi(i)$. Note that it is important to represent edges as pairs of symmetrical arcs as they will have different modified costs. If an arc is not part of the tree, then $c'_{ij} \geq 0$. If an arc that was not part of the tree has $c'_{ij} = 0$, it means there might be more than one possible shortest path tree.

Next, arcs in G' that are part of the shortest path from the source node s to the target node t should be replaced by arcs with zero cost and pointing from t to s.

Dijkstra's algorithm is now used to calculate the shortest path between s and t, p', this time in the modified graph G'. If p' does not contain any reversed arc from p, the solution is $p^* = p$ and $q^* = p'$. Otherwise, paths p and p' are *interlaced* by the reversed arcs of p present in p'. The removal of the interlacing part of both paths (the shortest paths from s to t in G and G') will result in two paths that compose the min-sum path pair. From here on, this process will be called *deinterlacing*.

According to [21], the running time is $O(|A| \log_{(1+|A|/|N|)} |N|)$.

2.4.2 Deinterlacing

Let C be a set that contains the reversed arcs from path p' and the corresponding symmetrical arcs from path p. Deinterlacing can be summarized as removing the arcs that are part of C from both paths and exchanging the remaining arcs to form the edge-disjoint path pair.

Let (p^*, q^*) be the shortest path pair solution that results from deinterlacing p and p'. Path p^* can be obtained through Algorithm 1. The procedure to get q^* is very similar and can be found in Appendix A. The deinterlacing process can be performed in both directions, but, in this case, it will be done from source to target node to simplify the explanation. This means that p^* and q^* will be defined by the first arc of p and p', respectively, otherwise they would be defined by the last arc.

Algorithm 1 Deinterlacing p^*

Require: Paths p and p' from s to t (described by the successive successors of s).
Ensure: Calculates path p^* of path pair (p^*, q^*).

1: $k \leftarrow 1$ ▷ Starts from the first element of p
2: $p^* \leftarrow s$ ▷ s is the first node of p^*
3: **repeat** ▷ p^* gets arcs and nodes from p
4: **if** $a'_k \notin C$ **then** ▷ If arc from p is not reversed in p'
5: $p^* \diamond (v'_k, v'_{k+1})$ ▷ Arc $a'_k = (v'_k, v'_{k+1})$ and next node v'_{k+1} are added to p^*
6: $k \leftarrow k+1$ ▷ Increment k
7: **else**
8: $k \leftarrow k'$, *with* $v''_{k'} = v'_k$ ▷ k gets p''s index of the last node in p^*
9: **repeat** ▷ p^* gets arcs and nodes from p'
10: **if** $a''_k \notin C$ **then** ▷ If arc from p' is not reversed in p
11: $p^* \diamond (v''_k, v''_{k+1})$ ▷ Arc $a''_k = (v''_k, v''_{k+1})$ and next node v''_{k+1} are added to p^*
12: $k \leftarrow k+1$ ▷ Increment k
13: **else**
14: $k \leftarrow k'$, *with* $v'_{k'} = v''_k$ ▷ k gets p's index of the last node in p^*
15: **break** Line 9 cycle
16: **end if**
17: **until** $v''_{k'} = t$ ▷ Algorithm terminates when t is reached
18: **end if**
19: **until** $v'_k = t$ ▷ Algorithm terminates when t is reached
20: **return** p^* (as a set of alternating nodes and arcs)

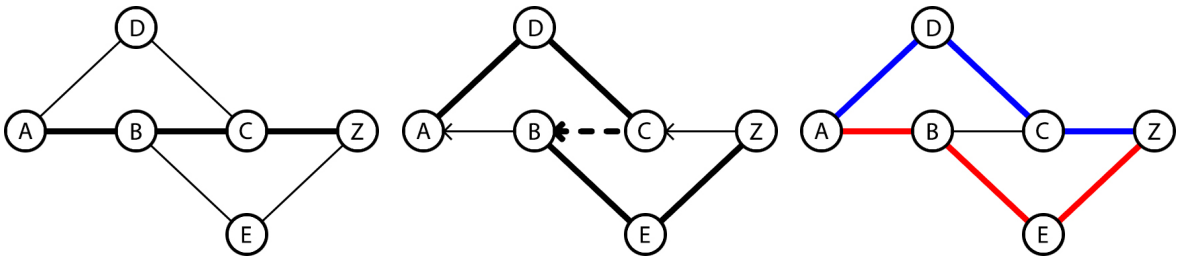
The deinterlacing process delineated in Algorithm 1 can be applied to path pairs obtained through any metric, so it is also used in Bhandari's algorithm and the algorithm proposed in [14], but the individual costs and/or bandwidths of the paths p^* and q^* need to be recalculated. Nevertheless, two things are guaranteed:

$$c(p^*) + c(q^*) = c(p) + c(p') \tag{2.14}$$

and

$$b_m(p, p') = b_m(p^*, q^*) \tag{2.15}$$

In Figure 2.5 there is a simple example of deinterlacing. For this purpose and for a simpler description, let edge weights be ignored. Let the set of edges (and arc) in thick lines in Figure 2.5(a) and Figure 2.5(b) be the optimal paths p found in G and p' found in G'.



(a) Path p in G. (b) Path p' in G'. (c) Path pair.

Figure 2.5: Deinterlacing example.

Reversed arcs that are part of p' are in dashed thick lines. In this case, it is only one, arc (C,B).

Let p^*'s first arc (the one whose tail node is s) be the first arc of p. Starting from s, p^* will be equal to p until an arc that is reversed in path p' is found (arc $B-C$). So far, $p^* = \langle A,(A,B),B\rangle$. From this point, p^* is concatenated with a section of p' that goes from the last node in p^* to t or until another reversed arc is found. In this example, there is only one set of consecutive reversed arcs in p', so p^* alternates between p and p' only once. In the end, $p^* = \langle A,(A,B),B,(B,E),E,(E,Z),Z\rangle$ (in red in Figure 2.5(c)).

The same logic applies to q^*, except its first arc is the first arc of p'. Hence, $q^* = \langle A,(A,D),D,(D,C),C\rangle$ when the first reversed arc is found and $q^* = \langle A,(A,D),D,(D,C),C,(C,Z),Z\rangle$ (in blue in Figure 2.5(c)) when Z is reached.

2.4.3 Bhandari's Algorithm

Bhandari's algorithm, presented in [3], is somewhat similar to Suurballe's and can be summed up as: calculating the shortest path between s and t using Dijkstra's algorithm, transforming the graph G into G', recalculating the shortest path in G' using the modified version of Dijkstra or BFS and deinterlacing the two paths.

The main difference is the transformation of the graph. Instead of calculating the reduced cost of every arc in G, only the links that are part of the first shortest path (let it be p) are substituted by reversed arcs with symmetrical costs. This, in turn, forces the use of a shortest path algorithm that can be ran on graphs with negative costs. The fact that this transformation makes common arcs have negative cost in p' means that the sum of the costs of p and p' is equal to the sum of the costs of the resulting path pair.

Another difference is the fact that there is no need to calculate the complete shortest path tree in the first step, as the shortest path from s to t is sufficient.

Because of its graph transformation and in contrast with Suurballe's algorithm, Bhandari's needs to deal with negative cost arcs. This takes more time than running the standard version of Dijkstra on G'. However, Bhandari's algorithm may have an advantage in the first steps, especially for larger and denser (high arc-to-node ratio) networks, as it does not need to calculate the entire tree nor to change the cost of every arc. According to author, in [3, page 92], his algorithm and

Suurballe's have similar efficiency and the advantage of his is that it is *straightforward, direct and convenient for a practicing engineer*.

The maximization of the bandwidth of the narrowest path, i.e., the one with minimum bandwidth in the pair, is an important problem in QoS routing, because this bandwidth is the one that is actually protected and can be guaranteed in case of failure of the active path.

This problem was formulated in Equation (2.12) and can be solved by an adaptation of Bhandari's algorithm, which makes use of the widest path variant of Dijkstra's algorithm and a graph transformation that includes concave metrics.

When it comes to the graph transformation, bandwidth values are altered differently. While costs of reversed arcs take (negative) symmetrical values, the bandwidths of the reversed arcs of the widest path must take very large values to mimic the effect of the cost modification. These altered values should be larger than the bandwidth of the widest path in the original network for a given source-target node pair. As no path can be composed solely by reversed arcs after the graph transformation into G', no path will have bandwidth larger than the widest path in the original graph G. This value can be used as a threshold so that any higher value is considered to be a practical representation of infinity.

2.4.4 Maximally Edge-Disjoint Path Pairs

The difference between edge-disjoint and maximally edge-disjoint path pairs is somewhat subtle and so is the difference between the methods used to obtain them. In fact, it all lies in the graph transformation. The calculation of a maximally edge-disjoint path pair implies less strictness when it comes to paths sharing resources.

The first step is (as in Bhandari's algorithm) the calculation of the shortest (widest) path: p. Then the network is transformed into G', reversing the arcs of p with negative cost (infinite bandwidth), but without removing the directed arcs from s to t in p. The cost (bandwidth) of the arcs in p must now take values should be sufficiently large (sufficiently small) [3, 14]. In practical terms, this means the cost should be higher than the sum of the costs of all arcs in the original graph, as no path can possibly have a worse cost; the bandwidth must be take a value less than the lowest bandwidth in the original graph.

For maximally edge-disjoint path pairs, an arc of p in the s-to-t direction will have its cost/bandwidth changed according to:

$$\begin{cases} c'_{ij} = c_{ij} + \sum_{a_k \in A} c(a_k) \\ b'_{ij} = \min_{a_k \in A} b(a_k)/2 \end{cases} \tag{2.16}$$

This has the effect that the directed arcs in p (from s to t) will only be used if no other option is left. In this modified graph (G'), the shortest (widest) path p' is calculated. If p' contains no reversed arcs from p, p and p' are the solution: $(p^* = p, q^* = p')$. Otherwise the p and p' must be deinterlaced to obtain the solution path pair: (p^*, q^*). If $A_{p*} \cap A_{q^*} \neq \emptyset$, the pair is not edge-disjoint, and it only shares unavoidable edges.

Chapter 3

Addressed Problems

In this chapter, the problems addressed in this work are explained and formalized. They are, in the order they are presented, Widest Edge-disjoint Path Pair Lexicographic Optimization problem, the maximization of the sum of the bandwidths of the pair and the Widest Pair of Disjoint Paths Decoupled problem.

The problems are organized this way because the first one is a development from the problem solved by Bhandari, discussed in the last chapter, and the second one is also an optimization problem. In contrast with these two, the last one is not an optimization problem, but, instead, is a search for any edge-disjoint path pair that respects the given restrictions. Each problem is also presented in an integer linear programming (ILP) formulation.

In directed graphs, symmetrical arcs are considered different. However, for the purpose of calculating edge-disjoint paths for protection in communication networks, using arc (v_i, v_j) in one path should prevent the use of arc (v_j, v_i) in the corresponding protection path. The ILP formulations presented here avoid this type of solution, which may arise when considering concave metrics. Hence, if applied to directed graphs all arcs are required to have a topologically symmetrical arc (which may be enforced by adding arcs of sufficiently high cost and zero bandwidth).

3.1 Widest Edge-disjoint Path Pair Lexicographic Optimization

The problem presented in this section spawns from the widest edge-disjoint path pair problem, which optimizes solely the bandwidth of the minimum bandwidth path of the pair.

The widest edge-disjoint path pair lexicographic optimization problem designates the maximization of the capacity of the minimum bandwidth path, $b_m(p,q)$, followed by the maximization of the capacity of the maximum bandwidth path, $b_M(p,q)$, where $(p,q) \in \bar{P}_{st}$.

Finding this edge-disjoint path pair can be achieved by solving the problem that will be formalized next [12]: Let

$$\begin{cases} f_1 = 1/b_m(p,q) \\ f_2 = 1/b_M(p,q) \end{cases}, \text{ with } (p,q) \in \bar{P}_{st}. \tag{3.1}$$

The problem addressed in this work is the lexicographic optimization of $f_i, i = 1,2$:

$$(p^*, q^*) = \arg \min_{(p,q) \in \bar{P}_{st}} f_i, \quad i = 1,2 \tag{3.2}$$

The minimization of f_1 and f_2 corresponds to the maximization of $b_m(p,q)$ and $b_M(p,q)$, respectively.

This optimization problem Widest Edge-disjoint Path Pair Lexicographic Optimization will be denoted as WEDLO. Let us assume, without loss of generality, that $b_m(p,q) = b(p)$ and that $b(q) \geq b(p)$. If more than one edge-disjoint path with p exists or if more than one path with bandwidth equal to $b(p)$ exists, the minimization of f_1 problem has multiple optimal solutions (multiple path pairs that maximize $b_m(p,q)$). Solving the problem in Equation (3.2) allows to obtain the widest possible path q for the pair of widest bandwidth.

In [15], a sequential lexicographic optimization procedure is formulated. This approach is not required for solving WEDLO, because a simple algorithm is used to obtain the minimum value of the first objective function, and a single ILP formulation for the second (and last) objective function can be written. In [4], a hierarchical multi-criteria routing model associated with a two-path traffic splitting routing method is proposed. This type of approach could also be used to solve WEDLO, considering $b_m(p,q)$ as the first level objective function and $b_M(p,q)$ as the second level objective function.

For the ILP formulation, some additional notation is necessary:

Parameters:

s	source node; $s \in N$
t	target node; $t \in N$
B_m	is $\max_{(p,q) \in \bar{P}_{st}} b_m(p,q)$, with $B_m > 0$
M	a sufficiently large constant, in this case, $\max_{p \in P_{st}} b(p)$

Variables:

x_{ij}^k	1 if path k is using arc $(i,j) \in A$, 0 otherwise, with $k = 1,2$
y_k	bandwidth of path k

The ILP formulation is as follows:

max y_1

Subject to:

$$y_k \geq B_m, \ k \in \{1,2\} \tag{3.3a}$$

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}, \ \forall i \in N, \ k \in \{1,2\} \tag{3.3b}$$

$$x_{ij}^1 + x_{ji}^1 + x_{ij}^2 + x_{ji}^2 \leq 1, \ \forall (i,j) \in A \tag{3.3c}$$

$$y_k \leq b_{ij} x_{ij}^k + M(1 - x_{ij}^k), \ \forall (i,j) \in A, \ k \in \{1,2\} \tag{3.3d}$$

$$x_{ij}^k \in \{0,1\}, \ \forall (i,j) \in A, \ k \in \{1,2\} \tag{3.3e}$$

This ILP formulation (Equation (3.3)) is modified from [18, Fig. 5] to suit our needs for WEDLO. Note that this formulation does not prevent the resulting paths from having cycles. In Equation (3.3) we maximize the bandwidth of path $k = 1$, while inequality (3.3a) ensures both paths must have at least a bandwidth equal to B_m. Equation (3.3b) expresses the standard flow conservation law for variables x_{ij}^k. Variables x_{ij}^k are additionally bounded by inequality (3.3c), which ensures no edge (represented by a pair of symmetrical arcs) can be shared by the paths and no two arcs representing the same edge can be part of the same path. Inequality (3.3d) identifies the capacity of a bottleneck arc for each path, thus defining the corresponding bandwidth. Equation (3.3e) defines the bounds for decision variables.

3.2 Maximum Sum Optimization

The path pair which maximizes $b_S(p,q)$ is given by:

$$(p^*, q^*) = \arg \max_{(p,q)\in \bar{P}_{st}} b_S(p,q) \tag{3.4}$$

This a well-known problem and more common than WEDLO. The aim is to supply the maximum possible bandwidth with two edge-disjoint paths.

Shen et al. proved in [18] that the problem addressed in this section is a NP-Complete problem, proposed an heuristic to tackle it, and gave an ILP formulation for this problem, which is next adapted to fit the directed representation of undirected networks:

max $\quad y_1 + y_2$

Subject to:

$$y_k \geq 0,\ k \in \{1,2\} \tag{3.5a}$$

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases},\ \forall i \in N,\ k \in \{1,2\} \tag{3.5b}$$

$$x_{ij}^1 + x_{ji}^1 + x_{ij}^2 + x_{ji}^2 \leq 1,\ \forall (i,j) \in A \tag{3.5c}$$

$$y_k \leq b_{ij} x_{ij}^k + M(1 - x_{ij}^k),\ \forall (i,j) \in A,\ k \in \{1,2\} \tag{3.5d}$$

$$x_{ij}^k \in \{0,1\},\ \forall (i,j) \in A,\ k \in \{1,2\} \tag{3.5e}$$

The formulation in Equation (3.5) and Equation (3.3) differ on the constraint of y_k (see (3.5a) and (3.3a)). While in WEDLO's formulation the B_m limit imposed that the minimum bandwidth of the pair, $b_m(p,q)$, should be maximum and only $b_M(p,q)$ was maximized, in this problem no positive limit is set and the sum is maximized. Restrictions related to the disjointness of the pair and edge utilization are maintained.

3.3 Widest Pair of Disjoint Paths Decoupled

This problem is the search for an edge-disjoint path pair between nodes s and t that comply with minimum limits. These limits are denoted X_1 and X_2, with $X_1 > X_2$. Note that if $X_1 = X_2$, the problem is solved by Bhandari's algorithm for widest edge-disjoint path pairs, because it is known that this algorithm maximizes the minimum bandwidth of the pair, $b_m(p,q)$, so, if this bandwidth is inferior to the set limits, then there is no path pair (for the given pair of nodes) that can satisfy the restrictions,

The objective is to find a pair (p,q) that has:

$$\begin{cases} b_M(p,q) \geq X_1 \\ b_m(p,q) \geq X_2 \end{cases} \tag{3.6}$$

In contrast with the two problems presented before, there may be more than one solution to this problem and these solutions may not be comparable. In other words, instead of being a search for the best possible path pair according to some criterion, it can be solved by finding a path pair that satisfies the constraints. If such a path exists, the problem is solved. In [18], the authors prove this problem is NP-Complete.

This is of great practical interest, because, even though the solutions may not be optimal, solving this problem allows to find paths that meet specific requirements that may be needed to guarantee QoS.

The ILP formulation is the following:

max ρ

Subject to:

$$\rho \geq 0 \tag{3.7a}$$

$$\sum_{(i,j)\in A} x_{ij}^k - \sum_{(j,i)\in A} x_{ji}^k = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = t \\ 0 & \text{otherwise} \end{cases}, \ \forall i \in N, \ k \in \{1,2\} \tag{3.7b}$$

$$x_{ij}^1 + x_{ji}^1 + x_{ij}^2 + x_{ji}^2 \leq 1, \ \forall (i,j) \in A \tag{3.7c}$$

$$\rho X_k \leq b_{ij} x_{ij}^k + M(1 - x_{ij}^k), \ \forall (i,j) \in A, \ k \in \{1,2\} \tag{3.7d}$$

$$x_{ij}^k \in \{0,1\}, \ \forall (i,j) \in A, \ k \in \{1,2\} \tag{3.7e}$$

The value of ρ resulting from the execution of the ILP can be used to measure the difference between the solution and the target values. If $\rho < 1$, the constraints are not satisfied. If $\rho > 1$, then the solution has surpassed the limits. If $\rho = 1$, then the path pair has bandwidths that are equal to the limits. However, during experiences using CPLEX, there were some cases in which $\rho = 1^-$ ($\rho < 1$ and $1 - \rho \approx 0$) and, upon verification, it was clear that the pair did meet the criteria.

It is important to note that the maximization of ρ transforms this into an optimization problem. As two conflicting objective functions are being optimized simultaneously, the problem is bicriteria and may not have optimal solutions, but Equation (3.7d) constraints the result and allows to verify if the limits X_1 and X_2 are respected. The fact that there may be no optimal solution, but only non-dominated solutions, should be kept in mind during the performance analysis of the proposed heuristics in Chapter 6. A non-dominated solution is a feasible solution such that no improvement in any criterion may be achieved without sacrificing at least one of the other criteria [19].

Chapter 4

Auxiliary Implemented Algorithms

This section enumerates various algorithms that are of great importance, either as inspiration or as a term of comparison. First, Dijkstra's algorithm for shortest, widest, shortest-widest and widest-shortest paths. Second, the modified version of Dijkstra's algorithm for shortest path calculation in graphs with negative costs (but no negative cycles), which is used by the last auxiliary algorithm, Bhandari's edge-disjoint (and maximally edge-disjoint) path pair algorithm.

4.1 The Dijkstra's Algorithm

Although Dijkstra's algorithm [7] is well known, it is reviewed here in detail, because it is the foundation of the proposed and implemented algorithms in this dissertation. Dijkstra's algorithm can very efficiently obtain the shortest path tree or just the shortest path between two nodes (source node s and target node t), depending on the condition used to terminate it. It assumes all costs are positive, $c_{ij} \geq 0$, for all $(i, j) \in A$.

Algorithm 2 (Dijkstra's algorithm) requires some additional notation:

k	Present node.
u	Candidate successor of the present node.
S	Set of non-permanent nodes.
$\psi(j)$	Node preceding node j.
$\pi(j)$	Cost of the computed path from s to j.
$\beta(j)$	Bandwidth of the computed path from s to j.

Algorithm 2 Dijkstra's Algorithm for the Calculation of a Chosen Variant Path

Require: $G = (N,A)$, B matrix with arcs' bandwidth $(b_{ij}, (i,j) \in A)$, C matrix with arcs' costs $(c_{ij}, (i,j) \in A)$, nodes source s and target t.

Ensure: Calculates p_{st} depending on the chosen variant and its cost $\pi(t)$ and/or bandwidth $\beta(t)$. If it exists, then $\pi(t) \neq \infty$ and/or $\beta(t) \neq 0$.

1: **for** all $i \in N$ **do**
2: $\quad\psi(i) \leftarrow s$ ▷ s is the predecessor of all nodes
3: $\quad\pi(i) \leftarrow \infty$ ▷ No path has been found from s to i
4: $\quad\beta(i) \leftarrow 0$ ▷ No path has been found from s to i
5: **end for**
6: $\pi(s) \leftarrow 0$ ▷ The cost of reaching s, starting from s is 0
7: $\beta(s) \leftarrow \infty$ ▷ The bandwidth to s from s is ∞
8: $k \leftarrow s$ ▷ First node is s
9: $S \leftarrow N - \{s\}$ ▷ s is permanently labeled
10: **repeat**
11: $\quad$**for** every arc $(k,u), u \in S$ **do** ▷ For every non-permanent neighbor node of k
12: $\quad\quad$**if** "condition" **then** ▷ If the path to u can be improved (Table 4.1)
13: $\quad\quad\quad\psi(u) \leftarrow k$ ▷ k becomes the predecessor of u
14: $\quad\quad\quad\pi(u) \leftarrow \pi(k) + c_{ku}$ ▷ $\pi(u)$ is updated
15: $\quad\quad\quad\beta(u) \leftarrow \min[\beta(k), b_{ku}]$ ▷ $\beta(u)$ is updated
16: $\quad\quad$**end if**
17: $\quad$**end for**
18: $\quad k \leftarrow$ "best candidate" ▷ Another node is chosen (Table 4.1)
19: $\quad S \leftarrow S - \{k\}$ ▷ Node is permanently labeled
20: **until** $k = t$
21: **return** p_{st} (described by the successive predecessors of t if $\pi(t) \neq \infty$ or $\beta(t) \neq 0$) and $\pi(t)$ and/or $\beta(t)$

Variant	"condition"	"best candidate"
shortest path	$\pi(u) > \pi(k) + c_{ku}$	$\arg\min_{j \in S} \pi(j)$
widest path	$\beta(u) < \min[\beta(k), b_{ku}]$	$\arg\max_{j \in S} \beta(j)$
widest-shortest path	$\pi(u) > \pi(k) + c_{ku} \vee$ $(\pi(u) = \pi(k) + c_{ku} \wedge$ $\beta(u) < \min[\beta(k), b_{ku}])$	$\arg\max_{j' \in S'} \beta(j')$ $S' = \{j : j = i^* \wedge \pi(i^*) = \min_{i \in S} \pi(i)\}$
shortest-widest path	$\beta(u) < \min[\beta(k), b_{ku}] \vee$ $(\beta(u) = \min[\beta(k), b_{ku}] \wedge$ $\pi(u) > \pi(k) + c_{ku})$	$\arg\min_{j' \in S'} \pi(j')$ $S' = \{j : j = i^* \wedge \beta(i^*) = \max_{i \in S} \beta(i)\}$

Table 4.1: Variants of the implemented Dijkstra's algorithm

This is a slightly altered version of Dijkstra's algorithm. While it is very similar to the one presented in [7], this one is designed to calculate shortest, widest, shortest-widest or widest-shortest paths, depending on the chosen variant. To perform the last two variants, shortest-widest and

widest-shortest, nodes have both cost and bandwidth labels. The algorithm needs to adjust to every type of path, but the differences are set solely on Line 12 and Line 18. These differences are displayed in Table 4.1.

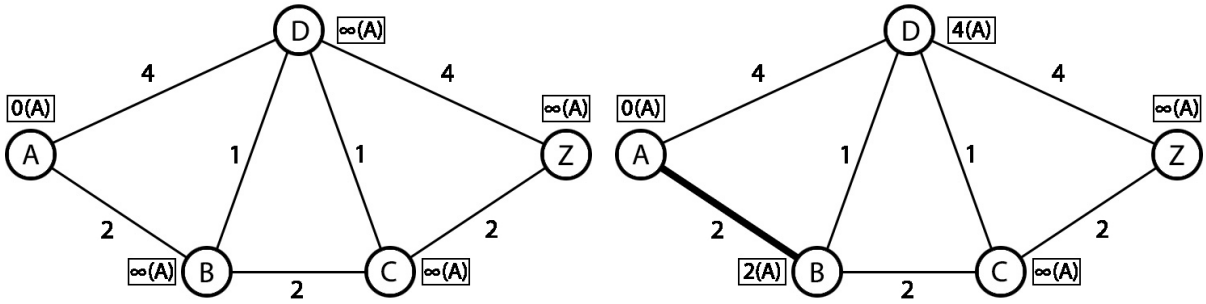
Note that the shortest path variant corresponds to the Dijkstra's algorithm proposed in [7], which will be exemplified next, followed by a short explanation regarding the differences between this and the other variants.

For the sake of simplicity, this example will ignore the bandwidth labels. The algorithm starts by attributing a path cost equal to infinity to every node except for *s*. In practical terms, infinity is represented by a large number, larger than the sum of the costs of all arcs in the network. This is to make sure these paths can be improved, because a path can't have more arcs than the graph.

Then, starting from *s*, it scans the arcs that spawn from this node and labels the head nodes according to the cost of the path to them. Out of all labeled nodes (nodes with path cost that is not infinity), the one with the least path cost is marked as permanent and becomes *k*. The procedure is repeated until the condition in Line 20 is met.

If the algorithm was set to stop when all nodes have been marked as permanent, then the product would be a complete shortest path tree. However, if there is no interest in computing the entire tree, which is the case in this work, the algorithm can be terminated once *t* is marked as permanent. Due to this premature termination, the tree may be incomplete, but some time might be saved.

Figure 4.1 and Figure 4.2 show, as an example, how the shortest path from the source node *A* to the target node *Z* depicted in Figure 2.2(a) was obtained. Every node label is initialized in Figure 4.1(a). The cost to every node starts by being equal to infinity except to the source node, which is 0. In this first step, every node is considered to be preceded by the source node *A*. Figure 4.1(b) shows the result of the first iteration of the Line 10 cycle. *B* and *D* are labeled with finite costs equal to the path that leads to them and *B* is picked as the best candidate and is marked as permanent.

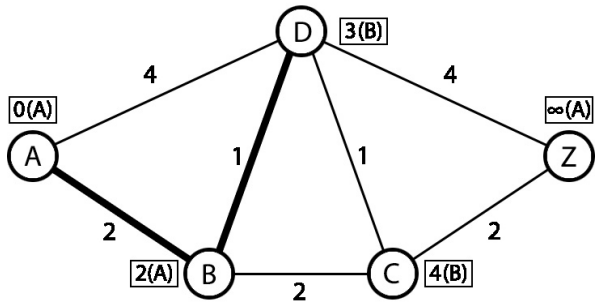


(a) Initialization. (b) Labeling *A*'s neighbor nodes.

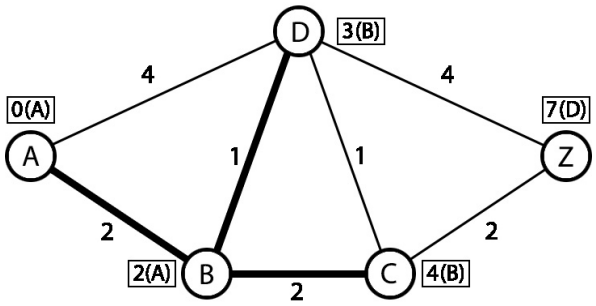
Figure 4.1: First steps of Dijkstra's algorithm.

The same happens to *B*'s neighbors in Figure 4.2(a) with the exception of the already permanent node *A*. *D*'s path is improved and its cost decreases from 4 to 3 when *B* becomes its new

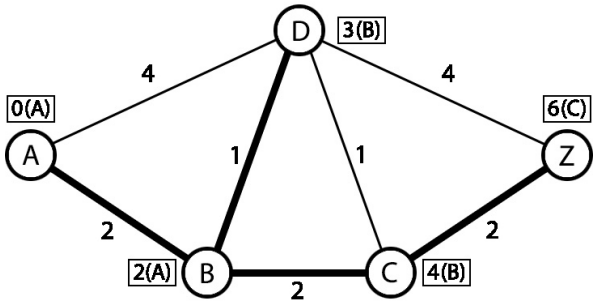
predecessor. After comparing the possible candidates, C and D, $k = D$ and D is marked as permanent. In Figure 4.2(b), the labeling of D's neighbor nodes includes the target node Z. However, C is a better candidate, because its cost is lower, so C becomes a permanent node. Z is relabeled through C and, since it's the last non-permanent node, is finally labeled permanent in Figure 4.2(c).



(a) Labeling B's neighbor nodes.



(b) Labeling D's neighbor nodes.



(c) Complete tree.

Figure 4.2: Middle and final steps of Dijkstra's algorithm.

In this case, even though the condition defined in Line 20 was to stop once the target node was reached, the tree was completely calculated.

If there are multiple optimal solutions, a different optimal path might be found, depending on the order of selection of node k (of identical cost) in Line 18.

The adaptation of the Dijkstra's algorithm to the concave metric requires the few changes that are enunciated in Table 4.1. Paying attention to the bandwidth labels, the first important detail to note is the result of the metric change. Labels are now updated if the condition

$$\beta(u) < \min[\beta(k), b_{ku}] \tag{4.1}$$

is met, because the arc from node k to node u may limit the bandwidth that was already guaranteed up to node k. Thanks to this condition, the bandwidth $b(p_{st})$ (given by $\beta(t)$) is equal to the minimum bandwidth among all arcs that are part of p_{st}.

The second and last major change happens when choosing the new best candidate, k, in Line 18. It is now equal to

$$\arg\max_{j \in S} \beta(j) \tag{4.2}$$

because the widest path problem is a maximization problem.

In the widest-shortest and shortest-widest path variants, the condition to label (or relabel) a node is based on a combination of the shortest and widest path variants, as is the choice of the next node k.

For the widest-shortest path variant, the node is labeled if the cost can be reduced or, if the cost is maintained, the bandwidth can be increased. The best candidate, k, is chosen as the node that has maximum $\beta(j')$, where $j' \in S'$ and S' is the set of nodes that have minimum cost label among all non-permanent nodes. Similarly, for the shortest-widest variant, labeling happens if the bandwidth can be improved or, if bandwidth is maintained, the cost can be decreased. k is a node that has minimum cost among all non-permanent nodes with maximum bandwidth label.

This algorithm can be implemented using language C++. To determine k in Line 18, i.e., the node that leads to the path with larger bandwidth and/or minimum among all non-permanent nodes (depending on the variant), a multi-map from the C++ Standard Library can be used. A multi-map is a container whose elements are the association of keys and values. The key is used to order the elements according to an order relation. In Dijkstra's algorithm (as well as in Algorithm 3 and the proposed algorithms), the elements are ordered by bandwidth in a decreasing fashion and/or by cost in an increasing fashion, depending on the chosen variant. Given two candidate nodes, a and b, being evaluated based on bandwidth, if $\beta(a) > \beta(b)$, then a takes the top position in the multi-map. If the values are equal, the element that was first introduced remains on top. Likewise, if the candidate nodes are evaluated based on cost, then a takes the top if $\pi(a) < \pi(b)$. The shortest-widest and widest-shortest variants make use of both parameters in different order.

Removal of the top element of the multi-map has a constant cost. Insertions, removals and searches on the multi-map have logarithmic complexity on the number of elements, so the complexity of this algorithm is $O(|A| \log_2 |N|)$, which is identical to the complexity of Dijkstra's algorithm using a binary heap [1].

4.2 The Modified Dijkstra's Algorithm

Dijkstra's algorithm, in its simplest form, fails when facing graphs with negative arc costs. However, if no negative loops exist, this problem can be solved by a modified version of the same algorithm. A negative loop (or cycle) is a set of arcs that form a path from a node to itself with negative additive cost. Since it is a minimization problem, the algorithm would not terminate after passing through that loop, because the cost of the path to said node will be reduced every time the cycle is traversed.

In this work, the calculation of shortest paths in graphs with negative cost arcs (and no negative cycles) is very useful to find disjoint path pairs using Bhandari's algorithm. A modified version of Dijkstra's algorithm that solves this problem can be found in [3].

Algorithm 3 Modified Dijkstra's Algorithm for the Calculation of the Shortest Path

Require: $G = (N,A)$, C matrix with arcs' costs $(c_{ij}, (i,j) \in A)$, nodes source s and target t.
Ensure: Calculates p_{st} and its cost $\pi(t)$. If it exists, then $\pi(t) \neq \infty$.

1: **for** all $i \in N$ **do**
2: $\quad \psi(i) \leftarrow s$ ▷ s is the predecessor of all nodes
3: $\quad c(i) \leftarrow \infty$ ▷ No path has been found from s to i
4: **end for**
5: $c(s) \leftarrow 0$ ▷ The cost of reaching s, starting from s is 0
6: $k \leftarrow s$ ▷ First node is s
7: $S \leftarrow N - \{s\}$ ▷ s is permanently labeled
8: **repeat**
9: $\quad$ **for** every arc $(k,u), u \in N$ **do** ▷ For every neighbor node of k
10: $\quad\quad$ **if** $\pi(u) > \pi(k) + c_{ku}$ **then** ▷ If the path to u can be improved
11: $\quad\quad\quad \psi(u) \leftarrow k$ ▷ k becomes the predecessor of u
12: $\quad\quad\quad c(u) \leftarrow c(k) + c_{ku}$ ▷ $c(u)$ is updated
13: $\quad\quad\quad S \leftarrow S + \{u\}$ ▷ u becomes non-permanent
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad k \leftarrow \arg\min_{j \in S} \pi(j)$ ▷ Another node is chosen
17: $\quad S \leftarrow S - \{k\}$ ▷ Node is permanently labeled
18: **until** $k = t$
19: **return** p_{st} (described by the successive predecessors of t if $\pi(t) \neq \infty$

It contrasts with the original version of the Dijkstra's algorithm in the fact that it scans all neighbor nodes, including previously marked permanent nodes. The main difference between Algorithm 2 and Algorithm 3 can be found in the *for* cycle in Line 9. First, all arcs are scanned, not just for non-permanent nodes. And second, if a better path is found, the node is reinserted in S (becomes non-permanent again) in Line 13.

Because bandwidths values are considered to be nonnegative and the widest path problem does not include additive metrics, the modified Dijkstra's algorithm is used just for shortest path calculation.

4.3 Bhandari's Edge-Disjoint Path Pair Algorithm

This algorithm makes use of the two previously presented algorithms to calculate paths, that, through deinterlacing, result in edge-disjoint path pairs of min-sum cost or maximum bandwidth, depending on the considered optimization problem. So, if, for example, the paths calculated in G and then in G' were the shortest paths in these graphs, the resulting path pair would be the shortest edge-disjoint path pair and the sum of the costs of both pairs would be minimum among all path pairs between the two given nodes. On the other hand, if the paths obtained from G and G' were to be the widest paths, then the resulting path pair would be an widest edge-disjoint path pair and $b_m(p^*, q^*)$ would be maximized.

Algorithm 4 Bhandari's Edge-Disjoint Path Pair Algorithm

Require: $G = (N,A)$, B matrix with arcs' bandwidth $(b_{ij},(i,j) \in A)$, C matrix with arcs' costs $(c_{ij},(i,j) \in A)$, nodes source s and target t, and selected variant "widest" or "shortest".

Ensure: Calculates edge-disjoint (or maximally edge-disjoint) path pair (p^*,q^*) (if it exits), such that the sum $c(p^*)+c(q^*)$ is minimized or $b_m(p^*,q^*)$ is maximized, depending on the chosen "variant"

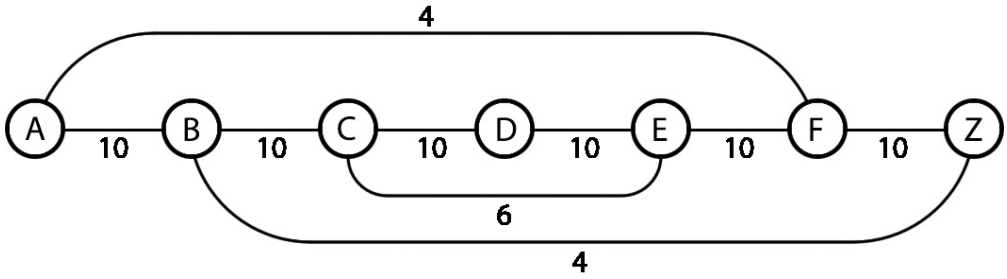
$(p^*,q^*) \leftarrow (\emptyset,\emptyset)$ ▷ No solution
Application of the "variant" Dijkstra's algorithm, calculating path p
if p exists **then**
    Network is transformed in G', so that every (edge) directed arc $a_{ij} \in p$ is replaced by an arc a_{ji} in the t-to-s direction with altered weights:

$$\begin{cases} c'_{ji} = -c_{ij} & \text{, for min cost path pair} \\ b'_{ji} = b(p)+1 & \text{, for widest path pair} \end{cases} \tag{4.3}$$

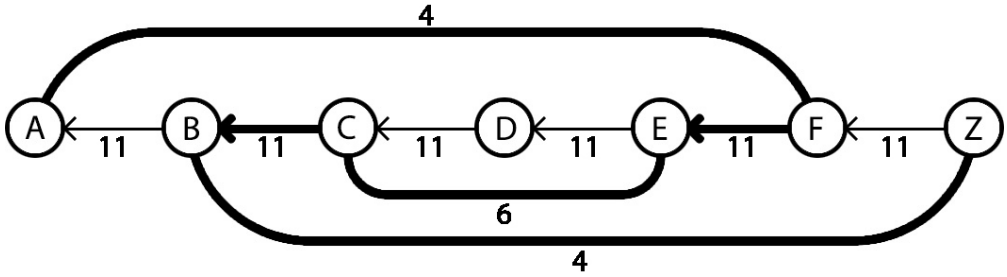
    if maximally disjoint pairs are admissible **then**
        Keep p and change the cost/bandwidth of $a_{ij} \in p$ according to Equation (2.16).
    end if
    if "variant" = widest **then**
        Application of the widest path Dijkstra's algorithm, calculating path p'
    else
        Application of the modified Dijkstra's algorithm, calculating path p'
    end if
    Restores G ▷ Undoes all changes required to obtain the transformed network (G').
    if $p' \neq \emptyset$ **then**
        $(p^*,q^*) \leftarrow$ path pair resulting from the deinterlacing of p and p'.
    end if
end if
return (p^*,q^*) ▷ Either an optimal solution or $(\emptyset,\emptyset)$

In [3, pages 65 and 88] and [14, page 10], it is said that the arcs that are not removed during the deinterlacing process are part of the path pair solution. While this is true when the metric is additive, it is not always true when the criterion is based on a concave metric. Next there is an example that proves that, when calculating an widest edge-disjoint path pair, some arcs may not be removed and yet not be part of the solution. It is important to note that a widest edge-disjoint path pair is still found and the only thing being pointed here is the curious fact that not all arcs are part of the solution.

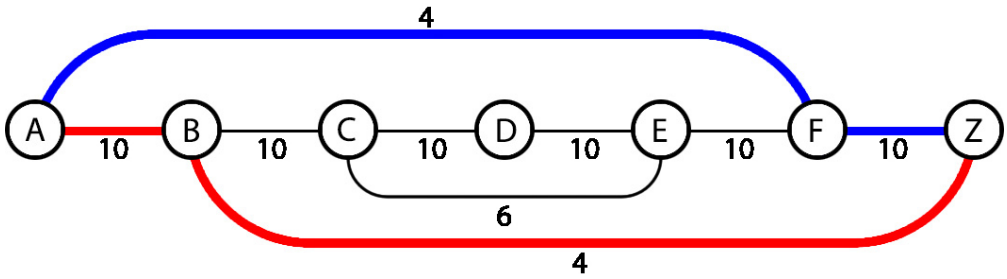
Figure 4.3 depicts the calculation of a widest edge-disjoint path pair that does not contain all non-reversed arcs from the two paths obtained with the Dijkstra's and the modified Dijkstra's algorithms. The widest path p in Figure 4.3(a) is $A-B-C-D-E-F-Z$. After modifying G into G', the widest path is $A-F-E-C-B-Z$ (Figure 4.3(b)). After deinterlacing, the resulting path pair (p^*,q^*) is $A-F-Z$ and $A-B-Z$ (see Figure 4.3(c)), of which the non-reversed edge between E and C is not a part of.



(a) Example graph G.



(b) Modified graph G' and widest path p'.



(c) Deinterlacing and resulting path pair.

Figure 4.3: Example of widest edge-disjoint path pair that does not contain all non-reversed arcs from p'.

Note that this can only happen if b_{EC} does not limit $b(p')$, so ignoring this arc does not affect the bandwidth of path p' or the resulting path pair.

Chapter 5

Proposed Heuristics

Here are described the proposed and implemented heuristic algorithms. An heuristic algorithm based on Bhandari's is presented for addressing the WEDLO problem, formalized in Section 3.1. This heuristic algorithm will make use of the Dual Path Label Dijkstra's heuristic (DPLD). It will also serve as a mold for two similar path pair heuristics that are used to solve the problems in Section 3.3 and Section 3.2, by using the Dual Path Label Dijkstra's heuristic with Minimum Limits (DPLD-ML) and the Dual Path Label Dijkstra's heuristic for Maximum Sum (DPLD-MS), respectively.

The proposed heuristics are exposed in this order, which differs from the order the corresponding problems were presented, because each of these heuristics is based on the previous one.

The relative performances of the proposed heuristics will be presented in Chapter 6.

5.1 Heuristic for Widest Edge-disjoint Path Pair Lexicographic Optimization

In this section is presented an heuristic for the Widest Edge-disjoint Path Pair Lexicographic Optimization problem [5]. As such, it is denominated HLO.

The main idea behind the resolution approach to the problem in Section 3.1 is the use of dual bandwidth labels at the nodes: each node will have primary and secondary node labels[1].

The heuristic algorithm first calculates the widest path p from node s to node t. Then it changes the network into G' in a way similar to the procedure in [3]: the directed arcs in the path from s to t are removed and the reversed (symmetrical) arcs are altered to have infinite bandwidth (in practical terms, a value larger than the bandwidth of the widest edge in the network).

To obtain the path pair that solves Equation (2.12), all that is required is to calculate p', the widest path in the transformed network G'. The interlacing arcs, i.e., the reversed arcs of p which appear in p', are removed and the remaining arcs define the widest edge-disjoint path pair (p^*, q^*).

The heuristic used to obtain the path pair is described in pseudocode in Algorithm 5. It makes use of the Dual Path Label Dijkstra's heuristic (DPLD) to calculate p' in Line 5. DPLD will be explained and detailed in pseudocode (Line 5) in the next section.

[1] This approach was inspired in [11], where the authors employed dual labels to finding link-disjoint paths for $\alpha + 1$ path protection.

Algorithm 5 Heuristic for Widest Edge-disjoint Path Pair Lexicographic Optimization (HLO): heuristic for the problem described in Section 3.1

Require: $G = (N,A)$, B matrix with arcs' bandwidth $(b_{ij},(i,j) \in A)$, nodes source s and target t.
Ensure: Returns a path pair (p^*,q^*) to problem in Equation (3.2), (possibly sub-optimal) or $(\emptyset,\emptyset)$ if no solution was found.

1: $(p^*,q^*) \leftarrow (\emptyset,\emptyset)$ ▷ No solution
2: Application of the widest path Dijkstra's algorithm, calculating path p, such that $p = \arg\max_{p^+ \in P_{st}} b(p^+)$
3: **if** p exists **then**
4: Network is transformed in G' as described in Section 4.3.
5: $p' \leftarrow DPLD(G',B',s,t)$ ▷ Algorithm 6
6: Restores G ▷ Undoes all changes required to obtain the transformed network (G').
7: **if** $p' \neq \emptyset$ **then**
8: $(p^*,q^*) \leftarrow$ path pair resulting from the removal of the interlacing edges of p and p'.
9: **end if**
10: **end if**
11: **return** (p^*,q^*) ▷ Solution (possibly sub-optimal) to Equation (3.2) or $(\emptyset,\emptyset)$

5.1.1 The Dual Path Label Dijkstra's Heuristic

When the calculation of the widest path in the transformed network G' begins, the source node has its labels equal to the bandwidth of the path p. In the modified graph, the widest path is determined using the primary node labels. If interlacing arcs exist in p', it is known that the interlacing arcs will be removed and the resulting sub-paths between interlacing chains will belong alternately to the resulting path pair p and q.

During the calculation of p', whenever the non-permanently labeled node presently with the largest bandwidth label (in G') is selected, let it be node v_k, then node v_k becomes a permanently labeled node. Let the predecessor of v_k be v_{k-1}. The secondary label of v_k takes the value of the secondary label of its predecessor, v_{k-1}.

Let the new permanently labeled node be v_{k_1}, such that the arc (v_{k_1-1},v_{k_1}) belongs to the reversed p path (see Figure 5.1). If the predecessor of v_{k_1-1}, let it be v_{k_1-2}, is such that (v_{k_1-2},v_{k_1-1}) does not belong to the reversed p path, then the primary and secondary labels of node v_{k_1} swap, because arc (v_{k_1-1},v_{k_1}) is the first reversed arc permanently added to the tree of widest paths and is a candidate to be part of p'.

If this is the first time a reversed arc has appeared as possible candidate to be on p', then the sub-path of p' from s to v_{k_1-1}, $p'_{sv_{k_1-1}}$, is the candidate sub-path of one of the paths of the final path pair (say p^*) and the sub-path of p from s to the exit node (v_{x_1}) of the chain of reversed arcs (in p') starting in node v_{k_1-1}, $p_{sv_{x_1}}$, will belong to the other path of the path pair (say q^*). Note that v_{x_1} may coincide with v_{k_1} if the chain of reversed arcs is made of a single arc.

The label swapping ensures that, from this point onwards, the bandwidth of the sub-path starting in v_{k_1} will be calculated independently from the sub-path that ended in the predecessor of v_{k_1}. If v_{x_1} is the tail node of the sub-path of q^*, its bandwidth will initially be the bandwidth of p,

because the first sub-path of q^* will be $p_{sv_{x_1}}$ and $b(p_{sv_{x_1}}) \geq b(p)$. Hence the first label swapping ensures v_{x_1} has now as primary label $b(p)$. The bandwidth of q^* will be at most $b(p)$, regardless of the bandwidth of the sub-path $p_{sv_{x_1}}$. Hence, this is the correct label for calculating q^*'s bandwidth if the chain from v_{k_1-1} to v_{x_1-1} does in fact belong to p' – which implies $q^*_{sv_{x_1}} = p_{sv_{x_1}}$. If the chain from v_{k_1-1} to v_{x_1} is not part of p', then the fact that we have modified the label of node v_{k_1} in that chain is irrelevant for the calculation of p'. As the tree of widest paths calculation progresses, the sub-paths with tail node v_{x_1} and v_{k_1-1} will be calculated as widest as possible, taking into account the bandwidth of $p_{sv_{x_1}}$ and $p'_{sv_{k_1-1}}$.

If, after exiting the first chain of reversed arcs (from v_{k_1-1} to v_{x_1}), a new reversed arc appears in p', that is, if the selected node v_{k_2} (and arc (v_{k_2}, v_{k_2-1})) is permanently added to the tree of widest paths and it is the first reversed arc of the second chain of reversed arcs, the primary and secondary labels of v_{k_2} swap. Let v_{x_2} be the exit node of the chain of reversed arcs (in p') starting in node v_{k_2-1}. The candidate path p^* will be $p'_{sv_{k_1-1}} \diamond p_{v_{k_1-1}v_{x_2}}$, and the candidate sub-path of q^* path will be $p_{sv_{x_1}} \diamond p'_{v_{x_1}v_{k_2-1}}$. Due to the label swapping, the calculation of the widest sub-paths with tail nodes v_{x_2} and v_{k_2-1} will be made taking into account the bandwidth of $p'_{sv_{k_1-1}} \diamond p_{v_{k_1-1}v_{x_2}}$ and of $p_{sv_{x_1}} \diamond p'_{v_{x_1}v_{k_2-1}}$, respectively.

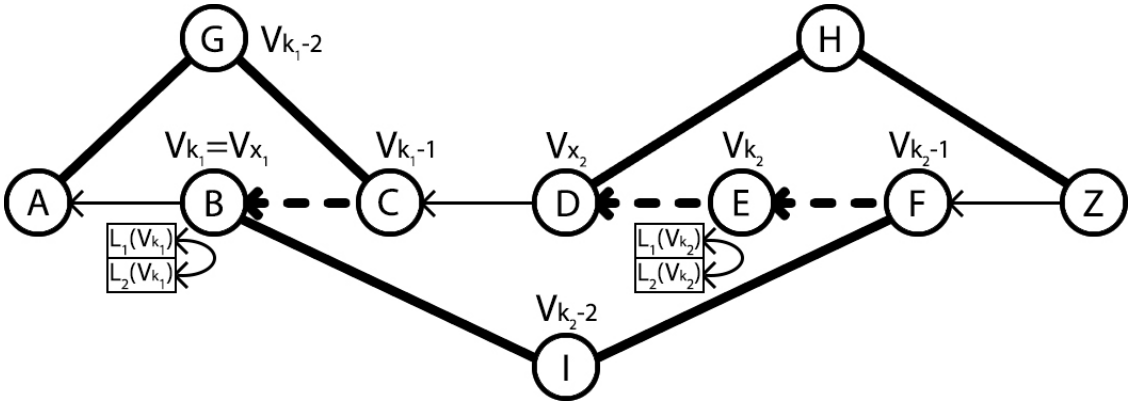


Figure 5.1: Illustration of the label swapping procedure.

The procedure that has been described and illustrated in Figure 5.1 can be generalized. Let the first reversed arc of the i-th ($i > 2$) chain of reversed arcs in p' be (v_{k_i-1}, v_{k_i}). This arc is permanently added to the tree of widest paths (in construction) when v_{k_i} is permanently labeled. Because v_{k_i} is the head of the first reversed arc of the i-th chain of reversed arcs in p', the primary and secondary labels of v_{k_i} swap. Let v_{x_i} be the exit node of the i-th chain of reversed arcs in p' starting in node v_{k_i-1}. If i is even, the candidate sub-path of p^* will be:

$$p'_{sv_{k_1-1}} \diamond p_{v_{k_1-1}v_{x_2}} \diamond \cdots \diamond p_{v_{k_{i-1}-1}v_{x_i}} \tag{5.1}$$

and candidate sub-path of q^* will be:

$$p_{sv_{x_1}} \diamond p'_{v_{x_1}v_{k_2-1}} \diamond \cdots \diamond p'_{v_{x_{i-1}}v_{k_i-1}} \tag{5.2}$$

If i is odd, the candidate sub-path of p^* will be:

$$p'_{sv_{k_1-1}} \diamond p_{v_{k_1-1}v_{x_2}} \diamond \cdots \diamond p'_{v_{x_{i-1}}v_{k_i-1}} \tag{5.3}$$

and the candidate sub-path of q^* will be:

$$p_{sv_{x_1}} \diamond p'_{v_{x_1}v_{k_2-1}} \diamond \cdots \diamond p_{v_{k_{i-1}-1}v_{x_i}} \tag{5.4}$$

Due to the label swap, the calculation of the widest sub-paths with tails v_{x_i} and v_{k_i-1} will be made taking into account the bandwidth of the candidate paths which end in v_{x_i} and v_{k_i-1} (according to Equations (5.1) to (5.4)), thus ensuring that when node t is reached, the primary and secondary labels of t will contain the bandwidth of p^* and q^*.

If the number of interlacing chains is even (odd), the *primary label* of t will be equal to $b(p^*)$ ($b(q^*)$) and the *secondary label* of t will be equal to $b(q^*)$ ($b(p^*)$). If no interlacing between p and p' takes place, the primary label will be the bandwidth of p' and the secondary label of t will be the bandwidth of p. In this resolution approach, the paths p^* and q^* result from the union of the arcs in p and p', discarding every arc whose reversal appears on the other.

The Dual Path Label Dijkstra's heuristic, which is similar to Dijkstra's algorithm, will be described next in pseudocode and requires the notation used in Section 4.1 plus the following:

- $L_1(j)$ Label 1 of node j – primary label.
- $L_2(j)$ Label 2 of node j – secondary label.

This heuristic algorithm (DPLD) has the same complexity as the implemented version of Dijkstra's algorithm (see the end of Section 4.1). Next is an illustrated example of how it works.

5.1.2 Illustrative Example of the Dual Path Label Dijkstra's Heuristic

Consider the example network represented by the graph in Figure 5.2(a). The first step is to calculate the widest path p between the source node A and the target node Z, which is path $A-B-C-Z$ (thicker lines in Figure 5.2(a)) with bandwidth $b(p) = 100$. The next step is to transform the network and assign labels to each node, as pictured in Figure 5.2(b). The nodes belonging to the widest path, $A-B-C-Z$, will require two labels in order to perform label swapping, while the rest of the nodes use the second label just to carry its value across the tree. In the figures, the top label represents the primary label, L_1, and bottom label represents the secondary label, L_2.

Also, since L_2 will be passed on from node to node as the tree is calculated, the only labels that need to be initialized are the ones related to the source node, A. $L_1(A)$ and $L_2(A)$ receive the bandwidth value of the previously computed widest path, $b(p) = 100$. The remaining nodes start with both their labels set as 0.

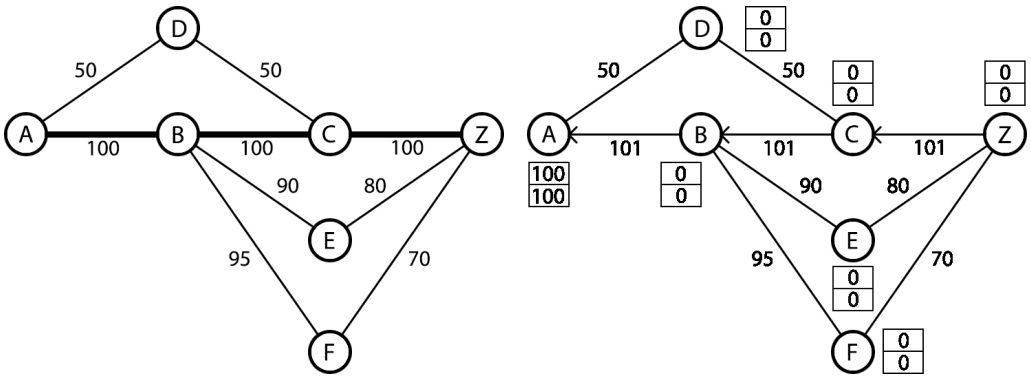
Algorithm 6 Dual Path Label Dijkstra's Heuristic (DPLD)

Require: $G' = (N, A')$, modified graph as described in step 4 of Algorithm 5, B' matrix with arcs' bandwidth $(b_{ij}, (i,j) \in A')$, nodes source s and target t.

Ensure: Calculates p' (if it exits), the widest path in G', ensuring that p' and p contain the arcs solving Equation (3.2)

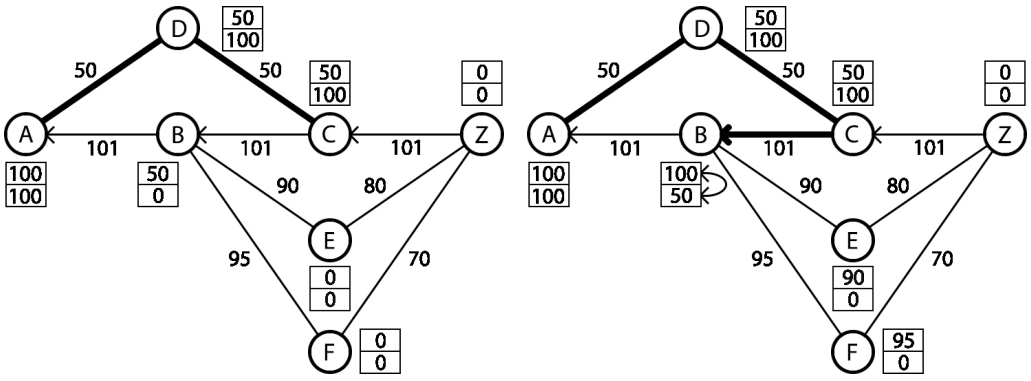
1: **for** all $i \in N$ **do**
2: $\psi(i) \leftarrow s$ ▷ s is the predecessor of all nodes
3: $L_1(i) \leftarrow 0, L_2(i) \leftarrow 0$ ▷ No path has been found from s to i
4: **end for**
5: $L_1(s) \leftarrow b(p), L_2(s) \leftarrow b(p)$ ▷ Labels are initialized with the bandwidth of the widest path
6: $k \leftarrow s$ ▷ First node is s
7: $S \leftarrow N - \{s\}$ ▷ s is permanently labeled
8: **repeat**
9: **for** every arc $(k,u), u \in S$ **do** ▷ For every non-permanent neighbor node of k
10: **if** $L_1(u) < \min[L_1(k), b_{ku}]$ **then** ▷ If $L_1(u)$ can be improved
11: $\psi(u) \leftarrow k$ ▷ k becomes the predecessor of u
12: $L_1(u) \leftarrow \min[L_1(k), b_{ku}]$ ▷ $L_1(u)$ is updated
13: **end if**
14: **end for**
15: $k \leftarrow \arg\max_{j \in S} L_1(j)$ ▷ The non-permanent node with highest L_1 is chosen
16: $S \leftarrow S - \{k\}$ ▷ Permanently labeled node
17: $L_2(k) \leftarrow L_2(\psi(k))$ ▷ L_2 is passed on from the predecessor of k
18: $l \leftarrow \psi(k)$ ▷ Auxiliary variable
19: **if** $b_{\psi(k)k} = \infty \wedge b_{\psi(l),l} \neq \infty$ **then** ▷ If $(\psi(k), k)$ is the first reversed arc of a chain
20: $L_1(k)$ and $L_2(k)$ swap values.
21: **end if**
22: **until** $k = t$ ▷ Terminates
23: **if** $L_1(t) = 0$ **then**
24: $p' \leftarrow \emptyset$ ▷ No path from s to t
25: **else**
26: p' is described by the successive predecessors of t.
27: **end if**
28: **return** p'.

To compute the widest path tree in the transformed network G', the Dual Path Label Dijkstra's heuristic is used. Similarly to the widest path Dijkstra's algorithm, the first step is to label A's neighbor nodes and pick the one with largest bandwidth to be part of the tree. Seeing as this is a small network and there are not many options, node C is easily reached (see Figure 5.3(a)). The value of $L_2(A)$ is passed on from A to D and then to C as these nodes become permanent. C's non-permanent neighbor, B, gets labeled with the current path bandwidth, $L_1(B) \leftarrow L_1(C) = 50$. As B becomes part of the tree, $L_2(B)$ takes the value of $L_2(\psi(B) = C)$. Since the arc to B is the first (and in this case only) in a chain of arcs that were the result of edge reversal, the labels are swapped (see Figure 5.3(b)). If there was a node Y between C and B, then the primary and secondary labels of Y would swap once the node became permanent, but there would be no swapping when B was made permanent. Meaning that, regardless of the number of reversed edges that consecutively



(a) Widest path. (b) Network transformation.

Figure 5.2: Initial steps of the DPLD algorithm.



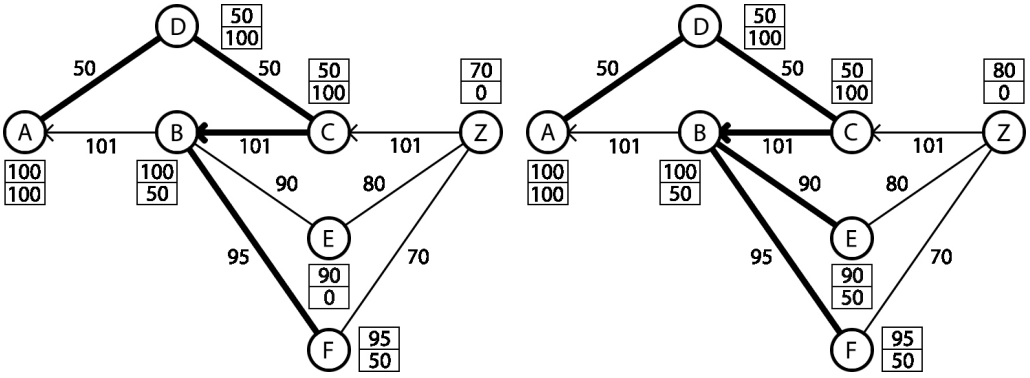
(a) After node C is permanently labeled (b) Labels swap

Figure 5.3: The Dual Path Label Dijkstra's heuristic first iterations.

become part of the tree, the labels will swap only once per chain of reversed edges. It is also worth noting that since the reversed edges have bandwidth equal to the practical equivalent of infinity, the only difference between the labels at the start and at the end of the chain is that they were swapped.

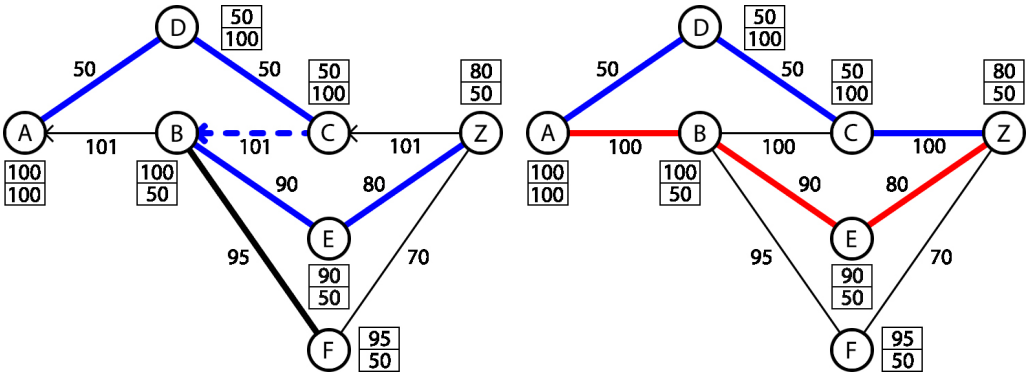
As the labels swapped, the current path bandwidth is now $L_1(B) = 100$, which means that the next choice between E and F is not irrelevant, as would happen in the case of the widest path Dijkstra's algorithm. Instead, $L_1(E) < L_1(F)$, so F is the preferred node and, becoming part of the tree, labels the target node, Z, with $L_1(Z)$ equal $\min[L_1(F), b_{FZ}] = 70$ (see Figure 5.4(a)). Also, $L_2(F)$ takes the value of $L_2(B) = 50$. However, now the best candidate is E with $L_1(E) > L_1(Z)$. In Figure 5.4(b), E becomes part of the tree and labels the target node Z with a larger value, $L_1(Z) \leftarrow 80$, because $\min[L_1(E), b_{EZ}] > L_1(Z)$. E also gets the secondary label from B, $L_2(E) \leftarrow L_2(B) = 50$.

The target node is reached in Figure 5.5(a) and $L_2(Z) \leftarrow L_2(E) = 50$. In thick lines, we have the widest paths tree; in blue, the widest path on the modified network; and in a dashed line, the edge that will not be part of the widest path pair. In Figure 5.5(b), the interlacing edge is removed and the path pair calculation is finished. Note that $L_1(Z)$ has the bandwidth of the red path, which was the current path when the Dual Path Label Dijkstra's heuristic ended, and $L_2(Z)$ has the bandwidth



(a) Node F is permanently labeled (b) Node E is permanently labeled

Figure 5.4: Impact of the dual labels.



(a) Path (and tree) completion. (b) Path Pair computation.

Figure 5.5: Final steps.

of the blue path. The (C,Z) edge does not affect the veracity of the last affirmation because it was part of the widest path in the original network and no path in the pair can have bandwidth larger than that. It should also be noted that although (p^*,q^*) is used to represent the solution returned by HLO for problem in Section 3.1, this may not be the optimal path pair. This notation is used to recall that (p^*,q^*) is the desired target of the heuristic.

5.1.3 Important Remarks on the Heuristic for Widest Edge-disjoint Path Pair Lexicographic Optimization

Similarly to the case discussed in Figure 4.3, some edges that are not removed during deinterlacing may not be part of the solution, though for this heuristic those cases need to be more elaborate due to the use of the dual labels. The use of labels would prevent the arc (E,C) from being part of p' in the example of Figure 4.3, thus making every non-reversed arc of p' be part of the solution. However, it might still happen in more ellaborate cases. To ensure it does not happen at all, in the case of a tie the reversed edges in G' should be preferred to be part of p'. This led to the creation of two versions of this heuristic: HLO_f and HLO_l, where, in case of a tie, the unlabeled

nodes adding a reversed arc to the tree of widest paths under construction (in G') are ranked first and last, respectively. In other words, should more than one candidate node have maximum L_1, HLO_f gives priority to a node if the arc that leads to it is reversed and HLO_l gives priority if it's not reversed. This applies not just to choosing node k, but also to labeling. In case of a tie in the other conditions, a node can be relabeled if the arc that leads to it is reversed and the previous label was associated with a non-reversed arc. The existence of these two versions will be justified during the performance analysis, but for now it should be said that the dual labels calculated by HLO_f meet the following condition with perfect accuracy:

$$\begin{cases} L_1(t) = b(p^*) \\ L_2(t) = b(q^*) \end{cases} \quad \text{or} \quad \begin{cases} L_1(t) = b(q^*) \\ L_2(t) = b(p^*) \end{cases} \tag{5.5}$$

On the other hand, due to the exclusion of arcs from p', HLO_l may overestimate $b_M(p^*, q^*)$, i.e., the maximum bandwidth of the pair found in the bandwidth labels of the target node may be larger than the actual bandwidth of the path. This happens in cases where not all non-reversed arcs in p and p' are part of the path pair solution (p^*, q^*). The ignored arc forces an extra label swapping without actually affecting the bandwidth of the path pair. These cases are rare, but should not be ignored as they require the recalculation of the bandwidths of the obtained paths. Nevertheless, as it will be seen ahead, this is no reason to discard this version of the heuristic.

5.2 Heuristic for Edge-disjoint Path Pairs with Minimum Limits

The Widest Pair of Disjoint Paths Decoupled problem (in Section 3.3) is approached in this section, where an heuristic to solve it, called Heuristic for Edge-disjoint Path Pairs with Minimum Limits (HML), is presented.

The use of dual path labels has room to be explored in several ways. The innovative and very exploitable part of it is the fact that, at a given node, the maximum bandwidths that the path pair can have, if that node is part of the solution, are known. This can be used to solve the problem formalized in Section 3.3, where a path pair (any path pair) that has bandwidths larger than or equal to the limits X_1 and X_2, with $X_1 > X_2$, is sought.

After successfully obtaining a path p' by running DPLD, if the target node's dual path labels verify the following conditions:

$$\begin{cases} \max[L_1(j), L_2(j)] \geq X_1 \\ \min[L_1(j), L_2(j)] \geq X_2 \end{cases}, \text{ with } j \in N \tag{5.6}$$

Then, the resulting path pair (p^*, q^*) also meets the limits X_1 and X_2. Additionally, if Equation (5.6) is true for the target node, then it is also true for every other node in path p'. So, by

including these conditions in the path calculation process, it is possible to obtain a path p' that leads to a feasible path pair when deinterlaced with the widest path p. The heuristic that obtains said path p' is denominated Dual Path Label Dijkstra's heuristic with Minimum Limits.

As $X_1 > X_2$, the minimum label should be compared with X_2 and the maximum label with X_1. A path pair that includes a node that fails to meet any of these conditions will also not respect the limits X_1 and X_2 and is not a feasible solution.

Let Y be a node whose neighbors are being scanned for labeling and let W be one of those nodes. W is labeled only if the condition

$$\min[L_1(Y), b_{YW}] > L_1(W) \tag{5.7}$$

and the conditions set in Equation (5.6) are verified.

It was explained in the last section that if the widest path from s to t in the modified graph G', p', is calculated using HLO_f and contains at least one chain of reversed arcs, then these arcs aren't part of the final path pair and the remaining arcs are divided between p^* and q^*, with at least one arc to each one. Bearing in mind that reversed arcs have bandwidth equal to the practical equivalent of infinity, it can be said that if there is one arc in p' that has bandwidth lower than X_2, then path pair (p^*, q^*) won't pass the condition in Line 10, namely the X_2 one. It can also be said that if p' has two arcs with bandwidths lower than X_1 with an odd number of reversed arcs chains (that precede and are preceded by non-reversed arcs) between them, (p^*, q^*) fails the X_1 condition in Line 10. Knowing this, if an arc leads to a node with bandwidth labels that do not meet the criteria, then a path pair that includes it cannot satisfy Line 10. Any other arcs can be part of the path pair solution and, if the target node is marked as permanent, then the resulting path pair is feasible.

A feasible path pair (p^*, q^*) is obtained by running HML, which is very similar to HLO (Algorithm 5), except for the fact that it calls DPLD-ML instead of DPLD in Line 5 and passes the two limits, X_1 and X_2, as additional parameters.

Because this heuristic relies heavily on the bandwidth labels, the condition to label nodes (Line 10 in Algorithm 7) and choose k (Line 19) is the same used in HLO_f (in case of a tie, priority is given to a node if the arc that leads to it is reversed).

The heuristic for calculating p' is described in the pseudocode of Algorithm 7.

Algorithm 7 follows the same line of thought as Algorithm 6, with the exception of the added Line 11 and Line 12 and the need for two more parameters. However, just like in the adaptations of Dijkstra's algorithm, these small changes make a great difference in the result. While Algorithm 6 was guaranteed to maximize $b_m(p^*, q^*)$ like Bhandari's algorithm did, Algorithm 7 may sacrifice that maximization in order to meet the criteria set on Line 10.

Next is an illustrated example of this heuristic that also exemplifies what was discussed in the

Algorithm 7 Dual Path Label Dijkstra's Heuristic with Minimum Limits (DPLD-ML)

Require: $G' = (N, A')$, modified graph as described in step 4 of Algorithm 5, B' matrix with arcs' bandwidth ($b_{ij}, (i,j) \in A'$), nodes source s and target t, bandwidth guaranteed limits (X_1, X_2).
Ensure: Calculates p' (if it exits) ensuring that the deinterlacing of p' and p results in a path pair (p^*, q^*) that is feasible according to Line 10

1: **for** all $i \in N$ **do**
2: $\psi(i) \leftarrow s$ ▷ s is the predecessor of all nodes
3: $L_1(i) \leftarrow 0$, $L_2(i) \leftarrow 0$ ▷ No path has been found from s to i
4: **end for**
5: $L_1(s) \leftarrow b(p)$, $L_2(s) \leftarrow b(p)$ ▷ Labels are initialized with the bandwidth of the widest path
6: $k \leftarrow s$ ▷ First node is s
7: $S \leftarrow N - \{s\}$ ▷ s is permanently labeled
8: **repeat**
9: **for** every arc $(k,u), u \in S$ **do** ▷ For every non-permanent neighbor node of k
10: **if** $L_1(u) < \min[L_1(k), b_{ku}] \vee (L_1(u) = \min[L_1(k), b_{ku}] \wedge b_{ku} = \infty \wedge b_{\psi(u)u} \neq \infty)$ **then**
11: **if** $\max[\min[L_1(k), b_{ku})], L_2(k)] \geq X_1$ **then** ▷ If the limit X_1 is respected
12: **if** $\min[\min[L_1(k), b_{ku}], L_2(k)] \geq X_2$ **then** ▷ If the limit X_2 is respected
13: $\psi(u) \leftarrow k$ ▷ k becomes the predecessor of u
14: $L_1(u) \leftarrow \min[L_1(k), b_{ku}]$ ▷ $L_1(u)$ is updated
15: **end if**
16: **end if**
17: **end if**
18: **end for**
19: $k \leftarrow \arg\max_{j \in S} L_1(j)$, with priority to the last arc being reversed ▷ Next k is chosen
20: $S \leftarrow S - \{k\}$ ▷ Permanently labeled node
21: $L_2(k) \leftarrow L_2(\psi(k))$ ▷ L_2 is passed on from the predecessor of k
22: $l \leftarrow \psi(k)$ ▷ Auxiliary variable
23: **if** $b_{\psi(k)k} = \infty \wedge b_{\psi(l),l} \neq \infty$ **then** ▷ If $(\psi(k), k)$ is the first reversed arc of a chain
24: $L_1(k)$ and $L_2(k)$ swap values.
25: **end if**
26: **until** $k = t$ ▷ Terminates
27: **if** $L_1(t) = 0$ **then**
28: $p' \leftarrow \emptyset$ ▷ No path from s to t
29: **else**
30: p' is described by the successive predecessors of t.
31: **end if**
32: **return** p'.

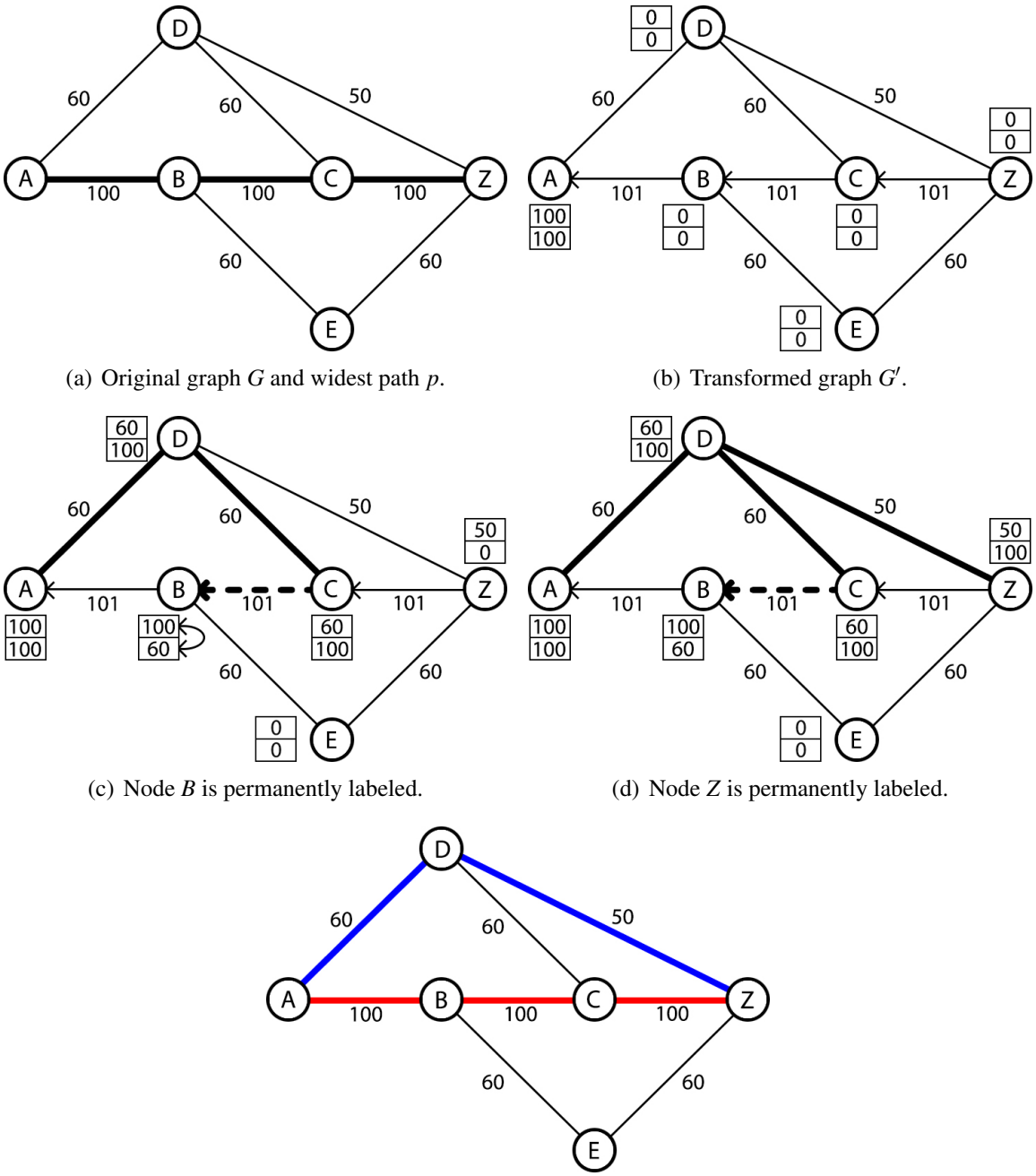
last paragraph. In this example,

$$\begin{cases} X_1 = 80 \\ X_2 = 40 \end{cases} \tag{5.8}$$

Figure 5.6(a) shows a network and the widest path from node A to Z and Figure 5.6(b) shows the transformed network G' with node labels initialized as they would be in DPLD (Figure 5.2(b)).

Nodes D, C and B are labeled like they were in the DPLD example and label swapping in node B also occurs. The difference surfaces in Figure 5.6(c), where through arc (B,E), node E would be

labeled with $L_1(E) \leftarrow \min[L_1(B), b_{BE}] = \min[100, 60] = 60$ (and $L_2(E) \leftarrow L_2(B) = 60$, if it were to become permanent). This fails the *if* condition in Line 11 of Algorithm 7 and, therefore, the node is not labeled. The only candidate is then Z, which is made permanent with labels $L_1(Z) = 50$ and $L_2(Z) = L_2(D) = 100$. Path p' has no interlacing arcs with p, so there's no need to deinterlace



(a) Original graph G and widest path p.

(b) Transformed graph G'.

(c) Node B is permanently labeled.

(d) Node Z is permanently labeled.

(e) Feasible path pair.

Figure 5.6: Illustrative example of HML.

them. The edge-disjoint path pair solution is $(p^*, q^*) = (p, p')$ (see Figure 5.6(e)), with

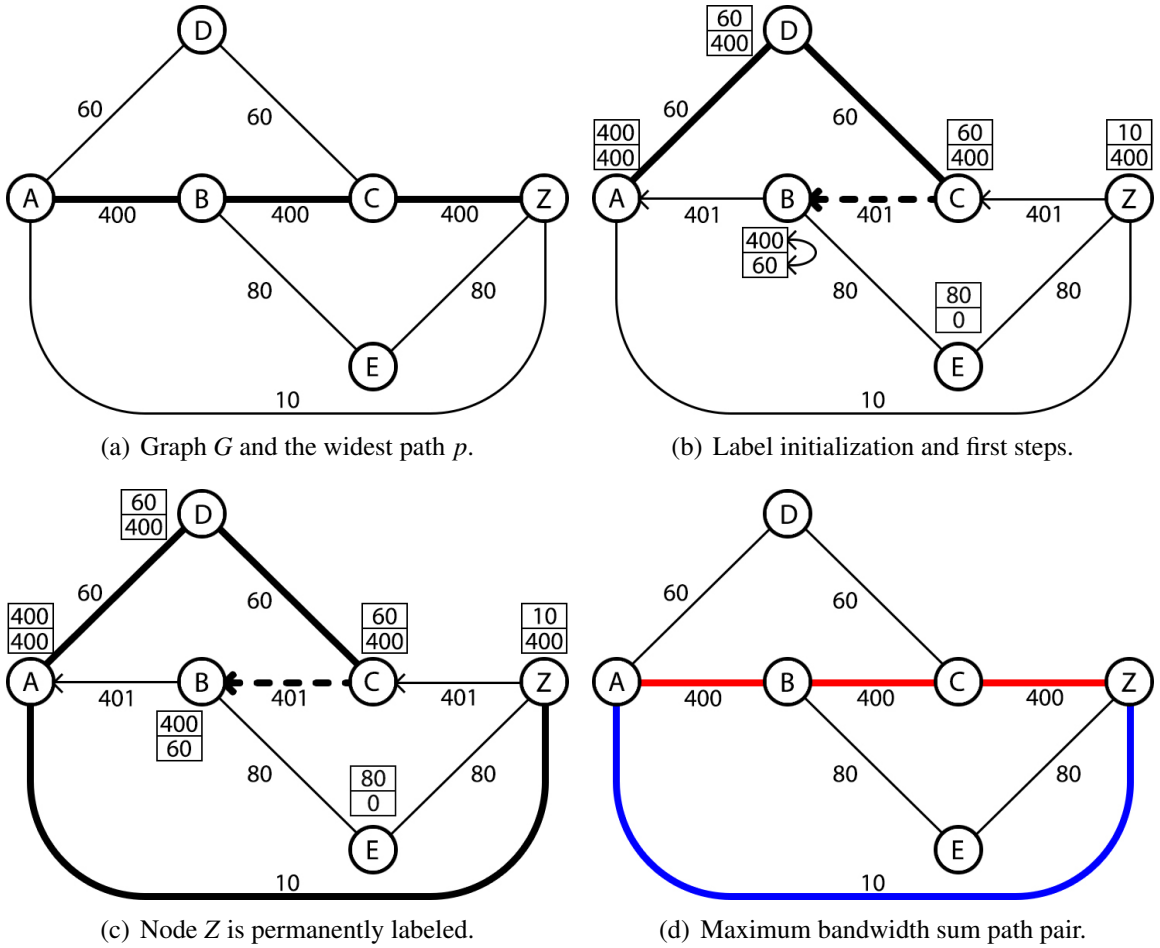
$$\begin{cases} b_M(p^*, q^*) = 100 > X_1 = 80 \\ b_m(p^*, q^*) = 50 > X_2 = 40 \end{cases} \tag{5.9}$$

5.3 Heuristic for Maximum Bandwidth Sum

The last proposed heuristic, Heuristic for Maximum Bandwidth Sum (HMS), addresses the problem formalized in Section 3.2.

In this heuristic, the optimal (or sub-optimal) pair (p^*, q^*) is obtained in the same manner as the desired path pairs in the previously described heuristics: through an heuristic in all equal to HLO (Algorithm 5), except for calling the Dual Path Label Dijkstra's Heuristic for Maximum Sum instead of DPLD in Line 5.

If, at each node, it is possible to know the maximum bandwidths each path will have in the final path pair, then the problem might be solved by choosing the nodes that have maximum label sum. In case of a tie, the decision to label or relabel (Line 10) a node or pick it as the next k (Line 15) is made like in HLO_f and HML, i.e., reversed arcs have priority over non-reversed arcs.



(a) Graph G and the widest path p.

(b) Label initialization and first steps.

(c) Node Z is permanently labeled.

(d) Maximum bandwidth sum path pair.

Figure 5.7: Illustrative example of the HMS heuristic.

Figure 5.7 illustrates an example of this heuristic. In Figure 5.7(a) the widest path is calculated in graph G. After the graph transformation, labels are initialized like in the previously described heuristics. In its first steps (Figure 5.7(b)), the heuristic labels nodes D, C and B as permanent, swapping the labels of the last one, and labels node Z with $L_1(Z) = 10$ and A as its predecessor.

After B becomes permanent, E is labeled with $L_1(E) = 80$. At this point, $L_1(E) > L_1(Z)$, but taking into account the secondary labels they will receive from their predecessors,

$$L_1(E) + L_2(B) = 140 < L_1(Z) + L_2(A) = 410 \tag{5.10}$$

The maximum sum path pair is then achieved without the need to deinterlace p and p'.

Algorithm 8 Dual Path Label Dijkstra's Heuristic for Maximum Sum (DPLD-MS)

Require: $G' = (N, A')$, modified graph as described in step 4 of Algorithm 5, B' matrix with arcs' bandwidth $(b_{ij}, (i,j) \in A')$, nodes source s and target t.

Ensure: Calculates p' (if it exits) ensuring that the deinterlacing of p' and p results in a path pair (p^*, q^*) that maximizes the sum of the bandwidths of the paths (possibly sub-optimal).

1: **for** all $i \in N$ **do**
2: $\psi(i) \leftarrow s$ ▷ s is the predecessor of all nodes
3: $L_1(i) \leftarrow 0, L_2(i) \leftarrow 0$ ▷ No path has been found from s to i
4: **end for**
5: $L_1(s) \leftarrow b(p), L_2(s) \leftarrow b(p)$ ▷ Labels are initialized with the bandwidth of the widest path
6: $k \leftarrow s$ ▷ First node is s
7: $S \leftarrow N - \{s\}$ ▷ s is permanently labeled
8: **repeat**
9: **for** every arc $(k,u), u \in S$ **do** ▷ For every non-permanent neighbor node of k
10: **if** $L_1(u) + L_2(\psi(u)) < L_1(k) + L_2(k) \vee (L_1(u) + L_2(\psi(u)) = L_1(k) + L_2(k) \wedge b_{ku} = \infty \wedge b_{\psi(u)u} \neq \infty)$ **then**
11: $\psi(u) \leftarrow k$ ▷ k becomes the predecessor of u
12: $L_1(u) \leftarrow \min[L_1(k), b_{ku}]$ ▷ $L_1(u)$ is updated
13: **end if**
14: **end for**
15: $k \leftarrow \arg\max_{j \in S}(L_1(j) + L_2(\psi(j)))$, with priority to the last arc being reversed ▷ Next k is chosen
16: $S \leftarrow S - \{k\}$ ▷ Permanently labeled node
17: $L_2(k) \leftarrow L_2(\psi(k))$ ▷ L_2 is passed on from the predecessor of k
18: $l \leftarrow \psi(k)$ ▷ Auxiliary variable
19: **if** $b_{\psi(k)k} = \infty \wedge b_{\psi(l),l} \neq \infty$ **then** ▷ If $(\psi(k), k)$ is the first reversed arc of a chain
20: $L_1(k)$ and $L_2(k)$ swap values.
21: **end if**
22: **until** $k = t$ ▷ Terminates
23: **if** $L_1(t) = 0$ **then**
24: $p' \leftarrow \emptyset$ ▷ No path from s to t
25: **else**
26: p' is described by the successive predecessors of t.
27: **end if**
28: **return** p'.

Chapter 6

Performance Analysis of the Proposed Heuristics

In this chapter, the performances of the three heuristics proposed in Chapter 5 are analyzed. These analysis are delivered in the same order the heuristics were proposed.

The performance of the heuristics are evaluated using fourteen networks from the SNDLib [16], where nodes of degree one were removed (and the corresponding networks have an '*' appended to their name in the figures). The deatils regarding the number of nodes, edges and the edge-to-node ratio (density of the network) is presented in Table B.1 in Appendix B. The bandwidth b_{ij} of each edge (i, j) was defined to be $1000/\log d(v_i, v_j)$, where $d(v_i, v_j)$ is the distance between nodes v_i and v_j, based on the GPS coordinates of v_i and v_j. These networks appear in ascending order of number of nodes. The heuristics are applied to every node pair in each of these networks. For comparison, the ILP formulations of the problems that were defined in Chapter 3 are used.

Every heuristic is compared with CPLEX 12.6 [9] in terms of average CPU time per node pair using a Desktop with an Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz processor and 6GB of RAM.

6.1 Heuristic for Widest Edge-disjoint Path Pair Lexicographic Optimization

In Figure 6.1, we present the average CPU time per node pair for solving HLO considering all node pairs for each network. The error bars represent the minimum and maximum value observed for each network. Note the different vertical axis for the heuristics and the solver. The CPU time in all cases grows with size of the networks (ordered in the figures by increasing number of nodes, from 10 to 64), but the CPU values for the heuristics are about 1% of the corresponding values obtained by CPLEX. Although the average value of HLO_l is slightly less than the corresponding value for the HLO_f, the maximum values of the former are not always less than the maximum values of the latter. Therefore, regarding CPU time, the two heuristics may be considered to have similar performance.

The percentage of node pairs where HLO (Algorithm 5) is able to find an optimal solution (verified using the solution obtained by CPLEX) is presented in Figure 6.2. It can be seen that HLO_l

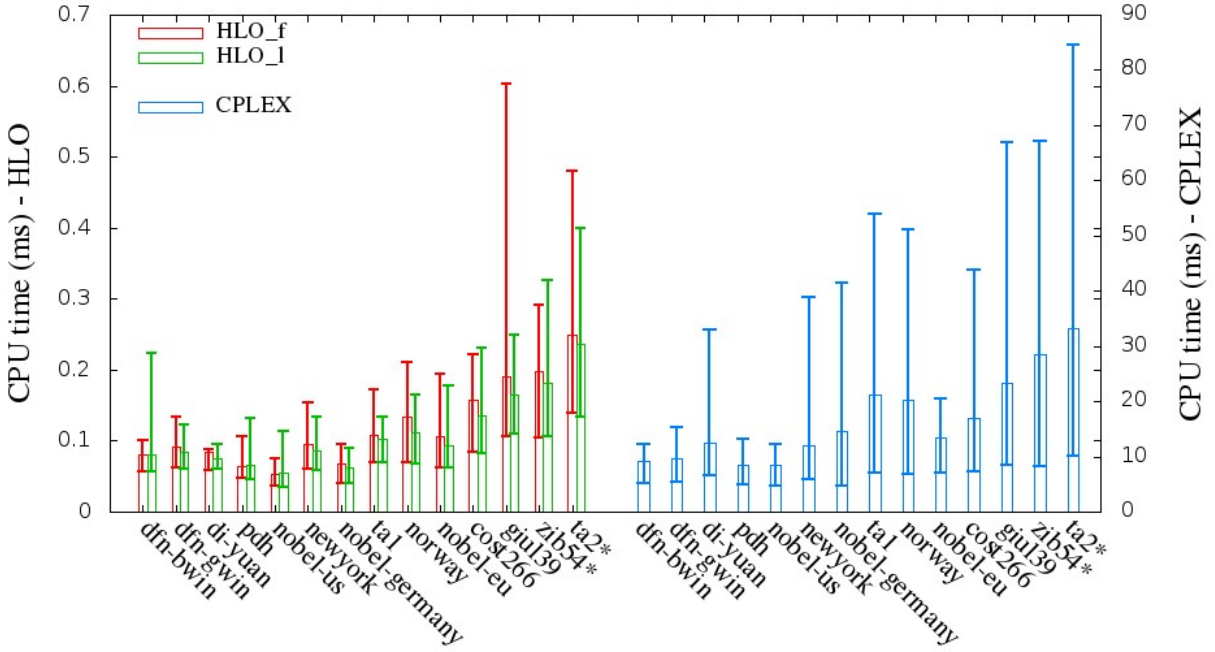


Figure 6.1: HLO's CPU time in milliseconds per node pair.

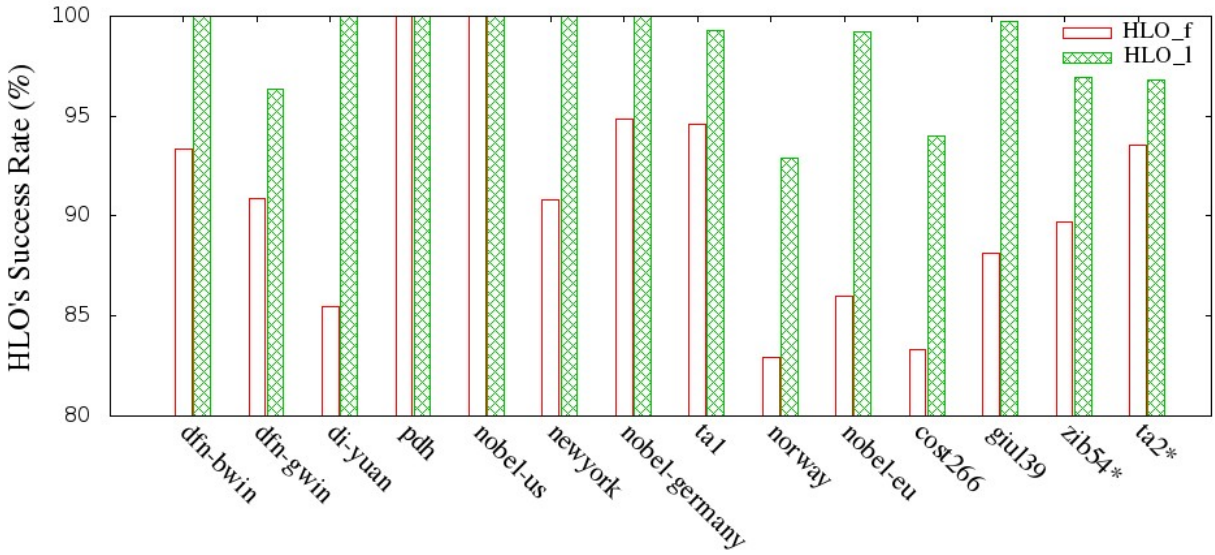


Figure 6.2: Percentage of optimal solutions found by HLO_f and HLO_l.

performs much better than HLO_f regarding the number of optimal solutions found. HLO_l found 100% of the optimal solutions for six of the fourteen tested networks. Out of the remaining eight networks, HLO_l obtained over 99% of the optimal solutions in three of them and at least 92% in the rest. HLO_f only managed to obtain 100% of the optimal solutions in two networks, has six networks with 90%-95% and the remaining six with 85%-90% optimal solutions.

Regarding the cases in which HLO_l returned labels that overestimated bandwidth, there were two out of all node pairs in all fourteen networks. In these two cases, the bandwidth $b_M(p,q)$ was overestimated 1.923% of the actual value.

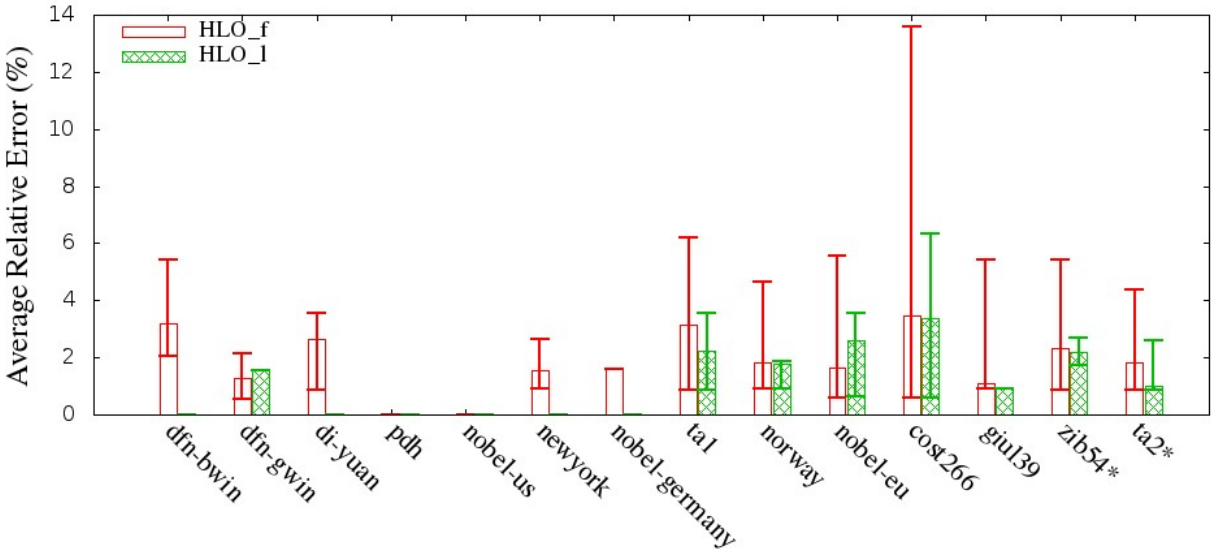


Figure 6.3: Average relative error of the solutions found by HLO_f and HLO_l.

Regarding the sub-optimal solutions of Algorithm 5, the average relative error of $b_M(p,q)$ using the y_1 value as reference is presented in Figure 6.3. Only the error in $b_M(p,q)$ is accounted for, because $b_m(p,q)$ is guaranteed to be maximized. The error bars represent the minimum and maximum relative error for all node pairs in each network (some columns have zero error and no error bars are shown, because all optimal solutions were found). In both cases, the relative error is, in average, less than 4% and, in the worst case, less than 7% and 14%, for HLO_l and HLO_f, respectively.

6.2 Heuristic for Edge-disjoint Path Pairs with Minimum Limits

The performance analysis of this heuristic rests on success rate and CPU time. Since this heuristic addresses a problem solved by a yes/no answer, it is not relevant to study the differences in the bandwidths of the solutions found by it and the ones found by CPLEX (when both satisfy the constraints).

To analyze the success rate, the heuristic was applied to the same fourteen networks with ten different combinations of the parameters X_1 and X_2 for each one. These parameters were calculated specifically for each network according to the following equations. Let a_m and a_M be, respectively, the narrowest and widest arcs in the studied network, and B_m be the maximum $b_m(p,q)$ among all path pairs between every pair of nodes (v,w) which maximize $b_m(p,q)$:

$$B_m = \max_{(v,w),\ v,w \in N} \max_{(p,q) \in \bar{P}_{vw}} b_m(p,q) \tag{6.1}$$

The constant that determines the number of (X_1, X_2) pairs is k. In this case, $k = 4$. A larger k means the intervals are shorter.

$$X_2 = b(a_m) + i \times \Delta_2, \quad i = 0, 1, \ldots, k-1 \tag{6.2}$$

$$X_1 = X_2 + j \times \Delta_1, \quad j = 1, \ldots, k-i \tag{6.3}$$

where Δ_1 and Δ_2 are

$$\Delta_1 = \frac{b(a_M) - b(a_m)}{k+1} \tag{6.4}$$

$$\Delta_2 = \frac{B_m - b(a_m)}{k} \tag{6.5}$$

This guarantees $X_1 > X_2$, while also providing values that are challenging to the heuristic. If X_1 and X_2 are too low, it is very easy to find paths, and if one of them is too high, no feasible path pairs exist and the test is futile. The heuristic results can be seen in Table B.2 in Appendix B.

In Table B.2, it is shown the heuristic's success rate, which is the percentage of times the heuristic found a feasible solution when feasible solutions existed. Feasible path pairs were assumed to be nonexistent when CPLEX failed to find one.

The effective success rate of HML should be calculated considering the frequency the heuristic and the CPLEX agree: both answer "yes" (a solution could be found) or both answer "no" (no solution exists). This, however, could favourably polarize the performance analysis of HML for more demanding values of (X_1, X_2), when there are very few "yes" answers. Hence, in the analysis, we considered the heuristic's success rate to be calculated as the percentage of times the heuristic and CPLEX agree when a solution exists.

To illustrate the percentage of solutions found by HML, Figure 6.4 shows the results, in order of ascending X_1 and X_2, for each network. The first thing of note is the range on the vertical axis. The worst case of all is a little over 94%. Excluding that and five others, all the results were over 96%, with the great majority being 100%. As mentioned before, the complete data regarding these rates can be found in Appendix B.

The main reason why this heuristic fails is the same reason why HLO fails. The widest path calculated in graph G, for a given pair of nodes, affects the widest path p' obtained in G', limiting the possibilities and sometimes keeping a solution from being found. Of course, if the network is denser and the arcs have bandwidth values that satisfy the minimum limits, then the probability of success increases.

There are two noticeable things in Table B.2: sometimes the success rate decreases when X_2 increases (see networks "newyork", "ta1", "norway", "nobel-eu", "cost266" and "ta2*") and sometimes it increases with X_1. The first happens for obvious reasons. If the limit is increased, it is harder to find a path pair and, given the fact that the heuristic's capacity to analyze every option is more limited than CPLEX's, it is understandable why it fails when the constraints are tighter. On the other hand, when X_1 is raised, the number of node pairs with feasible solutions goes down.

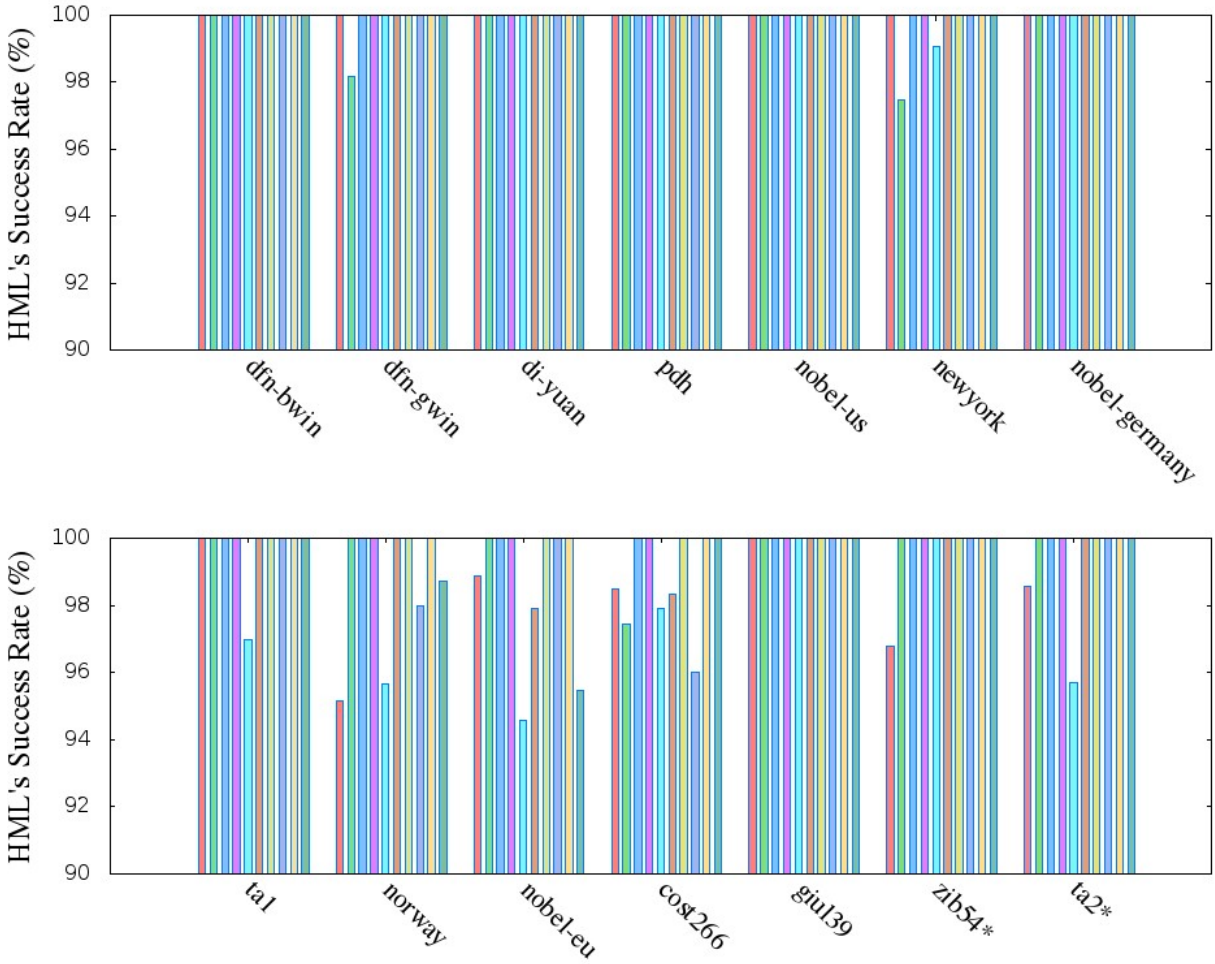


Figure 6.4: HML's Success Rate.

If some of the cases that the heuristic was unable to solve disappear, its success rate might grow.

Figure 6.5 displays in detail the values of the CPU times obtained during a run of HML and CPLEX on the fourteen tested networks. In those histograms, the bars represent average times, with the error bars being the minimum and maximum values. It is important to note the scale, but even ignoring it, it is possible to see that HML's average and minimum times are more constant and are not as affected by the different values of X_1 and X_2. On the other hand, for a constant X_2 on any network, when X_1 rises, CPLEX's CPU time is reduced. When X_2 increases and X_1 goes back to a lower value, CPLEX's average time goes up again.

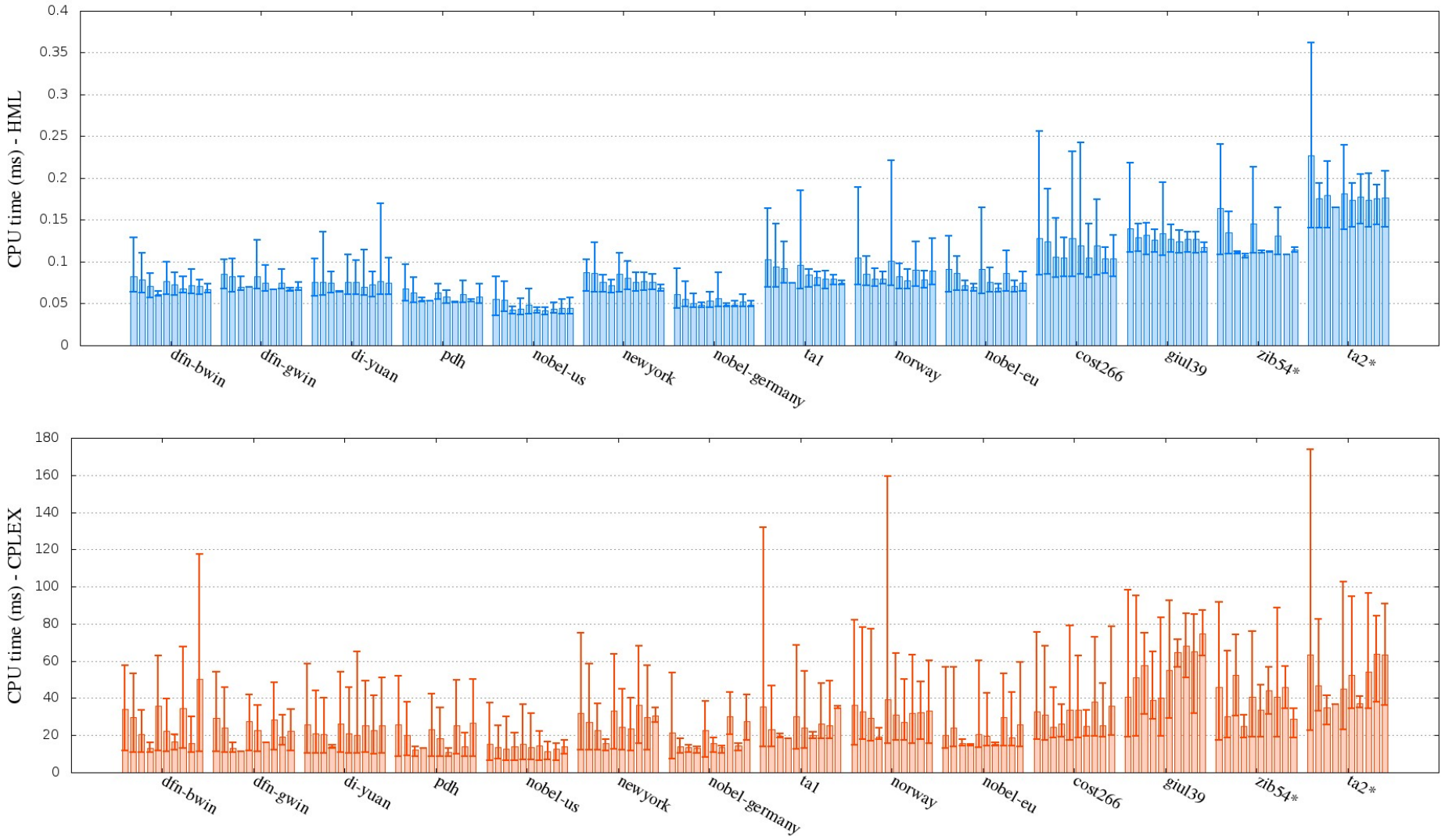


Figure 6.5: HML's and CPLEX's CPU time in milliseconds per node pair.

6.3 Heuristic for Maximum Bandwidth Sum

The Heuristic for Maximum Bandwidth Sum is evaluated in terms of CPU time, success rate and average relative error, similarly to HLO.

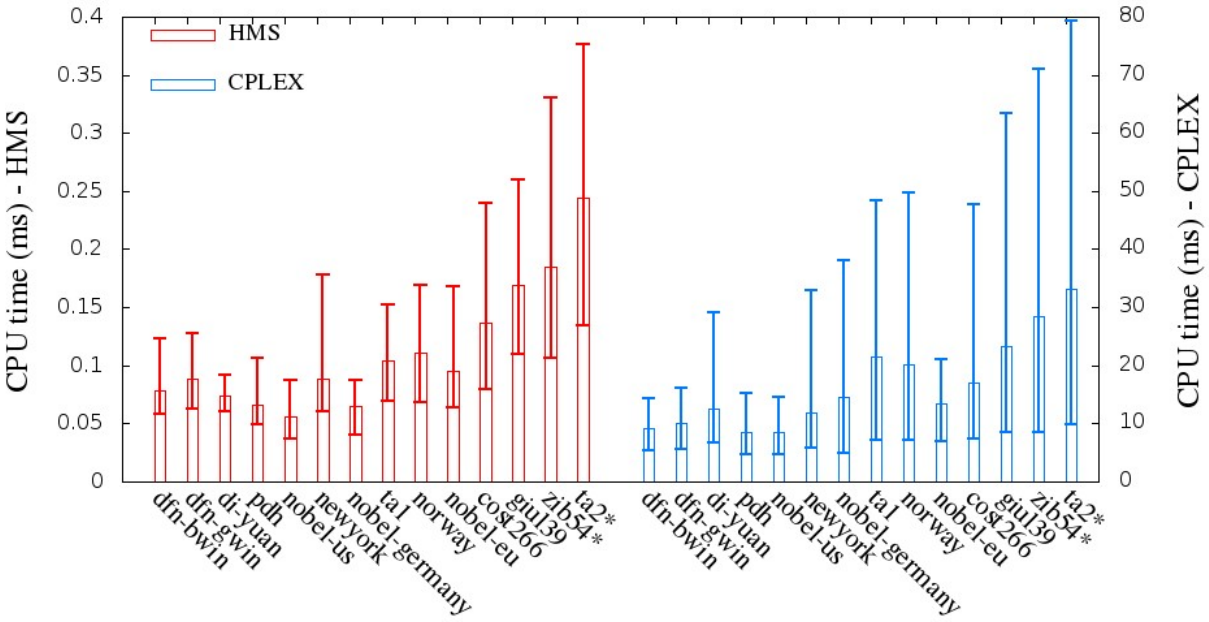


Figure 6.6: HMS's CPU time in milliseconds per node pair.

The average CPU times for each pair of nodes per network is displayed in Figure 6.6, where minimum and maximum times are also presented. These graphs show the same shape as they did for the HLO. Taking into account the number of nodes and density of the networks, exhibited in Table B.1 in Appendix B, it is possible to note that the average times increase with the number of nodes, but also slightly decrease when the networks are less dense. On the other hand, the success rates displayed in Figure 6.7 do not seem to be ruled by this logic.

Finally, the average relative error is shown in Figure 6.8. The values are smaller than the ones presented for HLO. The compared values are in this case the result of a sum and are, therefore, larger than the bandwidth of a single path. On the other hand, the difference between did not scale.

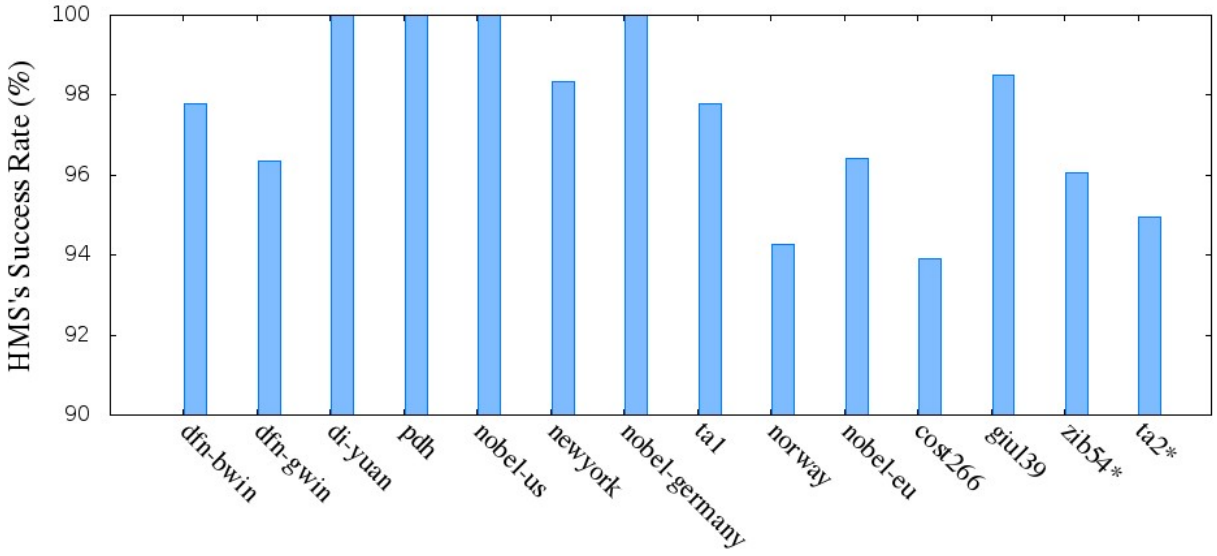


Figure 6.7: HMS's success rate.

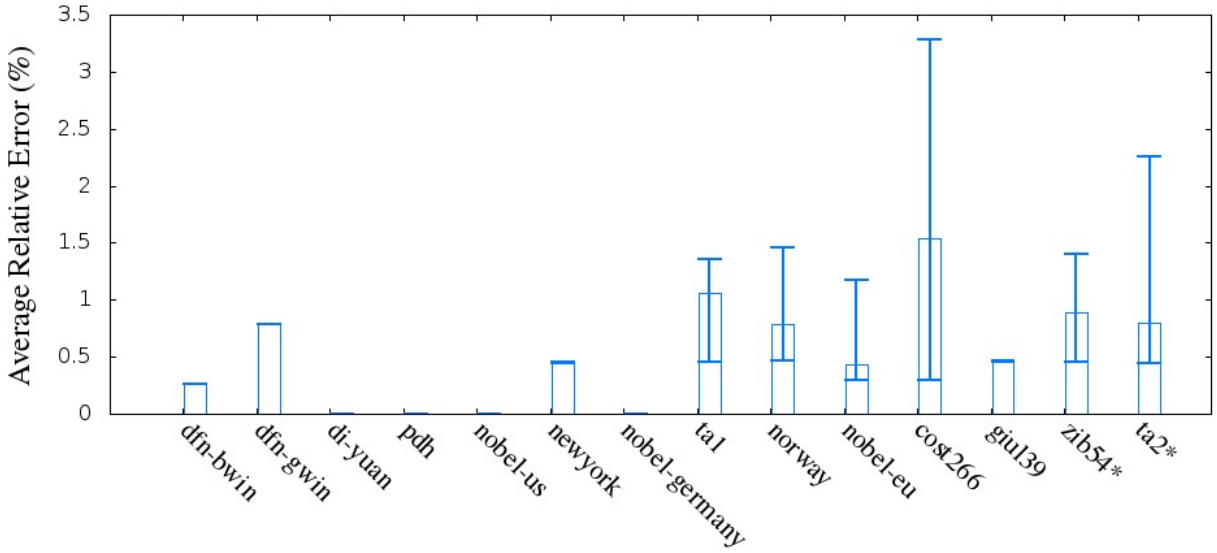


Figure 6.8: HMS's average relative error.

Chapter 7

Conclusion

Routing with protection while ensuring adequate QoS is necessary in telecommunication networks. Path protection requires the calculation of a pair of disjoint paths, satisfying some constraints, namely cost or bandwidth.

The work developed in this thesis led to the development of new heuristics for the calculation of edge-disjoint path pairs. The first proposed heuristic seeks to obtain a widest edge-disjoint path pair, while maximizing the bandwidth of the widest path of the pair; the second one attempts to obtain a path pair such that each path satisfies a specific bandwidth requirement; the third one tries to calculate a path pair maximizing the sum of the bandwidth of the paths. Prior to developing these heuristics, several other related problems were studied, as well as their respective algorithms, some of which were implemented. In particular, algorithms for solving the shortest, widest, shortest-widest, and widest-shortest path problems were implemented; algorithms for solving the min-sum edge-disjoint, widest edge-disjoint, min-sum maximally edge-disjoint and widest maximally edge-disjoint path problems were also implemented.

The performance of the proposed heuristics was evaluated. Each optimization heuristic was shown to have a large optimal solution rate and a low relative error (less than 5% on average). Both of the HLO heuristics took less than 1% of CPLEX's time, while presenting high success rates (over 82% in HLO_f's case and over 92% in HLO_l's case). Also, HLO_f provided the dual labels precision that was necessary to implement the other two proposed heuristics (HML and HMS), which present even higher solution ratios. HML was on a par with CPLEX in most cases and only took about 0.3% of the CPU time, regarding obtaining a feasible path pair. HMS took roughly 1% of CPLEX's CPU time, the average relative error was below 1.5%, and the largest observed relative error value was only 3.5%.

This makes these heuristics interesting and less costly alternatives to CPLEX when it comes to solving these problems, because in real-world applications, the networks may be larger and denser and the time needed to solve a problem may render CPLEX and similar tools impractical. To sum up, the considered problems were successfully addressed.

Regarding future work, it may be possible to combine HLO_f's label accuracy with the success rate of HLO_l.

Appendices

Appendix A

Deinterlacing Algorithm for q^*

Algorithm 9 deinterlaces two paths to obtain path q^* that is part of an edge-disjoint path pair (p^*, q^*), similarly to Algorithm 1.

Algorithm 9 Deinterlacing q^*

Require: Paths p and p' from s to t (described by the successive successors of s).
Ensure: Calculates path q^* of path pair (p^*, q^*).

1: $k \leftarrow 1$ ▷ Starts from the first element of p'
2: $q^* \leftarrow s$ ▷ s is the first node of q^*
3: **repeat** ▷ q^* gets arcs and nodes from p'
4: **if** $a''_k \notin C$ **then** ▷ If arc from p' is not reversed in p
5: $p^* \diamond (v''_k, v''_{k+1})$ ▷ Arc $a''_k = (v''_k, v''_{k+1})$ and next node v''_{k+1} are added to q^*
6: $k \leftarrow k+1$ ▷ Increment k
7: **else**
8: $k \leftarrow k',\ with\ v'_{k'} = v''_k$ ▷ k gets p's index of the last node in q^*
9: **repeat** ▷ q^* gets arcs and nodes from p
10: **if** $a'_k \notin C$ **then** ▷ If arc from p is not reversed in p'
11: $p^* \diamond (v'_k, v'_{k+1})$ ▷ Arc $a'_k = (v'_k, v'_{k+1})$ and next node v'_{k+1} are added to q^*
12: $k \leftarrow k+1$ ▷ Increment k
13: **else**
14: $k \leftarrow k',\ with\ v''_{k'} = v'_k$ ▷ k gets p''s index of the last node in q^*
15: **break** Line 9 cycle
16: **end if**
17: **until** $v'_{k'} = t$ ▷ Algorithm terminates when t is reached
18: **end if**
19: **until** $v''_k = t$ ▷ Algorithm terminates when t is reached
20: **return** q^* (as a set of alternating nodes and arcs)

Appendix B

Additional Information Regarding Performance Analysis

B.1 Characteristics of the Tested Networks

Network	$\|N\|$	$\|E\|$	$\|E\|/\|N\|$
dfn-bwin	10	45	4.5
dfn-gwin	11	47	4.273
di-yuan	11	42	3.818
pdh	11	34	3.091
nobel-us	14	21	1.5
newyork	16	49	3.063
nobel-germany	17	26	1.529
ta1	26	51	1.962
norway	27	51	1.889
nobel-eu	28	41	1.464
cost266	37	57	1.541
giul39	39	86	2.205
zib54*	53	80	1.509
ta2*	64	107	1.672

Table B.1: Networks from SNDLib [16].

B.2 Results of the HML Heuristic

In the following table are the results of the comparison of the proposed heuristic algorithm that addresses the problem in Section 3.3 and CPLEX. The contents of the columns are, from left to right: network's name, bandwidth limit X_2, bandwidth limit X_1 and heuristic success rate (HSR). This success rates are relative to the results obtained from CPLEX, in other words, a 100% success rate means that the heuristic found solutions for all cases where CPLEX also found solutions. Cases for which CPLEX did not find a feasible path pair are discarded.

Network	X_2	X_1	HSR (%)	Network	X_2	X_1	HSR (%)
dfn-bwin	156	169	100	ta1	101	108	100
dfn-bwin	156	182	100	ta1	101	115	100
dfn-bwin	156	195	100	ta1	101	122	100
dfn-bwin	156	208	100	ta1	101	129	100
dfn-bwin	165	178	100	ta1	104	111	96.970
dfn-bwin	165	191	100	ta1	104	118	100
dfn-bwin	165	204	100	ta1	104	125	100
dfn-bwin	174	187	100	ta1	107	114	100
dfn-bwin	174	200	100	ta1	107	121	100
dfn-bwin	183	196	100	ta1	110	117	100
dfn-gwin	156	170	100	norway	101	105	95.157
dfn-gwin	156	184	98.182	norway	101	109	100
dfn-gwin	156	198	100	norway	101	113	100
dfn-gwin	156	212	100	norway	101	117	100
dfn-gwin	165	179	100	norway	102	106	95.667
dfn-gwin	165	193	100	norway	102	110	100
dfn-gwin	165	207	100	norway	102	114	100
dfn-gwin	174	188	100	norway	103	107	98
dfn-gwin	174	202	100	norway	103	111	100
dfn-gwin	183	197	100	norway	104	108	98.718
di-yuan	101	105	100	nobel-eu	143	154	98.860
di-yuan	101	109	100	nobel-eu	143	165	100
di-yuan	101	113	100	nobel-eu	143	176	100
di-yuan	101	117	100	nobel-eu	143	187	100
di-yuan	103	107	100	nobel-eu	149	160	94.587
di-yuan	103	111	100	nobel-eu	149	171	97.917
di-yuan	103	115	100	nobel-eu	149	182	100
di-yuan	105	109	100	nobel-eu	155	166	100
di-yuan	105	113	100	nobel-eu	155	177	100
di-yuan	107	111	100	nobel-eu	161	172	95.455
pdh	161	177	100	cost266	135	148	98.498
pdh	161	193	100	cost266	135	161	97.430
pdh	161	209	100	cost266	135	174	100
pdh	161	225	100	cost266	135	187	100
pdh	169	185	100	cost266	143	156	97.898
pdh	169	201	100	cost266	143	169	98.344

Network	X_2	X_1	HSR (%)	Network	X_2	X_1	HSR (%)
pdh	169	217	100	cost266	143	182	100
pdh	177	193	100	cost266	151	164	96
pdh	177	209	100	cost266	151	177	100
pdh	185	201	100	cost266	159	172	100
nobel-us	125	135	100	giul39	101	111	100
nobel-us	125	145	100	giul39	101	121	100
nobel-us	125	155	100	giul39	101	131	100
nobel-us	125	165	100	giul39	101	141	100
nobel-us	134	144	100	giul39	104	114	100
nobel-us	134	154	100	giul39	104	124	100
nobel-us	134	164	100	giul39	104	134	100
nobel-us	143	153	100	giul39	107	117	100
nobel-us	143	163	100	giul39	107	127	100
nobel-us	152	162	100	giul39	110	120	100
newyork	102	106	100	zib54*	101	111	96.779
newyork	102	110	97.468	zib54*	101	121	100
newyork	102	114	100	zib54*	101	131	100
newyork	102	118	100	zib54*	101	141	100
newyork	105	109	99.048	zib54*	105	115	100
newyork	105	113	100	zib54*	105	125	100
newyork	105	117	100	zib54*	105	135	100
newyork	108	112	100	zib54*	109	119	100
newyork	108	116	100	zib54*	109	129	100
newyork	111	115	100	zib54*	113	123	100
nobel-germany	175	199	100	ta2*	101	113	98.573
nobel-germany	175	223	100	ta2*	101	125	100
nobel-germany	175	247	100	ta2*	101	137	100
nobel-germany	175	271	100	ta2*	101	149	100
nobel-germany	189	213	100	ta2*	105	117	95.694
nobel-germany	189	237	100	ta2*	105	129	100
nobel-germany	189	261	100	ta2*	105	141	100
nobel-germany	203	227	100	ta2*	109	121	100
nobel-germany	203	251	100	ta2*	109	133	100
nobel-germany	217	241	100	ta2*	113	125	100

Table B.2: Comparison results of the Heuristic for Edge-disjoint Path Pairs with Minimum Limits

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms and applications*. Prentice Hall, 1993.

[2] Anteneh Beshir and Fernando Kuipers. Variants of the min-sum link-disjoint paths problem. In *Proceedings of the 16th Annual IEEE Symposium on Communications and Vehicular Technology (IEEE SCVT'09)*, Louvain-la-Neuve, Belgium, November 2009.

[3] R. Bhandari. *Survivable Networks, Algorithms for Diverse Routing*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.

[4] J. Craveirinha, J. Clímaco, M. Pascoal, and L. Martins. Traffic splitting in MPLS networks - a hierachical multicriteria approach. *Journal of Telecommunications and Information Technology*, 4:3–10, 2007.

[5] P. Cruz, T. Gomes, and D.Medhi. An heuristic for widest edge-disjoint path pair lexicographic optimization. *Submitted*, 2014.

[6] Mostafa H. Dahshan. Maximum-bandwidth node-disjoint paths. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 3(3), 2012.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269 – 271, 1959.

[8] TC Hu. The maximum capacity route problem. *Operations Research*, 9(6):898–900, 1961.

[9] IBM. IBM ILOG CPLEX Optimization Studio v12.6, 2013.

[10] H. Leng, J. Song, M. Liang, Z. Xie, and J. Zhang. Routing on shortest pair of disjoint paths with bandwidth guaranteed. In *Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, pages 557–561, 2009.

[11] Soung-Yue Liew and Ming-Lee Gan. An exact optimum paths-finding algorithm for $\alpha + 1$ path protection. In *Information Networking (ICOIN), 2012 International Conference on*, pages 210 –215, Feb. 2012.

[12] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[13] D. Medhi and K. Ramasamy. *Network Routing: Algorithms, Protocols, and Architectures*. Morgan Kaufmann Publishers (an imprint of Elsevier), 2007.

[14] R. Ogier, B. Bellur, and N. Taft-Plotkin. An efficient algorithm for computing shortest and widest maximally disjoint paths. Technical Report ITAD-1616-TR-170, SRI International, November 1998.

[15] W. Ogryczak, M. Pioró, and A. Tomaszewski. Telecommunications network design and max-min optimization problem. *Journal of Telecommunications and Information Technology*, (3):43–56, 2005.

[16] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. SNDlib 1.0–Survivable Network Design library. *Networks*, 55(3):276–286, 2010.

[17] Maurice Pollack. The maximum capacity through a network. *Operations Research*, 8(5):733–736, 1960.

[18] B. H. Shen, B. Hao, and A. Sen. On multipath routing using widest pair of disjoint paths. In *2004 Workshop on High Performance Switching and Routing*, pages 134 – 140, 2004.

[19] C. Simões, T. Gomes, J. Craveirinha, and J. Clímaco. Performance analysis of a bi-objective model for routing with protection in WDM networks. *Journal of Telecommunications and Information Technology*, (3):25–35, 2010.

[20] J. W. Suurballe. Disjoint paths in networks. *Networks*, 4:125–145, 1974.

[21] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[22] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14:1228–1234, 1996.