Multicolour Paths in Graphs: NP-hardness, Algorithms, and Applications on Routing in WDM Networks

Rafael F. Santos[1], Alessandro Andrioni[1], Andre C. Drummond[2], and Eduardo C. Xavier * † [1]

[1]Institute of Computing, University of Campinas, Campinas, Brazil
[2]University of Brasilia, Distrito Federal, Brazil

February 17, 2016

Abstract

In this paper we study a problem of finding coloured paths of minimum weight in graphs. This problem 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 let $C = \{1, 2, \ldots, x\}$ 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. We say that edge e contains colours $c(e) \subseteq C$. 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.

Key Words: Minimum Paths in Coloured Graphs, NP-Hardness, Heuristics, WDM Networks

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

*Corresponding Author: eduardo@ic.unicamp.br. Av. Albert Einstein 1251, Institute of Computing, UNICAMP, Campinas-SP, Brazil. Fax:(+55) (19) 3521-5847

†This work was supported by CNPq and FAPESP.

properties. There are classic examples of such problems in the literature, such as finding shortest paths [10, 13], node-disjoint or edge-disjoint paths [18, 26], routing, line planning [4], 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 Wavelength-Division Multiplexing (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. Two lightpaths can share a link, but if this happens they must use different wavelengths. 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 a certain number of colours. Chen et al. presented in [7] 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 on the following page. 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). For $k = 2$, the solution is the path (s, v_2, t), highlighted on Figure 2 on the next page, which uses colours 2 and 3. For $k \geq 3$, there are no possible solutions, since the highest number of common colours between the edges on any given path is at most 2.

This article is organized as follows. In Section 1.1, we discuss related works involving coloured graphs and WDM optical network design. In Section 1.2 we summarize our contributions. Section 2 introduce definitions, notations, and terms needed throughout the article. In Section 2.1 and Section 3 we define the problem formally and give a NP-hardness proof for it. Section 4 presents exact Branch and Bound and Integer Linear Programs algorithms to the problem, whereas Section 5 presents the developed heuristics. In Section 6, we evaluate the performance of the proposed algorithms through a simulation for execution time, solution cost (Section 6.1) and blocking ratio (Section 6.2). Conclusions are drawn in Section 7.

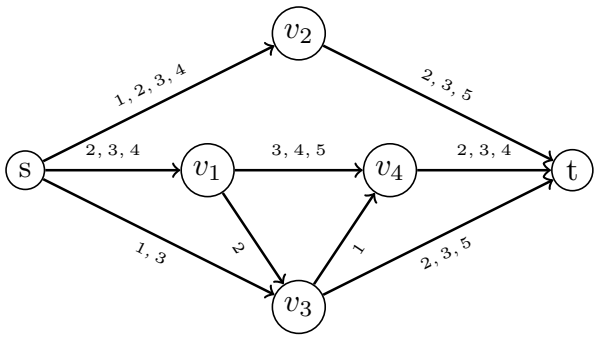

Figure 1: Edge-coloured graph example

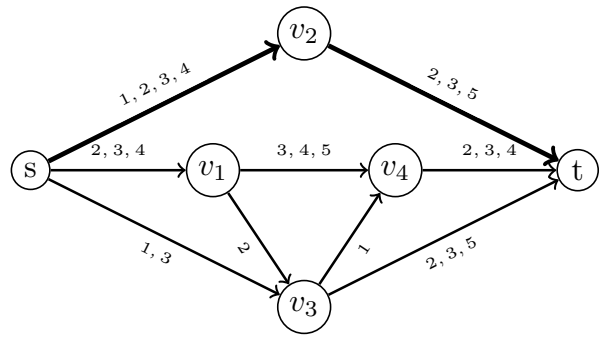

Figure 2: Feasible path for 2 colours

1.1 Related works

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

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 problem 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 $\chi(G)$. The vertex colouring problem was proven to be NP-complete by Karp in 1972 [17]. A survey on the algorithmic and computational results obtained for the vertex colouring problem can be found in [24, 21].

A related problem studied by Granata et al. [14] is that of finding a path from a vertex s that meets all the $\chi(G)$ colours in a coloured graph G. They proved that, in properly coloured directed graphs, finding a path that starts at a specific vertex and meets all $\chi(G)$ colours is NP-hard. It is also NP-hard to find a shortest (or longest) path 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 distinct colours, a path in G whose internal vertices have the same colours is called a vertex-monochromatic path [5]. 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 [5], Erdös-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 [19] 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 said to be a rainbow path between s and t if its length is $d(s, t)$, the length of the minimum path between s and t. The survey in [19] also discusses rainbow connectivity in edge-coloured graphs. Li and Sun [20], 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 [20], a 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 [27] studied the following problem: given two vertices $s, t \in V$, find the maximum number of mutually vertex-disjoint uni-colour paths between s and t. We call this problem by 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 for Max CDP by the use of a greedy strategy by [27].

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. He also proved 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 for the ℓ-LCDP problem.

Finding disjoint paths in graphs has many applications in other areas of Computer Science. In [23], Ntafos and Hakimi studied some variants of the problem of covering some elements of a digraph by disjoint paths. In one version of the problem one is asked to find the minimum number of paths that covers all vertices (or edges) of a given digraph. In another version of the problem, the objective is to find the minimum number of paths that covers selected pairs of vertices, this last version being proved to be NP-Hard. They explain how these problems arise in testing software systems. The problem of finding a cover of paths of a graph such that selected pairs of vertices belongs to the same path has also applications in Bioinformatics (see Beerenwinkel et. al. [2] and Rizzi et. al. [25]).

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

A Wavelength-Division Multiplexing (WDM) approach has been proposed to handle the ever-increasing bandwidth demands for users of 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 [1].

Chlamtac et al. [8] 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 [8] 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-hard. To solve the RWA problem, one can decouple it into two separate sub problems: routing and wavelength assignment. Many approaches relying on ILP formulations and heuristics have been presented to solve the problem and its components [29, 16].

Applications in the scientific and engineering communities and emerging media applications require extremely high bandwidth connections, typically larger than one wavelength [7]. To accommodate such requests, Chen et al. [7] propose a modification to the routing problem in WDM networks, 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 proposed approaches to path provisioning and traffic grooming in more wavelength-flexible networks (flexgrid optical network) [6, 22, 9].

1.2 Contributions

We address the problem of routing in WDM optical networks with extremely high bandwidth requests presented in [7]. We consider both versions of single path as well as multi paths. Heuristics and ILP models were developed

for these problems before (see [7]), but their complexity class was still unknown. We formulate these problems as an edge-coloured graph problem and study their complexity proving that they are NP-hard and also hard to approximate.

Unless P = NP, there are no efficient polynomial-time algorithms to solve these problems optimally. Therefore we develop heuristics and branch-and-bound algorithms and study their performance through a set of computational experiments also comparing to ILP models presented by Chen et al. [7].

2 Preliminaries

In this section we introduce some definitions, notations, and terms that will be needed throughout the paper. When describing problems and algorithms for paths in graphs, we make use of the following terms.

A graph $G = (V, E)$ has set of nodes V and set of edges E. The terms node and vertex will be used interchangeably. The terms arc and edge will be used interchangeably. 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; node j is an i-neighbour if (i, j) is in G. $\delta^+(i)$ denotes the set of out-neighbours of a node i, i.e., the set of nodes j for which there exists the edge (i, j) in G. Likewise, $\delta^-(i)$ denotes the in-neighbours, the set of nodes j for which $(j, i) \in E$. A *directed path* is determined by a sequence of nodes $i_1, i_2, \ldots, i_k$; it consists of these nodes and the arcs connecting them in sequence, $(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k)$. We say such path *connects* nodes i_1 and i_k and represent it as (i_1, i_k)-path. 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).

Unless otherwise stated, we assume any given graph is directed. We also assume a graph has no multiple arcs, that is, there is at most one arc (i, j) from node i to node j. Each arc has an associated non-negative weight represented as $w(i, j)$.

2.1 Problem definition

In this section we formally describe the k-multicolour path problem in edge-coloured graphs. For the following definitions, we refer to an edge-coloured graph simply as a graph unless otherwise stated. When exemplifying any of the definitions, we use the graph depicted in Figure 1 on page 3 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 $G = (V, E)$, a set of x colours $C = \{1, \ldots, x\}$, $x \in \mathbb{N}$ and a function $c : E \mapsto 2^C$ 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).

Let P be a path between two nodes of G and $E(P)$ the edge set of P. The available colours for the path P, denoted by $AC(P)$, is defined as $AC(P) = \bigcap_{e \in E(P)} c(e)$. We then associate a subset of colours from $AC(P)$ to P. Let $C(P) \subseteq AC(P)$ be the set of colours associated with P. 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 these k colours.

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

Definition 1 (Feasible path) *A path P_i in G is feasible with respect to k_i if it contains at least k_i colours, i.e., $|C(P_i)| \geq k_i$.*

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

Definition 2 (Compatible paths) *Two feasible paths P_i and P_j in G, with respect to k_i and k_j respectively, are compatible if one of the following two cases holds:*

(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, 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.

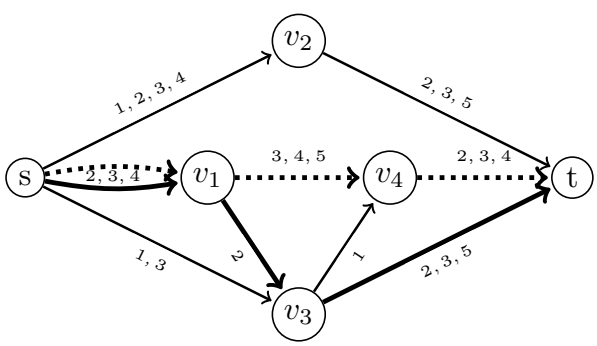

Figure 3: Compatible paths

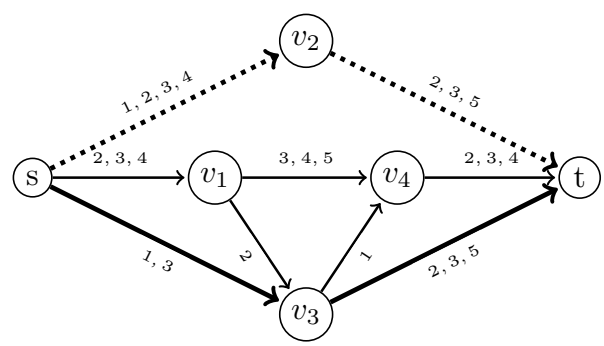

Figure 4: Absolutely compatible paths

Definition 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 they are mutually arc-disjoint.*

The highlighted paths in Figure 4, $P_1 = (s, v_2, t)$ and $P_2 = (s, v_3, t)$ are absolutely compatible even though both use colour 3, since they do not share any edges.

We are now ready to formally state our problems.

An input instance consists of a graph $G = (V, E)$, a set of colours C, where each edge $e \in E$ is coloured with colours $c(e) \subseteq C$, 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.

Definition 4 (Single k-Multicolour Path Problem (SMP)) *Input: A graph G, a set of colours C, edges colour function c, and a tuple $(s, t, k, 1)$ where s and t are source and target nodes, k is the number of required colours and 1 stands for the number of paths. Output: A single feasible shortest (s, t)-path containing at least k colours.*

Considering the graph from Figure 1 on page 3, the solution to the input $(s, t, 2, 1)$ would be the path $P = (s, v_2, t)$ (highlighted in Figure 2). Although the path $P' = (s, v_1, v_4, t)$ is feasible with respect to $k = 2$, P is the only shortest feasible path. For $k > 2$, there is no solution possible for that graph.

Definition 5 (Multiple k-Multicolour Paths Problem (MMP)) *Input: A graph G, a set of colours C, edges colour function c, and a tuple (s, t, k, p) where s and t are source and target nodes, k is the number of required colours and p stands for the number of required paths. Output: A set of p compatible (s, t)-paths, $P_1, \ldots, P_p$, with total minimum weight (i.e. minimum $\sum_{i=1}^{p} \sum_{e \in P_i} w(e)$), so that the sum of colours used by all paths is at least k, i.e., $\sum\limits_{i=1}^{p} |C(P_i)| \geq k$. We require that each path contains at least one colour, i.e, $|C(P_i)| \geq 1$.*

In Figure 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 6 (Absolute Multiple k-Multicolour Paths Problem (AMMP)) *The problem is the same as in the MMP problem, but now we require that the p paths must be pairwise absolutely compatible.*

Referring to Figure 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 $\mathrm{AMMP_d}$).

We assume that the colour requirement is $k \geq 2$, since the case $k = 1$ can be easily solved by computing $|C|$ shortest paths (or computing p minimum multi paths), one for each possible colour, and picking the path (or paths for MMP and AMMP) with minimum weight.

Also note that if k is bounded by a constant then all problems can also be solved in polynomial time, since one can enumerate all $\binom{|C|}{k}$ combinations of k colours in polynomial time, then for each combination construct a graph with the edges containing only these colours and solve a minimum path (multi path) problem for each possible graph.

3 Problem complexity

In this section we prove that the $\mathrm{SMP_d}$, $\mathrm{MMP_d}$, and $\mathrm{AMMP_d}$ problems are NP-complete. As a consequence, the optimization versions are NP-hard. We also establish that these problems are hard to approximate.

3.1 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 polynomial time that a given path is a (s,t)-path and uses k colours.

We present now a reduction from the 3CNF-SAT [17] to the $\mathrm{SMP_d}$ 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 $(G, C, s, t, n, 1)$ of 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 5 and a structure representing a clause K_j is depicted in Figure 6.

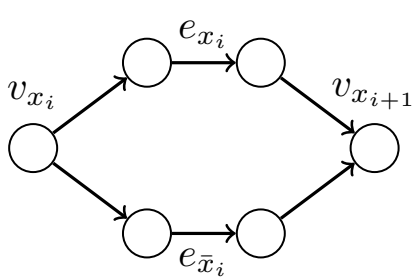

Figure 5: Structure for variable x_i

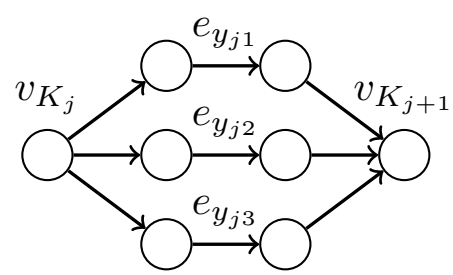

Figure 6: Structure for clause K_j

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 the one which represents the negation of z_i, and (2) unlabelled edges containing all $2n$ colours.

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 makes the clause be satisfied.

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 polynomial time. The complete graph is depicted in Figure 7 on the next page.

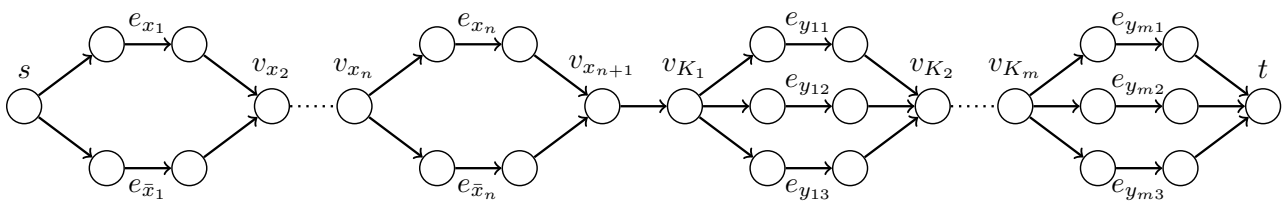

Figure 7: Graph built from a 3CNF-SAT instance

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 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. $\square$

3.2 Multiple paths decision problems

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

Theorem 2 *The MMP$_d$ problem is NP-complete.*

Proof. We reduce the Set Cover problem [17] to the MMP$_\mathrm{d}$ 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 polynomial time. The complete graph is depicted in Figure 8.

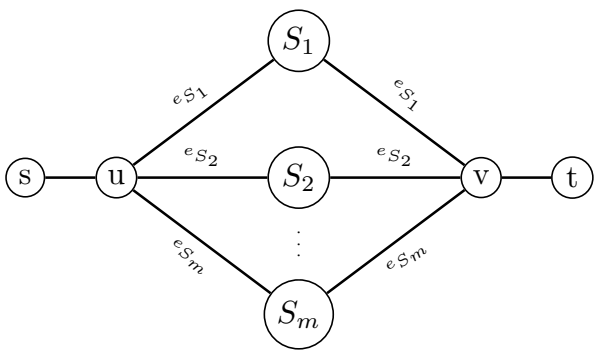

Figure 8: Graph built from an instance of the Set Cover problem

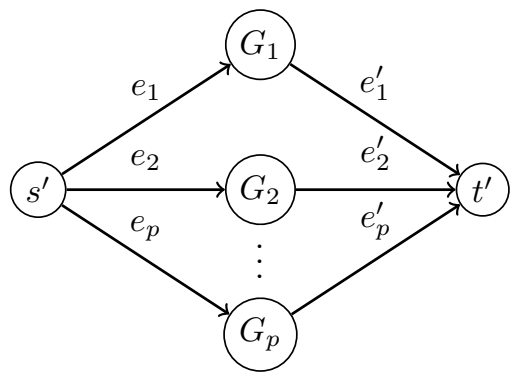

Figure 9: Reduction for absolutely compatible paths

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. Note that the n colours used must be distinct, one for each element, since all paths are compatible and share the edges (s,u) and (v,t). 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, $|\mathcal{S}'| \leq K$ (the less than or equal comes from the fact that different paths may use the same edges as long as they use different colours). 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.

Now assume $\mathcal{S}' = \{S_1, \ldots, S_{K'}\}$, $K' \leq K$, is a solution for an instance I of the Set Cover. First 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. For each set S_i in the solution $\mathcal{S}'$ we will have a (s,t)-path P_i. Path P_i uses the edges e_{S_i} corresponding to 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. In order to obtain the remaining $K - K'$ paths we can construct extra compatible paths. For paths P_i that use more than one

colour, we can split P_i into two or more paths using different colours in each one. We repeat this process until we obtain K paths. These paths are compatible since we split the colours of one path among the created ones and it is always possible to construct K paths since $K \leq n$.

□

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 9 on the previous 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 absolutely compatible paths in G using exactly np colours. □

3.3 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 *For any polynomial time computable value $\alpha > 1$, there is no α-approximation algorithm for the SMP, MMP, or AMMP problems unless $P = NP$.*

Proof. Let I be an instance of the decision version of the problem SMP_d (or MMP_d, or AMMP_d). Suppose for the purpose of contradiction that $\mathcal{A}$ is an α-approximation algorithm for the optimization problem SMP (or MMP, or AMMP respectively). Let I' be an instance of the optimization version of SMP (MMP, AMMP, respec.) that is equal to I except that each arc in the graph is given a weight equal to 1, and a new arc (s, t) is included with weight $\alpha|V| + 1$ (or p new (s, t) arcs with weight $\alpha|V|p + 1$ each, for MMP and AMMP), and with k new colours. Note that the new added arcs can be replaced with new paths with the same cost using new vertices in order to construct a graph without multiple edges. If I admits a feasible solution then $\mathcal{A}(I')$ must produce a solution with cost $\leq \alpha|V|$ (or $\leq \alpha|V|p$ for MMP and AMMP). If I does not admit a feasible solution then $\mathcal{A}(I') \geq \alpha|V| + 1$ since the only solution must use the new added arc (or $\geq \alpha|V|p + 1$ for MMP and AMMP). Then $\mathcal{A}(I')$ can be used to decide SMP_d (respec. MMP_d, AMMP_d) in polynomial time. □

It is interesting to think of another optimization measures, instead of path weights, in theses problems. As an example, an interesting problem is the following: suppose a SMP (MMP or AMMP) problem where edges do not have weights and the objective is to find a path (or multi paths) that maximizes the number of used colours. We leave the study of approximation algorithms for these variations of the problem as a future research topic.

4 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 Sections 4.1 and 4.2. ILP formulations for SMP, MMP, and AMMP are presented in Section 4.3.

4.1 Branch and Bound for SMP

For the description of our branch and bound solution, we need the concept of a *partial path* and *potential solution*. A partial path P_s^l is just a regular path from the source node s to some node l. A partial path P_s^l is a potential solution if it satisfies $|c(P_s^l)| = |\cap_{e \in P_s^l} c(e)| \geq k$, i.e the number of colours available in the path is at least k. We call it *potential* because it has not yet reached the destination node t. The weight of a potential solution P_s^l is $w(P_s^l) = \sum_{e \in P_s^l} w(e)$.

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, which is the one with the minimum estimated cost to reach t. Let P_s^l be this promising potential solution. We expand it by considering the node l and checking, for each out-neighbour u of l: (1) whether the number of common colours in the expanded path including the neighbour u is 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, the node representing P_s^u is discarded; otherwise, a new potential solution $P_s^u = P_s^l + (l,u)$ including the edge (l,u) to P_s^l is added to the search tree (in a heap). The cost of this new potential solution is $w(P_s^u) = w(P_s^l) + w(l,u)$.

The starting potential solution is composed of the 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 potential solution to evaluate, we use a best bound strategy: we select the potential solution with least *estimated weight* to the destination. The estimated weight is the sum of weights of the edges in the potential solution plus a lower bound in the weight of a 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. In short, the **estimated weight** of a potential solution is $w(P_s^l) + d(l,t)$, where $d(l,t)$ is the shortest distance between l and t disregarding the colours.

Let G_{rev} be the path obtained from G by reversing the direction of all arcs. In our algorithm, we get the shortest distance $d(i,t)$ between all nodes i and t, 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 potential solution, if it contains the destination node as the last node, then this path corresponds to the optimal solution, since we are using a best-bound strategy.

4.2 Branch and Bound for AMMP

A set of partial paths is arc-disjoint if and only if their partial paths are pairwise mutually arc-disjoint. A potential solution for the AMMP consists of p arc-disjoint partial paths that have at least k colours available in total. Then the same algorithmic idea in the SMP problem can be expanded to the AMMP problem. The weight of a potential solution is the sum of weights of its p partial paths. The estimated weight is the sum of the estimated weights to t of these partial paths.

The starting potential solution is composed of p copies of P_s^s. At each iteration of the algorithm, we expand one partial path of the most promising potential solution, repeating this procedure until all partial paths in a potential solution reach the destination node or no potential solution is left to evaluate.

4.3 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 presence 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.

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:

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

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.

4.3.1 ILP model for SMP:

The formulation presented in this section was proposed in [7] and we replicate it here for easy of comparison with our own formulation (ILP fast) to be introduced in the following.

$$
\text{(ILP original)}\quad
\begin{aligned}
&\min \sum_{(i,j)\in E} f_{ij}\cdot w_{ij} \\
&\text{s.t.} \quad \sum_{j\in\delta^+(i)} f_{ijx} - \sum_{j\in\delta^-(i)} f_{jix} = \begin{cases} c_x, & \text{if } i=s \\ -c_x, & \text{if } i=t, \\ 0, & \text{otw} \end{cases} && \forall i\in V, \forall x\in C && (1)\\
&\qquad \sum_{x=1}^{C} c_x = k && && (2)\\
&\qquad \sum_{x=1}^{C} f_{ijx} = k\cdot f_{ij} && \forall (i,j)\in E && (3)
\end{aligned}
$$

Constraints (1), are flow conservation constraints, such that a path is formed from s to t for each used colour. Constraint (2) ensures the number of colours used is equal to k. Constraints (3) ensure that just one path from s to t is formed.

4.3.2 New Faster ILP model for SMP:

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

$$
\text{(ILP fast)}\quad
\begin{aligned}
&\min \sum_{(i,j)\in E} f_{ij}\cdot w_{ij} \\
&\text{s.t.} \quad \sum_{j\in\delta^+(i)} f_{ij} - \sum_{j\in\delta^-(i)} f_{ji} = \begin{cases} 1, & \text{if } i=s \\ -1, & \text{if } i=t, \\ 0, & \text{otw} \end{cases} && \forall i\in V && (4)\\
&\qquad \sum_{x=1}^{C} c_x = k && && (5)\\
&\qquad \sum_{x\in c(i,j)} c_x \geq k\cdot f_{ij} && \forall (i,j)\in E && (6)\\
& && && (7)
\end{aligned}
$$

Constraints (4) are flow conservation constraints, such that a single path is formed from s to t. Constraint (5) ensures that k colours are used. Constraints (6) ensure that each edge in the path contains the same k colours.

4.3.3 ILP models for MMP and AMMP:

Now we present a formulation for both versions of the multipath problem: 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. We denote by $[P]$ the set $\{1, \ldots, P\}$.

$$
\begin{aligned}
&\min \sum_{p\in[P]} \sum_{(i,j)\in E} f_{pij} \cdot w_{ij} \\
&\text{s.t.} \qquad \sum_{p\in[P]} \sum_{x\in C} c_{px} = k && (8)\\
&\qquad f_{pijx} \leq c_{px} \qquad (i,j) \in E, p \in [P], x \in C && (9)\\
&\qquad \sum_{j\in\delta^+(i)} f_{pijx} - \sum_{j\in\delta^-(i)} f_{pjix} = \begin{cases} c_{px}, & \text{if } i = s \\ -c_{px}, & \text{if } i = t \\ 0, & \text{otw} \end{cases} \qquad i \in V, p \in [P], x \in C && (10)\\
&\qquad f_{pijx} \leq f_{pij} \qquad (i,j) \in E, x \in C, p \in [P] && (11)\\
&\qquad \sum_{j\in\delta^+(i)} f_{pij} - \sum_{j\in\delta^-(i)} f_{pji} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otw} \end{cases} \qquad i \in V, p \in [P] && (12)\\
&\qquad \sum_{x\in C} c_{px} \geq 1 \qquad p \in [P] && (13)\\
&\qquad \sum_{p\in[P]} f_{pijx} \leq 1 \qquad (i,j) \in E, x \in C && (14)
\end{aligned}
$$

(ILP Mult)

Constraints (8) ensure that the P paths have k colours in total. Constraints (10) are flow conservation constraints and together with (9) ensure that if a path is using some colour then that colour is accounted for in the variable c_{px}. Constraints (11) and (12) guarantee that for each p, a single path using exactly the same colours is formed. Constraint (13) ensures at least one colour is used in each path, and constraint (14) ensures the paths are compatible.

By modifying ILP Mult, we can extend it for the absolutely compatible paths case. To do that, replace constraint (14) by a restriction ensuring the paths are disjoint:

$$\sum_{p\in[P]} f_{pij} \leq 1, \quad (i,j) \in E$$

5 Heuristics

We developed some heuristics to these problems. They 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 times.

5.1 Dijkstra-based heuristics for SMP

We propose three heuristics for the SMP problem based on the Dijkstra's algorithm [10]. 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 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 the 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.

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 them in details.

5.1.1 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 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 $O(|V|^2)$.

5.1.2 DijkstraX heuristic:

The problem with the previous DijkstraQ algorithm 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. 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 $\text{score}(i) = \text{dist}(i) - \frac{D}{k}c(i)$, where $\text{dist}(i)$ is the shortest distance from the source to node i, $c(i)$ is the set of common colours in the path from the source to i found so far, and D is the minimum path distance between the source s and target t. Notice that if the minimum path between s and t has k colours then its score is 0. The idea of this score function is that among the paths that have at least k colours, we are searching for the one with minimum difference from the shortest path. The time complexity of the DijkstraX algorithm is the same as Dijkstra's.

5.1.3 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 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.

5.2 Dijkstra-based heuristics 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 a Dijkstra-based algorithm for the multipath 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 one 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 colours selected for P on its arcs in G. For the AMMP, because we want arc-disjoint paths, we removed all arcs in P from G. The time complexity for both algorithms is $O(p\{|E| + |V| \log |V|\})$. We call this heuristic MMPMin.

5.3 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 [28], 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 (we call this algorithm by Intersection).

Performance-wise, the IntersectionFast algorithm can be implemented with time complexity $O(|E||C| + |V|\log|V|)$. For the Intersection algorithm, it can be implemented with time complexity $O(|C|(|E| + |V|) + |V|\log|V|)$.

6 Computational results

Because heuristic solutions give no guarantee on the quality of the result obtained, we performed a simulation with the developed algorithms and in this section we present the results obtained. Our aim is to assess the performance of the algorithms in practical terms, comparing them with the exact Branch and Bound and ILP algorithms.

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 an edge-coloured graph G. Each instance is solved by the algorithm which then returns the p-paths found. If no path was found, we say the algorithm *blocked* on that instance. For the SMP problem, $p = 1$. For the MMP and AMMP, $p = 2$ due to the fact that few applications require a large number of multiple paths in practice [7].

For the ILP models we used the CPLEX tool [15]. All simulations rounds were executed in a 2.4GHz quad-core machine with 8GB of RAM memory. All graphs displaying averages in this section show a 95% confidence interval.

In Section 6.2 we present results with instances representing real networks making use of a discrete event simulator to assess the blocking ratio of the algorithms.

6.1 Execution time and solution cost

For each execution round we keep track of the execution time of each algorithm. If valid p-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 $G(n, d_a, d_c)$ 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 reach about $|E||C| \cdot d_c$ colours in total in the graph. The purpose of varying the edge and colour densities is to simulate scenarios with different network loads. The pairs (d_a, d_c) used, in percentage, were: $(10, 10)$, $(10, 40)$, $(10, 70)$, $(30, 40)$, $(30, 60)$, $(30, 80)$, $(40, 10)$, $(40, 40)$, $(40, 70)$, $(60, 30)$, $(60, 60)$, $(60, 80)$, $(70, 10)$, $(70, 40)$, $(70, 70)$, $(90, 40)$, $(90, 60)$, and $(90, 80)$.

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.

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 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.

Following the methodology in [7], 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 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 10 depicts the effect of the densities on the Branch and Bound algorithm for the SMP problem. 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.

Effect of arc and colour densities on Branch and Bound algorithm for SMP problem

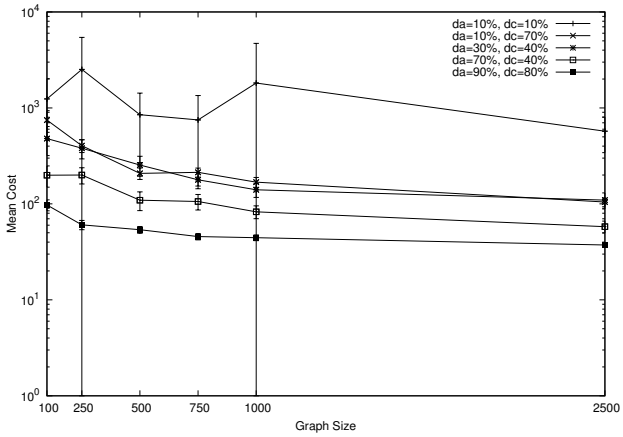

Figure 10: Mean cost

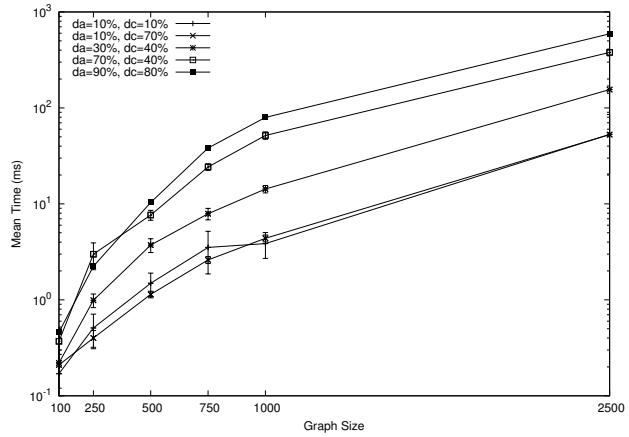

Figure 11: Mean execution time

Likewise, Figure 11 on the previous 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.

Figure 12 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 since more nodes are removed in the search tree when k is high, and then the algorithm tends to run faster. (Figure 13).

Effect of number of colours in request for Branch and Bound algorithm for SMP problem

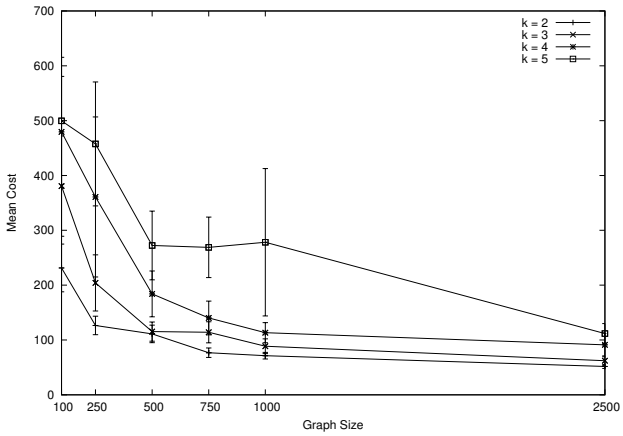

Figure 12: Mean solution cost

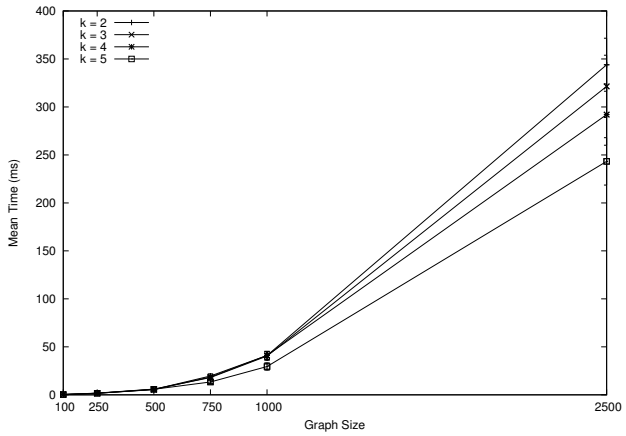

Figure 13: Mean execution time

To compare the algorithms graphically, we present their *performance profiles*, a comparison method proposed by Dolan and More [11]. 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 14 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 14 and Figure 15 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 15, the point $(100, 0.901)$ for the Intersection algorithm indicates that it solved 90.1% of the instances in no more than 100 milliseconds.

Some relevant remarks regarding the simulations for the SMP problem. As it is easily seen from the graphs in Figure 14 on the next page and Figure 15 on the following page, the ILPs are easily outperformed both in solution cost and 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

Performance profiles for the SMP problem

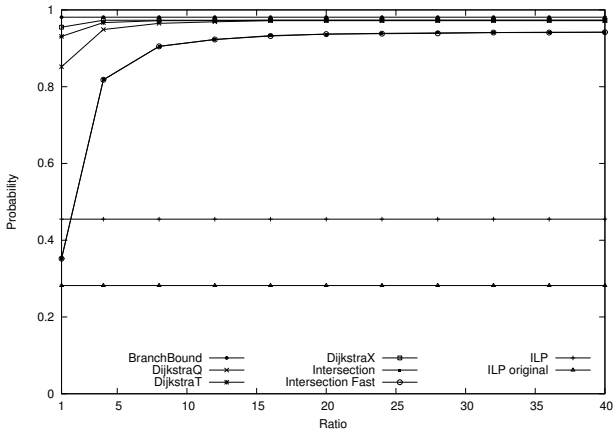

Figure 14: Solution cost

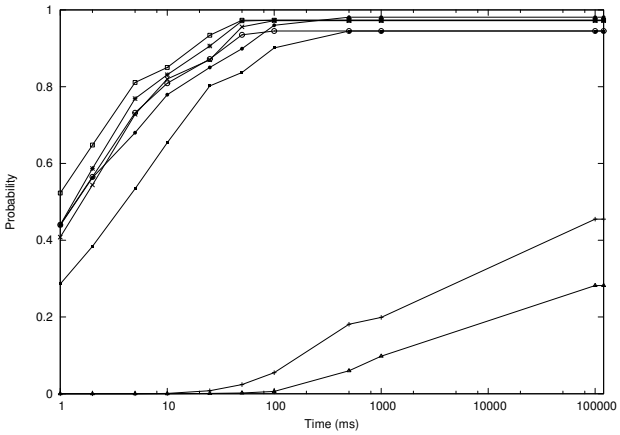

Figure 15: Execution time

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, DijkstraX was best both in terms of solution cost (95% solved with best cost) but also in terms of time, being able to solve 53% 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 16 and Figure 17 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 18 and Figure 19 show the performance profiles for the MMP problem. For this problem, there was no Branch and Bound implementation. 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.

Performance profiles for the AMMP problem

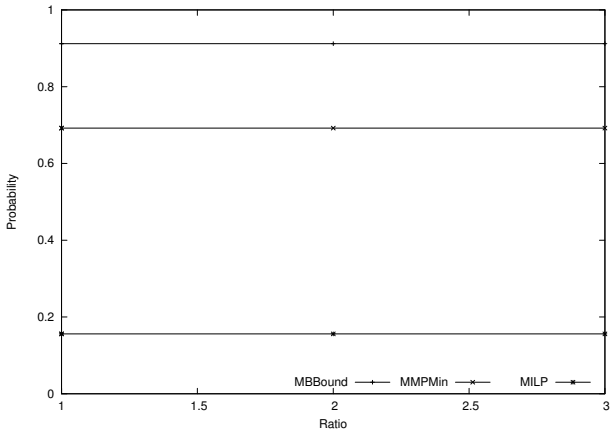

Figure 16: Solution cost

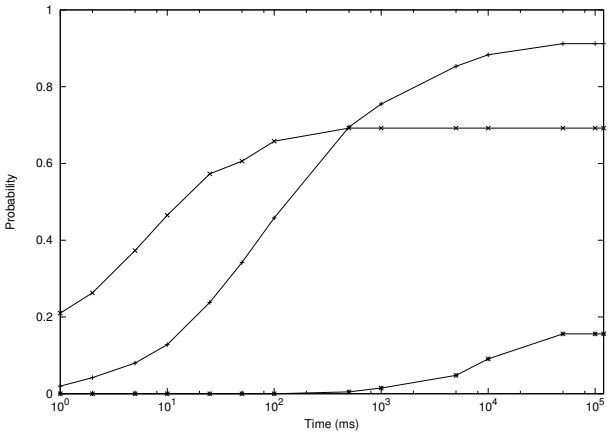

Figure 17: Execution time

Performance profiles for the MMP problem

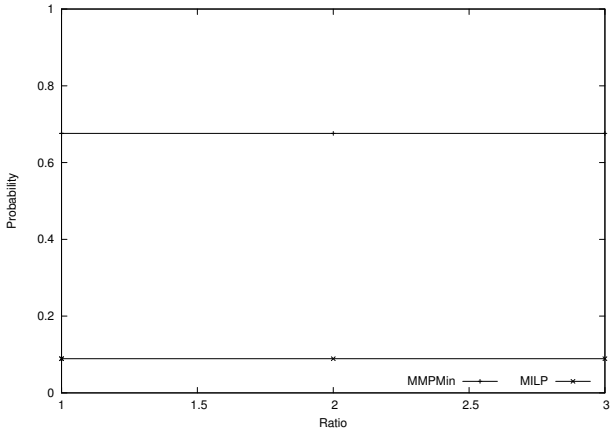

Figure 18: Solution cost

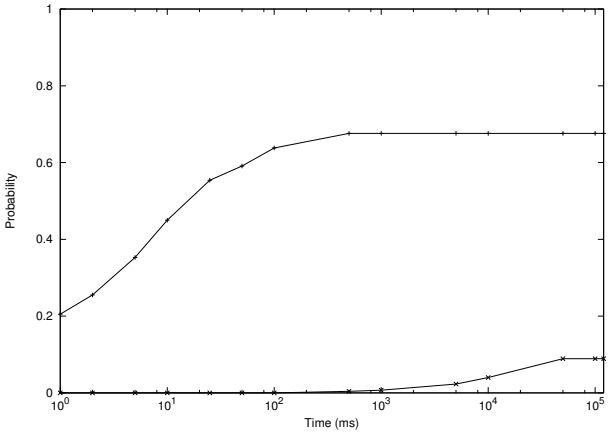

Figure 19: Execution time

6.2 Blocking ratio

We now discuss the blocking ratio results. The blocking ratio defined in [7] 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 [7, 12] 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.

The network topologies used for measuring blocking ratios are a grid 5x5 network, with 25 nodes and

40 bidirectional links, the USA long-distance Mesh Network, with 24 nodes and 43 bidirectional links, the NSFNet network, with 16 nodes and 25 bidirectional links, and the Pan-European Network, with 28 nodes and 41 bidirectional links. It is assumed each fibre carries 8 colours in all topologies.

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.

In Figures 20, 21, 22 and 23 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.

Blocking ratios for the SMP problem

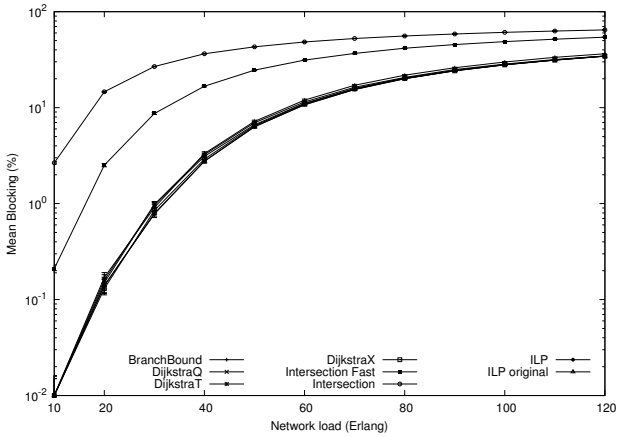

Figure 20: Grid 5x5 network

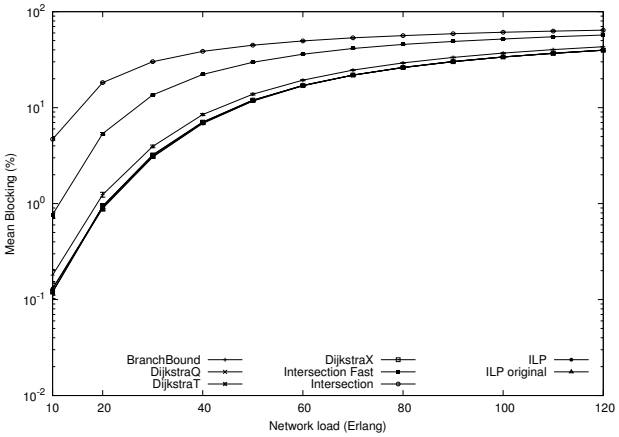

Figure 21: NSFNet network

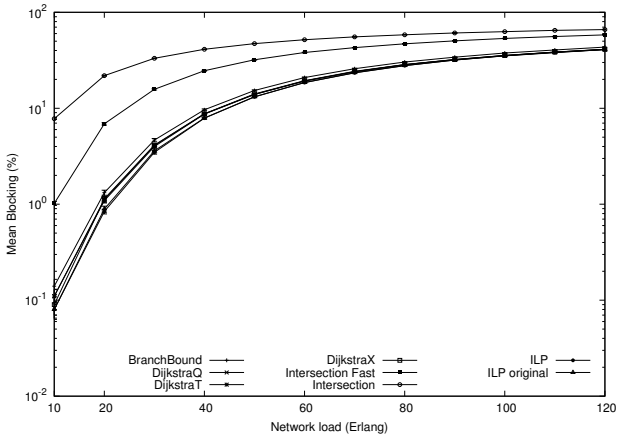

Figure 22: PanEU network

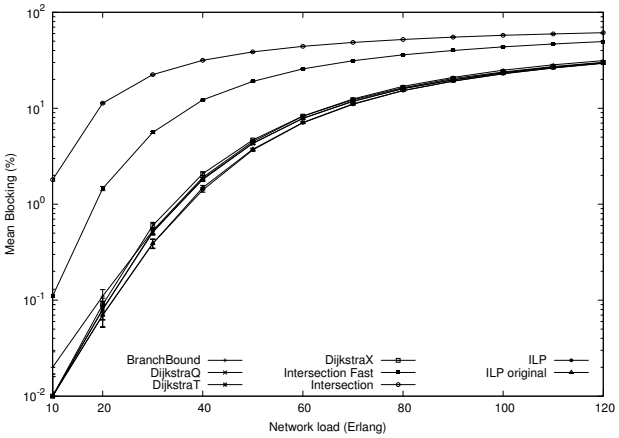

Figure 23: USA network

In Figures 24, 25, 26 and 27 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 show in the Grid 5x5 for 10 and 20 Erlang; MMPMin blocked 10 times more than the Branch and 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.

Blocking ratio for the AMMP problem

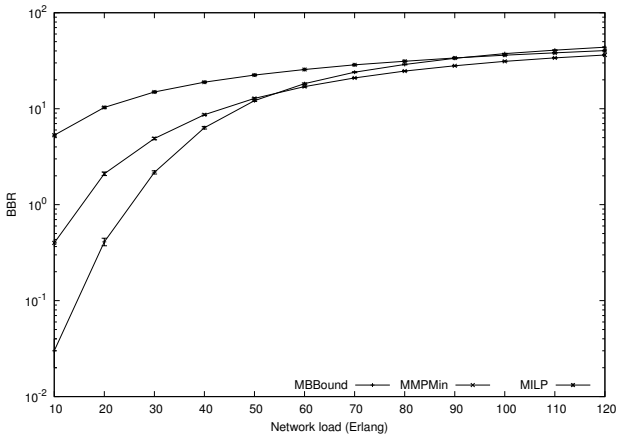

Figure 24: Grid 5x5 network

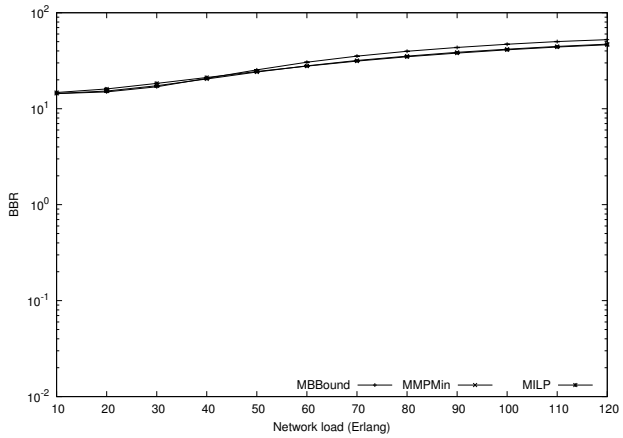

Figure 25: NSFNet network

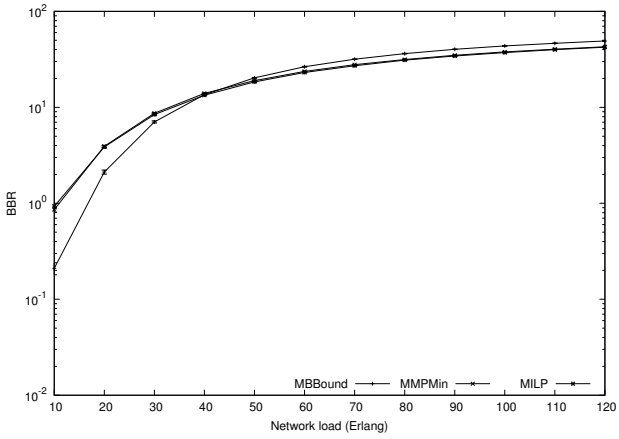

Figure 26: PanEU network

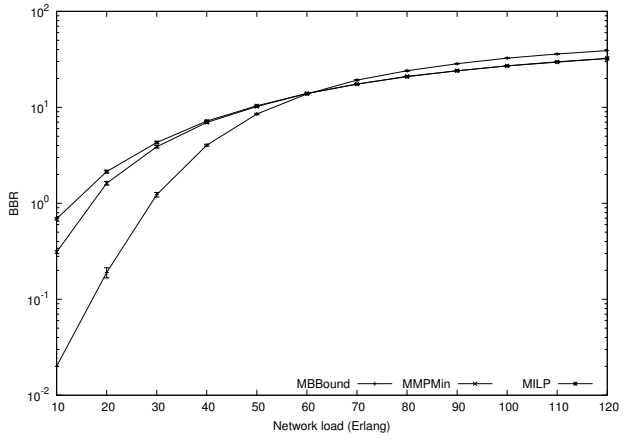

Figure 27: USA network

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 Figures 28, 29, 30, and 31. As with the AMMP problem, the biggest difference between the algorithms is seen for the Grid 5x5 network.

Blocking ratio for the MMP problem

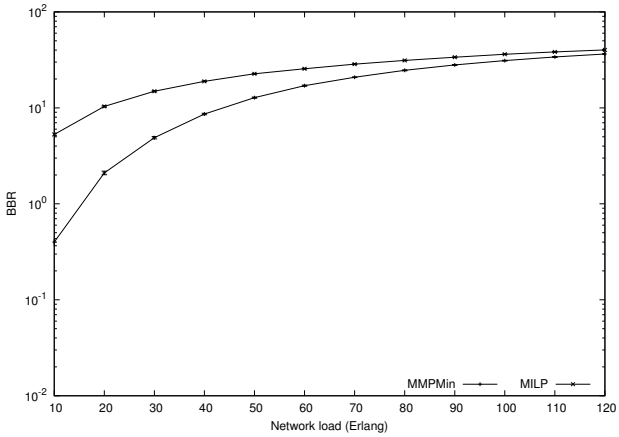

Figure 28: Grid 5x5 network

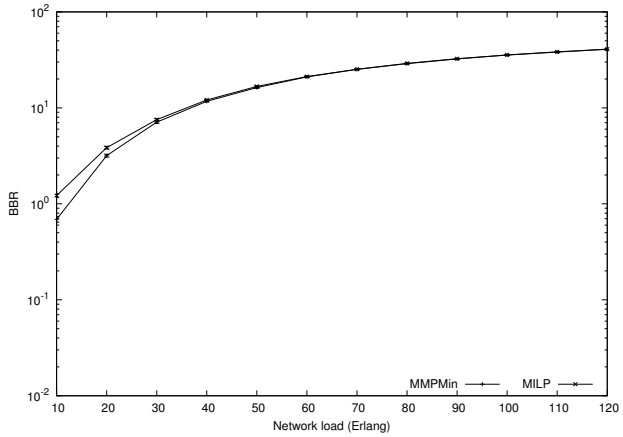

Figure 29: NSFNet network

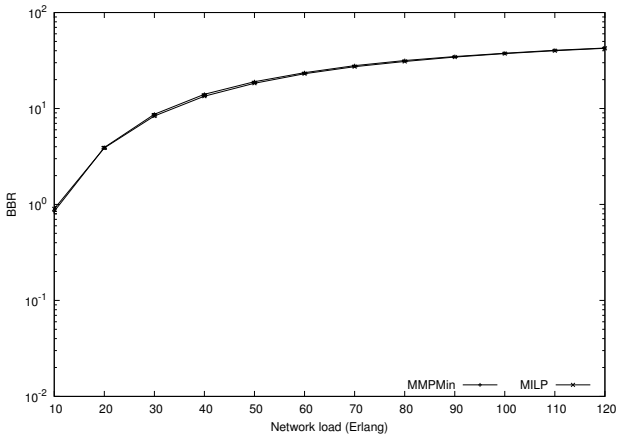

Figure 30: PanEU network

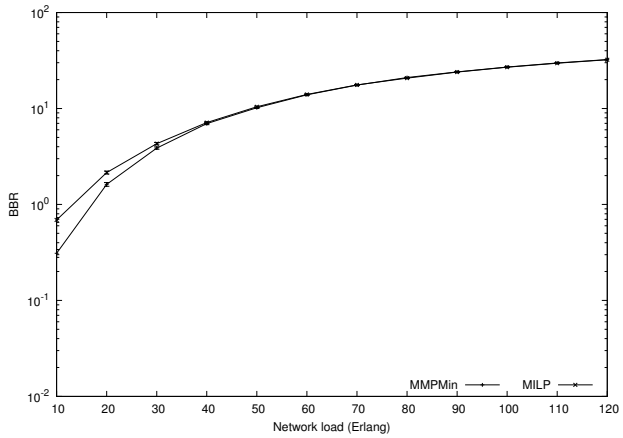

Figure 31: USA network

7 Conclusions

This work discussed the problem of finding multicolour paths in edge-coloured graphs. The network related 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 an 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 not much difference on the blocking behaviour of all algorithms.

Further work based on these problems can explore classes of graphs in which exact solutions can be easily obtained.

Another possibility is to investigate approximation algorithms when the objective function is to find a (s, t)-path that contains the maximum number of possible colours.

8 Acknowledgements

This work was partially supported by CAPES and CNPq.

References

[1] 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.

[2] Niko Beerenwinkel, Stefano Beretta, Paola Bonizzoni, Riccardo Dondi, and Yuri Pirola. Covering pairs in directed acyclic graphs. *Comput. J.*, 58(7):1673–1686, 2015.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

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

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

[14] 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.

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

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

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

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

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

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

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

[22] 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.

[23] Simeon C. Ntafos and S. Louis Hakimi. On path cover problems in digraphs and applications to program testing. *IEEE Trans. Software Eng.*, 5(5):520–529, 1979.

[24] Panos M Pardalos, Thelma Mavridou, and Jue Xue. The graph coloring problem: A bibliographic survey. In *Handbook of combinatorial optimization*, pages 1077–1141. Springer, 1998.

[25] Romeo Rizzi, Alexandru I. Tomescu, and Veli Mäkinen. On the complexity of minimum path cover with subpath constraints for multi-assembly. *BMC Bioinformatics*, 15(S-9):S5, 2014.

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

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

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

[29] 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.