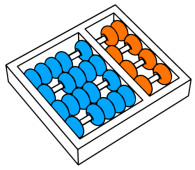


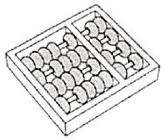
.

Rafael Fonseca dos Santos

Finding multicolour paths in graphs: NP-hardness and algorithms

Encontrando caminhos com múltiplas cores em grafos: NP-dificuldade e algoritmos

CAMPINAS
2015



University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

Rafael Fonseca dos Santos

Finding multicolour paths in graphs: NP-hardness and algorithms

Supervisor:
Orientador(a): Prof. Dr. Eduardo Cândido Xavier

Encontrando caminhos com múltiplas cores em grafos: NP-dificuldade e algoritmos

MSc Dissertation presented to the Graduate Program of the Institute of Computing of the University of Campinas to obtain a Master degree in Computer Science.

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Mestre em Ciência da Computação.

This volume corresponds to the final version of the Dissertation defended by Rafael Fonseca dos Santos, under the supervision of Prof. Dr. Eduardo Cândido Xavier.

Este exemplar corresponde à versão final da Dissertação defendida por Rafael Fonseca dos Santos, sob orientação de Prof. Dr. Eduardo Cândido Xavier.

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2015

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

Sa59f

Santos, Rafael Fonseca dos, 1988-
Finding multicolour paths in graphs : NP-hardness and algorithms / Rafael Fonseca dos Santos. – Campinas, SP : [s.n.], 2015.

Orientador: Eduardo Cândido Xavier.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Algoritmos em grafos. 2. Análise de algoritmos. 3. Multiplexação por divisão de comprimento de onda. 4. Simulação (Computadores). 5. Otimização combinatória. I. Cândido, Eduardo Marcelo,1979-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Encontrando caminhos com múltiplas cores em grafos : NP-dificuldade e algoritmos
Palavras-chave em inglês:
Algorithms in graphs
Algorithm analysis
Wavelength division multiplexing
Computer simulation
Combinatorial optimization
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Eduardo Cândido Xavier [Orientador]
André Costa Drummond
Juliana Freitag Borin
Data de defesa: 03-06-2015
Programa de Pós-Graduação: Ciência da Computação

T E R M O D E A P R O V A Ç Ã O

Defesa de Dissertação de Mestrado em Ciência da Computação, apresentada pelo(a) Mestrando(a) **Rafael Fonseca dos Santos**, aprovado(a) em **03 de junho de 2015**, pela Banca examinadora composta pelos Professores(as) Doutores(as):

Prof(a). Dr(a). André Costa Drummond
Titular

Prof(a). Dr(a). Juliana Freitag Borin
Titular

Prof(a). Dr(a). Eduardo Cândido Xavier
Presidente

Institute of Computing /*Instituto de Computação*
University of Campinas /*Universidade Estadual de Campinas*

Finding multicolour paths in graphs: NP-hardness and algorithms

Rafael Fonseca dos Santos[1]

June 03, 2015

Examiner Board/*Banca Examinadora*:

- Prof. Dr. Eduardo Cândido Xavier (Supervisor/*Orientador*)
- Prof. Dr. Juliana Freitag Borin
  Institute of Computing - UNICAMP
- Prof. Dr. André Costa Drummond
  Institute of Computing - UNB
- Prof. Dr. Diego de Freitas Aranha
  Institute of Computing - UNICAMP (Substitute/*Suplente*)
- Prof. Dr. Yuri Abtibol
  Institute of Computing - UFF (Substitute/*Suplente*)

[1]Financial support: CNPq scholarship (process 133039/2011-0) 2011 and Capes 2012–2013

Abstract

In this work we study a problem of finding minimum coloured paths in graphs that has applications in WDM optical networks when high bandwidths are required to send data between a pair of nodes in the graph. Let $G = (V, E)$ be a (directed) graph with a set of nodes V and a set of edges E in which each edge has an associated positive weight $w(i, j)$, and $C = \{1, 2, \ldots, x\}$ be a set of x colours, $x \in \mathbb{N}$. The function $c : E \mapsto 2^C$ maps each edge of the graph G to a subset of colours, where 2^C is the power set of C. Given a positive integer $k > 1$, a k-multicolour path is a path in G such that there exists a set of k colours $K = \{c_1, \ldots, c_k\} \subseteq C$, with $K \subseteq c(i, j)$ for each edge (i, j) in the path.

The problem of finding one or more k-multicolour paths in a graph has applications in optical network and social network analysis. In the former case, the available wavelengths in the optical fibres are represented by colours in the edges and the objective is to connect two nodes through a path offering a minimum required bandwidth. For the latter case, the colours represent relations between elements and paths help identify structural properties in such networks.

In this work we investigate the complexity of the multicolour path establishment problem. We show it is NP-hard and hard to approximate. Additionally, we develop Branch and Bound algorithms, ILPs, and heuristics for the problem. We then perform an experimental analysis of the developed algorithms to compare their performances.

Resumo

Neste trabalho estudamos um problema de encontrar caminhos coloridos mínimos em grafos. Tal problema tem aplicações práticas em redes ópticas WDM quando elevada largura de banda é necessária para enviar dados entre um par de nós do grafo. Seja $G = (V, E)$ um grafo (orientado) com conjunto de vértices V e conjunto de arestas E no qual a cada aresta está associado um peso positivo $w(i, j)$ e seja $C = \{1, 2, \ldots, x\}$ um conjunto de x cores, $x \in \mathbb{N}$. A função $c : E \mapsto 2^C$ mapeia cada aresta do grafo G para um subconjunto de cores, onde 2^C é o conjunto potência de C. Dado um inteiro positivo $k > 1$, um caminho k-colorido é um caminho em G para o qual existe um conjunto de k cores $K = \{c_1, \ldots, c_k\} \subseteq C$ tal que $K \subseteq c(i, j)$ para toda aresta (i, j) no caminho.

O problema de encontrar um ou mais caminhos k-coloridos em um grafo tem aplicações em redes ópticas e análise de redes sociais. No primeiro caso, os comprimentos de onda disponíveis nas fibras ópticas são representados por cores nas arestas e o objetivo é conectar dois nós da rede através de um caminho com um mínimo de banda requerida. No caso de redes sociais, as cores representam relacionamentos entre os elementos e caminhos ajudam a identificar propriedades estruturais nessas redes.

Neste trabalho investigamos a complexidade do problema de estabelecer caminhos com múltiplas cores. Mostramos que é NP-difícil e difícil de aproximar. Também desenvolvemos algoritmos *Branch and Bound*, Programas Lineares Inteiros e heurísticas para o problema. Por fim realizamos uma análise experimental dos algoritmos desenvolvidos e comparamos seus desempenhos.

To my family and friends

Acknowledgements

In a world where lists are loved and overused in every click-bait title ("The X things you should know", "The Y ways of eating healthily", ...), even though the matters presented should not be just listed, but also accompanied by a lengthy and thorough explanation, one might not understand the frustration of creating a simple thanking list. On the attempt to avoid listing everyone and unintentionally forgetting to mention an important person, thus offending said person, one might try to show appreciation in generic terms. Which is also doomed to fail, for no one will feel rightfully thanked.

So a disclaimer is added, saying to the unthanked to feel thanked even though not mentioned by name. In the partial justice of that act, at least some justice is done. Therefore, some names are mentioned, some in a specific way, others in a more generalised manner, those not named are covered by the disclaimer. Everybody is thanked. In a way. Q.E.D.

I would like to thank anyone with whom I talked about my research. Sometimes, just talking makes things more clear, more organized in your head. For the same reason, I thank all my friends that not only listened (more than once!) but also offered some form of encouragement or friendly word. Counter-intuitively I thank those who dragged me away from the research. Progress includes periods of not-thinking too. Everyone who ever convinced me to stop working a few hours for a drink, a dinner, or just a chat, thank you!

I thank everyone in the LOCo lab who contributed for its functioning. I thank the IC staff for being there when needed, answering questions and dealing with all the boring bureaucracy. I thank all my teachers who (in)directly contributed in some way or another to this research. And thank Alessando Andrioni for helping in the early stages of my research as part of an undergraduate research project.

I thank my house mates, Ania and DJ, for hearing my complaints without having any idea of what I was talking about, for wanting to know my progress nevertheless and offering encouraging words. I thank my friend Ciarán and his girlfriend Laura for putting up with me and my never-ending research for the past two years; for reviewing my thesis while going on holidays; for dragging me outside or leaving me alone to work, all at the right moment. Also Tom for the grammar check and Jen for the (delicious!) food breaks.

I thank my advisor and tutor, Eduardo, for believing that I would keep working even when I moved to the other side of the Sea and took two more years than intended to finish this text. Thank you for putting up with me all this time, even if unsure of my conviction of finishing the research. Eventually. It was bigger than me, you see? I would keep going until the end because there was a part of me that is important and will be needed for posterity.

And last but not least, I thank the financial support from CNPq and Capes.

"You see? Size defeats us. [...] one might take the tip of the pencil and magnify it. One reaches the point where a stunning realization strikes home: The pencil tip is not solid; it is composed of atoms which whirl and revolve like a trillion demon planets. What seems solid to us is actually only a loose net held together by gravity. Viewed at their actual size, the distances between these atoms might become league, gulfs, aeons. The atoms themselves are composed of nuclei and revolving protons and electrons. One may step down further to subatomic particles. And then what? Tachyons? Nothing? Of course not. Everything in the universe denies nothing; to suggest an ending is the one absurdity."

Stephen King – The Dark Tower

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Finding paths in a computer network is a basic problem in combinatorial optimization: given a network and two of its nodes, a source and a target, we want to find one or multiple paths between these nodes with specific properties. There are classic examples of such problems in the literature, such as finding shortest paths [13, 17], node-disjoint or edge-disjoint paths [24, 31], routing, line planning [5], etc.

The shortest path problem is of fundamental importance in network optimization and arises as a subproblem in many different scenarios: from purely graph theory problems, to VSLI and network design, and even social network analysis.

Consider the example of social network analysis. Social networks can be represented and studied as graphs; the vertices represent the elements being analysed and the edges are binary relations between them. Different kinds of relations might be represented by distinct colours on the edges. Connectivity properties or the availability of paths between vertices help to identify structural properties like group cohesiveness and centrality in such networks [3].

Finding paths on edge-coloured graphs can also be used for solving routing problems in WDM optical networks. In a WDM network, at any given moment, each optical link (edge) has a set of available wavelengths through which data can be transmitted in parallel. Data from a source to a target node is sent by the establishment of a *lightpath* between this pair of nodes. A lightpath is a special path for it uses the same wavelength throughout all its links. In order to send data between any pair of nodes, one has to find a lightpath between them. When a high bandwidth is required, it is necessary to find multiple lightpaths going through the same set of edges so that no two lightpaths that share an edge use the same wavelength. This routing problem in WDM networks is equivalent to finding paths in edge-coloured graphs: the wavelengths can be directly mapped to colours on the edges, and the objective is to find a path between a source and a sink node with certain number of colours. Chen et al. presented in [8] the problem of

finding shortest paths between a pair of nodes such that the total number of wavelengths is k with the restriction that all edges in those paths must have the same k available wavelengths.

To illustrate this problem, refer to Figure 1.1. The numeric labels on the edges are the colours available in them. We assume arcs with unitary weights, thus omitting the weights. Suppose we want to find a path between s and t. A solution for $k = 1$ is (s, v_2, t) or (s, v_3, t). The path (s, v_1, v_4, t), although having 2 colours in common, is not a shortest path from s to t. For $k = 2$, the solution is the path (s, v_2, t). For $k \geq 3$, there are no possible solutions, since the highest number of common colours on any given path is 2.

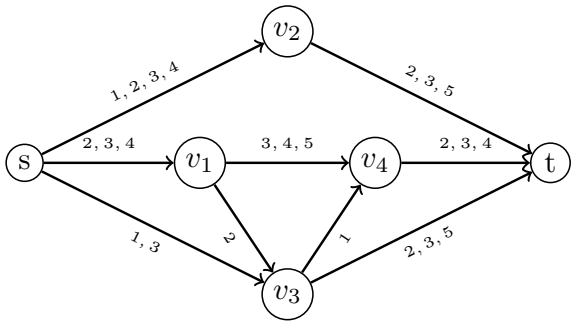


Figure 1.1: Edge-coloured graph example

In this work, we propose to study the problem presented in [8] as an edge-coloured graph problem. We analyse its complexity and prove it is NP-hard. Unless P = NP, there are no efficient polynomial-time algorithms to solve NP-hard problems optimally. Therefore we develop heuristics and study their performance through a set of computational experiments.

1.1 Objectives

We address the problem of multipath routing in optical networks with extremely high bandwidth requests [8]. First, we formulate the problem as an edge-coloured graph problem. We study its complexity and prove it is NP-hard and hard to approximate. Computational results obtained from a simulation show the performance of the developed algorithms against exact Branch and Bound and Integer Linear Programming algorithms.

1.2 Text organization

This text is organized as follows. In Section 1.3, we discuss works related to problems in graphs involving colours and optical networks design. In Chapter 2, we present terms, notations and assumptions that will be used throughout the text (Section 2.1). We also discuss approximation algorithms (Section 2.2), the Branch and Bound paradigm (Section 2.3), and problems related to finding paths in graphs (Section 2.4).

Chapter 3 introduces a formal description of the problems being tackled and a proof of their NP-hardness is given on Chapter 4. Chapter 5 and Chapter 6 present the developed algorithms including exact branch and bound (Section 5.1) and Integer Linear Programs (Section 5.2), as well as Dijkstra-based (Section 6.1) and intersection-based (Section 6.3) heuristics.

In Chapter 7 we discuss the computational results obtained through simulations, separating them into two criteria: execution time and solution cost (Section 7.1), and blocking ratio (Section 7.2).

Finally, we draw our conclusions on Chapter 8.

1.3 Related works

In this section we discuss works related to coloured graphs and WDM optical networks design.

A **colouring** of a graph is an assignment of colour labels to elements of a graph. These elements can be either vertices (vertex colouring), edges (edge colouring), or both (total colouring). Colouring problems consist in colouring elements in a way to satisfy a certain property. Another variant, given an already coloured graph, is to find paths satisfying some requirement. Practical problems can be modelled as problems involving colours in graphs, for example timetabling, frequency assignment in telecommunication networks, social network analysis, reliability in networks, and so on.

The most common form of graph colouring is the vertex colouring. In this type of problem, the goal is to colour the vertices of a graph such that no two adjacent vertices share the same colour. The minimum number of colours with which a graph can be vertex-coloured is called *chromatic number*, and it is represented by $\chi(G)$. A graph G whose vertices are coloured with exactly $\chi(G)$ colours is called a properly coloured graph. For paths and bipartite graphs, the chromatic number is trivially 2. From Brook's theorem [4] we know that $\chi(G) \geq \Delta(G)$ for connected graphs that are not complete graphs or odd cycles, where $\Delta(G)$ represents the maximum degree of G. The vertex colouring problem was proven to be NP-complete by Karp in 1972 [22]. A survey on the algorithmic and computational results obtained for the vertex colouring problem can be found in [27].

A related problem studied by Granata et al. [19] is that of finding a path from a vertex s that meets all the $\chi(G)$ colours, that is, a path that contains all $\chi(G)$ colours. They prove that, in properly coloured directed graphs, finding a path that starts at a specific vertex and meets all $\chi(G)$ colours is NP-complete. The same complexity still holds to finding a shortest and longest of all paths between two given vertices meeting all the colours in a properly coloured directed graph.

When given a vertex-coloured graph G, in which adjacent vertices not necessarily have to be coloured with distinct colours, a path in G whose internal vertices have the same colours is called a vertex-monochromatic path [6]. We define a monochromatic vertex-connectivity (MVC)- colouring to be a vertex colouring so that there is a vertex-monochromatic path between any two vertices in the graph. For a connected graph G, we define as $mvc(G)$ the maximum number of colours used in a MVC-colouring of G. In [6], Erdős-Gallai-type problems are investigated for the monochromatic vertex-connectivity number $mvc(G)$ as well as the Nordhaus-Gaddum-type inequality for $mvc(G)$ is given.

The exact opposite counterpart of determining the mvc number is the well-studied problem of the vertex-rainbow connectivity number. A vertex-coloured graph is rainbow vertex-connected if any two of its vertices are connected by a path whose internal vertices have distinct colours. A descriptive survey is presented in [25] covering problems related to rainbow connectivity in graphs.

Now we turn our attention to edge-colouring. A problem that has been gaining popularity because of its use in social network analysis is that of finding paths in edge-coloured graphs. When social networks are represented as graphs, the vertices represent the elements analysed while the edges represent a binary relation between these elements. That way, the vertex connectivity is a measure of the information flowing from one vertex to another. This information can be used for determining group cohesiveness and centrality [3].

Let $G = (V, E)$ be an edge-coloured graph on which there is a colouring $c : E \mapsto \{1, 2, \ldots, n\}$, $n \in \mathbb{N}$. In this case each edge has just one colour. A path in G is a rainbow path if no two edges are coloured the same. A rainbow (s, t)-path in G is a rainbow path between s and t of length $d(s, t)$, where $d(s, t)$ is the length of the shortest path between s and t. Note that this problem is the edge-coloured equivalent of the vertex-rainbow connectivity problem. The survey in [25] also discusses rainbow connectivity in edge-coloured graphs. Li and Sun [26], discuss the strong rainbow connectivity number of a graph. A graph is said to be strongly rainbow connected if there exists a rainbow (s, t)-path with length $d(s, t)$ for any two vertices s and t in the graph. Denoted by $src(G)$, the strong rainbow connectivity number of a graph is the minimum number of colours needed to make a graph G strongly rainbow connected. In [26], sharp upper bound for $src(G)$ is given in terms of the number of edge-disjoint triangles in G. They

also investigate the graphs with large strong rainbow connectivity numbers.

Now let $G = (V, \mathcal{E})$ be an edge-coloured graph, where V is the set of nodes and $\mathcal{E} = \{E_1, E_2, \ldots, E_c\}$ is a collection of c not necessarily disjoint edge sets. Each $E_i \subseteq V \times V$, $1 \leq i \leq c$, is the edge set of colour i. Let $G_i = (V, E_i)$ be the graph with only the edges of colour i.

Wu [34] studied the problem of finding, given two vertices $s, t \in V$, the maximum number of mutually vertex-disjoint uni-colour paths between s and t. We call this problem the Max CDP. He proved it is NP-hard and cannot be approximated with ratio less than 2 in polynomial time, unless P = NP for $c \geq 2$. Bonizzoni [3] later showed that the Max CDP is not approximable within factor $c^{1-\epsilon}$ for any $\epsilon > 0$. For $c = 1$, it can be reduced to a maximum flow problem, and is therefore polynomial time solvable. A c-approximation algorithm is given to Max CDP by the use of a greedy strategy. The idea of the algorithm is the following. Let $\kappa_i(s, t)$ denote the maximum number of vertex-disjoint paths between s and t in G_i. Find $\kappa_i(s, t)$ for each i. Then repeat until no paths remain: select i with maximum $\kappa_i(s, t)$ and put all the paths into the solution; remove all internal nodes of these paths. This procedure runs in $O(mn)$-time.

For the length-bounded case, ℓ-LCDP, where the solution paths' lengths are required to be upper bounded by a fixed integer ℓ, Wu proved it can be solved polynomially for $\ell = 3$ through graph matching and it is NP-hard for $\ell \geq 4$ and can be approximated with ratio $(\ell - 1)/2 + \epsilon$ for any $\epsilon > 0$. Bonizzoni et al. also gave a fixed-parameter algorithm to the ℓ-LCDP problem.

Although not stated as purely a graph problem, the problems of finding paths on optical networks can be easily mapped to a graph problem by mapping the availability of a wavelength in a given link to a colour on that link (edge). With that in mind, we now turn our attention to the problem of paths establishment in optical networks.

To handle the ever-increasing bandwidth demands of network users, a Wavelength-Division Multiplexing (WDM) approach has been proposed for optical fibre networks. WDM can divide the high bandwidth of a fibre into many non overlapping wavelengths (WDM channels) thus allowing multiple channels coexistence on a single fibre. In such networks, an optical signal passing through an optical switch may be routed from an input fibre to an output fibre without undergoing optoelectronic conversion [2].

Chlamtac et al. [10] proposed the *lightpath* architecture as a means of end users to communicate with one another via all-optical WDM channels. A lightpath is a path spanning multiple fibre links. Assuming the absence of wavelength converters, a lightpath must occupy the same wavelength on all the fibre links through which it traverses (*wavelength-continuity constraint*). The problem of setting up lightpaths by routing and assigning a wavelength to each connection in a set of connections is called the Routing and Wavelength-Assignment (RWA) problem. The objective is to route lightpaths and

assign wavelengths in a manner that minimizes the amount of network resources consumed, while also ensuring that no two lightpaths share the same wavelength on the same fibre link. In [10] is shown that even when all the connections to be established are known in advance (Static RWA), the problem is equivalent to a vertex-colouring in a graph and, therefore, is NP-complete. To solve the RWA problem, one can decouple it into two separate sub problems: routing and wavelength assignment. Many approaches relying in ILP formulations and heuristics have been presented to solve the problem and its components [36, 21].

Applications in the scientific and engineering communities and emerging media applications require extremely high bandwidth connections, typically larger than one wavelength [8]. To accommodate such requests, Chen et al. propose a modification to the RWA problem to allow for more than one wavelength to be assigned to a lightpath. The only restriction is that the set of wavelengths assigned to a lightpath must be available on all of its links. A connection request is *blocked* when it is either not possible to route a path between its end points or to assign the required wavelengths to the lightpath found. When single path routing is deployed, due to a small chance of more than one wavelength per fibre being free for a given path, a higher blocking is observed. For that reason, a multipath approach is also proposed. Besides improving blocking ratio when compared to single routing, multipath routing also reduces network resource consumption, like minimizing bandwidth required for backup paths in case of link failures.

Recent works have been proposing approaches to path provisioning and traffic grooming in more wavelength-flexible networks (flexgrid optical network) [7, 28, 12].

1.4 Contributions

We address the problem of multipath routing in WDM optical networks with extremely high bandwidth requests [8]. We formulate the problem as an edge-coloured graph problem. We study its complexity and prove it is NP-hard and hard to approximate. Unless P = NP, there are no efficient polynomial-time algorithms to solve NP-hard problems optimally. Therefore we develop heuristics and study their performance against exact Branch and Bound and Integer Linear Program algorithms through a set of computational experiments. A network simulation evaluates our algorithms according to solution cost, execution time and blocking ratio.

Chapter 2

Preliminaries

This chapter contains a summary of definitions, notations and basic notions which will be needed in the following work. We present basic definitions about approximation algorithms, giving related definitions about the subject. We briefly discuss techniques used when developing approximation algorithms. We also talk about paths in graphs, showing terms and definitions used.

2.1 Terms, notations and assumptions

When describing problems and algorithms for paths in graphs, we make use of the following terms:

1. The terms arc and edge will be used interchangeably.
2. The terms node and vertex will be used interchangeably.
3. An arc from a node i to a node j is represented as (i, j), its given direction pointing from i to j. Arc (i, j) is called an input for j, and an output for i.
4. Node j is an i-neighbour if (i, j) is in G.
5. $\delta^+(i)$ denotes the set of out-neighbours of the node i, that is, the set of nodes j for which there exists the edge (i, j) in E. Likewise, $\delta^-(i)$ denotes the in-neighbours, the set of nodes j such that $(j, i) \in E$.
6. A *directed path* is determined by a sequence of nodes $n_1, n_2, \ldots, n_k$; it consists of these nodes and the arcs connecting them in sequence, $(n_1, n_2), (n_2, n_3), \ldots, (n_{k-1}, n_k)$. We say such a path *connects* nodes n_1 and n_k and represent it as (n_1, n_k)-path.

7. A path is node-disjoint (arc-disjoint) if its node sequence (arc sequence) has no repetitions.

8. Two paths are mutually node-disjoint (arc-disjoint) if they do not share any vertex (arc) except at the extremes. Likewise, a set of paths is mutually node-disjoint (arc-disjoint) if they are pairwise mutually node-disjoint (arc-disjoint).

9. Given a path from an origin to a node j, node i is a *predecessor* of j, and denoted by p_j, in either one of these cases: *a*) arc (i, j) is in the path and *b*) j is preceded in the path by a continuous, non-empty sequence of reverse arcs beginning at i.

10. $w(i, j)$ = weight of arc (i, j).

Assumptions:

- Unless otherwise stated, the given graph is directed. That is because undirected graphs can be made directed by replacing any undirected arc by two oppositely directed arcs [31].
- Because of the previous assumption, arc means directed arc.
- The graph has no multiple arcs, i.e. there is at most one arc (i, j) from node i to node j.
- There are no node weights associated with the nodes of the graph. In the case there are, it is convenient to interpret them as arc weights by a node splitting technique [31]: for each node i with weight $w_i \neq 0$, introduce an auxiliary node i', and reassign all outputs on i as outputs on i'. Leave all arc weights unchanged. Connect i and i' by an auxiliary arc (i, i') with weight w_i.
- Weights on arcs are non-negative.
- In an (s, t)-path, $s \neq t$.

2.2 Approximation algorithms

In this section we give a brief overview of approximation algorithms, introducing basic concepts about the subject.

Approximation algorithms are those with polynomial time complexity to find approximate solutions to optimization problems. Given an algorithm $\mathcal{A}$ for an minimization problem $\mathcal{P}$, let I be an input instance of this problem. We denote by $\mathcal{A}(I)$ the value of

the solution returned by algorithm $\mathcal{A}$ when applied to the instance I and by $\mathcal{OPT}(I)$ the corresponding value of an optimal solution for the same instance. We say an algorithm has approximation factor α, or it is an α-approximation algorithm, if $\mathcal{A}(I)/\mathcal{OPT}(I) \leq \alpha$, for any instance I.

When developing an approximation algorithm, the first step is to search for a proof of its approximation factor. Another aspect is to check if the approximation factor α is the best possible. For that, we have to find an instance whose ratio between the obtained solution and the optimal one is equal to α or as close as possible to α. In that case, we say the approximation factor is tight, that is, it is impossible to improve it. From a theoretical point of view, the most desired algorithms are those whose solutions values are as close as possible to the optimal value. It is possible to show for some problems the there are algorithms with approximation factor $(1 + \epsilon)$ or $(1 - \epsilon)$ for minimization and maximization problems, respectively, where ϵ is a small enough constant value, $\epsilon > 0$.

A common strategy for treating combinatorial optimization problems is to formulate them through an integer linear programming system and solve its linear relaxation, which can be done in polynomial time. Linear programming has been used to obtain approximation algorithms in many different ways. A common one is rounding the fractional solutions of a linear program. Another technique is to solve the dual system of the linear program, instead of the primal, obtaining in that way a solution based on dual variables. A more recent technique is the primal-dual approximation method, used for developing combinatorial algorithms based on the duality theory in linear programming. In that case, the method is combinatorial em general, not requiring the linear programs to be solved and consists of a generalization of the traditional primal-dual method.

There are also problems to which it is impossible to find an algorithm with a certain approximation factor. Given a problem $\mathcal{P}$, we say it is α-inapproximable if there is no α-approximation algorithm for it. A way to prove this is to show that whenever there is an α-approximation algorithm for the problem $\mathcal{P}$, then it is possible to solve optimally and in polynomial time an NP-hard problem [33].

2.3 Branch and Bound algorithms

In this section we summarize the Branch and Bound paradigm using the terminology by Clausen [11]. We will restrict ourselves to minimization problems without loss of generality; maximization problems can be dealt with in a similar manner.

Many optimizations problems are easy to state, have a finite but large number of feasible solutions and no polynomial method to find the best solution is known. The Branch and Bound paradigm is a useful and efficient tool in solving NP-hard discrete optimization problems to optimality. A Branch and Bound algorithm searches the complete

space of solutions for the best solution by using bounds for the function to be optimized combined with the value of the current best solution rather than explicitly enumerating the exponentially increasing number of potential solutions.

A Branch and Bound algorithm consists of three main components:

1. a *bounding function* providing a lower bound for the best value obtainable in a given subspace of the solution space,
2. a *selection strategy* for selecting the solution subspace to be investigated in the current iteration, and
3. a *branching rule* to be applied in case the subspace cannot be discarded, thereby subdividing the subspace into more subspaces to be investigated in subsequent iterations.

Let $\mathcal{P}$ be a problem in which the objective function is

$$\min_{x \in S} f(x)$$

on the variables $(x_1 \dots x_n)$, where S is a region of feasible solutions. Now consider P, a set of *potential solutions* for which f is still well defined, containing S, and a *bounding function* $g(x)$ defined on S (or P) with the property that $g(x) \leq f(x)$, for all x in S (resp. P).

The solution of a problem with a Branch and Bound algorithm is described as a search through a search tree. The root node corresponds to the original problem to be solved and the children nodes correspond to subproblems of the original one. The leaves correspond to solutions. For all NP-hard problems there are instances for which an exponential number of leaves exist in the search tree. To each node in the tree, the bounding function g associates a real number, called *bound*. On internal nodes such numbers are lower bounds for the value of any solution in the subspace corresponding to that node; for leaves, the bound corresponds to the value of the solution.

The bounding function is such that for any leaf it agrees with the objective function and provides a not worse bound for subspaces added to the tree. With that said, g is required to satisfy the following three conditions:

1. $g(P_i) \leq f(P_i)$ for all node P_i in the tree,
2. $g(P_i) = f(P_i)$ for leaves, and
3. $g(P_i) \geq g(P_j)$ if P_j is ancestor of P_i.

The search tree is developed dynamically during the search and initially consists of only the root node. A heuristic can be used to produce a feasible solution and its value is used as the current best solution. If no such heuristic exists, the initial value is set to infinity. In each iteration of a Branch and Bound algorithm, a node is selected for exploration using a lazy or eager selection strategy.

If the eager strategy is used, a branching is performed. That means the subspace is subdivided into smaller subspaces. For each of these the bound for the node is calculated. The subspace is discarded for further evaluation in one of two cases: (1) if the bound is no better than the best, since no feasible solutions can be better than that of the best solution found so far, and (2) in case no feasible solution to the subproblem exist. Otherwise the node (with the bound as part of the information stored) is then joined to the pool of live subspaces. The algorithm with the eager strategy is outlined in Algorithm 1.

Algorithm 1 Branch and Bound with eager selection strategy

```
Best ← ∞
LB(P_0) ← g(P_0)
Live ← {(P_0, LB(P_0))}
while Live is not empty do
    Select P from Live.
    Live ← Live \ {P}
    Branch on P generating P_1, ..., P_k.
    for i from 1 to k do
        LB(P_i) ← g(P_i)                                   ▷ Bound P_i
        if LB(P_i) = f(X) for some feasible solution X And f(X) < Best then
            Best ← f(X)
            Solution ← X.
            Go to EndBound
        if LB(P_i) ≥ Best then
            Discard P_i
        else
            Live ← Live ∪ {(P_i, LB(P_i))}.
    EndBound
return Solution with value Best.
```

If the lazy strategy is used, the order of bound calculation and branching is reversed, and the live nodes are stored with the bound of their fathers as part of the information on the node. Algorithm 2 on the following page outlines the procedure.

Algorithm 2 Branch and Bound with lazy selection strategy

```
Best ← −∞
Live ← {(P_0, −∞)}
while Live is not empty do
    Select P from Live.
    Live ← Live \ {P}
    LB(P) ← g(P)                                          ▷ Bound P
    if LB(P) = f(X) for feasible solution X And f(X) < Best then
        Best ← f(X).
        Solution ← X.
        Go to EndBound
    if LB(P) ≥ Best then
        Discard P.
    else
        Branch on P generating P_1, ..., P_k.
        for i from 1 to k do
            Live ← Live ∪ {(P_i, LB(P))}.
    EndBound
return Solution with value Best
```

The search terminates when there are no more unexplored parts of the solution space left. The optimal solution is then the one recorded as the "current best". For a review of the main principles of Branch and Bound algorithms as well as an illustration of the method and different design issues, see [11].

2.4 Paths in graphs

Finding paths in graphs is a basic problem in combinatorial optimization. Given a graph and a pair of nodes, we want to find one or more paths connecting these nodes ensuring certain properties on each path.

The theoretically most efficient known algorithm for the classic single-source shortest path problem with non-negative weights in the edges is the Dijkstra's algorithm [13]. Let G be a graph, s a source node, $\ell(u, v)$ the length of the edge between nodes u and v, and $d(s, v)$ denote the shortest distance from s to v in G. Dijkstra's algorithm is a labelling process, maintaining three properties:

1. *tentative distance* $d(v)$, $d(v) \geq d(s, v)$, for each node v.

2. a partition of the nodes into two sets: *labelled* and *unlabelled*. If v is a labelled node, $d(v) = d(s, v)$.

3. *tentative predecessor* $p(v)$ such that there is a path from s to v of length $d(v)$ whose last edge is $(p(v), v)$ for every $v \neq s$ with $d(v)$ finite.

Initially, $d(s) = 0$, $d(v) = \infty$ if $v \neq s$, $p(v) =$ undefined for all v and all nodes are unlabelled. Then the algorithm keeps repeating the following labelling procedure until there is no unlabelled node v with $d(v)$ finite:

Labeling procedure: choose an unlabelled node v such that $d(v)$ is minimum. Put v on the labelled set. For each edge (v, w), if $d(w) > d(v) + \ell(v, w)$, update $d(w) = d(v) + \ell(v, w)$ and set $p(w) = v$.

That way, the algorithm labels nodes in non-decreasing order by distance from s. When it terminates, $d(v) = d(s, v)$ for every v reachable from s, and the set of edges $\{(p(v), v) \mid v \neq s \text{ and } d(v) \text{ is finite}\}$ defines a shortest-path tree rooted at s. The running time of the algorithm is bounded by how the unlabelled nodes are maintained. It can be $O(|V|^2)$ for the naive case or $O(|E| + |V| \log |V|)$ if a heap is used. The procedure is outlined in Algorithm 3.

Algorithm 3 Dijkstra's algorithm outline

Input: Graph G and source node s
Output: shortest distance tree of G

```
Q ← ∅                                                    ▷ node queue
for all node n in V(G) \ {s} do
    dist[n] ← ∞
    previous[n] ← undefined
    Q ← Q ∪ {n}
dist[s] ← 0
previous[s] ← undefined
Q ← Q ∪ {s}
while Q is not empty do
    u ← node in Q with min dist[u]
    Q ← Q \ {u}
    for all v ∈ δ+(u) do
        alt ← dist[u] + ℓ(u, v)
        if alt < dist[v] then
            dist[v] ← alt
            previous[v] = u
return dist, previous
```

When negative costs are involved, other algorithms should be used instead of Dijkstra's, e.g. the Bellman-Ford-Moore algorithm. A description of these classic algorithms and their implementations appear in [17] whereas experimental evaluation of different implementations is carried out in [9].

Another interesting problem is that of finding multiple edge or node disjoint shortest paths between pairs of source and target nodes. In the edge-disjoint path problem (*EDP*), an input instance consists of a (directed or undirected) graph $G = (V, E)$ and a multi-set $\mathcal{T} = \{(s_i, t_i) : s_i, t_i \in V,\ \ i = 1, \ldots, k\}$ of k source-sink pairs of nodes. The objective is to connect the highest number of those pairs by paths which do not share edges, that is, to find a set of k edge-disjoint paths $P_1, \ldots, P_k$, where P_i is an (s_i, t_i)-path, $i = 1, \ldots, k$. EDP is a classic NP-hard problem [18] which, besides being applied to VLSI circuits, is also of great importance in combinatorial optimization and graph theory.

There are four basic versions of the problem, depending on the graph type (directed or undirected) and edge- or node-disjointness condition. In [24], it is shown that there exist polynomial time reductions among them, except for the directed to undirected case. An undirected graph can be reduced to its directed counterpart by replacing each edge with an appropriate gadget. Edge-disjoint problems can be reduced to its vertex-disjoint counterpart by replacing the graph with its line graph. Finally, directed vertex-disjoint paths reduce to directed edge-disjoint paths by replacing each vertex with a pair of new vertices connected by an edge. When k is part of the input, all four versions of the problem are NP-complete. For undirected graphs, the disjoint paths problems can be solved in polynomial time when the number k of source-sinks is fixed or when $s_1 = s_2 = \ldots = s_k$, as a special case of the maximum flow problem [24].

More recently, Kawarabayashi and Kobayashi [23] presented an $O(\log n)$-approximation algorithm for the maximum edge-disjoint paths problem, in which the objective is to find the maximum number of edge-disjoint paths between k pairs of given nodes. A c-approximation in this case is a polynomial time algorithm that finds at least $1/c$ of the maximum number of possible edge-disjoint paths. The used approach works in the special class of 4-edge-connected planar or Eulerian planar graphs.

A similar problem consists in finding vertex-disjoint paths (*VDP*). The input instance is the same as in the EDP problem but now the objective is to find paths connecting those pairs that are vertex-disjoint, that is, not two paths share a common vertex. It is a fundamental problem in routing, being also applied in the VLSI circuits context and network reliability [1, 29]. Fortune, Hopcroft, and Wyllie [16] proved that even the decision version of the VDP problem in which we want to test the existence of two vertex-disjoint paths is NP-complete in directed graphs. For undirected graphs, the decision version of the problem can be solved in $O(n^3)$ time for any fixed k number of paths, as shown by Robertson and Seymor [30], where n is the number of vertices in the

graph.

All problems above presuppose $k \geq 2$ pairs of source-sink nodes. A similar problem involves finding edge or vertex disjoint paths between the same two nodes s and t so that the sum of weights in the edges of the paths is minimum. Such paths are required for reliability in communication networks [31].

Back in 1974, Suurballe [31] presented an algorithm for the k node-disjoint paths problem between a pair of source-target nodes with minimum total length. With a minor modification, the same routine can be applied for arc-disjoint paths. The algorithm performs k iterations of a single shortest path routine (Dijkstra's). The outline of the algorithm is:

1. Run Dijkstra's to get a shortest path tree.
2. Use the cost of each edge in the tree to modify the weights of every edge in the graph to get a transformed network. The modified weight is $w'_{ij} = L_i + w_{ij} - Lj$, where L_k is the shortest distance from the origin to the node k if k is in the shortest path tree or the shortest path length from origin to destination, otherwise.
3. Rerun Dijkstra's to get a shortest path to destination.
4. Repeat steps 2 and 3 for the desired number of paths.
5. Remove overlapping edges in paths to destination from the different iterations of Dijkstra's.

For all n targets, the algorithm's runtime is bounded by $O(nm \log_{(1+m/n)} n)$ time. Later, Suurballe discovered a way to combine the Dijkstra calculations for the various targets, improving the running time for the n-targets version to $O(n^2 \log n)$ time.

Extending the work in [31], Suurballe and Tarjan [32] developed an efficient modification of the k node-disjoint paths algorithm to find a set of n pairs of edge-disjoint shortest paths between a source s and every node v in the graph. For that end, they employ a modified version of Dijkstra's to allow for negative edge weights. The algorithm runs in $O(m \log_{(1+m/n)} n)$ time and $O(m)$ space, the same bounds as in Dijkstra's algorithm when using a d-heap implementation.

1. Run Dijkstra's to get a shortest path tree.
2. Replace each edge in the path tree with a negative edge in the opposite direction
3. Rerun modified version of Dijkstra's
4. Repeat 2 and 3 once per each extra path desired

5. Remove overlapping edges from the paths generated.

In the survey [24], results for several versions of problems requiring disjoint paths on the vertices or edges are presented.

Chapter 3

Problem description

In this chapter we formally describe the k-multicolour path problem in multi-edge-coloured graphs. For the following definitions, we refer to a multi-edge-coloured graph as only a graph unless otherwise stated. When exemplifying any of the definitions, we use the graph depicted in Figure 1.1 on page 2 assuming unit edge weights, therefore omitting them. The labels in the edges represent the colours available.

An instance of the problem consists of a graph $G = (V, E)$, where V is the set of nodes, E is the set of arcs, and a set of x colours $C = \{1, \ldots, x\}$, $x \in \mathbb{N}$. Let $c : E \mapsto 2^C$ be a function mapping colours to the arcs of the graph, where 2^C is the power set of C. So, $c(i, j)$ is the set of colours associated with arc (i, j). Each arc (i, j) has a non-negative weight $w(i, j)$.

Let P be a path between two nodes of G and $E(P)$ the edge set of P. The colour set of the path P, denoted by $C(P)$, is defined as $C(P) = \bigcap_{e \in E(P)} c(e)$. We say P is a k-multicolour path, or a k-path for short, if $|C(P)| \geq k$; equivalently, we say P contains/uses k colours.

For the following definitions, consider two positive integers, k_i and k_j.

Definition 3.0.1 (**Feasible path**)**.** A path P_i in G is feasible with respect to k_i if and only if it contains at least k_i colours, i.e., $|C(P_i)| \geq k_i$.

In Figure 3.1, the path highlighted $P = (s, v_2, t)$ is feasible when $k \leq 2$ but it is not feasible for $k > 2$ since $C(P) = \{2, 3\}$.

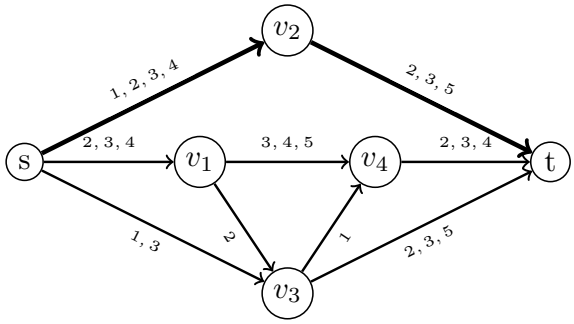


Figure 3.1: Feasible path in a graph

Definition 3.0.2 (**Compatible paths**)**.** Two feasible paths P_i and P_j in G, with respect to k_i and k_j respectively, are compatible in one of two cases:

(1) they are mutually edge-disjoint, or (2) they contain different colours, i.e., $C(P_i) \cap C(P_j) = \emptyset$.

In Figure 3.2, the path with thicker edges $P_1 = (s, v_1, v_3, t)$ has $C(P_1) = \{2\}$ and the path with dotted edges $P_2 = (s, v_1, v_4, t)$ has $C(P_2) = \{3, 4\}$. Even though they share the edge (s, v_1), they use different colours, therefore they are compatible.

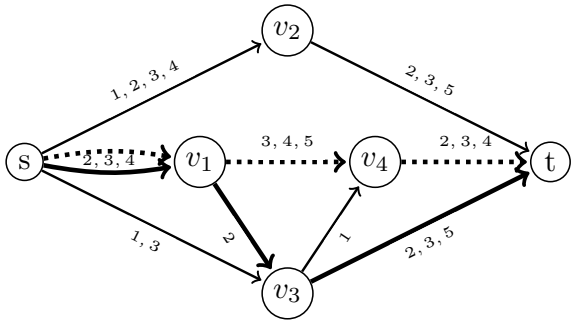


Figure 3.2: Compatible paths in a graph

Definition 3.0.3 (**Absolutely compatible paths**)**.** Two feasible paths P_i and P_j in G, with respect to k_i and k_j respectively, are absolutely compatible if and only if they are mutually arc-disjoint.

The highlighted paths in Figure 3.3, $P_1 = (s, v_2, t)$ and $P_2 = (s, v_3, t)$ are absolutely compatible even though both use colour 3, since they do not share any edges.

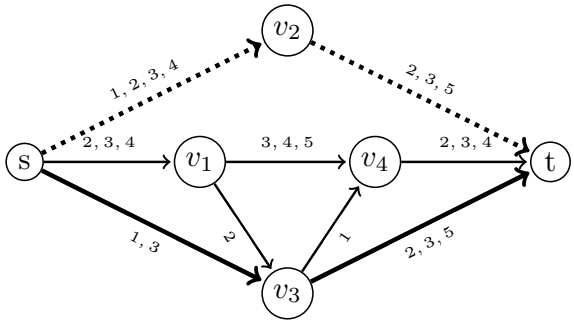


Figure 3.3: Absolutely compatible paths in a graph

We are now ready to formally state our problems.

The input consists of a graph $G = (V, E)$, a set of colours C and a tuple (s, t, k, p), where s is the source node, t is the target or destination node, k is the number of required colours, and p is the number of required paths. For the following definitions, a *shortest path* is shortest with respect to the arc weights. We assume that the colour requirement is $k \geq 2$, since the case $k = 1$ can be solved in polynomial time by the application of a shortest path procedure $|C|$ times, one for each colour, and then picking the shortest of the (s, t)-paths found. Because it makes no sense for paths not to have any colours assigned, we also assume $k \geq p$.

Definition 3.0.4 (**Single k-Multicolour Path Problem (SMP)**)**.** The input to this problem is G, C and a tuple $(s, t, k, 1)$. The objective is to find a single feasible shortest (s, t)-path containing at least k colours.

Considering the graph from Figure 1.1 on page 2, the solution to the input $(s, t, 2, 1)$ would be the path $P = (s, v_2, t)$ (highlighted in Figure 3.1). Although the path $P' = (s, v_1, v_4, t)$ is feasible with respect $k = 2$, P is the only shortest feasible path. For $k > 2$, there is no solution possible for that graph.

Definition 3.0.5 (**Multiple k-Multicolour Paths Problem (MMP)**)**.** The objective is to find p compatible shortest (s, t)-paths, $P_1, \ldots, P_p$, so that the sum of colours used by all paths is at least k, i.e., $\sum_{i=1}^{p} |C(P_i)| \geq k$.

In Figure 1.1, the paths $P_1 = (s, v_2, t)$ and $P_2 = (s, v_1, v_4, t)$ would be the solution for the input $(s, t, 4, 2)$. P_1 and P_2 are clearly compatible because they are arc-disjoint.

Definition 3.0.6 (**Absolute Multiple k-Multicolour Paths Problem (AMMP)**)**.** The objective is to find p absolutely compatible shortest (s, t)-paths containing at least k colours.

Referring to Figure 1.1 again, it is easy to see that there is no solution for an input instance $(s, t, 6, 4)$, since the fourth path would invariably share an edge with another one.

In the decision version of those problems we ignore the arc weights and the objective is to find only a single/multiple feasible/compatible (s, t)- path/paths containing the required k colours. The decision version is represented by a subscripted 'd' on the problem's name ($\mathrm{SMP_d}$, $\mathrm{MMP_d}$, and $\mathrm{AMMP_d}$).

Chapter 4

NP-hardness proof

In this chapter we prove that the SMP_d, MMP_d, and AMMP_d problems are NP-complete. As a consequence, the optimization versions are NP-hard.

4.1 Single path decision problem

Theorem 1. *The SMP_d problem is NP-complete.*

Proof. The problem is clearly in NP since we can check in linear time that a given path is a (s,t)-path and uses k colours.

We present now a reduction from the 3CNF-SAT [22] to the SMP_d problem. Let I be an instance of the 3CNF-SAT problem consisting of a logical formula over a set of n variables $\mathcal{X} = \{x_1, \ldots, x_n\}$ and containing m clauses $\mathcal{K} = \{K_1, \ldots, K_m\}$, where each clause K_j contains exactly three literals $K_j = (y_{j1} \vee y_{j2} \vee y_{j3})$. In the 3CNF-SAT problem, we need to decide if there is an assignment of truth values to the variables in $\mathcal{X}$ such that $\wedge_{j=1}^{m} K_j$ is true.

From I, we build an instance $(G, C, s, t, n, 1)$ to the SMP problem such that there is a feasible (s,t)-path P in G with n colours if and only if there is a truth assignment satisfying I.

The graph G consists of $n+m$ structures, one for each variable x_i, $1 \leq i \leq n$ and one for each clause K_j, $1 \leq j \leq m$. A structure representing a variable $x_i \in \mathcal{X}$ is depicted in Figure 4.1 on the following page and a structure representing a clause K_j is depicted in Figure 4.2 on the next page.

We create colours c_{z_i} for each literal $z_i \in \mathcal{Z} = \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. So $C = \{c_{x_1}, c_{\bar{x}_1}, \ldots, c_{x_n}, c_{\bar{x}_n}\}$ and $|C| = 2n$. There are two types of edges in the graph: (1) edges of the structure that have labels in the figure, such as e_{z_i}, containing all colours $\{c_{z_j} \mid z_j \neq \bar{z}_i \text{ and } z_j \in \mathcal{Z}\}$, that is, it contains all colours except that which represents the negation of z_i, and (2) unlabelled edges containing all $2n$ colours.

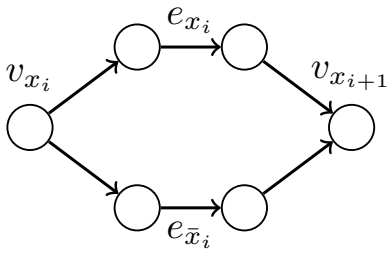


Figure 4.1: Structure for variable x_i

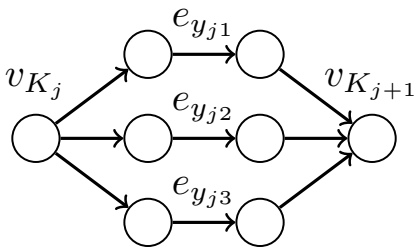


Figure 4.2: Structure for clause K_j

The meaning of a structure for a variable x_i is this: a path may use exactly one of the two edges e_{z_i} or $e_{\bar{z}_i}$. The edge used determines which of the literals z_i or $\bar{z}_i$ is going to be true. The structure for a clause $K_j = (y_{j1} \vee y_{j2} \vee y_{j3})$ has the following meaning: a path is going to use exactly one of the three edges $e_{y_{j1}}$, $e_{y_{j2}}$ or $e_{y_{j3}}$ representing a literal that is true and satisfying the clause.

The complete graph is created connecting the structures of the variables, one to another in sequence, where we set $s = v_1$, and then connecting the structures of the clauses, where the last vertex is t. It is clear that the whole transformation takes $O(n+m)$ time. The complete graph is depicted in Figure 4.3.

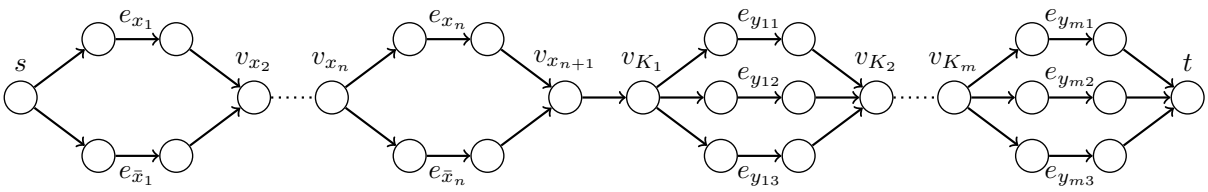


Figure 4.3: Graph built from a 3CNF-SAT instance

We show that there exists a valid assignment satisfying the formula of I if and only if there is a feasible (s,t)-path in G with n colours.

Let $A : \mathcal{X} \mapsto \{T, F\}$ be a truth assignment satisfying the formula of I. In G, the path P will use the colours whose respective literals have a truth assignment. We construct P as follows: starting from s, for each structure associated with a variable x_i, if $A(x_i) = T$, then P uses the subpath that goes through edge e_{x_i} (colour c_{x_i} is used); otherwise, P will use the subpath going over $e_{\bar{x}_i}$ (colour $c_{\bar{x}_i}$ is used). Since A is a valid assignment, it is not possible for both e_{x_i} and $e_{\bar{x}_i}$ to be used at the same time in P neither the colours c_{x_i} and $c_{\bar{x}_i}$. At the end of the variable graph structures, the path will be using exactly n colours, one for each truth literal. Since A is a truth assignment, at least one of the literals of K_j has a truth assignment, therefore, we can complete path P by picking the subpath in the K_j's structure whose edge has the least index and corresponds to a literal with truth assignment.

Now let P be a feasible (s,t)-path using n colours in G. First notice that any (s,t)-path cannot contain both edges e_{x_i} and $e_{\bar{x}_i}$ because of the bifurcation in the structure corresponding to the variable x_i. So it is not possible to use both colours c_{x_i} and $c_{\bar{x}_i}$ at the same time. At the end of the variable structures (at node $v_{x_{n+1}}$), P must have chosen exactly n colours: if P uses edge e_{x_i} it must use colour c_{x_i}, since colour $c_{\bar{x}_i}$ is not available in e_{x_i}; similarly, if P uses edge $e_{\bar{x}_i}$ it must use colour $c_{\bar{x}_i}$. The assignment A is obtained in the following manner: for each variable x_i, we make $x_i = T$ if e_{x_i} is used in P and $x_i = F$ if $e_{\bar{x}_i}$ is used in P.

To prove that A satisfies the formula of I, we show that every clause $K_j = (y_{j1} \vee y_{j2} \vee y_{j3})$ is satisfied. Some edge $e_{y_{ji}}$ is used in P for some $i \in \{1,2,3\}$, since P is feasible. This means that P must use the colour corresponding to the literal y_{ji}. By the definition of A, $y_{ji} = T$ then K_j is satisfied. □

4.2 Multiple paths decision problems

Even if the colours requirement can be split into multiple paths, i.e., $p > 1$ and $\sum_{i=1}^{p} |C(P_i)| \geq k$, we prove that the problem still remains NP-complete.

Theorem 2. *The MMP_d problem is also NP-complete.*

Proof. We reduce the Set Cover problem [22] to the $\mathrm{MMP_d}$ problem. Let I be an instance of the Set Cover consisting of a set of n elements $\mathcal{U} = \{u_1, \ldots, u_n\}$, a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of $\mathcal{U}$, and a positive integer $K \leq |\mathcal{S}|$. The problem is to decide if there are at most K subsets from $\mathcal{S}$ whose union equals $\mathcal{U}$. We construct an instance (G, C, s, t, n, K) to the $\mathrm{MMP_d}$ problem, such that there exists a solution to I if and only if there are K compatible (s,t)-paths in G using n colours.

Firstly, we create nodes s and t. There will exist n colours $C = \{c_{u_1}, \ldots, c_{u_n}\}$ each one representing one element from $\mathcal{U}$. There will also exist edges e_{S_i} representing each subset $S_i \in \mathcal{S}$. These edges contain the colours $\{c_{u_i} \mid u_i \in S_i\}$. The remaining unlabelled edges contain all the n colours. The number of required compatible paths is $p = K$ and the number of required colours is n. We remark that this procedure can be done in $O(m)$ time. The complete graph is depicted in Figure 4.4 on the next page.

We show that there is a set cover $\mathcal{S}' \subseteq \mathcal{S}$ for $\mathcal{U}$ such that $|\mathcal{S}'| \leq K$ if and only if there are K compatible (s,t)-paths in G containing n colours.

Let $P_1, \ldots, P_K$ be K compatible (s,t)-paths on G using n colours. If the path P_i contains the edge e_{S_i}, then the set S_i is used as part of the solution of the Set Cover, i.e., $\mathcal{S}' = \mathcal{S}' \cup S_i$, $1 \leq i \leq K$. Since there are exactly K paths, $|\mathcal{S}'| = K$. Furthermore, we know that the K paths are compatible and use n colours. Therefore, $\mathcal{S}'$ contains all the elements in $\mathcal{U}$, and it is a valid solution to the Set Cover problem.

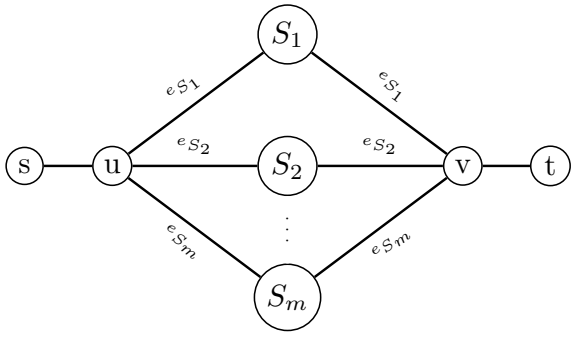


Figure 4.4: Graph built from an instance of the Set Cover problem

Now assume $\mathcal{S}' = \{S_1, \ldots, S_K\}$ is a solution for an instance I of the Set Cover (we can always assume that exactly K subsets are used since we can add subsets to a valid solution, and it remains valid). We construct K (s,t)-paths, $P_1, \ldots, P_K$ in G that use n colours as follows. All K (s,t)-paths will use the unlabelled edges. Each (s,t)-path P_i corresponds to some $S_i \in \mathcal{S}'$. Path P_i uses the edges e_{S_i} corresponding to a S_i and, consequently, uses colours c_{u_j} such that $u_j \in S_i$ as long as c_{u_j} is not already being used by another path. We need to prove that these (s,t)-paths are compatible. It is easy to see that each path P_i is feasible, for it uses colours $c_{x_1}, \ldots, c_{x_j}$ on the edges e_{S_i} and those colours are also present in the edges without labels. Moreover, any two paths are compatible because, by definition, each path uses only those colours not already being used by any other path. We know $P_1, \ldots, P_K$ use n colours because $\mathcal{S}'$ is a solution for the Set Cover thus it "covers" all the elements in $\mathcal{U}$, implying that we have K (s,t)-paths using n colours.

□

We also show that the AMMP_d problem, which requires absolutely compatible paths, is NP-complete.

Theorem 3. *The $AMMP_d$ problem is NP-complete.*

Proof. We can reduce the 3CNF-SAT to this problem: let G_i be the graph obtained from an instance of the 3CNF-SAT like in the reduction for the SMP_d problem. Consider $p > 1$ copies of this graph, each one with new colours (although still representing the same literals). In this way, there are $2np$ colours. Create nodes s' and t', edges e_i and e'_i, for $i = 1, \ldots, p$, containing all the colours and make s' adjacent to s of G_i through the edge e_i and t' adjacent to t of G_i through e'_i, as illustrated in Figure 4.5 on the following page.

Finally, let the required number of colours to the AMMP_d problem be np. It is not hard to see that a formula to the 3CNF-SAT can be satisfied if and only if there are p

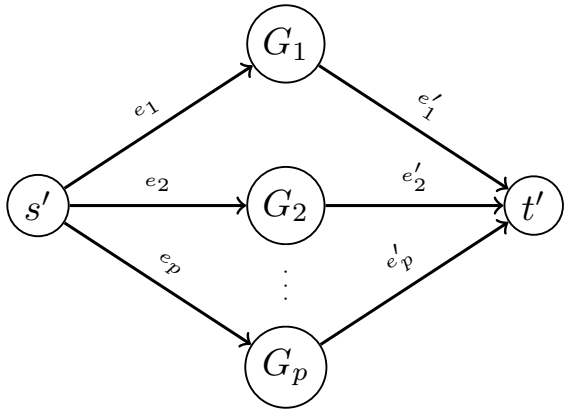


Figure 4.5: Reduction for absolutely compatible paths

absolutely compatible paths in G using exactly np colours. □

4.3 Single and Multipath optimization problems

The original optimization versions of the problems SMP, MMP, and AMMP assume weighted edges and the goal is to find multicolour paths of a minimum total weight. As a corollary of the previous theorems, the optimization versions of the SMP, MMP, and AMMP problems do not admit approximation algorithms, since we could use such algorithms to decide, in polynomial time, the decision version of theses problems.

Corollary 4. *There is no α-approximation algorithm for the SMP, MMP, or AMMP problems unless $P = NP$.*

Chapter 5

Exact algorithms

We now introduce two types of exact algorithms: Branch and Bound and Integer Linear Program (ILP). We present Branch and Bound algorithms for the SMP and AMMP problems in Section 5.1. ILP formulations for SMP, MMP, and AMMP are presented in Section 5.2.

5.1 Branch and Bound

As discussed on Section 2.3, Branch and Bound algorithms consist in a systematic enumeration of all candidate solutions, where by the employment of lower and upper bounds on the quantities being optimised one can discard big sets of unfruitful candidates and explore promising potential solutions. In this section we present Branch and Bound solutions for the SMP (Section 5.1.1) and AMMP (Section 5.1.2) problems.

5.1.1 Branch and Bound for SMP

For the description of our branch and bound solution, we need the concept of a *partial path* and a *potential solution* (PS).

Partial path (P_s^l): is a path starting at node s in which l is the last node in the path. We call it *partial* because it has not yet reached the destination t. The common colours in a partial path will be denoted by $c(P_s^l)$. If $|c(P_s^l)| = k$, P_s^l is called a partial k-path.

Potential solution (PS): is a partial k-path starting at node s. The weight of a potential solution, $w(PS)$, is the sum of weights of the edges in the partial path, $w(PS) = \sum_{e \in P_s^l} w(e)$. The common colours of a PS are the common colours in its partial path, $c(PS) = c(P_s^l)$.

In our algorithm, at any given point in time, the search tree is composed of a set of potential solutions. At each iteration, we choose and expand the most promising potential solution. Let P_s^l be a partial path in this promising potential solution set. We expand it by considering the node l and checking, for each out-neighbour u: (1) whether the number of common colours including the neighbour to the partial path is still at least k, i.e., $|c(P_s^l) \cap c(l,u)| \geq k$, and (2) whether the destination node t is reachable from u.

If any of these conditions is not met, then the node u is discarded; otherwise, a new potential solution with the neighbour node u as last node in the partial path P_s^u is added to the search tree.

The starting potential solution, is comprised of a partial "path" P_s^s with only the node s, containing all colours $c(P_s^s) = C$, with weight $w(P_s^s) = 0$ and estimated weight $d(s,t)$ (to be defined below).

When choosing a promising PS to evaluate, we use a best bound strategy: we select the PS with least *estimated weight* to the destination. The estimated weight is the sum of weights of the edges in the PS plus a lower bound in the weight of the path to the destination: the weight of the shortest path from the last node of the partial path to the destination, without considering the colour requirement.

Estimated weight of a PS: $w(P_s^l) + d(l,t)$, where $d(l,t)$ is the shortest distance between l and t disregarding the colours.

Let G_{rev} be the graph obtained from $G = (V,E)$ by reversing the direction of all arcs. In our algorithm, we get the shortest distance $d(i,t)$ between nodes i and $t, \forall i \in V$, from a shortest path tree rooted in t obtained from running the Dijkstra's shortest path algorithm on G_{rev}.

Notice that when evaluating a partial path, if it contains the destination as last node, then this path corresponds to the optimal solution, since we are using a best-bound strategy. The entire procedure is outlined in Algorithm 4 on the next page.

Algorithm 4 Branch and Bound algorithm for SMP

Input: instance I of the SMP problem
Output: k-path P or $\emptyset$ if none is found

PQ = priority queue of PSs with estimated weight as sorting key
Run Dijkstra's on G_{rev} with root node t
PQ $\leftarrow$ PQ $\cup \{P_s^s\}$ ▷ Insert starting PS into PQ
while PQ is not empty **do**
  PS_i = PS from PQ with least estimated weight
  PQ $\leftarrow$ PQ $\setminus \{\text{PS}_\text{i}\}$
  Let P_s^l be partial path associated with PS_i.
  if node l is t **then** ▷ Solution found
    return P_s^l
  for all $u \in \delta^+(l)$ **do**
    $c'(P_s^l) \leftarrow c(P_s^l) \cap c(l,u)$ ▷ Tentative common colours
    if $|c'(P_s^l)| \geq k$ **And** t is reachable from u **then**
      $P_s^u \leftarrow P_s^l \cup (l,u)$
      $w(P_s^u) = w(P_s^l) + w(l,u)$
      $c(P_s^u) \leftarrow c'(P_s^l)$
      Let PS_new have P_s^u with estimated weight $w(P_s^u) + d(u,t)$.
      PQ $\leftarrow$ PQ $\cup \{\text{PS}_\text{new}\}$ ▷ Add PS_new to PQ
return $\emptyset$ ▷ No solution found

5.1.2 Branch and Bound for AMMP

Define a pair of potential solutions PS_i and PS_j to be arc-disjoint if and only if their partial paths are mutually arc-disjoint. A set of potential solutions is arc-disjoint if and only if their partial paths are pairwise mutually arc-disjoint. Then the same algorithmic idea in the SMP problem can be expanded to the AMMP problem. Now a *multi-potential solution* (MPS) consists of a set of arc-disjoint PSs. The weight of a MPS is the sum of weights of the PSs, the estimated weight is the sum of estimated weights and the colour set is the union of the colours in the PSs.

The starting potential solution, MPS_0, is comprised of p copies of P_s^s. At each iteration of the algorithm, we expand one partial path of the most promising MPS, repeating this procedure until all partial paths in a MPS reach the destination node or no MPS is left to evaluate. The entire algorithm is outlined in Algorithm 5 on the following page.

Algorithm 5 Branch and Bound algorithm for AMMP

Input: instance I of the AMMP problem

Output: p absolutely compatible paths or $\emptyset$ if no solution found

```
PQ = priority queue of MPSs with estimated weight as sorting key
Run Dijkstra's on G_rev with root node t
PQ ← PQ ∪ {MPS_0}                                   ▷ Insert starting MPS to PQ
while PQ is not empty do
    MPS_i = MPS from PQ with least estimated weight
    PQ ← PQ \ {MPS_i}
    if node l is t for every partial path P_s^l in MPS_i then
        return paths in MPS_i
    for all PS_j in MPS_i do
        Let P_s^l be the partial path in PS_j
        if l is t then                              ▷ This partial path reached the target
            Ignore PS_j
        else
            for all u ∈ δ+(l) do
                if t is reachable from u And arc (l, u) is not in MPS_i then
                    c'(P_s^l) ← c(P_s^l) ∩ c(l, u)                ▷ Tentative common colours
                    total ← |c'(P_s^l)| + Σ_{PS_k ∈ MPS_i \ {PS_j}} |c(PS_k)|
                    if total ≥ k then
                        Create a copy MPS'_i of MPS_i
                        Update PS_j in MPS'_i
                        Add updated MPS'_i to PQ
return ∅                                            ▷ No solution found
```

5.1.3 Branch and Bound for MMP

For this problem, there was no Branch and Bound implementation because of the exponential growth of the search tree coupled with the difficulty of keeping track of colours in each path so that the final solution is feasible.

5.2 Integer Linear Programs

In this section we present two Integer Linear Programming (ILP) models for the SMP problem and one for the MMP and AMMP problems.

With respect to the formulations, consider the following variables:

f_{ij}**:** binary variable indicating the usage of edge (i, j).

f_{ijx}**:** binary variable indicating the usage of colour x on the edge (i, j).

f_{pij}**:** binary variable indicating the presence of the edge (i, j) in the path p.

f_{pijx}**:** binary variable indicating the usage of colour x on the edge (i, j) of path p.

c_x**:** binary variable indicating the usage of colour $x \in C$.

c_{px}**:** binary variable indicating if colour x is used by path p in the solution.

k**:** constant that represents the number of required colours.

w_{ij}**:** constant that represents the weight of the edge (i, j).

All the variables above are subject to:

$$f_{ij}, f_{ijx}, f_{pij}, f_{pijx} \in \{0, 1\}, c_x \in \{0, 1\}, c_{px} \in \{0, 1\}, x \in \{1, 2, \ldots, C\}, ij \in E$$

In Equations ILP.1, ILP.2, ILP.3, ILP.4 the objectives are to find path(s) between source and target nodes with minimum total weight. Equations (5.1), (5.4), (5.7), and (5.12) are flow conservation constraints. Equations (5.2), (5.6), (5.8), and (5.13) ensure the colours requirement is satisfied. Equations (5.3), (5.5), (5.9), and (5.14) ensure the paths are suitable.

5.2.1 ILP models for SMP

We present next two formulations for the SMP problem: one that can be easily extended for more than one path and a more compact (and faster) one.

Original ILP Model

This single path formulation was presented in [8] and we replicate it here for easy of comparison with our own formulation (ILP.2) to be introduced in the following.

(ILP.1)

$$\min \sum_{ijx} f_{ijx} \cdot w_{ij}$$

$$\text{subject to} \quad \sum_{j \in \delta^+(i)} f_{ijx} - \sum_{j \in \delta^-(i)} f_{jix} = \begin{cases} -c_x, & \text{if } i = \text{source} \\ c_x, & \text{if } i = \text{target}, \\ 0, & \text{otherwise} \end{cases} \quad \forall i, x \tag{5.1}$$

$$\sum_{x=1}^{C} c_x = k \tag{5.2}$$

$$\sum_{x=1}^{C} f_{ijx} = \{0, k\} \quad \forall (i,j) \tag{5.3}$$

Faster ILP model

Whereas the formulation ILP.1 was constructed so that it was extendible to the multipath case, we introduce here some modifications resulting in a faster model with a decreased number of total variables. We achieve that by decoupling the choice for arcs in the paths from the choice of colours.

(ILP.2)

$$\min \sum_{ij} f_{ij} \cdot w_{ij}$$

$$\text{subject to} \quad \sum_{j \in \delta^+(i)} f_{ij} - \sum_{j \in \delta^-(i)} f_{ji} = \begin{cases} -1, & \text{if } i = \text{source} \\ 1, & \text{if } i = \text{target}, \\ 0, & \text{otherwise} \end{cases} \quad \forall i \tag{5.4}$$

$$\sum_{x \in c(i,j)} c_x \geq f_{ij} \cdot k \quad \forall (i,j) \tag{5.5}$$

$$\sum_{x=1}^{C} c_x = k \tag{5.6}$$

5.2.2 ILP models for MMP and AMMP

Now we present formulation for the multipath versions, both for the compatible paths case and the absolutely compatible paths. The objective now is to find P paths with minimum total weight satisfying the k colours requirement.

ILP model for MMP

This ILP formulation for the MMP problem was presented in [8] and is reproduced here adapted to our own notation. Equation (5.10) ensures at least one colour is used in each path, and Equation (5.11) ensures the paths are compatible.

(ILP.3)

$$\min \sum_{pijx} f_{pijx} \cdot w_{ij}$$

$$\text{subject to} \quad \sum_{j \in \delta^+(i)} f_{pijx} - \sum_{j \in \delta^-(i)} f_{pjix} = \begin{cases} -c_{px}, & \text{if } i = \text{source} \\ c_{px}, & \text{if } i = \text{target} \\ 0, & \text{otherwise} \end{cases} \quad \forall i, p, x \tag{5.7}$$

$$\sum_{px} c_{px} = k \tag{5.8}$$

$$\sum_{x \in c(i,j)} f_{pijx} = \left\{ 0, \sum_{x} c_{px} \right\} \quad \forall (i,j), p \tag{5.9}$$

$$\sum_{x=1}^{C} c_{px} \geq 1 \quad \forall p \tag{5.10}$$

$$\sum_{p=1}^{P} f_{pijx} \leq 1 \quad \forall (i,j), x \tag{5.11}$$

ILP model for AMMP

By modifying ILP.3, we can extend it for the absolutely compatible paths case. In the formulation ILP.4, Equation (5.15) ensures at least one colour is used in each path, and Equation (5.16) ensures the paths are disjoint.

(ILP.4)

$$\min \sum_{pijx} f_{pijx} \cdot w_{ij}$$

$$\text{subject to} \quad \sum_{j \in \delta^+(i)} f_{pijx} - \sum_{j \in \delta^-(i)} f_{pjix} = \begin{cases} -c_x, & \text{if } i = \text{source} \\ c_x, & \text{if } i = \text{target} \\ 0, & \text{otherwise} \end{cases} \qquad \forall i, p, x \tag{5.12}$$

$$\sum_{px} c_{px} = k \tag{5.13}$$

$$\sum_{x \in c(i,j)} f_{pijx} = \left\{ 0, \sum_{x} c_{px} \right\} \qquad \forall (i,j), p \tag{5.14}$$

$$\sum_{x=1}^{C} c_{px} \geq 1 \qquad \forall p \tag{5.15}$$

$$\sum_{p=1}^{P} f_{pij} \leq 1 \qquad \forall (i,j) \tag{5.16}$$

Chapter 6

Heuristics

The heuristics developed are divided into two algorithmic ideas: Dijkstra-based and graph intersection-based. In the following sections, we outline each one of them and analyse their running time.

6.1 Dijkstra-based for SMP

We propose three heuristics for the SMP problem that are based on the Dijkstra's algorithm which was discussed in Section 2.4. The heuristics inherit Dijkstra's main characteristic: at each step, a node with minimum value according to some criterion is chosen, from which the search is expanded further towards the destination node. Nodes already chosen by the Dijkstra's algorithm have their minimum path discovered and are not visited any more. Each node not yet visited has an estimated value for the minimum path between the source and itself. The main difference introduced by our heuristics is with respect to how a node is chosen. Besides the estimated cost between source and each node, we take into account the number of available colours when including a node as part of some path. The heuristics keep track of the number of available colours through the path found so far. For a new node to be part of the path, the number of colours in the resulting path needs to be at least k.

The heuristics use the same structure depicted in the pseudocode Algorithm 3 on page 13. From the distinct ways on how to select a node, three heuristics arise: DijkstraQ, DijkstraT and DijkstraX. In what follows, we describe each one of these in details.

6.1.1 DijkstraQ heuristic

The DijkstraQ heuristic separates the unvisited nodes into "quadrants" between two axes. One of these axes considers the estimated distance between the source and the unvisited

nodes. The other corresponds to the number of common colours between a path and the edge linking that path to a node. We compute the average estimated distance (d_{avg}) and the average number of common colours (c_{avg}) considering all unvisited nodes for which a distance label was set. We are interested in the nodes with estimated distance shorter than the average and with number of common colours higher than the average. The algorithm chooses randomly one of the nodes satisfying this criterion. The heuristic choice in this case aims to find a shortest path while keeping a high number of common colours, in the hope that, upon finishing, there will be at least k common colours in the path. Because the step for finding the next node to be visited involves a linear search, the time complexity is $O(|V|^2)$. See Algorithm 6 on page 39 for details.

6.1.2 DijkstraT heuristic

The DijkstraT heuristic, in turn, makes use of a positive real parameter T. At each iteration, we choose the node with minimum estimated distance among those that have at least $\lfloor T \cdot k \rfloor$ common colours in the path from the source node. We then update T by decreasing its value by some constant amount. Roughly speaking, we choose nodes according to the shortest distance and a linear decreasing function in the number of common colours. The heuristic choice in this case is that we need more colours in common in the beginning, because of the higher number of possible nodes to choose from, but this requirement loosens as we approach the destination node. The algorithm is outlined in Algorithm 7 on page 40. As we can see from the outlined algorithm, the time complexity is the same as Dijkstra's, $O(|E| + |V| \log |V|)$. In our experiments, $T = 1.5$ and it is decremented by T/n, where n is the number of nodes in the graph, for each iteration of the algorithm.

6.1.3 DijkstraX heuristic

The problem with the previous DijkstraQ heuristic is that selecting a node to visit is not as efficient as using a heap. However, we can assign a *score* to the nodes. This score would take into account the distance from the source and the number of common colours. This way, the selection of the minimum node to visit can be made more efficiently than before. Besides, the score function can be adapted to give more weight to shortest paths or paths with more colours in common. The score function used in our algorithm is simply $\text{score}(i) = \text{dist}(i) - D \cdot c(i)$, where $\text{dist}(i)$ is the shortest distance from the source to node i, $c(i)$ is the set of common colours in the shortest path from the source to i found so far, and D is the longest shortest distance from the source to any other node. The role of the parameter D is to balance the score so that distance and number of colours are evenly accounted for. The time complexity of the DijkstraX algorithm is the same

as Dijkstra's: $O(|E| + |V| \log |V|)$. The entire procedure is outlined in Algorithm 8 on page 41.

6.2 MMPMin: Dijkstra-based heuristic for MMP and AMMP

Denote by p the number of paths to be found by the algorithm and by k the colour requirement. Denote by n_p the number of paths found so far, $n_p \leq p$. The algorithmic idea for MMPMin, a Dijkstra-based algorithm for the MMP and AMMP problems, is to repeat the following procedure for the number of paths desired:

1. compute a shortest (s, t)-path P;
2. select up to $n_k = k - p + n_p + 1$ colours in $C(P)$;
3. add P to the solution;
4. increment n_p by 1;
5. $k \leftarrow k - n_k$;
6. modify the graph G.

Step 2 above is necessary since each of the p paths must use at least one colour. The type of graph modification on step 6 depends on the problem we are solving. Let P be the shortest path found in the current iteration of the algorithm. For the MMP, we modify the graph by removing all the colours selected for P on its arcs in G. For the AMMP, because we want arc-disjoint paths, we remove all arcs in P from G. The time complexity for both algorithms is $O(p\{|E| + |V| \log |V|\})$. Both procedures are outlined in Algorithm 9 on the next page.

Algorithm 9 MMPMin heuristic for MMP and AMMP problems

Input: Instance I of MMP or AMMP

Output: p compatible paths using k colours or $\emptyset$ otherwise

```
k' = k
for i from 1 to p do
    P_i ← DIJKSTRA(G, s, t)
    if P_i does not exist then
        return ∅
    Select C'(P) ⊆ C(P) such that |C'(P)| ≤ k' − p + i + 1
    if MMP problem then
        Remove C'(P) from the arcs in E(P) from G
    else
        Remove E(P) from G
    k' ← k' − |C'(P)|
if Σ_{i=1}^{p} |c(P_i)| ≥ k then
    return P_1, ..., P_p
else
    return ∅
```

6.3 Intersection-based heuristic for SMP

Let E_i denote the set of edges in G that contains the colour i. Denote the set of all E_i's, $1 \leq i \leq x$, by $\mathcal{E} = \{E_1, E_2, \ldots, E_x\}$. The graph $G_i = (V, E_i)$ is the subgraph of G in which all edges contain the colour i. We say G_i is the subgraph of G induced by colour i. Let $\binom{\mathcal{E}}{k}$ denote the set of all k-combinations of $\mathcal{E}$. Then if $\mathcal{E}_k = \{E_{j1}, E_{j2}, \ldots, E_{jk}\}$, is a random selection of a set in $\binom{\mathcal{E}}{k}$, we denote by $G_k^* = (V, E_k^*)$, $E_k^* = \bigcap_{E_{jl} \in \mathcal{E}_k} E_{jl}$, the graph resulting from the intersection of the subgraphs $G_{jl} = (V, E_{jl})$, $\forall E_{jl} \in \mathcal{E}_k$. If there is a (s,t)-path in G_k^*, it is easy to see that it is also a k-path because every edge in G_k^* has exactly the same k colours. So all we need is to increase the odds that there will be a (s,t)-path in G_k^*. For that end, we could pick $\mathcal{E}_k$ so that the intersection of its elements results on the highest number of edges in G_k^*. In other words, we want to find an $\mathcal{E}_k$ such that $|\bigcap_{E_j \in \mathcal{E}_k} E_j| \geq |\bigcap_{E_l \in \mathcal{E}_{k'}} E_l|$, $\forall \mathcal{E}_{k'} \in \binom{\mathcal{E}}{k}$. However, because finding such $\mathcal{E}_k$ is, by itself, a hard problem [35], we have to employ a heuristic strategy instead. In this case, we sort E_i's by decreasing cardinality and pick the first k with highest number of edges. We call this algorithm IntersectionFast. We could further restrict the heuristic choice by

picking only those E_i's with highest cardinality as long as they contain a (s,t)-path (the Intersection algorithm). In Algorithm 10 we exemplify both procedures.

Algorithm 10 Heuristic based on graph intersection

Input: Instance I of the SMP problem
Output: feasible path P or $\emptyset$ if none is found
  for all i from 1 to x **do** ▷ Drop this step for IntersectionFast algorithm
    Let $G_i = (V, E_i)$ be the subgraph induced by colour i
    Discard E_i if G_i does not contain a (s,t)-path
  Let $\mathcal{E}_k = \{E_{j1}, \ldots, E_{jk}\}$ be the set of k E_i's with highest number of edges
  Let $E_k^* = \bigcap_{E_{jl} \in \mathcal{E}_k} E_{jl}$ be the intersection of the edge sets in $\mathcal{E}_k$.
  $G_k^* = (V, E_k^*)$ is the graph induced by the intersection
  Find a shortest (s,t)-path P in G_k^*
  if P can be found **then**
    return P
  else
    return $\emptyset$

For simplicity, denote $c = |C|$, $n = |V|$, and $m = |E|$. Performance-wise, the IntersectionFast algorithm has time complexity $O(cm + c\log c + m + m + n\log n)$. If we assume $c < m$, that can be simplified to $O(cm + n\log n)$. For the Intersection algorithm, the time complexity is $O(cm + c\{n+m\} + c\log c + m + m + n\log n)$, which can be simplified to $O(c\{n+m\} + n\log n)$ if $c < m$.

Algorithm 6 DijkstraQ heuristic

Input: Instance I of SMP
Output: k-path P or $\emptyset$ if none is found

```
for all node u in V(G) \ {s} do                          ▷ Initialization
    dist[u] ← ∞
    previous[u] ← undefined
    discovered[u] ← False
    visited[u] ← False
dist[s] ← 0
common[s] ← C
discovered[s] ← True
while there are nodes for which visited is False do
    Calculate d_avg for the discovered nodes             ▷ Average distance
    Calculate c_avg for the discovered nodes             ▷ Average colours
    u ← GetMinNode(d_avg, c_avg, discovered \ visited)
    visited[u] ← True
    if u is t then                                       ▷ Solution found
        return path rebuilt from previous
    for all v ∈ δ+(u) do
        c' ← c(u, v) ∩ common[u]
        if |c'| < k then                                 ▷ Not enough common colours
            Ignore node v
        else
            alt ← dist[u] + w(u, v)
            if alt < dist[v] then
                dist[v] ← alt
                common[v] ← c'
                discovered[v] ← True
                previous[v] ← u
return ∅

function GetMinNode(d_avg, c_avg, node_set[])
    for all node u in node_set do
        if dist[u] < d_avg And common[u] > c_avg then
            return u
                                                         ▷ If no node could satisfy the criterion
    return node with minimum distance in node_set
```

Algorithm 7 DijktraT algorithm outline

Input: Instance I of SMP
Output: k-path P or $\emptyset$ if none is found

```
Let PQ be a priority queue with shortest distance as key
for all node u ∈ V(G) \ {s} do                          ▷ Initialization
    dist[u] ← ∞
    previous[u] ← undefined
    common[u] ← ∅
common[s] ← C
dist[s] ← 0
PQ ← PQ ∪ {s}                                           ▷ Insert s into the queue
Let T be a positive real number
Let Dec be a positive real number, Dec < T
while PQ is not empty do
    u ← GetMinNode(T, k, PQ)
    PQ ← PQ \ {u}
    if u is t then                                      ▷ Solution found
        return path rebuilt from previous
    for all v ∈ δ+(u) do
        c' ← common[u] ∩ c(u, v)
        if |c'| < k then                                ▷ Not enough common colours
            Ignore node v
        else
            alt ← dist[u] + w(u, v)
            if alt < dist[v] then
                dist[v] ← alt
                common[v] ← c'
                previous[v] ← u
                PQ ← PQ ∪ {v}
    T ← T− Dec
return ∅                                                ▷ No solution found

function GetMinNode(T, k, node_set[])
    Let S = {v | v ∈ node_set and common[v] ≥ ⌊T * k⌋}
    if S is empty then
        return node in node_set with minimum distance
    else
        return node in S with minimum distance
```

Algorithm 8 DijkstraX heuristic

Input: Instance I of SMP
Output: k-path P or $\emptyset$ if none is found

```
for all node u in V(G) \ {s} do                          ▷ Initialization
    dist[u] ← ∞
    previous[u] ← undefined
    common[u] ← ∅
    score[u] ← ∞
common[s] ← C
dist[s] ← 0
score[s] ← 0
Let D be the longest shortest distance from s
Let PQ be priority queue with score as key
PQ ← PQ ∪ {s}                                            ▷ Insert s into the queue
while PQ is not empty do
    Let u be node in PQ with minimum score
    PQ ← PQ \ {u}
    if u is t then                                       ▷ Solution found
        return path rebuilt from previous
    for all v ∈ δ+(u) do
        c' ← c(u, v) ∩ common[u]
        if |c'| < k then                                 ▷ Not enough common colours in this path
            Ignore node v
        else
            alt ← dist[u] + w(u, v)
            if alt < dist[v] then
                dist[v] ← alt
                previous[v] ← u
                common[v] ← c'
                score[v] ← dist[v] - D · |c'|
                PQ ← PQ ∪ {v}
return ∅                                                 ▷ No solution found
```

Chapter 7

Computational results

Network simulations for our problems serve two purposes. In one hand they assess the performance of the algorithms in practical terms, since heuristic solutions give no guarantee on the quality of the result obtained. By comparing the results obtained for the heuristics and for the exact solutions, we have a basis in which to compare all algorithms. On the other hand, a metric like blocking ratio, defined in the next section, can only be assessed through a computational experiment.

In this chapter we present the results obtained through the simulations. We consider three types of metrics: execution time, solution cost and blocking ratio. Each algorithm is fed with an input instance (s, t, k, p), and with the current state of a network given by a multi-edge-coloured graph G. Each instance is solved by the algorithm which then returns the k-paths found. If no path was found, we say the algorithm *blocked* for that instance. For the SMP problem, $p = 1$. For MMP and AMMP, $p = 2$ due to the fact that few applications require multiple paths in practice [8].

For the ILP models we used the CPLEX tool [20]. Simulation rounds were executed in a 2.4GHz quad-core machine with 8GB of RAM memory. All the graphs in this chapter displaying averages show a 95% confidence interval.

7.1 Execution time and solution cost

For each execution round we keep track of the execution time of each algorithm. If valid k-paths are returned, the cost of the solution is defined as the cost of the paths returned, i.e., the cost of the solution is $\sum_{P} \sum_{(i,j)\in P} w(i, j)$. We use a simple simulator written in C++. The simulator generates a random connection request and run each algorithm on this instance.

The graphs representing the networks used in the simulations were generated in the following manner. We start with a random $G(n, d_a)$ graph. That means the graph has

n vertices and each edge is included in the graph with probability $0 < d_a < 1$. We also call d_a the arc density of the graph. Each added edge starts off with 8 colours available. Then, given a colour density $0 < d_c < 1$, we removed random colours from randomly selected edges until we reached about $|E||C| \cdot d_c$ colours in total in the graph. The purpose of variating the edge and colour densities is to simulate scenarios with different network loads. In Table 7.1 we list the pairs d_a, d_c used to generate our instances with an "X" telling which particular pair combinations were used to generate the graphs.

Arc density (%) \ Colour density (%)	10	40	60	70	80
10	X	X		X	
30		X	X		X
40	X	X		X	
60		X	X		X
70	X	X		X	
90		X	X		X

Table 7.1: Density pairs used for random graphs generation

For each d_a, d_c pair, 30 random instances were generated for each graph size. The graph sizes (number of nodes) used were 100, 250, 500, 750, 1000 and 2500. Figure 7.1 on the following page depicts the effect of the densities on the Branch and Bound algorithm for the SMP problem. To generated this graph, we calculated the mean cost for each size considering only those instances which were solved by the algorithm for all graph sizes. As expected, denser graphs yield "cheaper" solutions. For the case $d_a = 10\%, d_c = 10\%$, only 5 instances did not block for all sizes, therefore the high variance for this case.

Likewise, Figure 7.2 on the next page depicts the effects on execution time of the densities on the Branch and Bound algorithm. Unlike with the cost, execution time increases as the graphs become denser, due to the increase in processing needed for these kind of instances.

Effect of arc (da) and colour (dc) densities
on the Branch and Bound algorithm for the SMP problem

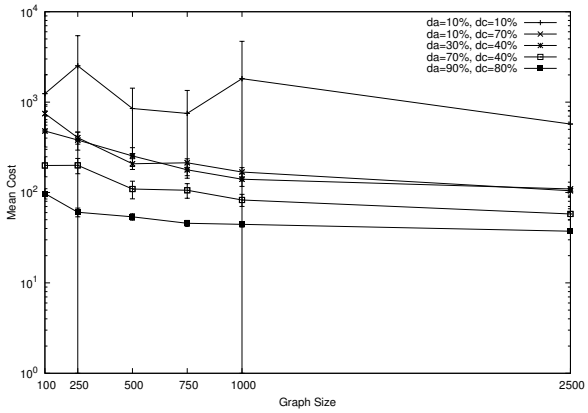


Figure 7.1: Mean cost

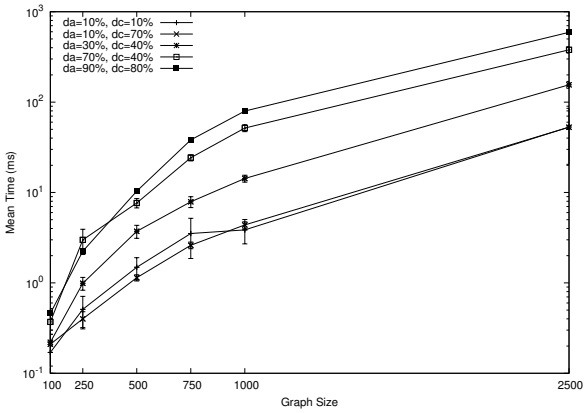


Figure 7.2: Mean execution time

Following the methodology in [8], each simulation round consisted in generating a connection request from a source node s to a destination node t, both s and t chosen uniformly among all nodes in the graph, $s \neq t$. The number k of colours in the request was chosen uniformly between 2 and 5. We set a time limit of 2 minutes for each algorithm so that the whole simulation would not take no more than 30 days. If upon reaching the 2-minutes mark no solution was returned, the algorithm is terminated and it is considered to have blocked for that instance.

Figure 7.3 on the following page shows the effect of the number of colours in the request on the solution cost for the Branch and Bound algorithm. A high colour number request results in higher solution costs. The opposite behaviour is observed with respect execution time (Figure 7.4 on the next page).

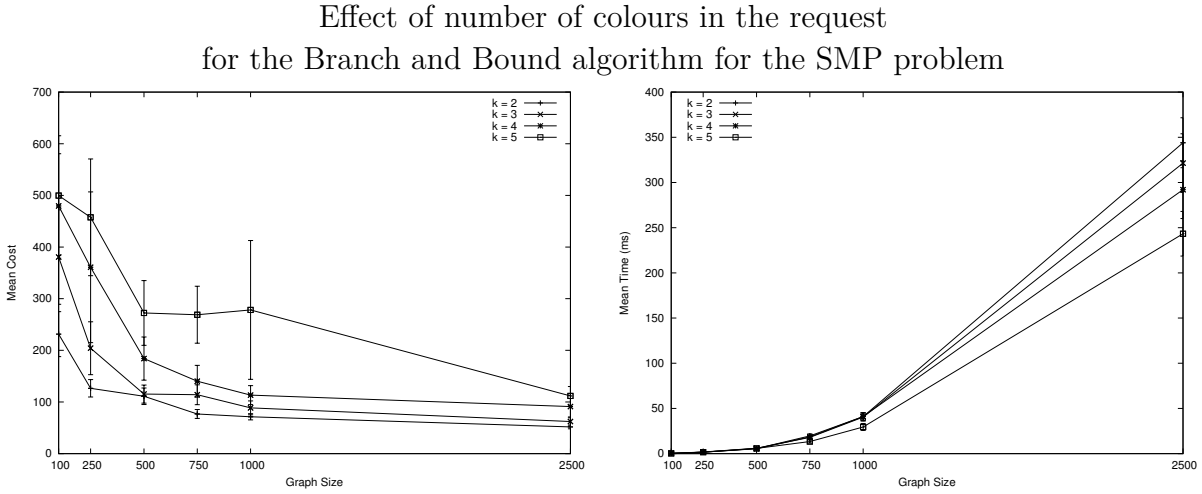


Figure 7.3: Mean solution cost

Figure 7.4: Mean execution time

Let $C^+(v) = \bigcup_{j \in \delta^+(v)} c(v,j)$ be the set of colours on the output arcs of a node v. Likewise, $C^-(v) = \bigcup_{j \in \delta^-(v)} c(j,v)$ is the set of colours in the input arcs of v. We claim that for any k-path between s and t in the SMP problem, the colours used will be a subset of $C' = C^+(s) \cap C^-(t)$. Therefore, we can execute a preprocessing step in which we remove every edge $e \in E$ with $|c(e) \cap C'| < k$. For the MMP and AMMP problems, we remove the edges with $c(e) \cap C' = \emptyset$. This preprocessing step was not included on the running time of the algorithms.

To compare the algorithms graphically, we present their *performance profiles*, a comparison method proposed by Dolan and More [14]. This method of comparing optimization software and algorithms consists in comparing the algorithms among themselves, computing for each instance the ratio between the value of some metric for that instance and the best result achieved for that instance considering all the algorithms. So, if some algorithm has ratio equal to 1 for a particular instance, that means it obtained the best result for that instance among all the other algorithms. In Figure 7.5 on the following page, the Intersection algorithm has point $(1, 0.35)$ meaning it solved 35% of the instances with best cost, whereas the point $(5, 0.85)$ indicates it solved 85% of the instances with cost at most 5 times the best cost. We compute all such ratios for all instances regarding solution cost and execution time.

In Figure 7.5 on the next page and Figure 7.6 on the following page we present the performance profiles for solution cost and execution time, respectively, for the SMP problem. Particularly for the time performance case, we replaced the ratio by the actual running time in milliseconds, so as to give a better perspective for the comparisons. For example, in Figure 7.6 on the next page, the point $(100, 0.901)$ for the Intersection algorithm indicates that it solved 90.1% of the instances in no more than 100 milliseconds.

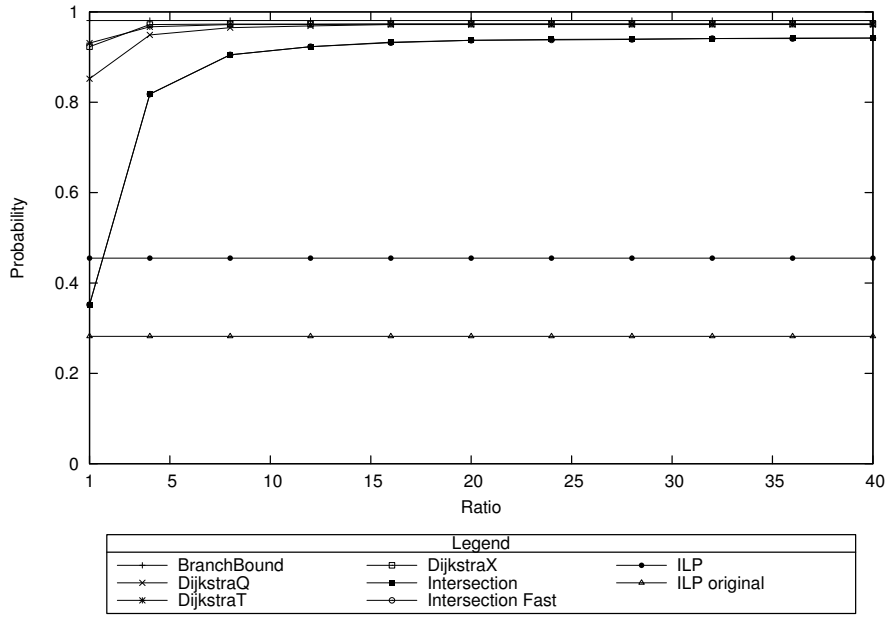


Figure 7.5: Solution cost performance profile for the SMP

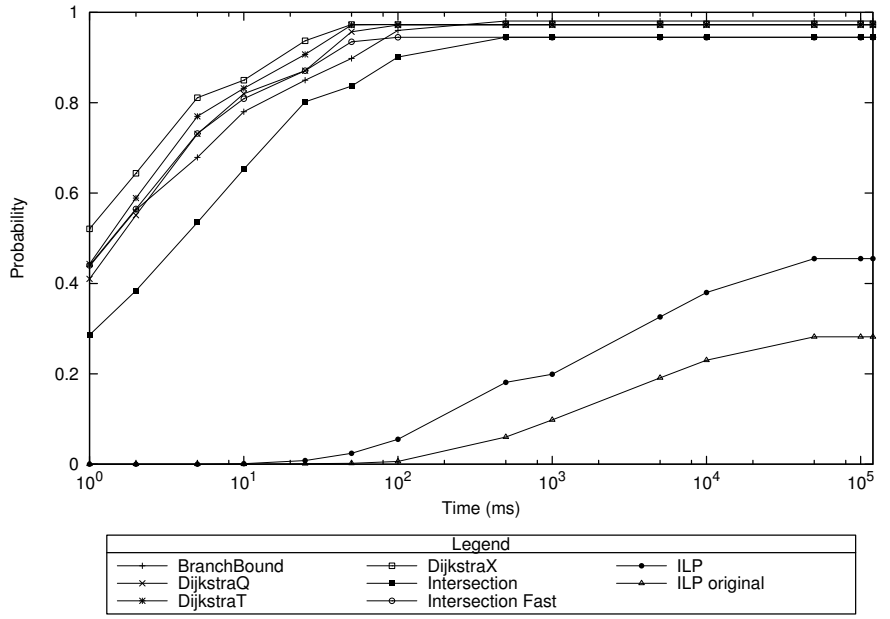


Figure 7.6: Execution time performance profile for the SMP problem

Some relevant remarks regarding the simulations for the SMP problem. As it is can be seen from the graphs in Figure 7.5 and Figure 7.6, the ILPs are easily outperformed in execution time. The original ILP formulation blocked on more than 70% of the instances

by not being able to solve them in 2 minutes. The faster ILP formulation, in turn, blocked on 55% of the instances. With a performance better than the ILPs but still slower than the Branch and Bound algorithm, the Intersection algorithm solved 82% of the instances with cost 4 times the optimum and solved 90% of the instances in less than 100ms, whereas the IntersectionFast version was able to solve 94.5% in the same time frame. The Branch and Bound algorithm performed surprisingly well, blocking less than 2% and solving the majority of the instances in less than 500ms. With respect to the heuristics, DijkstraT was best in terms of solution cost (93% solved with best cost) followed by DijkstraX (92%), whereas in terms of execution time, DijkstraX was able to solve 52% of the instances in up to 1ms.

To check at which point the Branch and Bound algorithm would start blocking the instances whereas the heuristics would solve them, we kept increasing the graph size. The issues encountered while executing this kind of test were the amount of memory needed (more than we had, causing the use of swap space) because of the size of the graphs; also, loading and preprocessing the graph took a long time, around 30 minutes each of these steps for a graph with 10000 nodes. For that case, the Branch and Bound blocked while the heuristics solved the instance within 20 seconds.

A similar simulation setup to the SMP was used for the MMP and AMMP problems. All requests required 2 paths and between 2 and 5 colours chosen uniformly.

Figure 7.7 on the following page and Figure 7.8 on the next page depict the performance profiles for the AMMP problem. Again the ILP algorithm is easily surpassed with respect execution time. It blocked on 85% of the instances because of the time limit. The heuristic MMPMin had the best runtime, solving all non-blocked instances in 500ms. It blocked on 30% of the instances. The Branch and Bound algorithm solved less than 2% of the instances with the best runtime and solved 91% of them with less than 50000ms. It blocked only on 1% of the instances.

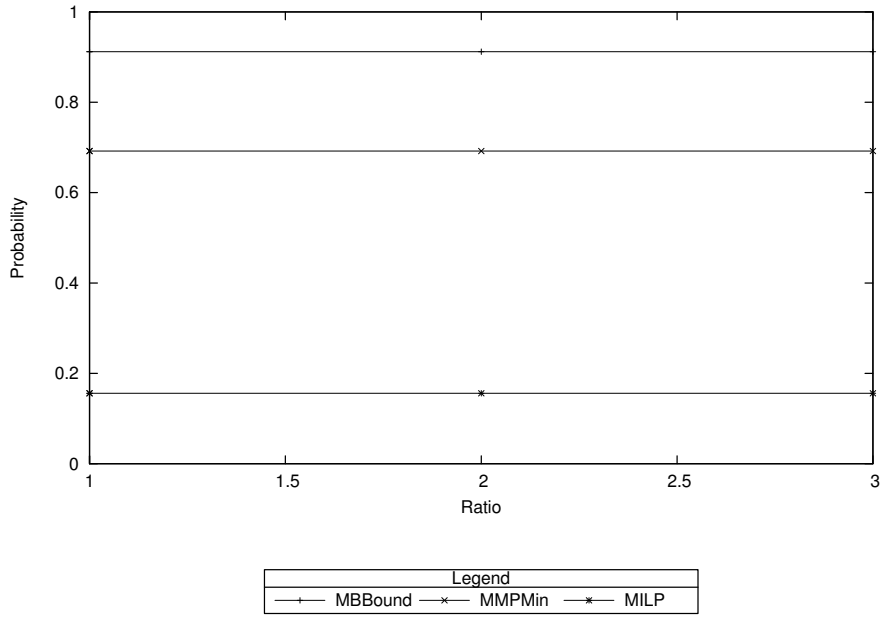


Figure 7.7: Solution cost performance profile for the AMMP problem

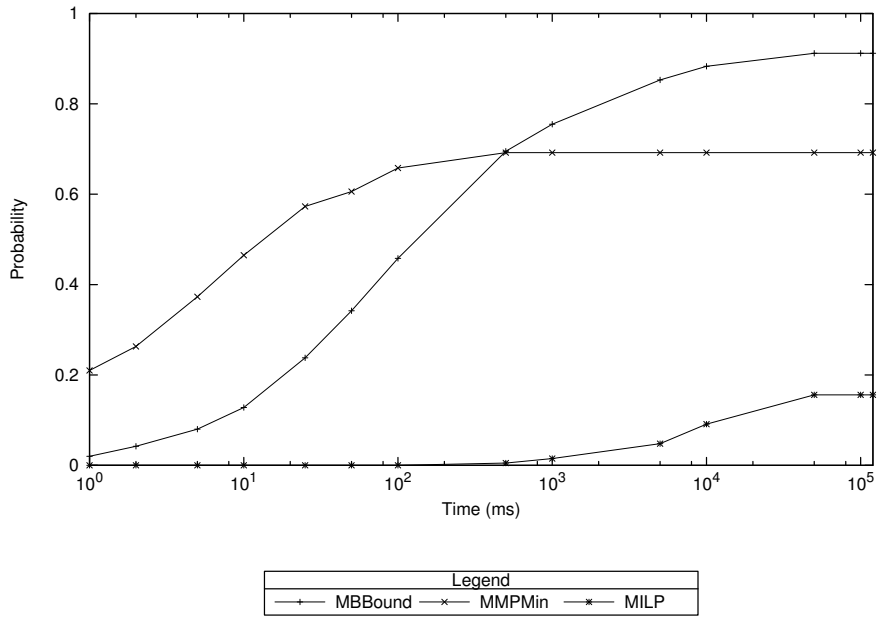


Figure 7.8: Execution time performance profile for the AMMP problem

Figure 7.9 on the following page and Figure 7.10 on the next page show the performance profiles for the MMP problem. The ILP algorithm blocked on 91% of the

instances. The heuristic blocked on 32% of the instances and solved the other 68% in less than 500ms.

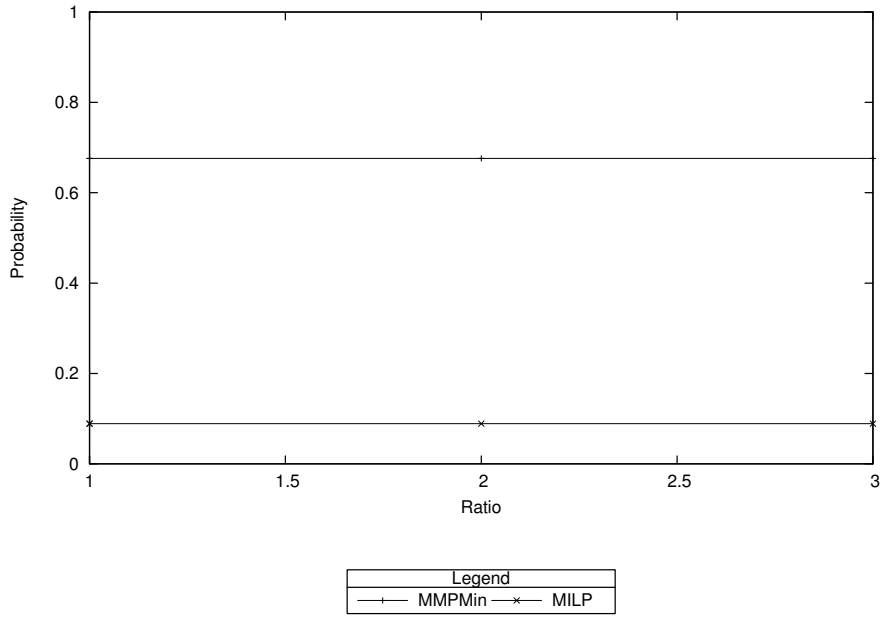


Figure 7.9: Solution cost performance profile for the MMP problem

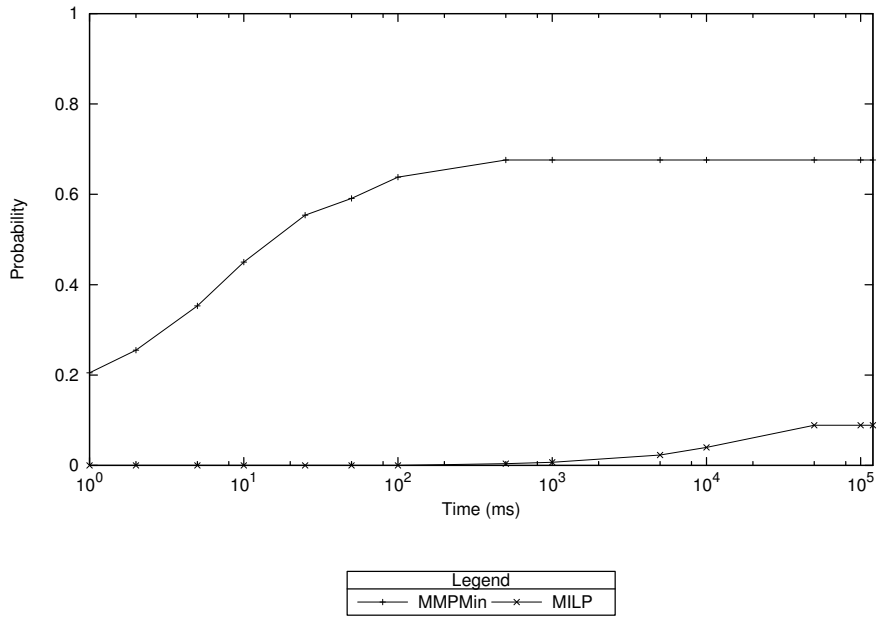


Figure 7.10: Execution time performance profile for the MMP problem

7.2 Blocking ratio

We now discuss the blocking ratio results. The blocking ratio defined in [8] is the bandwidth blocking ratio (BBR). It is the percentage of blocked bandwidth traffic relative to the total requested bandwidth during one simulation round for a WDM network. Since a bandwidth value can be directly converted to a number of colours given the bandwidth capacity of a wavelength, we use the term BBR hereafter. To calculate the blocking ratio we used a discrete event simulator developed in Java [8, 15] to generate dynamic connection requests and account for the graph states. Each simulation round is comprised of 10000 connection requests arriving with negative exponentially distributed inter-arrival time. For each connection request, if the algorithm fails to provide a suitable solution, the request is blocked and rejected. Otherwise, the paths to be established provided as output are used to update the graph state accordingly.

Instead of randomly generated graphs, we used the same real network topologies for measuring the blocking ratio as in [8]. The topologies used are:

Grid 5x5 network: with 25 nodes and 80 arcs (Figure 7.11),

USA long-distance Mesh network: with 25 nodes and 86 arcs (Figure 7.13 on the next page),

NSFNet network: with 16 nodes and 50 arcs (Figure 7.14 on page 52), and

Pan-European network: with 28 nodes and 82 arcs (Figure 7.12 on the next page)

In each of this topologies, the number in the nodes are identification labels. Each arc is labelled with a weight value representing the length in kilometres of that arc. In the case of the grid, every arc has a unitary weight. We assume each arc contains 8 colours in all topologies.

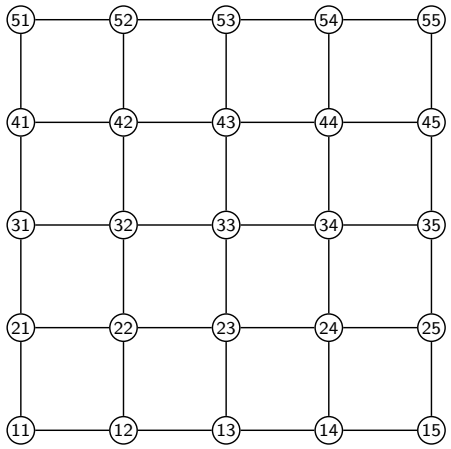


Figure 7.11: Grid 5x5 network

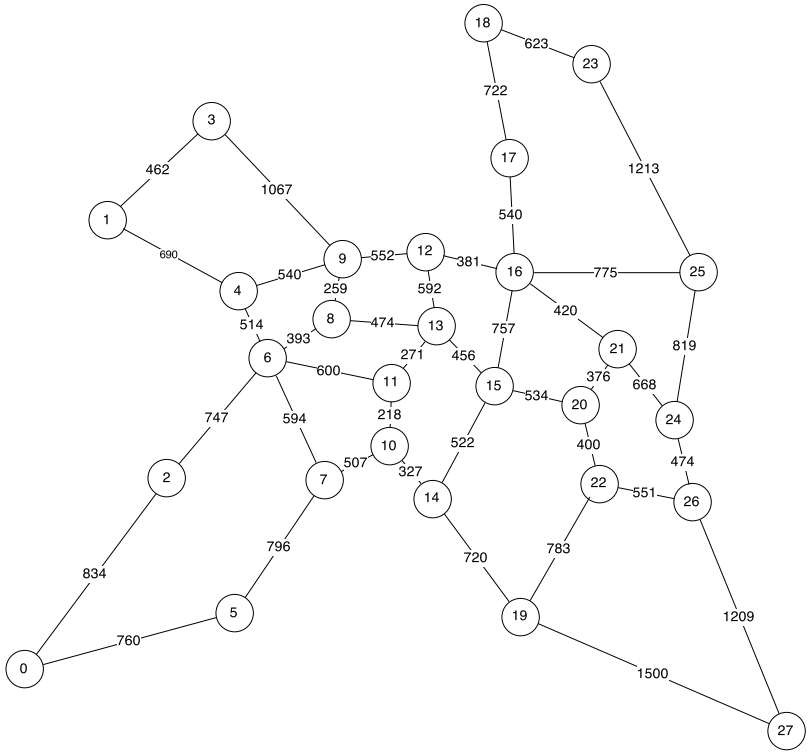


Figure 7.12: Pan-European Network

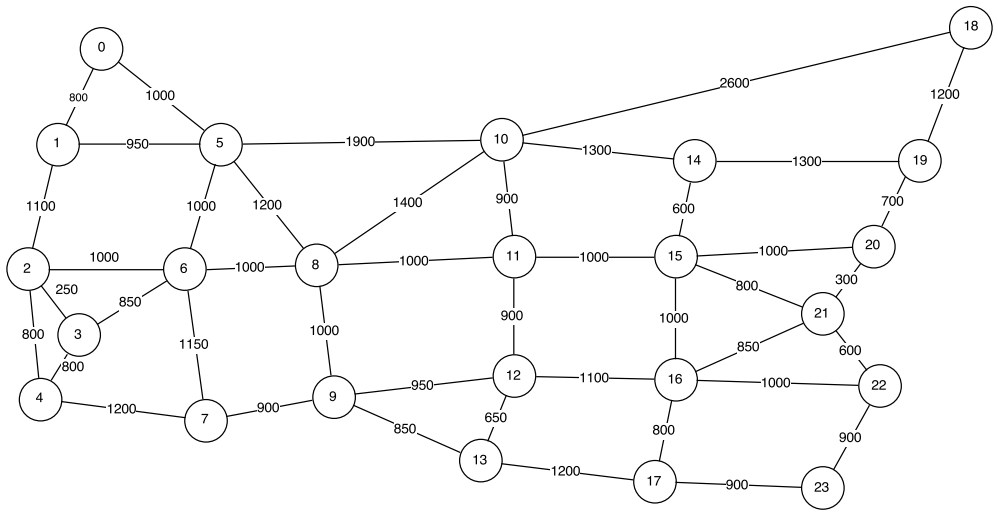


Figure 7.13: USA Long-distance Mesh Network

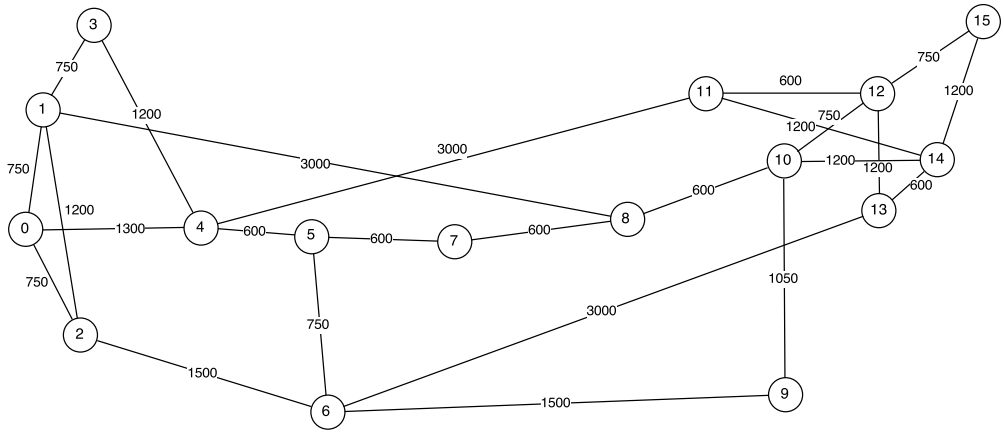


Figure 7.14: NSFNet network

For each simulation round, the networks start with all colours available in all arcs. For each one of the 10000 connection requests, source and destination nodes are chosen uniformly among all nodes. Colour requests range from 1 to 5 colours and they are generated with probability proportionally inverse to the number of colours, i.e., requests of 1 colour have five times more chance of being generated than those with 5 colours. The established paths are release after a defined *holding time* has passed. Connection holding times are sampled in a negative exponential distribution with mean value of a

single unit. For the MMP and AMMP problems, $p = 2$ for each connection request. All results in this section show confidence intervals of 95%.

In Figure 7.15 on the following page, Figure 7.16 on page 54, Figure 7.17, and Figure 7.18 on page 55 we present the BBR values for the SMP problem for the networks Grid 5x5, NSFNet, PanEU and USA, respectively.

Apart from the intersection-based algorithms, all algorithms had similar blocking ratios in all network types.

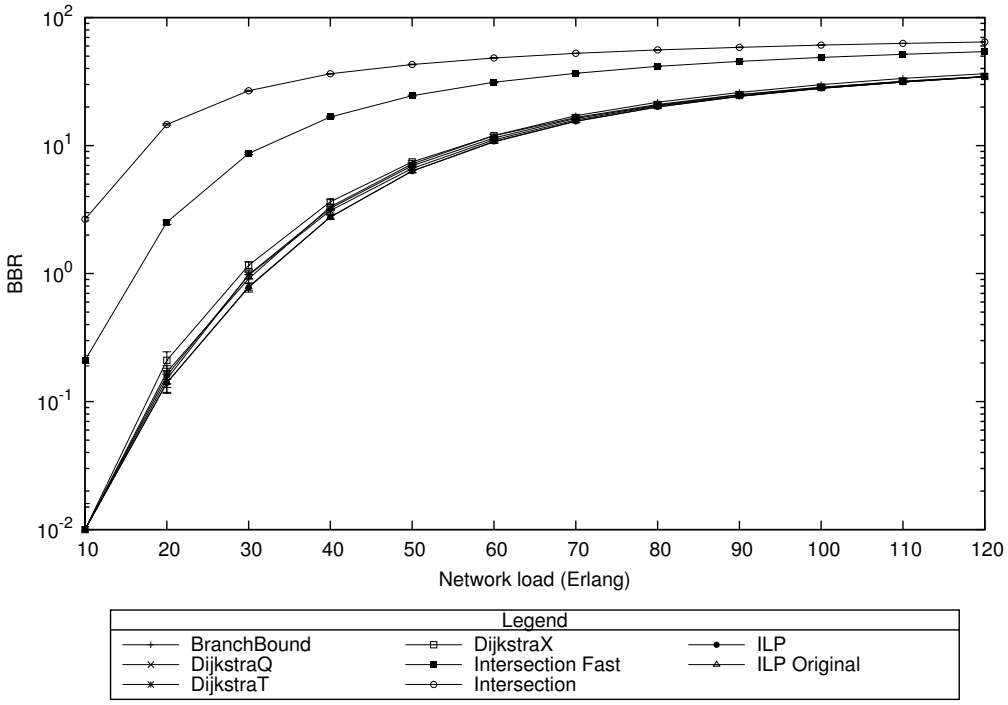


Figure 7.15: Grid 5x5 network blocking ratio for the SMP problem

In Figure 7.19 on page 55, Figure 7.20 on page 56, Figure 7.21 on page 56 and Figure 7.22 on page 57 we present BBR values for the AMMP problem for the Grid 5x5, NSFNet, PanEu and USA networks, respectively. Now, the Branch and Bound algorithm achieved the lowest ratio in all networks for loads up to 50 Erlang. The biggest difference in blocking behaviour is shown in the Grid 5x5 for 10 and 20 Erlang; MMPMin blocked 10 times more than the Branch Bound and the ILP algorithm blocked 10 times more than the heuristic. However, as the load increases, the blocking behaviour of the algorithms become similar, even slightly higher for the Branch and Bound.

The MMPMin heuristic had the best performance for the MMP problem, with blocking ratio equal ou lower to that of the ILP formulation, as show in Figure 7.23 on page 57,

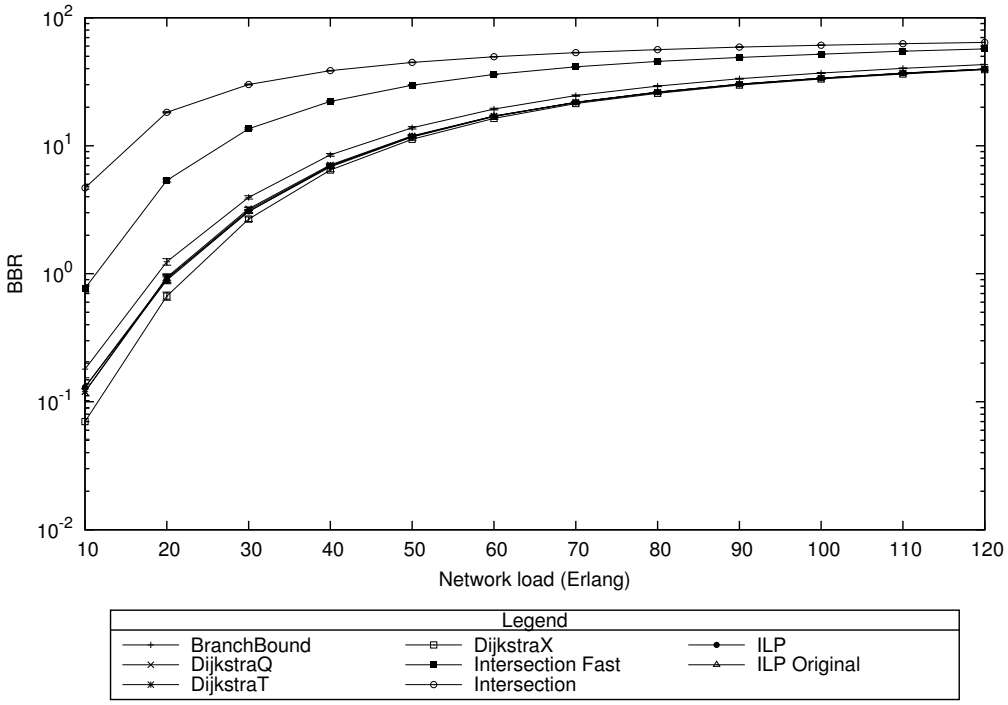


Figure 7.16: NSFNet network blocking ratio for the SMP problem

Figure 7.24 on page 58, Figure 7.25 on page 58, and Figure 7.26 on page 59. As with the AMMP problem, the biggest difference between the algorithms is seen for the Grid 5x5 network.

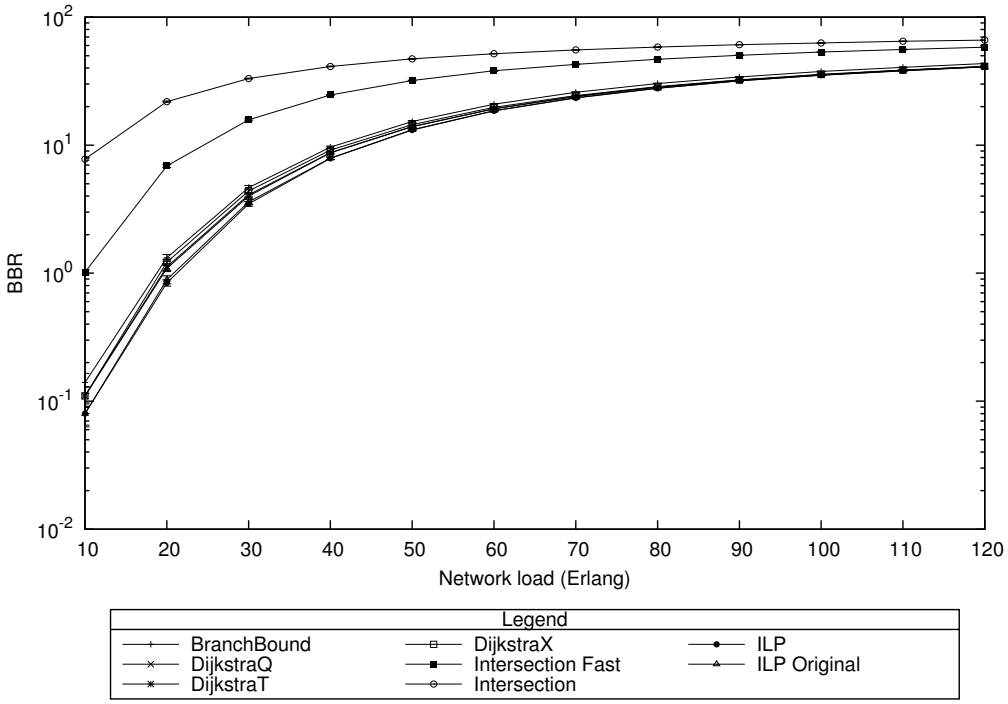


Figure 7.17: PanEU network blocking ratio for the SMP problem

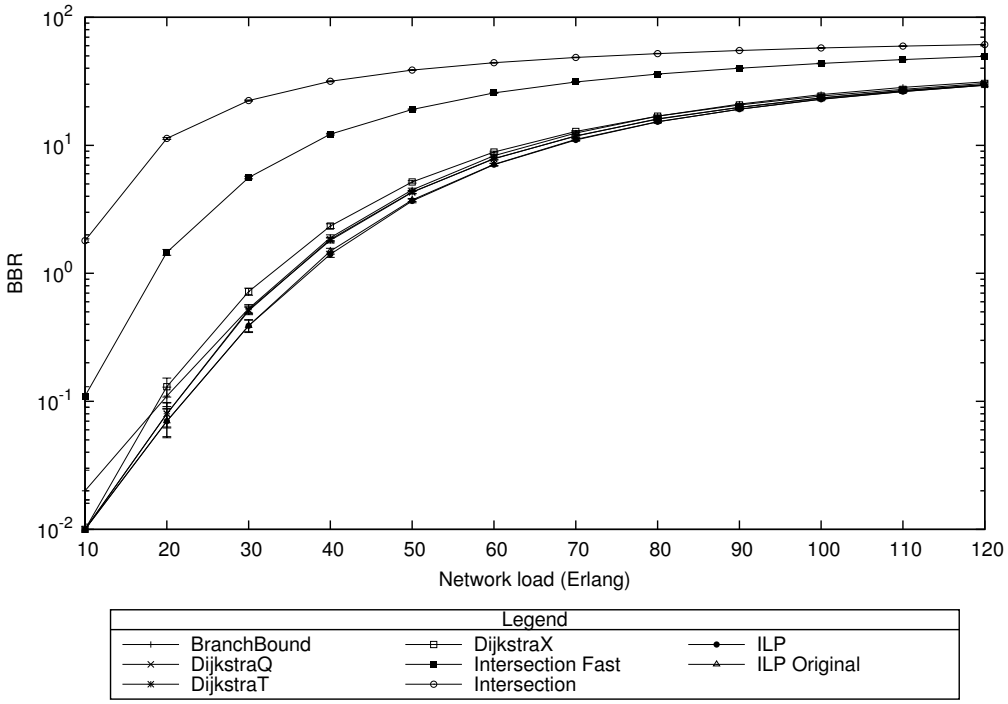


Figure 7.18: USA network blocking ratio for the SMP problem

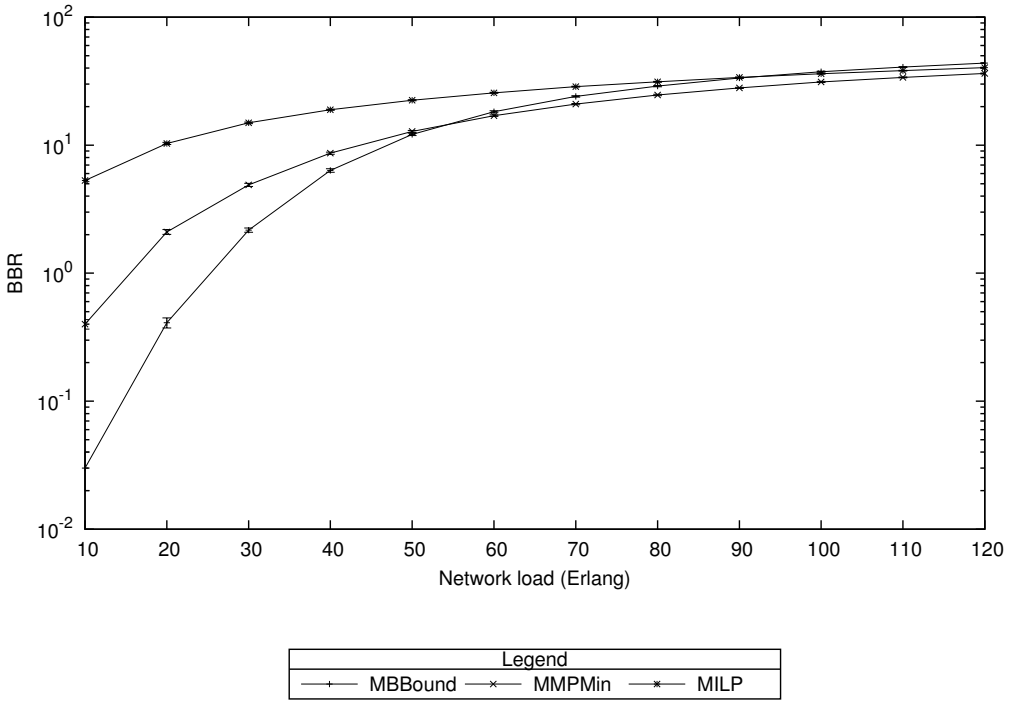


Figure 7.19: Grid 5x5 network blocking ratio for the AMMP problem

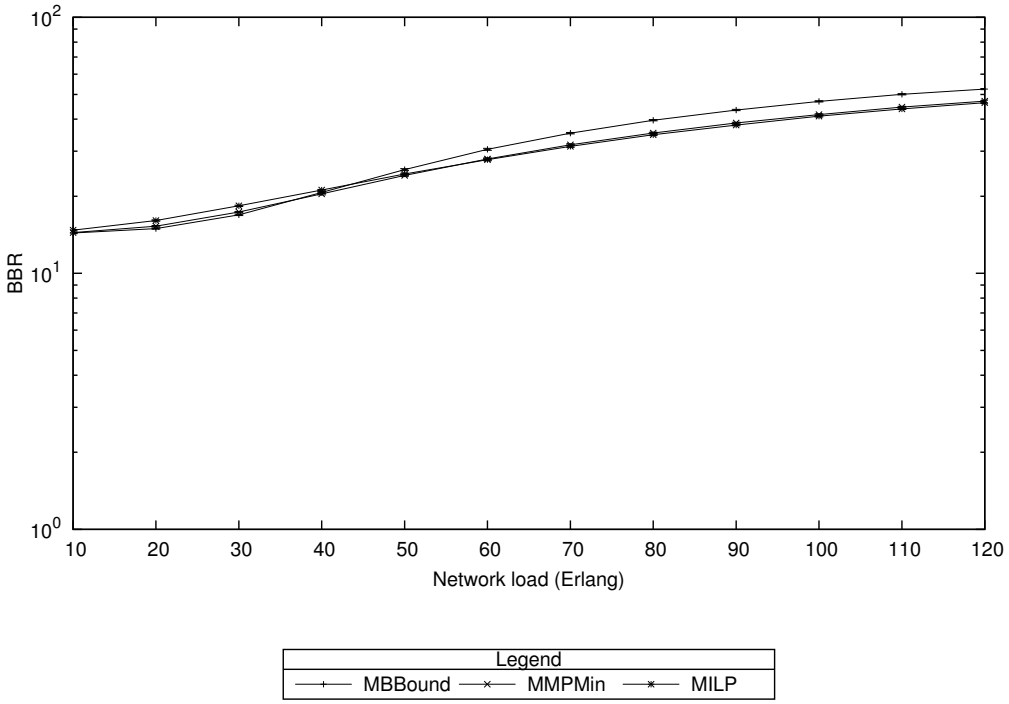


Figure 7.20: NSFNet network blocking ratio for the AMMP problem

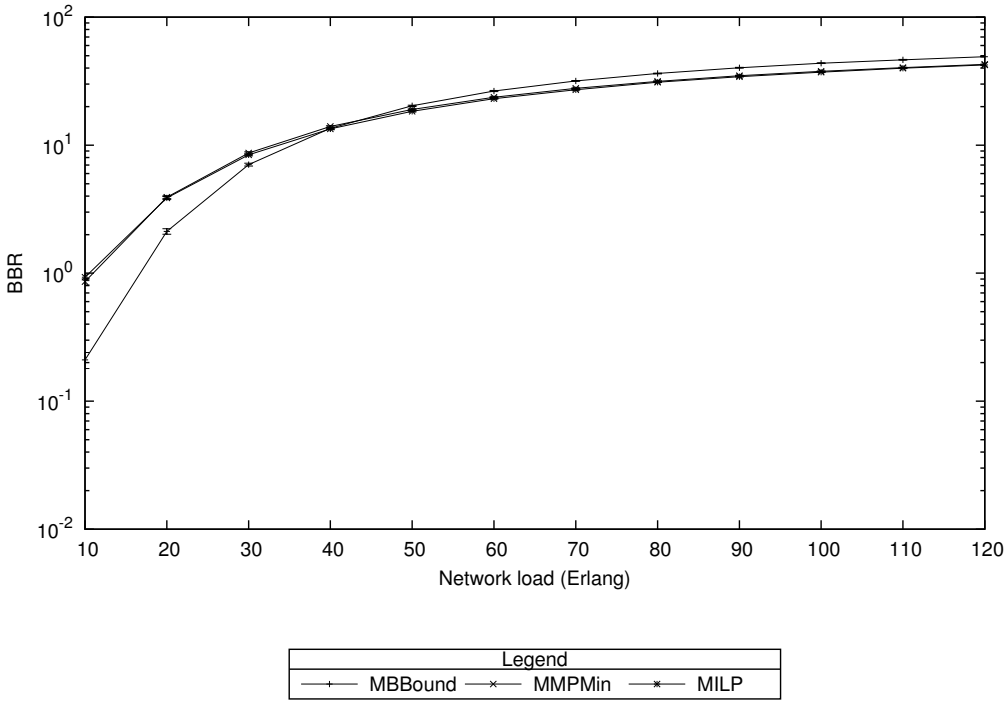


Figure 7.21: PanEU network blocking ratio for the AMMP problem

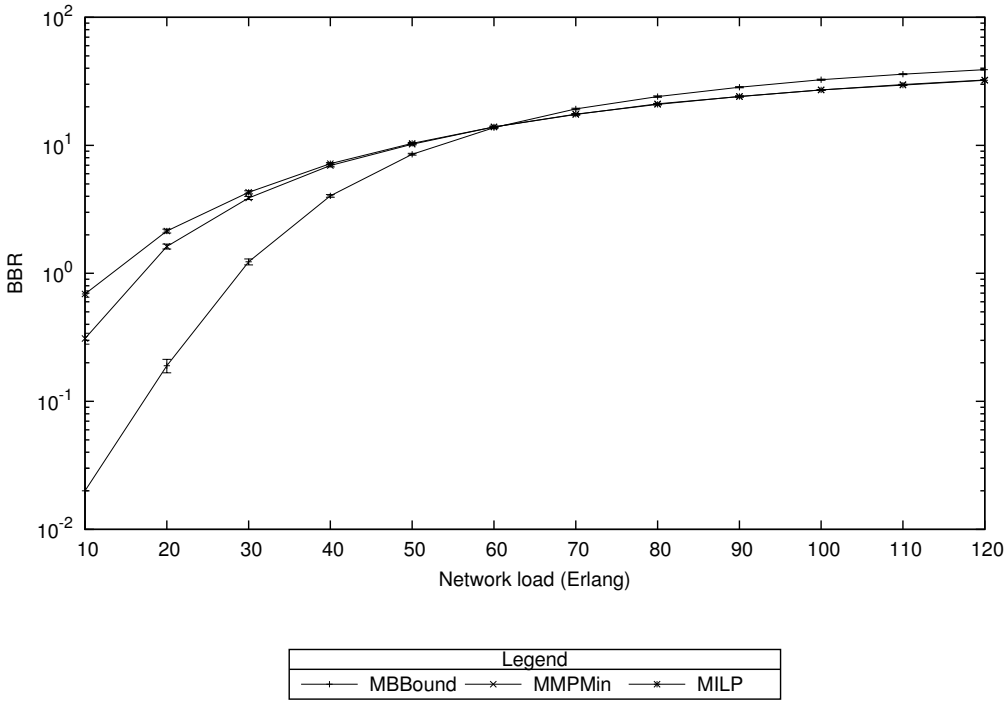


Figure 7.22: USA network blocking ratio for the AMMP problem

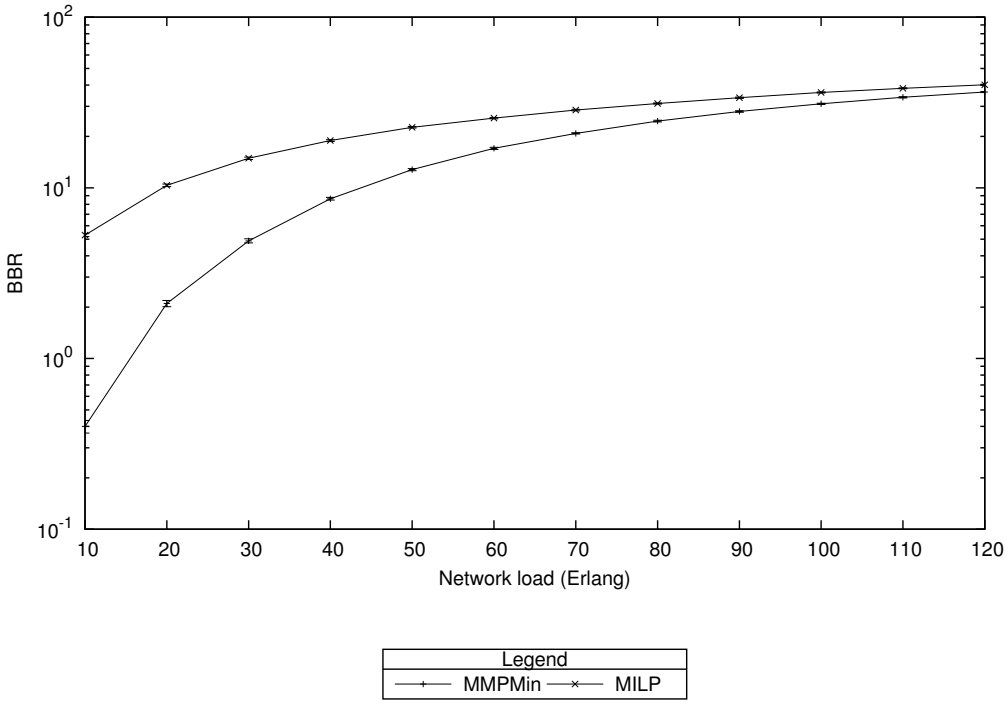


Figure 7.23: Grid 5x5 network blocking ratio for the MMP problem

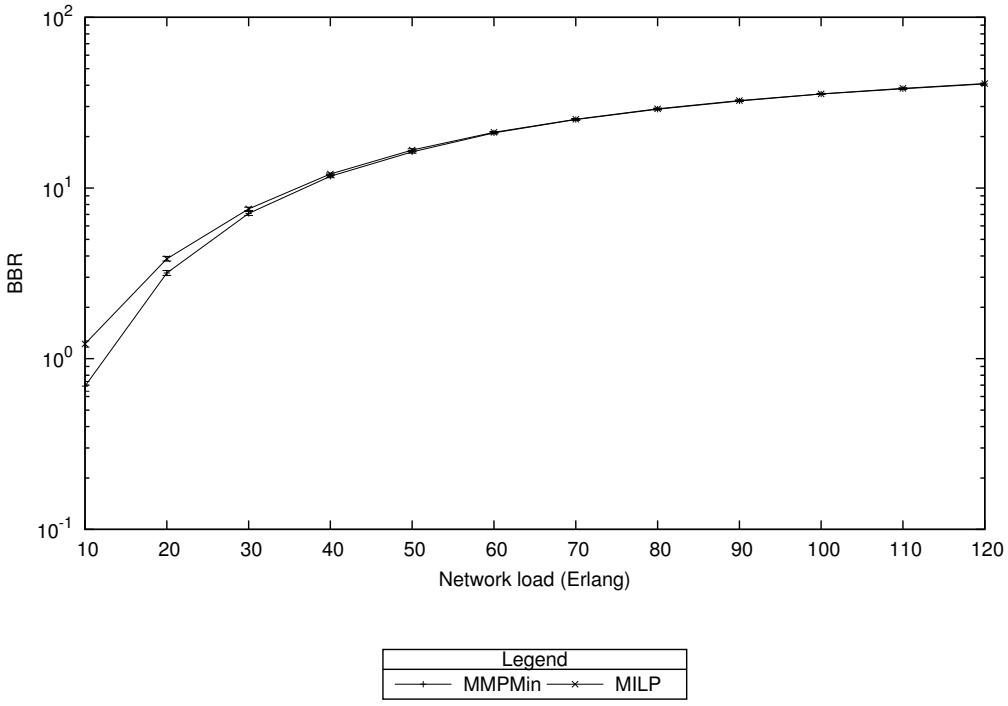


Figure 7.24: NSFNet network blocking ratio for the MMP problem

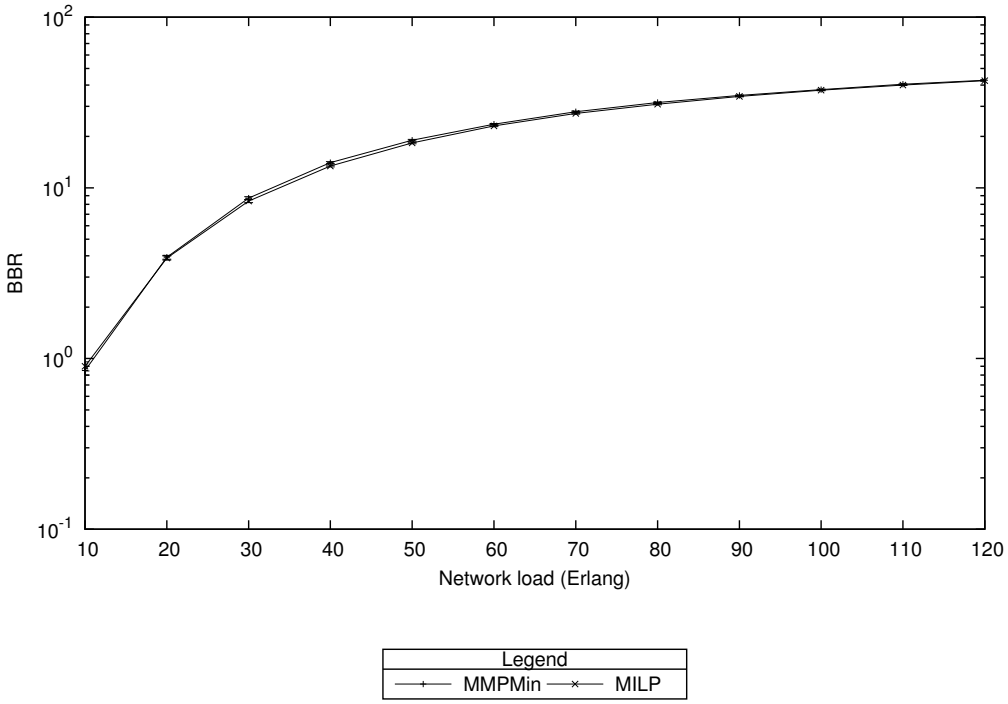


Figure 7.25: PanEU network blocking ratio for the MMP problem

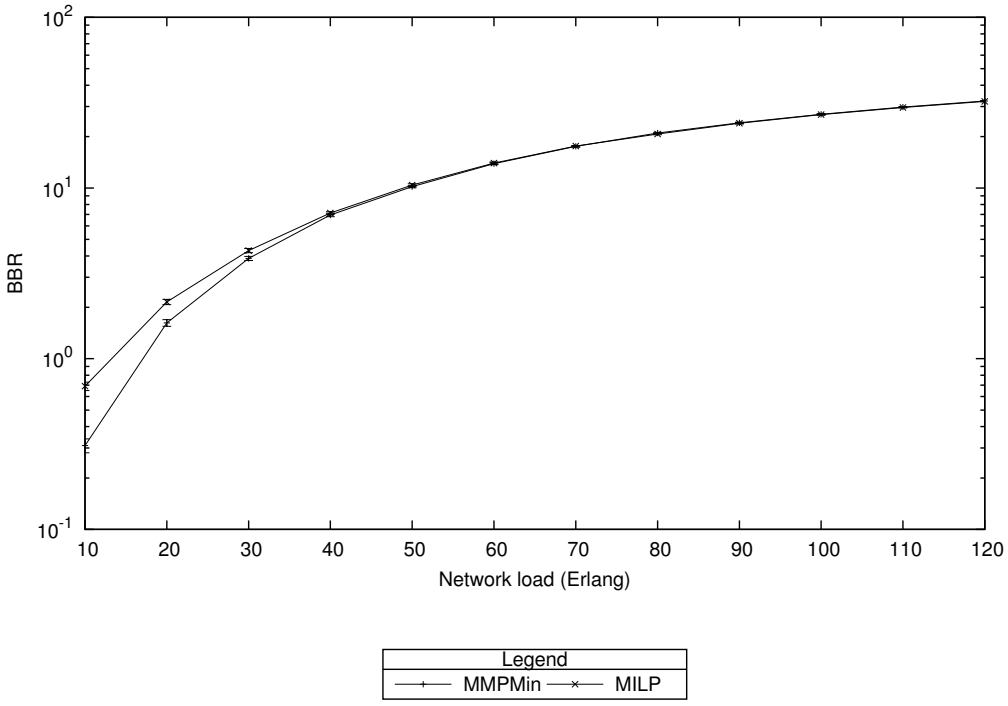


Figure 7.26: USA network blocking ratio for the MMP problem

Chapter 8

Conclusions

This work discussed the problem of finding multicolour paths in multi-edge-coloured graphs. The network counterpart problem of finding multiple lightpaths in WDM optical networks so as to satisfy a high bandwidth requirement was first formalised as a pure graph problem. Having a formal description of the problem and its variants, it proceeded then to prove NP-hardness results for the problem. Particularly, it was proven that finding a single k-multicolour path in a multi-edge-coloured graph is NP-hard; the same result holds for finding p multicolour paths so that the sum of colours used is k independently if the paths are required to be edge-disjoint or not. Assuming P $\neq$ NP, there are no efficient polynomial time algorithms to solve NP-hard problems optimally. Therefore, heuristics were proposed to the problems. These heuristics were based on two algorithmic ideas: the Dijkstra's shortest path algorithm and graph intersection. To assess the efficiency as well as the quality of the solutions returned by the heuristics, computational experiments were devised. These experiments measured the execution time, the cost of the solution returned and the blocking ratio of the algorithms. As comparison parameters, exact Branch and Bound and ILP formulations were developed.

The simulations results show that the Branch and Bound algorithm is a suitable alternative for the SMP problem, being able to solve instances with up to 10000 nodes in less than 2 minutes. It also displayed a low blocking behaviour for lower network loads. Regarding the heuristics, DijkstraX showed the best trade-off with respect to solution cost, execution time and blocking ratio. The use of a score function makes the implementation faster than the other Dijkstra-based heuristics; besides, the score function can be easily changed to accommodate different priorities, like prioritizing paths with more colours first, rather than shortest paths. For finding more than one path, the Branch and Bound solution starts suffering the effects of the exponential growth of the search tree regarding execution time. In the AMMP problem, the Branch and Bound implementation is more suitable for low network loads, up to 50 Erlang, whereas the

MMPMin heuristic is a better general choice for both problems with worst blocking ratio in the order of magnitude 10 when compared to the Branch and Bound solution. For high network loads, there is no much difference on all algorithms' blocking behaviour.

Further work based on these problems can explore classes of graphs in which exact solutions can be easily obtained or, on the other hand, classes for which the problem shows its NP-hard behaviour. Another possibility is to investigate approximation algorithms whose objective function is to satisfy the colours requirement instead of finding the shortest path.

Bibliography

[1] A. Aggarwal, J. Kleinberg, and D. P. Williamson. Node-disjoint paths on the mesh and a new trade-off in vlsi layout. *SIAM J. Comput.*, 29:1321–1333, February 2000.

[2] D. Banerjee and B. Mukherjee. A practical approach for routing and wavelength assignment in large wavelength-routed optical networks. *IEEE J.Sel. A. Commun.*, 14(5):903–908, June 1996.

[3] P. Bonizzoni, R. Dondi, and Y. Pirola. Maximum disjoint paths on edge-colored graphs: Approximability and tractability. *Algorithms*, 6(1):1–11, 2012.

[4] R. L. Brooks. On colouring the nodes of a network. In Ira Gessel and Gian-Carlo Rota, editors, *Classic Papers in Combinatorics*, Modern Birkhäuser Classics, pages 118–121. Birkhäuser Boston, 1987.

[5] C. Büsing and S. Stiller. Line planning, path constrained network flow and inapproximability. *Networks*, 57:106–113, January 2011.

[6] Q. Cai, X. Li, and D. Wu. Some extremal results on the colorful monochromatic vertex-connectivity of a graph. *arXiv preprint arXiv:1503.08941*, 2015.

[7] A. Castro, L. Velasco, J. Comellas, and G. Junyent. On the benefits of multi-path recovery in flexgrid optical networks. *Photonic Network Communications*, 28(3):251–263, 2014.

[8] X. Chen, A. Jukan, A. C. Drummond, and N. L. S. Da Fonseca. A multipath routing mechanism in optical networks with extremely high bandwidth requests. In *Proceedings of the 28th IEEE Conference on Global Telecommunications*, GLOBECOM'09, pages 2065–2070, Piscataway, NJ, USA, 2009. IEEE Press.

[9] B. Cherkassky, A. V. Goldberg, and T. Radzik. Shortest path algorithms: Theory and experimental evaluation. *Mathematical Programming*, 73:129–174, 1993.

[10] I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: An approach to high bandwidth optical wan's. *IEEE Transactions on Communications*, 40(7):1171–1182, 1992.

[11] J. Clausen. Branch and bound algorithms - principles and examples. *Parallel Computing in Optimization*, pages 239–267, 1997.

[12] J. de Santi, A. C. Drummond, N. L.S. da Fonseca, and X. Chen. Holding-time-aware dynamic traffic grooming algorithms based on multipath routing for wdm optical networks. *Optical Switching and Networking*, 16(0):21 – 35, 2015.

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

[14] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91, 2002.

[15] A. Drummond. Wdmsim optical wdm simulator. `http://www.lrc.ic.unicamp.br/wdmsim/about.html`, 2013. [Online; accessed 21-January-2013].

[16] S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[17] G. Gallo and S. Pallottino. Shortest path algorithms. *Annals of Operations Research*, 13(1):1–79, 1988.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., San Francisco, 1979. Standard reference on complexity theory. Includes extensive catalogue of NP-complete problems.

[19] D. Granata, B. Behdani, and P. M. Pardalos. On the complexity of path problems in properly colored directed graphs. *J. Comb. Optim.*, 24(4):459–467, November 2012.

[20] IBM. Ibm ilog cplex optimizer. "`http://www-01.ibm.com/software/integration/optimization/cplex-optimizer`", 2013. [Online; accessed 21-January-2013].

[21] J. P. Jue. Lightpath establishment in wavelength-routed wdm optical networks. In *Optical Networks*, pages 99–122. Springer, 2001.

[22] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 40(4):85–103, 1972.

[23] K. Kawarabayashi and Y. Kobayashi. An o(long n)-approximation algorithm for the edge-disjoint paths problem in eulerian planar graphs. *ACM Trans. Algorithms*, 9(2):16:1–16:13, March 2013.

[24] S. G. Kolliopoulos. Edge-disjoint paths and unsplittable flow. *Handbook of Approximation Algorithms and Metaheuristics*, pages 57–1, 2007.

[25] X. Li, Y. Shi, and Y. Sue. Rainbow connections of graphs: a survey. *Graphs and Combinatorics*, 29(1):1–38, 2013.

[26] X. Li and Y. Sun. On the strong rainbow connection of a graph. *Bull. Malays. Math. Sci. Soc.(2)*, 36(2):299–311, 2013.

[27] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 2010.

[28] P. M. Moura, N. L. S. da Fonseca, and R. A. Scaraficci. Fragmentation aware routing and spectrum assignment algorithm. In *Communications (ICC), 2014 IEEE International Conference on*, pages 1137–1142, June 2014.

[29] T. Ohtsuki. The two disjoint path problem and wire routing design. In *Proceedings of the 17th Symposium of Research Institute of Electric Communication on Graph Theory and Algorithms*, pages 207–216, London, UK, 1981. Springer-Verlag.

[30] N. Robertson and P. D. Seymour. Graph minors .xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65–110, 1995.

[31] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974.

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

[33] V. Vazirani. *Approximation Algorithms*. Springer-Verlog, 2000.

[34] B. Y. Wu. On the maximum disjoint paths problem on edge-colored graphs. *Discrete Optimization*, 9(1):50–57, 2012.

[35] Eduardo C. Xavier. A note on a maximum k-subset intersection problem. *Inf. Process. Lett.*, 112(12):471–472, 2012.

[36] H. Zang, J. P. Jue, and B. Mukherjee. A review of routing and wavelength assignment approaches for wavelength-routed optical wdm networks. *Optical Networks Magazine*, 1:47–60, 2000.