INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Exploring optimal solutions for planning optical networks with graph coloring techniques

Inês Ferreira Gomes

Master in Telecommunications and Computer Engineering

Supervisor
Dr. Luís Gonçalo Lecoq Vences e Costa Cancela, Assistant Professor
Iscte-Instituto Universitário Lisboa

Co-Supervisor
Dr. João Lopes Rebola, Associate Professor
Iscte-Instituto Universitário Lisboa

November, 2021

Department of Information Science and Technology

Exploring optimal solutions for planning optical networks with graph coloring techniques

Inês Ferreira Gomes

Master in Telecommunications and Computer Engineering

Supervisor
Dr. Luís Gonçalo Lecoq Vences e Costa Cancela, Assistant Professor
Iscte-Instituto Universitário Lisboa

Co-Supervisor
Dr. João Lopes Rebola, Associate Professor
Iscte-Instituto Universitário Lisboa

November, 2021

Às minhas avós

Acknowledgments

Thanks to Iscte, for being the home it has been over these five years full of sharing, teaching, new knowledge, experiences and opportunities, without which my academic path would have never been as rich as it was. My sincere thanks to every single person who crossed paths with me, in any adventure I committed myself to. A big thanks to Instituto de Telecomunicações for all the support necessary to carry out this work.

Thanks to my Professors, Luís Cancela and João Rebola, for giving me the basis, throughout my academic path, the background knowledge and the basis to choose this area of work and for the help, guidance, and advice in every detail during all these months.

To the special people that Lisbon brought to my life, thank you for welcoming me so well, for listening to me when I needed it the most and for being with me in all key moments. In particular, João Antão (there are no words!), André Glória, Rita Peixoto, Henrique Vitória and Eduardo Ferreira, for being the dedicated friends you are. You were a fundamental part in this process that did not seem as lonely as I thought it would be.

To my childhood friends and friends from Leiria, for always understanding me in those moments when I was not present, because of the focus I put into my studies and particularly into this work. Especially to Adriana, Mariana and Eugénia for always being there in all the moments when I most needed your companionship.

To my family, for being my handful of love, luck, courage, strength and dedication. To my uncles and cousins, for believing me in each step of my academic years. To my grandfather, for being one of the most present people in my life. A huge special thanks to my parents and brother, for always being there and for not letting the miles that separate us daily have the weight of distance, but the comfort of closeness. Thank you for every call, advice, help, which were always indispensable. Finally, to my twin sister, for literally living every doubt, vent and small victories with me: this really had to be done together!

Resumo

As redes óticas são fundamentais para as comunicações atuais, tendo o seu planeamento uma grande importância. O encaminhamento e a atribuição de comprimentos de onda são funções essenciais no planeamento do transporte de dados nestas redes. Este trabalho pretende estudar diferentes técnicas de coloração de grafos para a atribuição de comprimentos de onda nas redes óticas.

Analisamos e comparamos diferentes algoritmos de atribuição de comprimentos de onda, como a heurística Greedy, um algoritmo exato baseado em Programação Linear Inteira (PLI) e um algoritmo de procura meta-heurístico, o Tabu Search. Estudaram-se estes algoritmos para diferentes redes reais, diferentes topologias lógicas e de caminhos, com o objetivo de analisar o número de cores e o tempo computacional obtidos por cada algoritmo estudado.

Concluímos que o algoritmo exato baseado em PLI, embora devolva sempre um número de cores ótimo, aplica-se apenas a redes com menos de 20 nós, devido ao seu tempo computacional elevado. Também se concluiu que o Tabu Search devolve sempre os mesmos resultados que o algoritmo exato, para as redes reais, mas com mais rapidez computacional. Concluiu-se também para as redes reais, que o Greedy estima o mesmo número de cores que o Tabu Search. No entanto, com matrizes geradas aleatoriamente, nalguns casos, o Tabu Search retorna menos cores que o Greedy. Por exemplo, o Tabu Search devolve menos 35 cores que o Greedy para uma matriz de 1000 caminhos e 50% de probabilidade de cada caminho ter uma ou mais ligações pertencentes aos restantes caminhos.

Palavras-Chave: Atribuição de comprimentos de onda, Coloração de grafos, Greedy, Programação Linear Inteira, Redes óticas, Tabu Search

Abstract

Optical networks are crucial in today's communications, so their planning is of paramount importance. Routing and Wavelength Assignment (WA) are essential planning functions to transport the data in these networks. This work aims to study different graph coloring techniques for WA in static optical networks.

We analyze and compare different graph coloring algorithms such as the Greedy heuristic, an exact algorithm based on Integer Linear Programming (ILP) and the Tabu Search meta-heuristic, for WA in optical networks. The studies are performed considering several real network scenarios, different logical and path topologies, and the number of colors and computation time of each one of the studied algorithms is analysed.

We have concluded that the exact algorithm based on ILP, although giving always the optimum number of colors, is only applicable to networks with less than 20 nodes, due to its higher computation time. It was also concluded that the Tabu Search algorithm gives always the same results as the exact algorithm, for the real networks studied, but in a much faster computation time. Finally, we have concluded that the common Greedy algorithm performs as well as the Tabu Search algorithm, for all the real networks studied, but in some scenarios, with random path matrices, the Tabu Search gives a lower number of colors than the Greedy algorithm. For example, the Tabu Search gives less 35 colors than the Greedy algorithm for a random path matrix with 1000 paths and a 50% probability that each path has one or more links being used by the other paths.

Keywords: Graph Coloring, Greedy, Integer Linear Programming, Optical Networks, Tabu Search, Wavelength Assignment

Contents

List of Figures

List of Tables

List of Acronyms

B&S: Broadcast and Select.

CDC: Colorless, Directionless and Contentionless.

DSATUR: Degree of Saturation.

DWDM: Dense Wavelength Division Multiplexing.

GbE: Gigabit Ethernet.

ILP: Integer Linear Programming.

IP: Integer Programming.

OADM: Optical Add-Drop Multiplexer.

OTN: Optical Transport Network.

OTU: Optical Transport Unit.

R&S: Route and Select.

RLF: Recursive Largest First.

ROADM: Reconfigurable Optical Add-Drop Multiplexer.

RWA: Routing and Wavelength Assignment.

STM: Synchronous Transport Module.

WA: Wavelength Assignment.

WDM: Wavelength Division Multiplexing.

WSS: Wavelength Selective Switch.

YKSP: Yens k-Shortest Path.

CHAPTER 1

Introduction

1.1. Motivation and Context

Routing and Wavelength Assignment (RWA) are fundamental functions to transport data in an efficient way in optical networks [**1**]. Routing is responsible for finding the best path for a given traffic demand, and Wavelength Assignment (WA) is responsible for choosing an appropriate wavelength in that path to transport the given traffic demand taking into account the wavelength continuity and the distinct wavelength constraints [**2**].

Several techniques have been used to solve the WA problem, ranging from exact algorithms to heuristics that typically give a sub-optimal solution to the problem, but in a shorter time, like First-Fit or Most-Used algorithms [**2**], [**3**]. Graph coloring techniques, although applied to a large range of applications, such as constructing schedules, scheduling reservations or compiler register allocations, can be also used for WA in optical networks [**2**]. The most common graph coloring for WA is the Greedy algorithm [**4**]. Some studies have, however, used other Graph coloring algorithms for WA, such as the Degree of Saturation (DSATUR) and Recursive Largest First (RLF) [**5**], but for the majority of the networks studied, Greedy algorithm performs as well as these algorithms.

In this work the aim is to study more complex and more rigorous algorithms based on graph coloring techniques for WA in optical networks. In particular, we will study an exact algorithm based on an Integer Linear Programming (ILP) formalism [**4**] and the Tabu Search meta heuristic algorithm [**3**], [**4**], [**6**], for several network topologies. The respective number of colors and computation time are computed and compared with the Greedy algorithm performance [**5**]. The aim of these algorithms, considering a static network scenario, is to find the minimum number of wavelengths (i.e. colors) that satisfies all traffic demands, in a feasible and reasonable computational time.

Exact WA algorithms based on graph coloring techniques will give always the optimal solution, although they require enhanced computational resources and tend to have high computational times as the network size increases [**4**]. There are two common approaches for graph coloring using exact algorithms. The first one is called the backtracking approach, and the second one is to use Integer Programming (IP), which is a specific linear programming model type [**4**]. Linear programming is a general methodology for achieving optimal solutions to linear mathematical models. In this dissertation, we will use ILP to obtain exact solutions for the WA problem.

Due to the high computational requirements of exact algorithms, it is common to use heuristic algorithms when the network size increases, to achieve feasibility and reduce the computational time, but being aware that the solution could not be the optimal one [**7**]. In this dissertation, we also study the metaheuristic Tabu Search algorithm, which is used for solving different kinds of problems, such as optimization problems in network design [**8**]. For example, it has been used for solving RWA problems based on ILP formalisms [**6**], [**9**], [**10**]. Moreover, in [**7**], this algorithm has been used to solve graph coloring problems. But, to the best of our knowledge, there are no works that have used the Tabu Search algorithm as a graph coloring technique for WA in optical networks, as performed in this work.

1.2. Goals

The main goal of this work is to study more rigorous graph coloring algorithms for WA in optical networks, than the usual Greedy algorithm. In particular, we will study an exact algorithm based on ILP formalism and also the Tabu Search meta-heuristic algorithm. A performance study is going to be made for several network topologies - physical topologies, logical topologies and path topologies -, and a comparison between the studied algorithms and the Greedy is performed, in terms of the number of colors and also in terms of computation time. In this work, we consider the following real physical networks: rings with a different number of nodes, COST239 [**11**], NSFNET [**12**], UBN [**13**] and CONUS with 30 nodes network [**14**].

1.3. Dissertation Organization

This dissertation consists of five chapters with the following content.

In Chapter 1, the motivation of this study, goals and main contributions are presented.

Chapter 2 briefly describes the basic concepts in optical networks and describes summarily several algorithms to perform the RWA.

Chapter 3 presents the graph coloring basics. A brief discussion of the problem complexity is addressed. Also, the graph coloring algorithms that are studied in this work, Greedy, ILP and Tabu Search are explained and their corresponding pseudocodes presented.

Chapter 4 presents the application of the three graph coloring algorithms for WA in optical networks considering randomly generated path graphs, real regular and non-regular optical networks. For each algorithm, the number of colors/wavelenghts assigned is studied and conclusions regarding the computational time are presented. The Greedy algorithm is compared with the exact algorithm based on ILP for small sized networks. The Tabu Search algorithm is also compared with the Greedy algorithm considering random path graphs. Finally, all the three algorithms are applied to real networks and their performance is discussed.

Chapter 5 compiles the main conclusions of the whole work and presents future work possibilities.

1.4. Main Contributions

The main contributions of this work are the following:

- Performance study of an exact and a meta heuristic graph coloring algorithm for WA in optical networks. The exact algorithm is based on a ILP formalism and the meta-heuristic algorithm is the Tabu Search algorithm.
- The exact algorithm based on ILP was implemented and tested with several constraints, ranging from simple ones with slower computation times to more complex constraints with faster computation times.
- A performance comparison with the Greedy algorithm was performed. It was concluded that the Greedy with descending order gives similar number of colors than the Tabu Search for the real networks tested. However, using random path graphs, in several cases, the Tabu Search lead to a superior performance relatively to the Greedy.

CHAPTER 2

Planning Optical Networks Concepts

2.1. Introduction

RWA techniques are a complex problem in the context of optical network planning. These techniques are fundamental in establishing an optical path in a Wavelength Division Multiplexing (WDM) transport network. The routing function is responsible for selecting the set of connections between source and destination nodes and the WA corresponds to the reservation of a specific wavelength for the path resulting from the routing [**2**].

In section 2.2, some important basic network concepts are clarified, such as the different network topologies and the correspondent matrices used in the routing and WA functions. Section 2.3 contextualizes the RWA problem in optical networks describing the telecommunications network and its elements and architectures. In particular, subsections 2.3.1 and 2.3.2 describe, respectively, the routing and WA algorithms in order to understand the real context of this work.

2.2. Basic Network Concepts

A telecommunications network can be represented as a graph that is geometrically defined by a set of nodes or vertices joined through lines that represent the direct connection between two adjacent nodes, which are called edges or links [**4**]. This definition of a graph is represented as *G=(V, E)*, where the set of vertices is represented by $V = \{v_1, v_2,\dots, v_n\}$ and the set of links by the set $E = \{e_1, e_2,\dots, e_n\}$, where n is the total of vertices or edges. An example of a telecommunication network is shown in Figure 2.1 and associated to the respective graph $G = (\{v_1, v_2,\dots, v_n\}, \{e_1, e_2,\dots, e_n\})$. This example corresponds to a simple network with 5 nodes and 10 connections.

A link $e_1 = (v_1, v_2)$ between two nodes is characterized by its cost. Each link from e_1 to e_{10} has a distinct cost represented, e.g., by the connection delay, the distance between vertices, price, hops between nodes, examples can be found in [**2**]. A link can be unidirectional or bidirectional. Two unidirectional links implemented from a source node to a destination node in opposite directions result in a bidirectional link, which we assume is implemented by a pair of fibers, one for each transmission direction.

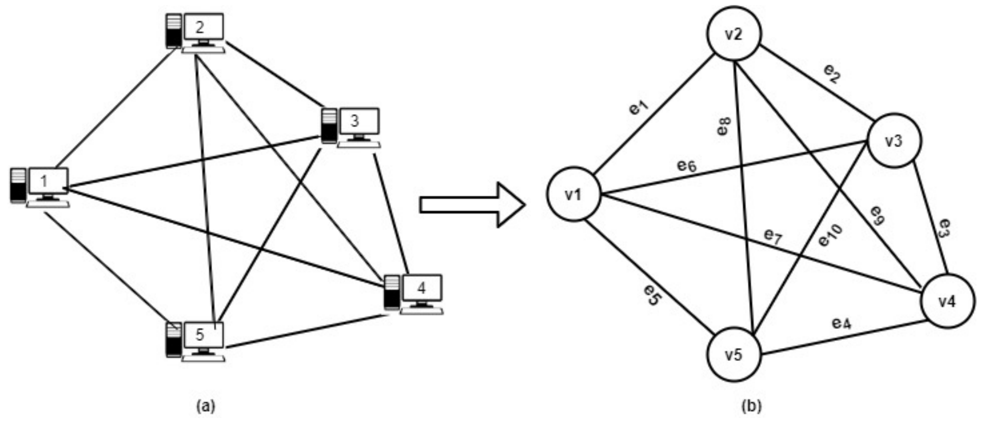


FIGURE 2.1. (a) Network and (b) Corresponding graph

A path can be described as a sequence of links that connect a sequence of distinct vertices. A *uv*-path is a path between two nodes: node *u* is a source node and node *v* is a destination node. A connection occurs when one of the possible paths between two nodes is reserved, depending on certain criteria or cost. If a *uv*-path exists between all pairs of vertices *u* and *v*, then a graph *G* is designated as connected; otherwise it is disconnected [**4**]. From another point of view, a path is directed if it can only be traversed in one direction. Otherwise, the path is undirected.

In the context of telecommunications networks, several topologies must be taken into account and the choice of the topology is one of the most important steps in network planning as it determines the development of the planning strategy and the type of service that the network provides [**2**].

The physical topology is defined by the way these nodes are physically linked and, on the other hand, the way how information is distributed between nodes determines the network logical topology [**2**].

Figures 2.2 (a) and 2.2 (b) represent, respectively, physical and logical topologies. Both topologies are respectively defined by the interconnection strategy between the nodes and by the way information (i.e. traffic) flows over the network and that traffic can be described by traffic demands or logical links. This is a result of assuming, according to the example of Figure 2.2, that the node v_1 distributes information to the other nodes and that all communications between the different nodes pass through node v_1.

The physical topology of the graph represented in Figure 2.2 (a) can be defined by a matrix called adjacency matrix A with $N \times N$ dimension, where N is the number of nodes, and can be represented by Equation 2.1. Taking into consideration that in an

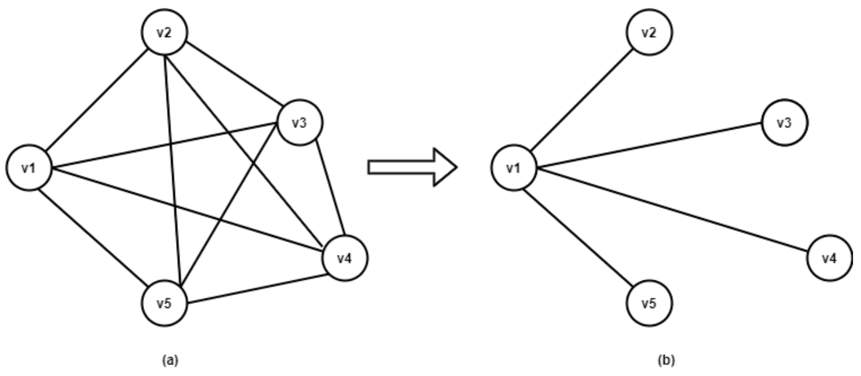


FIGURE 2.2. (a) Physical topology and (b) Logical topology

adjacency matrix bidirectional connections are considered, when $a_{ij} = 0$, then there is no physical connection between both nodes i and j. If this link exists, then $a_{ij} = 1$ [**15**].

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{2.1}$$

The logical topology can also be represented by a matrix, called demand matrix D, which represents the way traffic flows on the network. This traffic can be defined by the amount of logical connections or traffic requests. The logical topology is then described according to an demands matrix (which can be unidirectional or bidirectional). As in the matrix A, in the demand matrix D represented in Equation 2.2, the element $d_{ij} = 0$ if there is no traffic demand between nodes i and j and $d_{ij} = 1$ if that demand exists [**15**].

$$D = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.2}$$

The traffic matrix T represents the amount of traffic that flows between all network nodes in a certain time range and is represented in Equation 2.3. This matrix is associated to the Figure 2.3 that represents a 5 nodes network that transports traffic units in GbE [**16**].

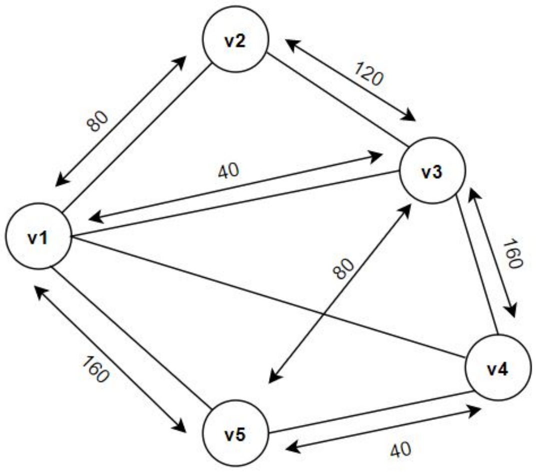


FIGURE 2.3. Traffic flow between nodes

Traffic units of various types of client signals can be transported on these networks, e.g., STM-1, GbE, 10 GbE. These units must be converted into traffic units that can be used on the network. For example, traffic units for Optical Transport Network (OTN) are defined considering Optical Transport Unit (OTU)-k signals. An example of these matrices are shown in Equation 2.4.

$$T = \begin{bmatrix} 0 & 80 & 40 & 0 & 160 \\ 80 & 0 & 120 & 0 & 80 \\ 40 & 120 & 0 & 160 & 40 \\ 0 & 0 & 160 & 0 & 5 \\ 160 & 0 & 80 & 40 & 0 \end{bmatrix} \tag{2.3}$$

$$T(OTU-2) = \begin{bmatrix} 0 & 10 & 5 & 0 & 20 \\ 10 & 0 & 15 & 0 & 0 \\ 5 & 15 & 0 & 20 & 40 \\ 0 & 0 & 20 & 0 & 5 \\ 20 & 0 & 10 & 5 & 0 \end{bmatrix} \tag{2.4}$$

2.3. RWA Problem in Optical Networks

A telecommunications network is composed by two layers: the service layer, and the transport layer (i.e. forward information from the network to the users), which is divided into the electrical and optical layers and is responsible for the information transmission, routing and multiplexing through the network [**17**].

In Figure 2.4, the physical topology of a known real optical network, called COST239, is represented [**11**].

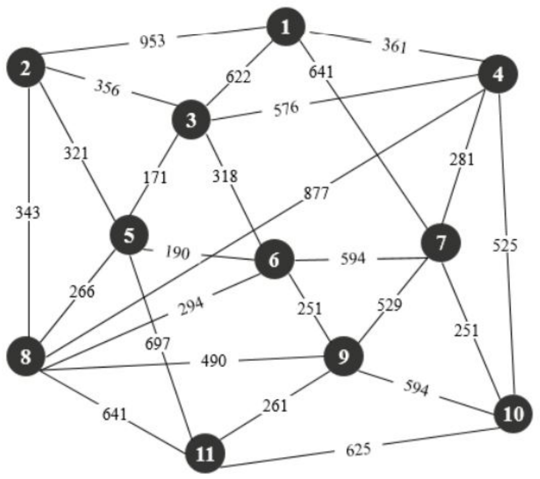


FIGURE 2.4. Physical topology of the COST 239 network [**11**]

In these networks, the source-destination nodes typically used to route information are the Reconfigurable Optical Add-Drop Multiplexer (ROADM), being a fundamental element in the network structure that allows the establishment and termination of optical paths according to the needs of metro/long-haul networks [**18**]. The reconfiguration property of the ROADM appeared due to the need to modify the wavelengths that are extracted/inserted/expressed in an Optical Add-Drop Multiplexer (OADM), to increase network flexibility [**2**].

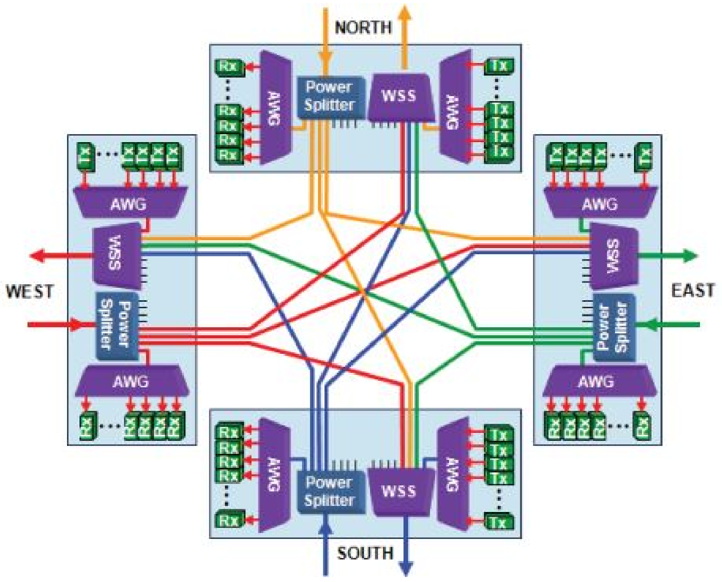


FIGURE 2.5. 4-degree Broadcast and Select (B&S) ROADM [**18**]

A 4-degree ROADM switches and transmits information in four directions, as shown in Figure 2.5, taken from [**18**]. However, the ROADM degree, which represents the switching direction and is related to a transmission fiber pair, may vary between two, for short distance networks, to a maximum of eight in metro or long-haul networks [**15**]. Nowadays, ROADM have several features: color independence, direction independence, wavelength contention free, which are known as Colorless, Directionless and Contentionless (CDC) ROADM and may support the flexible grid [**18**].

One of the main elements of ROADM is the Wavelength Selective Switch (WSS) [**2**]. These switches enable that an input wavelength can be selected to any of the output ports. Multiple wavelengths can be switched to the same output port [**18**]. The impact of WSS features on transmission is dependent on the ROADM node architecture: Route and Select (R&S) or B&S [**2**].

The B&S architecture is shown in Figure 2.5. Each ROADM node has a passive splitter (broadcast function) and a WSS associated (select function). The broadcast signals in all splitter outputs are selected (blocked or passed) at each WSS input port. The advantages of this architecture correspond to the reduction of optical and electrical complexity, cost and power consumption [**19**]. However, due to the division of the input signal power by the number of output ports, the insertion losses can be high. In this architecture, there may be degradation in the transmission due to the crosstalk because of the isolation on seletive ports [**19**]. There is no filtering in the splitter. This filtering happens in the WSS, which is responsible for choosing the wavelength at the output ports. The splitter is a passive element, where no filtering occurs; on the contrary, the WSS is active, that is, it filters the wavelengths [**19**].

To have a R&S architecture, the splitter in Figure 2.5 is changed by a WSS. This architecture uses, for each ROADM, two WSS, respectively 1xN and Nx1. The signal is routed from the first (routing function) to the second WSS (select function) which selects the desired signals, where filtering occurs [**19**]. By using this architecture, there is a reduction of physical layer impairments, like in-band crosstalk [**16**].

RWA problem is more complex than the problem of routing in networks operating electronically and this complexity is a consequence of two constraints. The first constraint is that there is a need for continuity of wavelength, i.e., all links forming an optical path must be assigned the same wavelength if there is no wavelength conversion in the network. The second constraint is that two distinct optical paths should not have the same wavelength on the same link [**2**].

In general, the RWA problem should be seen as an optimization problem where the main objective depends on whether the networks are static or dynamic. In the case of this work, since it focuses on static networks, the main objective is to try to minimize the number of wavelengths assigned to a given number of optical paths.

In order to simplify the approach to the RWA problem, it is advisable to divide it into two separate sub-problems, first the routing and then, WA, which will be solved through exact and heuristic solutions.

2.3.1. Routing Algorithms

Routing consists in selecting a path from among several possible paths between a source node and a destination node through which a demand will be sent [**3**]. The choice of the path should take into account its cost, which should be the one that least overloads the connection and the network, regardless of the cost criteria. These indicators or metrics used for routing are, for example, the lowest number of hops, the path with shortest distance or the minimization of path regeneration [**15**]. It is based on these metrics that the main goal of routing is obtained: given a physical topology G and a traffic matrix T, find the possible paths that support the traffic demands and choose the best paths according to the metric applied [**2**].

When the demand matrix D does not vary along time or if it varies depending on the arrival and the end of new demands as a function of time, the configuration of the path is made, respectively, in a static routing scenario or dynamic routing scenario [**2**].

There are two fundamental strategies to consider when selecting the set of paths for demand, fixed-path routing and alternative-path routing.

In the **Fixed-Path Routing** strategy, the set of paths was already generated before any demand being added to the network. It is important that from this set, the chosen path is always used for that request [**3**]. From each set of candidate paths, the lowest-cost path is preferably chosen. This path is responsible for routing all requests traffic from the source node to the destination node [**2**].

In fixed-path routing, the state of the network does not changes or adapt to any situation. This can block of demands in certain links or congestion in the network [**2**].

Given the graph G, which represents the physical topology of the network, and the respective traffic matrix T, to find out which are the shortest paths for all traffic requests, it is necessary to use, for example, the Dijkstra or the Yens k-Shortest Path (YKSP) algorithms [**3**].

As mentioned above, the Dijkstra algorithm is an algorithm that finds the shortest path from a source node to a destination node. Given a graph oriented $G = (V, E)$, where E are the links with their associated cost w, the following condition has to be verified in the characteristics of the graph: For all links (u, v), $w(u, v) \geq 0$; that is, all the costs of the links are not negative. This strategy allows to the minimization of end-to-end latency, becoming the main advantage of its use [**2**].

In the case of the YKSP, it is an algorithm that needs Dijkstra to calculate the shortest paths but with the objective of finding several shortest paths between two nodes, i.e., finding the first shortest path, the second shortest path, and so on, using the first that appears to make the demand transport [**2**].

After calculating the shortest paths, demands are ordered. For this, the following sorting strategies should be considered:

(1) **Shortest path first:** demands ordered in ascending order of cost;
(2) **Longest path first:** demands ordered in decreasing order of cost;
(3) **Random ordering:** randomly ordered.

Demands should always be forwarded according to the chosen ordering strategy. If several paths have the same associated cost, the path chosen is the one that represents the least connection overload.

In the **Alternative-Path Routing** strategy, mostly associated with dynamic networks, the process is the same for creating the paths before selection. However, it considers multiple alternative routes. There is always a backup route responsible for saving, for each node, an ordered list of alternative paths for each destination node. As the name indicates, it is possible that the state of the network may change and, therefore, these tables are updated accordingly [**3**].

This is an advantageous strategy to solve problems of blocking or network constraints caused by the overload of some connections in favor of less overloaded connections. This strategy it is able to adapt and/or re-organize the network paths in order to balance that traffic and avoid the problem. Also, it quickly changes the routing to an alternative path if, in case of failure, it is necessary to resort to the protection path [**2**]. This network's ability to change to an alternative path in case a certain service fails that allows the service to continue it is called survivability. This capacity is only checked and possible if both service and protection paths are disjoint, that is, they cannot have any node or connection in common except the source and destination nodes [**2**].

Protection can be shared or dedicated [**16**]. In the case of shared protection, a protection path is shared by more than one service path. If these paths have elements in common they fail simultaneously if one of these elements also fails. Therefore, it is ideal that these shared paths be independent [**16**].

In the case of the dedicated one, and unlike shared protection, each service connection is specifically associated with a particular demand, i.e., each service path has its own dedicated backup path [**2**]. This protection can be 1+1 and 1:1. In 1+1 dedicated protection, the protection path is always available, i.e., both paths between nodes are active and the destination node selects the better path, according to a desired criteria. On the other hand, in 1:1 dedicated protection, traffic uses only one fiber for transmission, the service fiber. In case of a failure, both nodes start to use the protection fiber [**16**].

Dedicated 1+1 protection regenerates the failure quickly. The 1:1 protection takes a longer time to respond after fault detection because it is still necessary to activate the service path [**2**].

2.3.2. Wavelength Assignment Algorithms

After using one of the ordering strategies described in subsection 2.3.1, it is necessary to associate a wavelength to each path. This is called WA. This problem associated to static networks consists of assigning each wavelength with a light path ensuring that two paths sharing the same physical link are assigned different wavelengths. This distinct wavelength assignment to different paths makes it possible to minimize the number of wavelengths, that should verify also the restriction of the wavelength continuity [**4**].

The algorithms typically used to perform the WA correctly on each network are the First-Fit, Most Used and Graph Coloring Algorithms (in particular, the Greedy algorithm).

First-Fit algorithm is based on one of the sorting strategy algorithms to index wavelengths. After this indexing from 1 to WL, where WL is the maximum number of lengths supported by the fiber, the available wavelength with the lowest index is chosen, that is, the availability of the wavelengths to be assigned by increasing index order is used to perform the assignment[**2**].

Most-Used algorithm, as First-Fit, is also based on sorting strategies to index wavelengths. However, in this case, each assigned wavelength is saved in a variable that counts the number of times that wavelength is assigned, for each link. The higher the value of the variable, the higher its priority. This priority is used to assign that wavelength to the new path if the continuity of the wavelength and the different wavelengths conditions are accomplished. If not, the wavelength with second higher priority is assigned [**2**]. **Graph Coloring Algorithms** which are the focus of this work, namely the Greedy algorithm [**4**], exact algorithm based on ILP [**4**] and Tabu Search algorithms [**6**], will be explained in detail in Chapter 3.

2.4. Conclusion

In this chapter, the basic concepts related to optical networks are introduced. In section 2.2, concepts related to graphs such as physical and logical topologies of a network are described, along with the definition of adjacency, demand and traffic matrices. Next, in section 2.3, the operation of ROADMs, element responsible for routing in optical networks, and their architectures, are briefly explained. Then, routing and WA functions are introduced. Some WA algorithms are described, in particular the First-fit and Most-used algorithms. The graph coloring techniques will be discussed in detailed in the following chapters.

CHAPTER 3

Graph Coloring Basics

3.1. Introduction

In this chapter, the basic concepts that allow to define and understand the graph coloring problem, described in Section 3.2, are exposed. Some real examples to illustrate that definitions are explained in Section 3.3 and the particular case of WA problem is presented. Next, in Section 3.4, the graph coloring problem complexity is analysed in order to recognize the complexity of the data and number of operations that the algorithms are dealing with. Also in this section, it is justified that it is possible to work with algorithms capable of solving these problems, regardless of their complexity, using the chromatic number of the graph. Finally, at Section 3.5 are exposed the three algorithms studied and compared in this work, Greedy, exact algorithm based on ILP and Tabu search algorithms. Each of the algorithms have its respective explanation, descriptions, pseudocode, and examples, which are fundamental to understanding how each one of them works.

3.2. Graph Coloring Problem Description

The problem of graph coloring consists in, given any graph G, assigning colors to the vertices so that there are no adjacent vertices with the same colors and the number of colors is as small as possible. This problem is present in many real cases and in several areas of work [**4**].

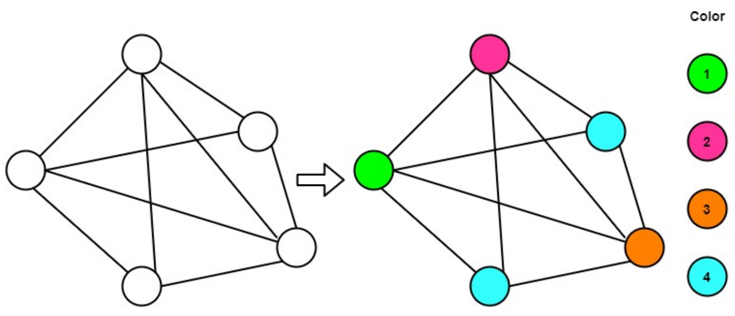


FIGURE 3.1. A graph example with corresponding 4-coloring.

The graph represented in Figure 3.1 is formed by 5 vertices and 9 links and its respective 4-coloring is presented. This is the correct coloring solution since it fulfills the two fundamental requirements: no pair of adjacent nodes has the same color between them and for this number of vertices and respective edges, 4 is the minimum number of colors that can be used.

Formally, given a graph *G=(V,E)*, the graph coloring problem consists in assigning to the vertices $v \in V$ an integer $c(v) \in \{1, 2, ..., k\}$ (each integer corresponds to a color up to k color) such that the following definitions are verified [**4**]:

(1) $c(v) \neq c(u)$ $\forall \{v, u\} \in E$, i.e., two nodes linked by the same link have different color;

(2) k is minimal, i.e., assign as few colors as possible;

To understand the graph coloring problem, it is important to define some concepts, which are the following [**4**]:

- When all vertices are colored $c(v) \in \{1, ..., k\}$ then the coloring of the graph is complete. If this is not the case, it is a partial coloring;
- When two adjacent vertices $\{u, v\} \in V$ have the same color assigned as $\{u, v\} \in E$ and $c(v) = c(u)$, a clash occurs and it is verified an improper coloring. Otherwise, the coloring is proper;
- A solution is said to be feasible if it is both complete and proper.
- An optimal solution of a feasible coloring of a graph G corresponds to the solution that uses exactly the minimum number of colors required in a feasible coloring of G. This minimum number of colors, $\chi(G)$, is called chromatic number;
- The set of all vertices of a graph assigned to a specific color in a solution is a color class: given a specific color $i \in \{1, ..., k\}$, the color class is defined as the set $\{v \in V : c(v) = i\}$;
- A subset of vertices $C \subseteq V$ mutually adjacent is called a clique: $\forall u, v \in C, \{u, v\} \in E$;
- A subset of vertices $I \subseteq V$ mutually nonadjacent is named the independent set: $\forall u, v \in I, \{u, v\} \subseteq E$.

Figure 3.2 illustrates a small generic example graph with its corresponding 5-coloring graph and some of the concepts exposed in this section. The graph is composed by 10 vertices and 21 edges. As explained in previous subsection, 5-coloring graph means that 5 colors were needed to color the graph respecting the previous definitions.

In Figure 3.2, based on the concepts discussed, the coloring is complete because all the nodes have a color and is a feasible solution because it is complete and proper. Each color assigned represents a different color class and, as a result, each color represents an

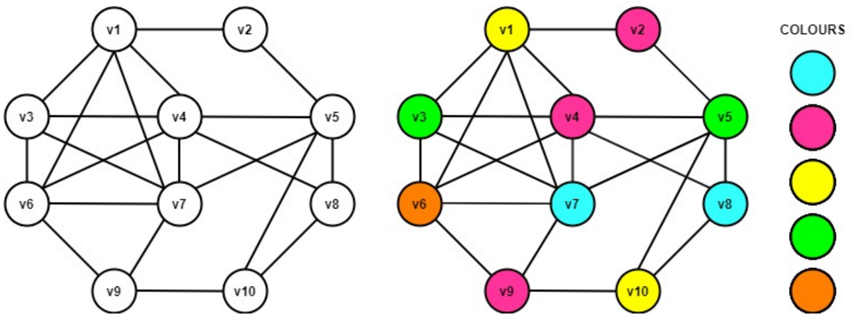


FIGURE 3.2. A graph example with corresponding 5-coloring.

independent set that is not adjacent to other independent set, i.e, there are 5 independent sets. There are no clashes since there are no pairs of adjacent nodes with the same assigned color. Vertex v_1 forms a clique of size 5 with its neighbours v_2, v_3, v_6, v_4 and v_7 .

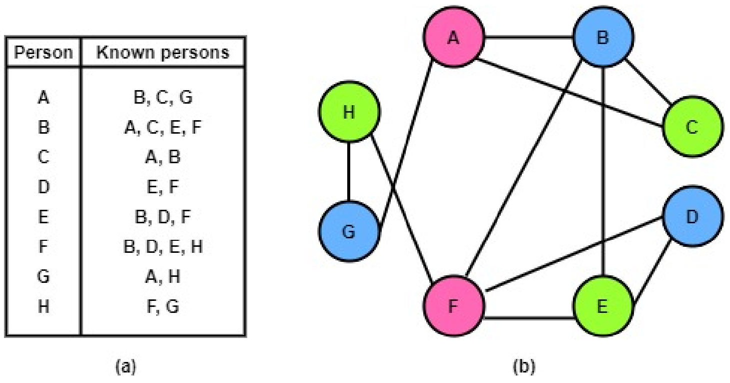
3.3. Pratical Examples for Graph Coloring

In this subsection, some examples of the different possible applications in the real world of graph coloring are explained. Color assignment can be made from simple to more complex situations such as building work groups, scheduling taxis according to customer orders, constructing schedules, help in compilation of programming code and the focus of this work, WA [**4**].

Team building example is the first example presented. The objective in this example is to group people - represented by the graph nodes - into groups, through their connection. To make this association of nodes it is necessary having a criterion to group them, for example, each person is allocated to a group that does not have one or more known persons, i.e., friends. Each person (i.e. node) is connected to all the nodes known to be its friends. This ensures that no group is formed by elements that are friends and ensures that groups of people are formed for the purpose of getting to know each other. The number of groups is the minimum possible, i.e., the nodes are assigned the minimum number of colors, since this is a graph coloring problem [**4**].

As can be seen in the table in Figure 3.3 (a), persons with names from A to H are associated with the respective known persons. In the graph of Figure 3.3 (b), each of the vertices, which represents a person, is linked again to the respective people they know [**4**].

As verified in the graph above, there are 3 groups of people who know each other: A, B and C; D, E and F; B, E and F. With this information, it is concluded that at least 3 groups will be built by assigning 3 different colors to each other (due to its adjacency). Other solutions also would be correct if the color assignment occurred in another order, but

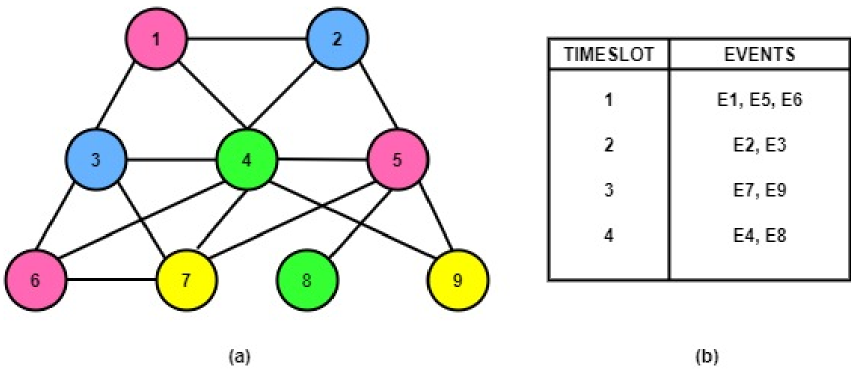


Person	Known persons
A	B, C, G
B	A, C, E, F
C	A, B
D	E, F
E	B, D, F
F	B, D, E, H
G	A, H
H	F, G

FIGURE 3.3. (a) Team Building table and (b) Team Building graph [4]

only the solution in Figure 3.3 is the optimal solution, as it corresponds to the minimum number of groups, i.e., minimum number of colors.

Constructing Schedules is the second example detailed. The graph coloring is treated between certain events and certain timeslots in order to organize timetables and schedules, following certain conditions that meet the achievement of the problem, such as the non-collision of events allocated in the schedule of a person who needs to attend both events. Two events that verify these conditions imposed by the problem cannot be assigned to the same timeslot, i.e, cannot be assigned to the same color [4]. This means that each vertex of the graph is a timeslot and therefore each available timeslot will have an assigned color. Then, two vertex are linked if there is between them a collision constraint, i.e, both events cannot occur ate the same time. At most, there will be as many colors assigned as the number of available timeslots.



TIMESLOT	EVENTS
1	E1, E5, E6
2	E2, E3
3	E7, E9
4	E4, E8

FIGURE 3.4. (a) Scheduling example graph and (b) Scheduling example table [4]

In Figures 3.4 (a) and (b) it can be seen that the first timeslot has one more event than the others. This solution is a feasible solution and correspond to assigning 4 colors to the timeslots. In this context, the availability of one room per event and per timeslot should also be taken into account, since it is the construction of a schedule. In this sense, it must be ensured that three rooms are available in timeslot 1 and two rooms in the remaining timeslots (2, 3 and 4). If there are only two rooms available per timeslot, a new timeslot will have to be added, increasing the complexity of the coloring problem.

Scheduling Taxis is the third example presented. When there is a need to make a taxi reservation, it is necessary to make an assessment of the availability of taxis before assigning them to a reservation. In this case it is necessary to know the bookings that exist, i.e., duration of each reservation (start and end time) so that the taxi can be booked. This is an example of interval graph, which are graphs where a pair of adjacent vertices means overlapping intervals [**4**].

The diagram in Figure 3.5 represents the duration of each reservation, i.e., black points represent the beginning of each reservation and white points represent its end. In terms of the application of graph coloring, each of these reserves is a node of the graph and the connections between them represent the reserves that are in a coincident time instant, i.e., adjacent vertices represent overlapping reserves. After the graph is completed, the colors are assigned to the nodes taking into account that reservations at the same time cannot have the same color assigned. The color represents the taxi assigned to the reservation and two adjacent nodes cannot have the same color, since one taxi cannot have more than one reservation at the same time. The solution found is as better as fewer colors are used.

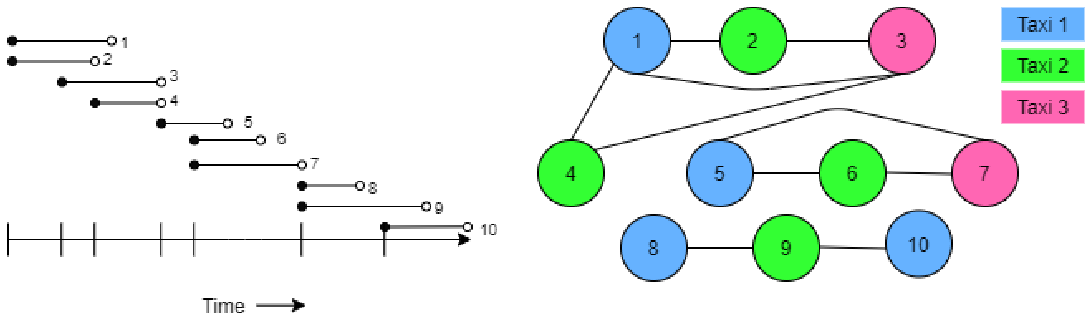
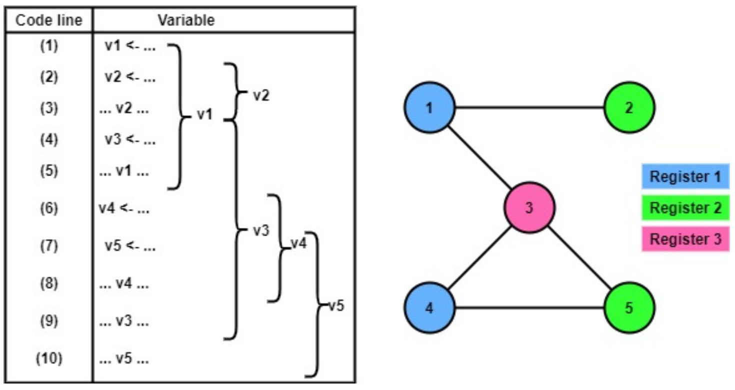


FIGURE 3.5. Taxis Booking example [**4**]

It is important to note that as soon as a reservation reaches its end, the taxi is available to cover another reservation. For example, reservations 2 and 4 may be assigned to the same taxi because the final time of the reservation 2 coincides exactly with the initial time of the reservation 4. In this case, optimal solution is achieved with 3 colors and therefore 3 taxis are sufficient for 10 bookings [**4**].

Another important function of the graph coloring techniques is to do **Compiler Register Allocation**. This example has the main goal of facilitating the compilation of programming code, assigning the variables to registers so that it is faster to access and update values by the processor.

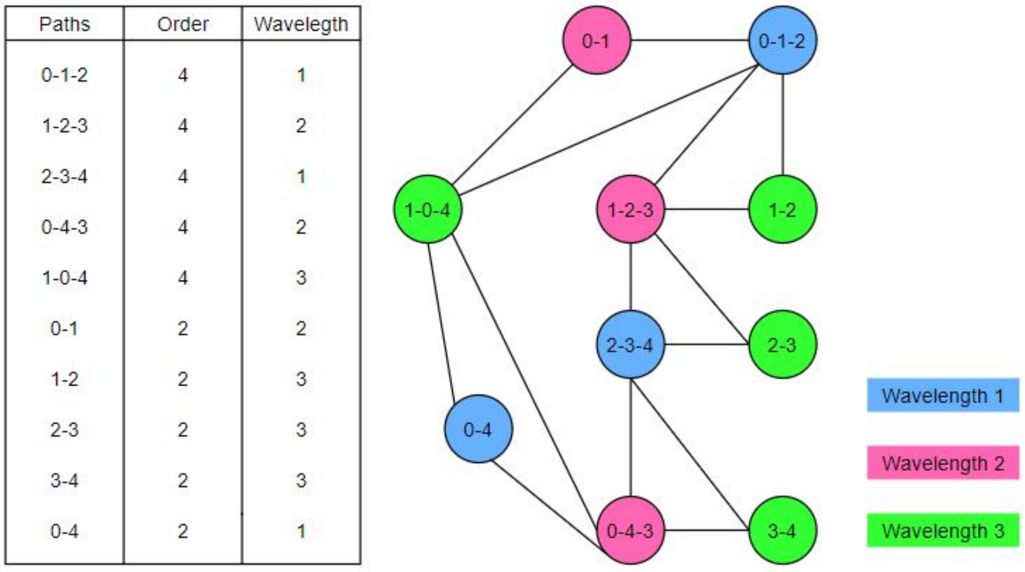


Code line	Variable
(1)	v1 <- ...
(2)	v2 <- ...
(3)	... v2 ...
(4)	v3 <- ...
(5)	... v1 ...
(6)	v4 <- ...
(7)	v5 <- ...
(8)	... v4 ...
(9)	... v3 ...
(10)	... v5 ...

FIGURE 3.6. Compiler Register Allocation example [4]

Figure 3.6 shows that graph coloring can solve these situations by assigning colors to the intervals each variable is running in the program. Some of these variable life cycles are overlapping. Each vertex of the graph represents a variable, and each of them connects to the others that are in simultaneous “live time”. The color assigned to the variables corresponds to the register allocated [4]. In this case, 3 registers are enough to run 5 variables in the program, since there are only 3 variables in the same ”live time”.

Finally, and the aim of this work, the **WA** example, which consists of assigning wavelengths to the optical paths of a network, which is the focus of this work.



Paths	Order	Wavelegth
0-1-2	4	1
1-2-3	4	2
2-3-4	4	1
0-4-3	4	2
1-0-4	4	3
0-1	2	2
1-2	2	3
2-3	2	3
3-4	2	3
0-4	2	1

FIGURE 3.7. Wavelength Assignment example [4]

In this case, the process consists in, from the graph $G(V, E)$ of physical topology of the network, create a graph $G(W, P)$ in which the vertices W are optical paths of the network ($W = (w_1, w_2 ... w_n)$, where n is the total number of paths W) and P is the set of links between these nodes. If one or more nodes share the same physical link, a link is established between them.

Depending on the number of incident edges on a vertex, the node is assigned to an order, i.e., if a node has 4 links, that node has order 4. The paths are sorted in descending order of degree and it is in this order that the colors (i.e. wavelengths) are assigned, so that each vertex is assigned the first color not assigned to any of the adjacent vertices [**4**].

The paths with 3 nodes and 2 links have order 4 and the remaining order 2. Given the set of paths with the same degree, the order is arbitrary. Following the assignment process, only 3 colors are needed for an optimal solution, as Figure 3.7 shows.

3.4. Graph Coloring Problem Complexity

After the detailed exposure of some real cases where graph coloring problem can be applied, the question is based on understanding which algorithm is capable of finding all feasible solutions, regardless of the topology or size of the problem. As said before, several solutions are possible when it comes to the graph coloring problem but only one is the optimal. However, and although ideal, it is almost impractical for an algorithm to discover all these solutions and return the optimal [**4**].

A viable solution can correspond to the solution where a total of maximum colors correspond to the total number of vertices, i.e, at most equal to V_n. To have a realistic perspective of the number of candidate solutions it is necessary to recognize that each vertex of the graph can have any of the colors $c(n)$. In this sense, since each vertex can have any color n associated, a graph can have n^n possible solutions. It is easily perceptible that this number is too large to be processed in practice. To make this perceptible in real context, taking into account a graph with $n = 50$ nodes, the problem has a space of solutions with $50^{50} \approx 8.8 \times 10^{84}$ possibilities, which is a really huge number of feasible solutions so that all these assignments can be checked. To have an idea of the complexity of the value, note that 10^{82} is the known estimated value of atoms in the universe, which really huge [**4**].

What is most important in this analysis is not effectively what each node in the graph represents but the number of different colors that are assigned to it [**4**].

A better approach for this context is to divide the vertices into independent groups of color: $S = \{S_1, ..., S_k\}$, where k is minimum. If the number of available colors were decreased to $k < n$, the space for solutions would decrease further.

Stirling numbers of second order, which define the number of n-item divisions in k nonempty subsets, are calculated by the expression 3.1.

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n \tag{3.1}$$

The *nth* Bell number, *Bn*, depends on Stirling numbers and describes the amount of ways of partitionating a set with n items into nonempty subsets containing the total vertices of a graph, as can be seen in expression 3.2.

$$B_n = \sum_{k=1}^{n} \left\{ {n \atop k} \right\} \tag{3.2}$$

The optimum solution resulting from this procedure corresponds to the solution with the smallest number of S groups [**4**].

This division allows a considerable decrease in the number of solutions to be analyzed. However, it is still not the ideal method because there are exponential growth rates in this solution space (various k-values) associated with Stirling numbers. To understand this in real context, taking into account a network with n=50 nodes again, with a division of 50 items in 10 subsets, would be obtained a value of possible solutions of $\left\{ {50 \atop 10} \right\} = 2.6 \times 10^{43}$ which is still a huge number.

Besides the complexity demonstrated in the description above, there are also decision problems in which the answers of a decision consist only between two options. It means that instead of testing a predefined set with all possible solutions and selecting the optimal solution, something computationally expensive, the test is done under a condition that does not require testing all solutions since only possible solutions that meet the defined condition will be tested. These options are:

(1) Giving k colors, can a graph be assign with a feasible coloring solution?
(2) For a feasible coloring, which is the minimum k number of colors?

Questioning whether k colors are sufficient to make a certain graph feasible, the problem is transposed to a decision problem. Given a possible solution and a k-value, it is possible to conclude whether the solution is viable in polynomial time.

Using heuristic methods in graph coloring problems has the intention not to solve the problem completely in polynomial time but to find more easily a solution as approximate as possible. It is possible to choose any algorithm A that is applied to a given graph G and whose solution is feasible. If it is, there are k colors accepted and it ends; otherwise, algorithm A's behavior is changed that can return an optimal solution, with the objective of decreasing the number of k colors compared to the solution that was not possible [**4**].

If the chromatic number of a given graph is known, is possible to draw the exact algorithm that produces the optimal solution. The simplest case is a graph composed of vertices not adjacent to each other. This feature allows all the nodes to be assigned the same color and the chromatic number is 1 [**4**]. The following graphs represent examples of solutions where the optimum number of colors can be found.

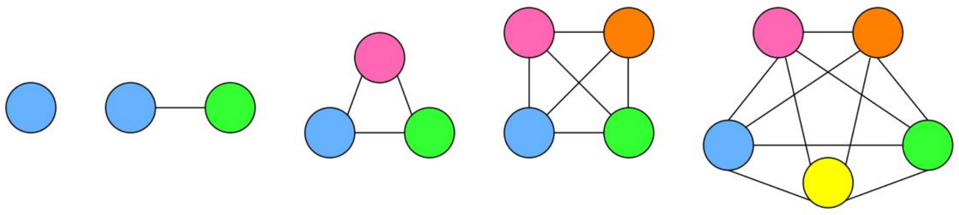


FIGURE 3.8. Optimal solution for complete graphs

Figure 3.8 shows 5 types of **complete graphs** with different number of vertices that are all adjacent to each other. This is the definition of a complete graph: there is no pair of nodes without an edge between them. As explained before, two adjacent nodes cannot contain the same color as each other and therefore the number of colors will be as much as the number of nodes in the graph. Thus, the chromatic number will also be equal to the number of vertices n: $\chi(G) = n$ [**4**].

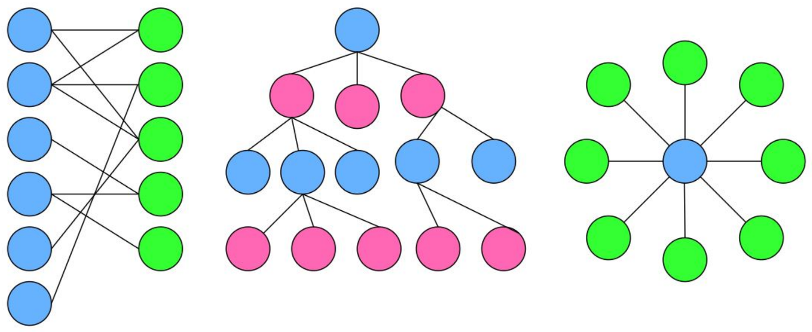


FIGURE 3.9. Optimal solution for bipartite graph, tree graph and star graph.

Analyzing the graphs drawn in Figure 3.9, which are called **bipartite graphs**, it is possible to conclude that it can be divided into two independent subsets of vertices, each with its associated color. This is because the nodes of each of the subsets are adjacent to the nodes of the subset from which it is independent and not adjacent to the nodes of the subset itself. Thus, the number of assigned colors is equal to the total number of subsets which is: $\chi(G) = 2$ [4]. If the graph was tripartite, then the chromatic number would be 3 and so on.

Cycle and Wheel Graphs are easily identifiable by their particular characteristics. These graphs have at least 3 vertices and the number of links corresponds to the number of vertices plus one. When it comes to **cycle graphs** as shown in Figure 3.10, the number of colors assigned and consequently chromatic number will be $\chi(G) = 2$ if the number of vertices is even and $\chi(G) = 3$ if the number of vertices is odd [4].

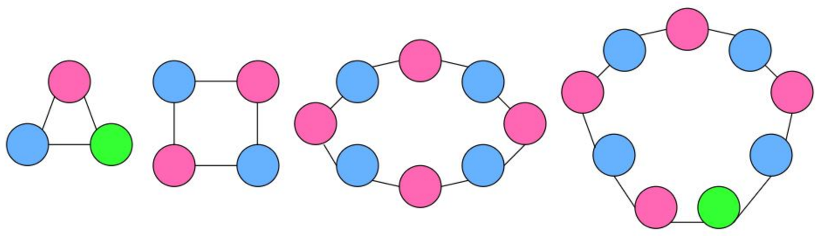


FIGURE 3.10. Optimal solutions for cycle graphs

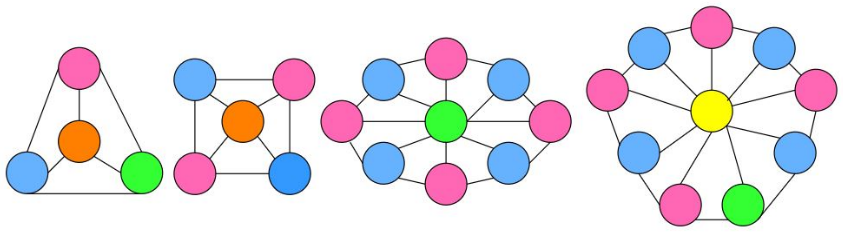


FIGURE 3.11. Optimal solutions for wheel graphs

A particular case of the cycle graphs, represented in Figure 3.11, are the **wheel graphs** that result from the addition of a central node to cycle graphs and respective edges of that node added to all the nodes already belonged to the topology. To each case, it adds a color to graph coloring and, consequently, the chromatic number increases one. Thus, unlike in cycle, graphs with vertices in odd number have an even chromatic number $\chi(G) = 3$ and graphs with vertices in even number have an odd chromatic number $\chi(G) = 4$ [4].

As can be seen in Figure 3.12, **Grid Graphs** can be sparse if each node is adjacent only to the top and bottom nodes and positioned on both sides and can be dense if each node is adjacent to all others, including diagonals [4]. Each of these graphs can be analyzed by dividing the vertices into subsets. The first graph, the sparse one, is equivalent to a bipartite graph, to which a chromatic number with $\chi(G) = 2$ is associated. In the case of the dense one, and because each node is adjacent with all the others around it, it is a

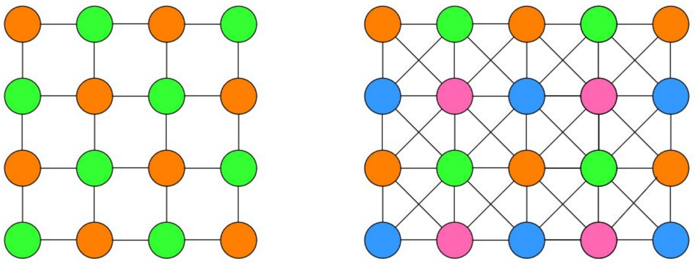


FIGURE 3.12. Optimal solution for (a) sparse and (b) dense grid graphs

4-coloring graph. In this sense, the chromatic number doubles with respect to the first case, i.e, $\chi(G) = 4$.

TABLE 3.1. Optimal chromatic number for different examples.

Graphs	**Chromatic Number**
Complete Graphs	$\chi(G) = n$
Bipartite Graphs	$\chi(G) = 2$
Cycle Graphs	$\chi(G) = 2$ (number of vertices even) and $\chi(G) = 3$ (number of vertices odd)
Wheel Graphs	$\chi(G) = 3$ (number of vertices even) and $\chi(G) = 4$ (number of vertices odd)
Grid Graphs	$\chi(G) = 2$ (sparse graph) and $\chi(G) = 4$ (dense grid graph)

Table 3.1 is a comparative table with respect to the chromatic number associated with each graph presented in this subsection. Bipartite graphs are those that have the smallest optimal chromatic number associated and complete or grid graphs are those that can reach a larger chromatic number. In case of cycle and wheel graphs, chromatic number depends directly on the parity of the number of vertices.

3.5. Graph Coloring Algorithms

Graph coloring algorithms are used to solve the WA problem.

As previously mentioned, WA algorithms aim to color the graph nodes, which represent optical paths connected by edges that represent links between them, so that adjacent nodes are assigned distinct colors and that this number is minimal. There are as many wavelengths used in the network as the number of colors assigned in the graph, i.e, one color corresponds to one wavelength [**2**].

The complexity of these algorithms lies in the fact that it is difficult to reach the optimal solution, although any solution found is a feasible one.

In order to improve this situation, chromatic number works as a reference that reflects the minimum number of colors attributed to a graph in a solution, which is the optimal number of a solution. Therefore, this chromatic number correspond to the solution of a WA problem. The upper bound corresponds to the upper limit with very low precision, i.e, the chromatic number as a minimum $D_g + 1$ distinct colors, and thus $D_g + 1$ wavelengths [**2**], considering D_g as the node degree of the graph.

In simple networks, there are many examples where reaching the chromatic number is simple because it can be calculated according to the topology of the graph, number of nodes and edges. For large dimensions, it becomes even more complex to obtain an optimal solution. In these cases, heuristic algorithms are used to found the number of assigned colors because they find solutions close to the optimal [**4**].

TABLE 3.2. Notation for pseudocodes.

Notation	**Description**
$\leftarrow$	Assignment operator
$\|\pi\|$	Length of set π
π	Set of vertices ordered of a certain strategy
S	Set of all colors
S_j	Color j from set S to assign to a vertex
independent set	Subset of vertices non adjacent mutually
$\cup$	Union operator

In the context of this work, the heuristic algorithms studied are the Greedy algorithm and the Local Search algorithm (Tabu search). An exact algorithm based on ILP is also studied.

Table 3.2 clarifies the notation of the pseudocodes that will be presented in this section.

3.5.1. Greedy Algorithm

The Greedy algorithm is the first graph coloring algorithm studied in this work and offers feasible coloring solutions, coloring the vertices one by one of any graph G, after ordering the nodes according to a certain strategy, normally in descending order of degree [**4**]. In general, the algorithm works as follows: it goes through the nodes of a path graph representing the connections of a particular network, one by one according to a previous given order and applies an available color to each node.

Figure 3.13 shows the pseudocode for Greedy algorithm implementation [**4**]. Initially S that represents the set of colors that are going to be assigned along the Greedy algorithm process is empty and π represents a possible permutation of the graph vertices, e.g. descending, ascending or random order of the vertices. The *for* cycle in line (1) of the pseudocode goes through the set of vertices π and, for each vertex of π tries to find a color class S_j belonging to S to which it can be associated. This process involves checking the color class of the adjacent vertices. If the working vertex belongs to an independent set then a color S_j can be assigned to this vertex. If this is not the case then a new color class must be assigned (lines 7 to 9).

```
GREEDY ($S \leftarrow 0, \pi$)

(1) for $i \leftarrow 1$ to $|\pi|$ do
(2)     for $j \leftarrow 1$ to $|S|$
(3)         if ($S_j \cup \{\pi_i\}$) is an independent set then
(4)             $S_j \leftarrow S_j \cup \{\pi_i\}$
(5)                 break
(6)         else $j \leftarrow j + 1$
(7)     if $j > |S|$ then
(8)         $S_j \leftarrow \{\pi_i\}$
(9)         $S \leftarrow S \cup S_j$
```

FIGURE 3.13. Pseudocode for Greedy algorithm.

Figure 3.14 based on [**4**] represents an example of the operation of the Greedy algorithm, assuming a coloring strategy based on the descending order of degree. Note that white nodes represent uncolored nodes. The vertex with the highest number of links connected, i.e., the highest degree, is colored first with the first available color, i.e. v_2 in step 1 is color with green (S_1={green}). In step 2 the algorithm continues the coloring with the following vertex with highest degree, v_8, which is adjacent to v_2 so it is assigned to a different color, pink (S_2={pink}). In step 3, since there are four vertices with degree 3, one of them is randomly chosen. We have choose v_1 with the color class $S_2 = \{pink\}$. This process continues until all vertices have been colored and in the end (step 8) we can see that three colors are used.

In the implementation of a graph coloring, it is important to analyze its computational effort. In this work, the computational effort of an algorithm is measured using the definition of constraint checks, depending on the list of total nodes of the graph, the list of adjacent nodes to a particular node u, $|Adj_u|$, i.e., on the inspection of the nodes to determine their degree and colors of class S that are attributed. There are 3 ways to define how constraints checks are counted according to [**4**]:

(1) To check adjacency from a u node to all other nodes, it is mandatory to access all Adj_u elements, i.e., elements adjacent to node u. In this step, $|Adj_u|$, i.e., length of the vector Adj_u, constraints are checked.

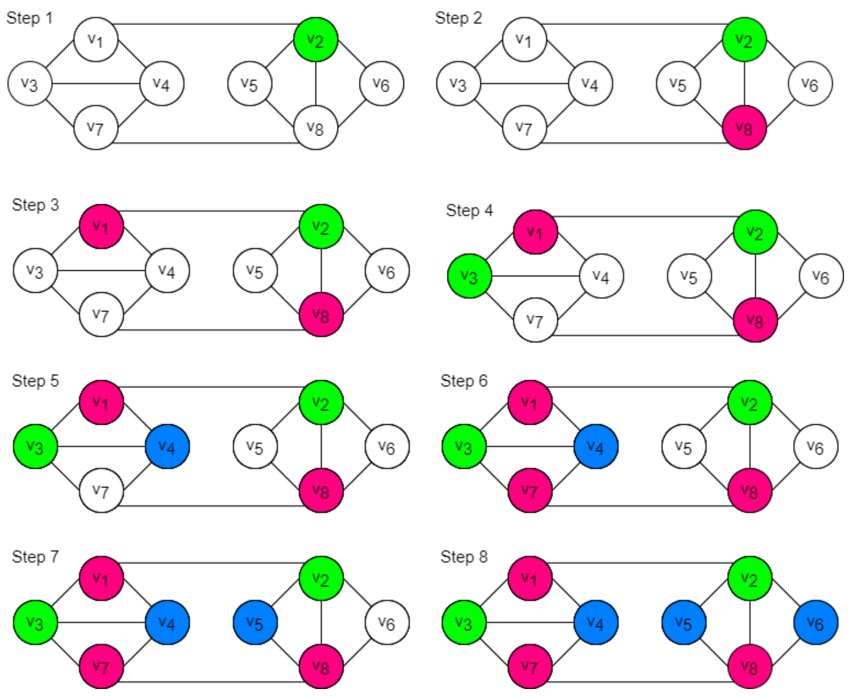


FIGURE 3.14. Greedy algorithm example.

(2) Accessing the list of adjacent nodes to determine each node degree corresponds to access the total list of nodes and corresponds to $|\pi|$ constraint checks.

(3) Accessing the set of colors assigned, which is denoted as C, that is, accessing each of the colors in the set to check the number of colors of class S adjacent to a given vertex u is another constraint check.

Hence, the total number of constraint checks in the Greedy algorithm, i.e., its computational effort, can be represented through

$$\#Constraints = |Adj_u| + |\pi| + |C| \tag{3.3}$$

3.5.2. Exact Algorithm

Exact algorithms always try to obtain the optimal solution to solve a computational problem such as the graph coloring problem, exhaustively searching the whole solution space. If this solution space increases then the solution search time becomes longer [4].

One way to achieve the optimal solution of computational problems such as graph coloring, instead of using exact solutions, such as the ILP, consists in using the "Backtracking" methodology. Algorithms such as Greedy or DSATUR algorithm are considered constructive algorithms that analyse vertex sequences ordered in a specific way and assign the colors for that specific vertices ordering. The "Backtracking" methodology uses multiple vertices orderings in conjunction with these algorithms in order to achieve the exact solution, taking a longer time. Since it is an algorithm that builds several options of solutions, with varying orders, it is possible that the execution may end prematurely and never give the optimal solution but a feasible approximation solution [**4**].

Another way to implement an exact algorithm consists in using ILP, a particular type of linear programming model used to search optimal solutions for linear mathematical models. In ILP problems, the decision variables are restricted to integer values [**4**]. These methods allow reaching optimal solutions to problems according to the definition of an objective function delimited to constraints defined according to the variables and the goal of the problem. The objective function is evaluated in order to reach a reasonable solution range and to find the solution that is the optimal one [**4**].

ILP can use several minimization methodologies, such as the Branch and Bound method that consists of a solution approach that splits the feasible solution space into smaller subsets of solutions, which restrict part of the solution value [**4**].

To solve the graph coloring problem through linear programming, it is necessary to define the restrictions and the objective function, which depend only on the logical topology of the network where a path graph $G(W, P)$ contains W vertices and P edges. Each vertex correspond to an optical path and an edge means that two optical paths are using one or several common links [**4**].

The variables for a graph coloring problem are defined by a binary matrix $X_{n\times n}$ and a binary vector Y_n, where n is the number of vertices of the path graph. This means that matrix $\mathbf{X}$, which represents the color state of the vertices, and vector $\mathbf{Y}$, which is the coloring solution, are composed by binary numbers. Each of them has its elements defined accordingly to conditions (3.4) and (3.5), respectively [**4**]:

$$X_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is assigned to color } j \\ 0 & \text{otherwise.} \end{cases} \tag{3.4}$$

$$Y_j = \begin{cases} 1 & \text{if at least one vertex is assigned to color } j \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

In matrix X_{ij}, there are two possible states for vertex v_i: having a color j assigned, which is represented by number 1, or not having any color j assigned, which is represented by number 0. The elements of vector $\mathbf{Y}$ will be 1 if at least one vertex is associated with color j and 0 otherwise. The resulting vector $\mathbf{Y}$ returns the solution to the graph coloring problem: the number of 1's in this vector corresponds to the number of colors assigned, and therefore the number of wavelengths.

The goal of implementing integer programming models for graph coloring problems is that this model is responsible for minimizing the number of colors according to a predefined objective function [**4**]

$$min \sum_{j=1}^{n} Y_j \tag{3.6}$$

This objective is achieved by implementing the two constraints defined in Equations (3.7) e (3.8). The first constraint guarantees that there are no adjacent vertices with the same color assigned and that the element Y_j has the value 1 only if any of these vertices is associated with a color j. The second condition imposes that each vertex should be associated to exactly one color [**4**].

$$X_{ij} + X_{lj} \leq Y_j \quad \forall \{v_i, v_l\} \in E, \quad \forall_j \in \{1, ..., n\} \tag{3.7}$$

$$\sum_{j=1}^{n} X_{ij} = 1 \quad \forall v_i \in V \tag{3.8}$$

The exact algorithm based on ILP is an algorithm that requires a very large computational time to provide a solution, due to the large space of possible solutions, $\frac{n!}{(n-k)!}$, where n is the number of vertices and k the number minimum of colors [**4**]. This last expression represents the number of possibilities to color the total number of n nodes with k colors. Hence, it is important to consider some implementation restrictions to the space of solutions in order to reduce the computational time [**4**]. The number of solutions can be really huge by the fact that any feasible k-color solution can be expressed in $\frac{n!}{(n-k)!}$ forms by simply permuting the columns of the matrix X. To surpass this constraint and reduce the ILP search space, the condition (3.9) guarantees that a k-color solution uses only the colors labelled 1 to k [**4**]. It also guarantees as possible solutions, solutions that have the colors associated with nodes on the left columns of the matrix.

$$Y_j \geq Y_{j+1} \quad \forall j \in \{1, ..., n-1\} \tag{3.9}$$

Equation (3.9) improves the ILP computational processing in a significant way, but it is possible to improve the computational time by exchanging the first k colors that arise due to the *k!* possibilities, which represent equivalent solutions, by replacing Equation (3.9) with

$$\sum_{i=1}^{n} X_{ij} \geq \sum_{i=1}^{n} X_{ij+1} \quad \forall j \in \{1, ..., n-1\} \tag{3.10}$$

Equation (3.10) guarantees that color j is associated with a total number of vertices greater than or equal to the total number of vertices with color $j{+}1$. Using this equation, there is an improvement in computational processing, but symmetric solutions are still verified due to the fact that color classes of the same size are interchangeable. To improve this, Equation (3.10) can be replaced by the the following constraints

$$X_{ij} = 0 \quad \forall v_i \in V, \quad j \in \{i+1, ..., n\}, \tag{3.11}$$

$$X_{ij} \leq \sum_{l=j-1}^{i-1} X_{lj-1} \quad \forall v_i \in V - \{v_i\}, \quad \forall j \in \{2, ..., i-1\} \tag{3.12}$$

Combining both constraints (3.11) and (3.12) specifies that only one solution from the first k permutations is a solution, eliminating possible repeated solutions. Thus, each column is assigned the minimum color of the color class according to each independent set. Equation (3.11) guarantees that the upper part of the matrix **X** above the main diagonal all its elements assigned to zero and, together with Equation (3.12), guarantees that for each possible k-coloring a unique permutation of the first k columns is the solution. Consequently eliminating these symmetric solutions greatly reduces the computational time [**4**].

Figure 3.15 illustrates a 8-node network topology colored using the ILP algorithm and the corresponding steps of the coloring problem. This network has 8 vertices and 12 edges [**4**] and the coloring order considered is from vertex v_1 to vertex v_8. At each algorithm step, the solutions matrix **X** with dimension $n \times n$ and vector **Y** with dimension n are being built and updated. This updating process is shown with detail in Table 3.3.

In this example, step 1 corresponds to the coloring of node v_1. Since there is no color associated yet, the yellow color is first assigned to the node. In this step, first element X_{11} of matrix **X** is set to 1, which means that color 1 has been assigned to vertex 1, and the other matrix positions in this line (X_{12} to X_{18}) are set to 0. This means that to the vertex v_1, which belongs to the total number of vertices *V*, yellow is the color assigned to position 11 of matrix **X**, and the Equation (3.8) it is verified. In this step

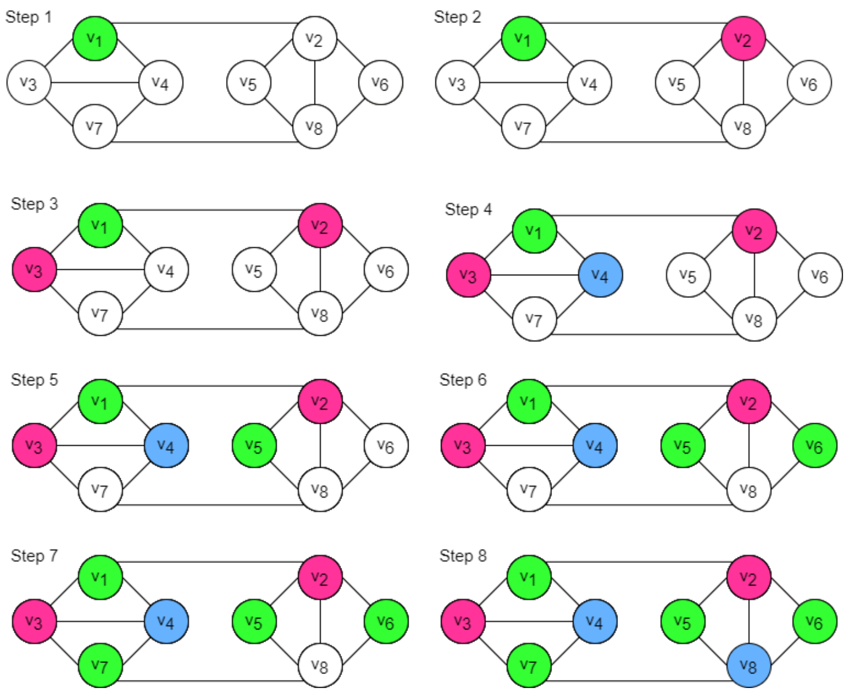


FIGURE 3.15. Exact algorithm based on ILP example.

also, position Y_1 is set to 1 because one color is already assigned. Step 2 corresponds to the coloring of node v_2. Each node is colored after evaluating all the adjacent nodes. After this evaluation, the color assigned is either the color attributed to non-adjacent vertices or a new color if all the colors already used correspond to adjacent vertices. Hence, step 2 must obey condition (3.9). Means that for position X_{21} of matrix $\mathbf{X}$, the color to be assigned cannot be the yellow color, so $X_{21} = 0$, to obey $X_{11} + X_{21} \leq Y_1$. Thus, position X_{22} will have the value 1 assigned and a new color, green is assigned. Again, Equation (3.8) it is verified because just the color green is assigned to vertex v_2. Y_2 will also have the value 1, representing the second assigned color. In step 3, after analyzing $v_1 - v_2$ and $v_1 - v_3$ edges, as node v_3 is adjacent to v_1 but not adjacent to v_2, the color assigned to v_3 is the same color of v_2, in order to use the available color in non-adjacent nodes and achieve the goal of minimum number of colors. Again, conditions (3.7) and (3.8) are verified and as $v_1 = 3$, $v_2 = 1$ and v_4 and v_7 are still uncolored, then this means position X_{32} is set to 1 in matrix $\mathbf{X}$.

Looking at the illustration in step 4, vertex v_4 is associated with a third color, blue, because this vertices is adjacent to vertex v_1 and v_3, which are colored yellow and green. Condition (3.7) analyzes the edges $v_1 - v_2$, $v_1 - v_3$, $v_1 - v_4$ and $v_3 - v_4$, which are the colored edges. In matrix $\mathbf{X}$, the position X_{47} of the fourth row is assigned to 1, because a new color has been assigned. The process continues until the entire graph is colored after analysing all the nodes and at each algorithm step, the matrix **X** has a new row corresponding to the vertices already analyzed and assigned to a specific color, until the solution, the vector **Y**, is found. In this example, $Y = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$, that is, 3 is the minimum number of colors found. The evolution of the matrix **X** and vector **Y** in relation to the steps in Figure 3.15 is shown in Table 3.3. For each step illustrated in Figure 3.15, a position $X_{ij} = 1$ when it has color j assigned to vertex i. The solution presented in Table 3.3 corresponds to one of $\frac{8!}{(8-3)!} = 336$ possible solutions of the graph coloring problem that arise from Equations (3.4) and (3.5). For this reason, to reduce these large number of solutions, Equations (3.11) and (3.12) have been developed.

To solve our ILP graph coloring problem, we have used the MATLAB function *intlinprog*, which is a mixed-integer linear programming solver developed to find the minimum of a problem specified by an objective function. The syntax of inputs and outputs of this function is presented in Table 3.4, with the optimization options set in *'optimooptions'* of MATLAB.

Through the inputs described in Table 3.4, *intlinprog* minimizes the objective function f described by Equation (3.6) through the Branch and Bound method and at the simulation end, displays the number of nodes and corresponding assigned colors, the total computational time, the integer number of solutions and the relative gap. The relative gap corresponds to the difference between internally generated upper and lower bounds that tends to zero, while all feasible solutions are checked. When the objective value is within a gap tolerance (absolute or relative) of the optimal value, the ILP minimization stops and the optimal solution is found. As an output, the method returns the solution x that minimizes the objective function and the 'exitflag' that identifies the algorithm stopping condition. If the method fails to perform the minimization, this 'exitflag' describes the error that has occurred.

In the next, the implementation of the ILP formulation in MATLAB to solve the graph coloring problem is explained. Figure 3.16 shows the implementation in MATLAB of the ILP algorithm, considering Equation (3.6) with the constraints (3.7) and (3.8).

In Figure 3.16, lines 62 and 63 of the code represent the information of the graph used: variable E represents the edges and N represents the total number of nodes of the path graph and defines the space of solutions that is sought for this problem. Variable represented in Equation (3.4) (line 114) corresponds to the matrix **X** with the colors solutions assigned to each edge of the path graph and variable represented in Equation

TABLE 3.3. Evolution of matrix **X** and vector **Y** in ILP algorithm.

Steps	**Evolution of Matrix *X***	**Evolution of Vector *Y***
Step 1	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 0 0 0 0 0 0 0
Step 2	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 0 0
	0 1 0 0 0 0 0 0	
Step 3	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 0 0
	0 1 0 0 0 0 0 0	
	0 1 0 0 0 0 0 0	
Step 4	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 1 0
	0 1 0 0 0 0 0 0	
	0 1 0 0 0 0 0 0	
	0 0 0 0 0 0 1 0	
Step 5	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 1 0
	0 1 0 0 0 0 0 0	
	0 1 0 0 0 0 0 0	
	0 0 0 0 0 0 1 0	
	1 0 0 0 0 0 0 0	
Step 6	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 1 0
	0 1 0 0 0 0 0 0	
	0 1 0 0 0 0 0 0	
	0 0 0 0 0 0 1 0	
	1 0 0 0 0 0 0 0	
	1 0 0 0 0 0 0 0	
Step 7	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 1 0
	0 1 0 0 0 0 0 0	
	0 1 0 0 0 0 0 0	
	0 0 0 0 0 0 1 0	
	1 0 0 0 0 0 0 0	
	1 0 0 0 0 0 0 0	
	1 0 0 0 0 0 0 0	
Step 8	$X =$ 1 0 0 0 0 0 0 0	$Y =$ 1 1 0 0 0 0 1 0
	0 1 0 0 0 0 0 0	
	0 1 0 0 0 0 0 0	
	0 0 0 0 0 0 1 0	
	1 0 0 0 0 0 0 0	
	1 0 0 0 0 0 0 0	
	1 0 0 0 0 0 0 0	
	0 0 0 0 0 0 1 0	

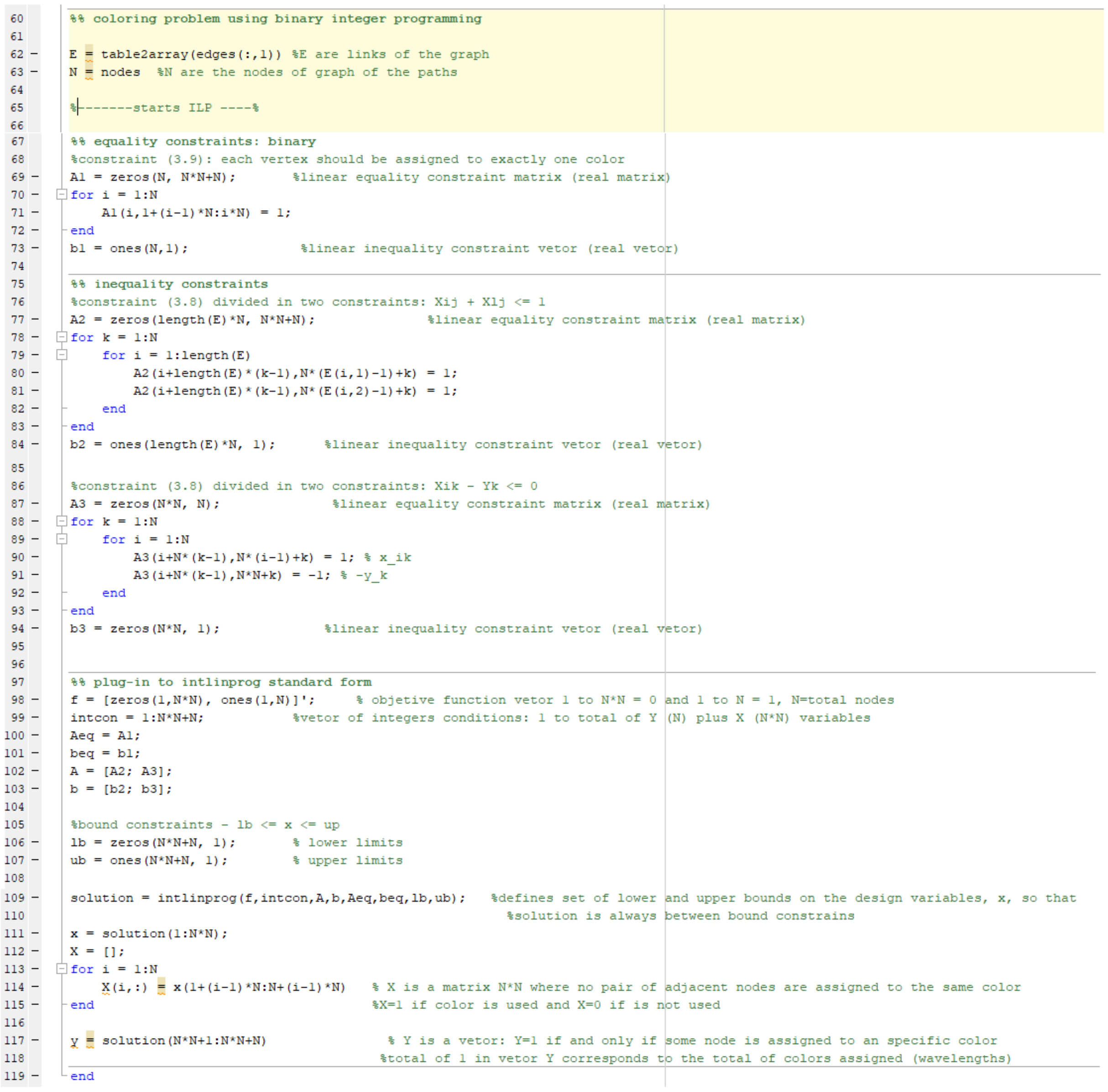
(3.5) (line 117) corresponds to the vector **Y**, representing the solution for the color class S.

Lines 69-73 impose the constraint (3.8), which specifies that each vertex should be assigned to just exactly one color, having the matrix A_1 dimensions N by $N^2 + N$. Constraint (3.7) is divided to guarantee two complementary situations: adjacent vertices have different colors associated and $Y_j = 1$ only if any vertex is associated with that color j. The first condition, in lines 77-84, corresponds to the implementation of the expression $X_{ij} + X_{lj} \leq 1$, where line 77 is the initialization of a matrix of zeros A_2 with dimension

Table 3.4. Syntax of *intlinprog* function.

Input Arguments	
f	Objetive function to minimize according to the problem
intcon	Vector of integers defining the number of variables of the problem
A	Constraint matrix to implement the inequality $A \cdot x \leq b$
b	Constraint vector to implement the inequality $A \cdot x \leq b$
Aeq	Constraint matrix to implement the equality $Aeq \cdot x = beq$
beq	Constraint vector to implement the equality $Aeq \cdot x = beq$
lb	Lower bound on the design value x
ub	Upper bound on the design value x
Options	To change *'intlinprog'* default options and improve processing
Options for *'intlinprog'* specified as the output of optimoptions	
AbsolGapTol	'intlinprog' stops if the difference between the internally calculated upper (U) and lower (L) bounds on the objective function is less than or equal to AbsoluteGapTolerance: $U - L \leq AbsoluteGapTolerance$
MaxNodes	Maximum number of nodes *'intlinprog'* explored using the branch-and-bound. Default value: 1e7.
MaxTime	Maximum time (seconds) that *'intlinprog'* runs. Default value: 7200.
LPMaxIter	Maximum number of simplex algorithms iterations per node during branch-and-bound
TolInteger	Maximum deviation from an integer that a component of the solution x can have and still be considered as an integer
TolGapRel	*'intlinprog'* stops if the relative difference between upper and lower bounds on the objective function is less than or equal to TolGapRel, which by default is 1e-4
Output Arguments	
x	Solution, returned as a vector that minimizes objective function
fval	Value of the objective function at x
exitflag	Algorithm stopping condition, returned as a integer identifying the reason for stopping the algorithm

N by $N^2 + N$. The second condition, on lines 87-94, corresponds to the expression $X_{ij} - Y_j \leq 0$, which guarantees that no pair of adjacent nodes are assigned to the same color.



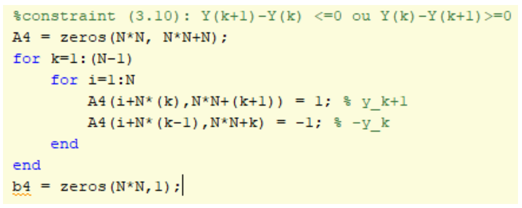
```
60       %% coloring problem using binary integer programming
61
62 -     E = table2array(edges(:,1)) %E are links of the graph
63 -     N = nodes  %N are the nodes of graph of the paths
64
65       %-------starts ILP ----%
66
67       %% equality constraints: binary
68       %constraint (3.9): each vertex should be assigned to exactly one color
69 -     A1 = zeros(N, N*N+N);       %linear equality constraint matrix (real matrix)
70 -     for i = 1:N
71 -         A1(i,1+(i-1)*N:i*N) = 1;
72 -     end
73 -     b1 = ones(N,1);              %linear inequality constraint vetor (real vetor)
74
75       %% inequality constraints
76       %constraint (3.8) divided in two constraints: Xij + Xlj <= 1
77 -     A2 = zeros(length(E)*N, N*N+N);              %linear equality constraint matrix (real matrix)
78 -     for k = 1:N
79 -         for i = 1:length(E)
80 -             A2(i+length(E)*(k-1),N*(E(i,1)-1)+k) = 1;
81 -             A2(i+length(E)*(k-1),N*(E(i,2)-1)+k) = 1;
82 -         end
83 -     end
84 -     b2 = ones(length(E)*N, 1);      %linear inequality constraint vetor (real vetor)
85
86       %constraint (3.8) divided in two constraints: Xik - Yk <= 0
87 -     A3 = zeros(N*N, N);              %linear equality constraint matrix (real matrix)
88 -     for k = 1:N
89 -         for i = 1:N
90 -             A3(i+N*(k-1),N*(i-1)+k) = 1; % x_ik
91 -             A3(i+N*(k-1),N*N+k) = -1; % -y_k
92 -         end
93 -     end
94 -     b3 = zeros(N*N, 1);             %linear inequality constraint vetor (real vetor)
95
96
97       %% plug-in to intlinprog standard form
98 -     f = [zeros(1,N*N), ones(1,N)]';     % objetive function vetor 1 to N*N = 0 and 1 to N = 1, N=total nodes
99 -     intcon = 1:N*N+N;           %vetor of integers conditions: 1 to total of Y (N) plus X (N*N) variables
100 -    Aeq = A1;
101 -    beq = b1;
102 -    A = [A2; A3];
103 -    b = [b2; b3];
104
105      %bound constraints - lb <= x <= up
106 -    lb = zeros(N*N+N, 1);       % lower limits
107 -    ub = ones(N*N+N, 1);        % upper limits
108
109 -    solution = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub);   %defines set of lower and upper bounds on the design variables, x, so that
110                                                             %solution is always between bound constrains
111 -    x = solution(1:N*N);
112 -    X = [];
113 -    for i = 1:N
114 -        X(i,:) = x(1+(i-1)*N:N+(i-1)*N)    % X is a matrix N*N where no pair of adjacent nodes are assigned to the same color
115 -    end                                    %X=1 if color is used and X=0 if is not used
116
117 -    y = solution(N*N+1:N*N+N)                % Y is a vetor: Y=1 if and only if some node is assigned to an specific color
118                                              %total of 1 in vetor Y corresponds to the total of colors assigned (wavelengths)
119 -    end
```

Figure 3.16. Implementation of the ILP algorithm in MATLAB.

Line 98 defines the objective function given by (3.6) that will return the solutions of the matrix $\mathbf{X}$ and vector $\mathbf{Y}$, where the $\sum Y_j$ corresponds to the number of colors assigned.

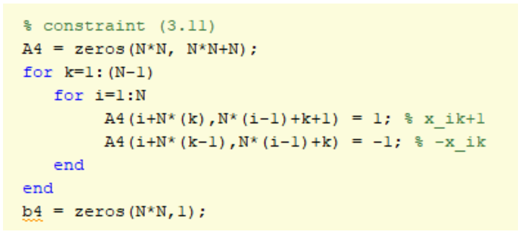
The input arguments of the MATLAB function *intlinprog* are specified in lines 98 to 107. Line 98 indicates that the variables to minimize correspond to the number of colors Y_j. Lines 106 and 107 correspond, respectively, to the definition of the lower limits and upper limits, i.e., bounds constraints where solution x must be found. The lower bound (lb) and upper bound (up) are restricted to 0 and 1 respectively, so that it restricts the result to binary results. Line 109 gives the integer linear programming solution.

Figures 3.17 (A), (B) and (C) refers to the code implemented to constraints (3.9), (3.10) and (3.11)+(3.12), respectively. Each of these blocks of code runs independently



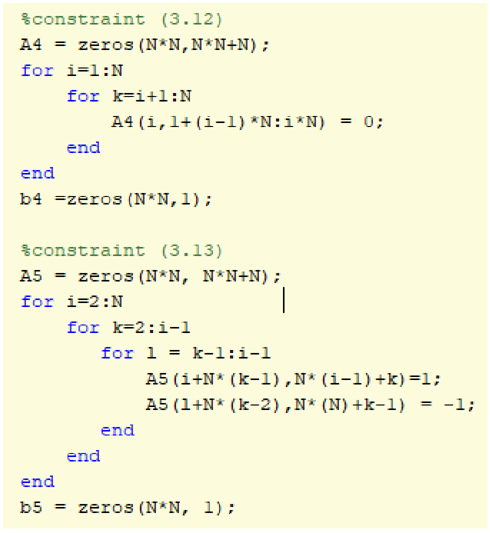
```
%constraint (3.10): Y(k+1)-Y(k) <=0 ou Y(k)-Y(k+1)>=0
A4 = zeros(N*N, N*N+N);
for k=1:(N-1)
    for i=1:N
        A4(i+N*(k),N*N+(k+1)) = 1; % y_k+1
        A4(i+N*(k-1),N*N+k) = -1; % -y_k
    end
end
b4 = zeros(N*N,1);
```

(A) Constraint 3.9



```
% constraint (3.11)
A4 = zeros(N*N, N*N+N);
for k=1:(N-1)
   for i=1:N
         A4(i+N*(k),N*(i-1)+k+1) = 1; % x_ik+1
         A4(i+N*(k-1),N*(i-1)+k) = -1; % -x_ik
   end
end
b4 = zeros(N*N,1);
```

(B) Constraint 3.10



```
%constraint (3.12)
A4 = zeros(N*N,N*N+N);
for i=1:N
    for k=i+1:N
        A4(i,1+(i-1)*N:i*N) = 0;
    end
end
b4 =zeros(N*N,1);

%constraint (3.13)
A5 = zeros(N*N, N*N+N);
for i=2:N
    for k=2:i-1
       for l = k-1:i-1
           A5(i+N*(k-1),N*(i-1)+k)=1;
           A5(1+N*(k-2),N*(N)+k-1) = -1;
       end
    end
end
b5 = zeros(N*N, 1);
```

(C) Constraints 3.11 and 3.12

FIGURE 3.17. Optimization Constraints

from the others in order to reduce the space of solutions under analysis and, consequently, to improve the computational performance, changing the code of Figure 3.17 (A) by 3.17 (B) and then 3.17 (B) by 3.17 (C). When these constraints are considered for computational time study, they are included in the construction of matrix **A** and vector **b** of lines 102 and 103 of the code in Figure 3.16. Taking again into account the example in Figure 3.15 and since it is an 8-node topology, if it is running the original ILP algorithm code with constraints (3.7) and (3.8), there are $\frac{8!}{(8-3)!} = 336$ possible candidate solutions. If constraint (3.9) is applied then the space of solutions decreases significantly to $3! = 6$ and when it is applied the restrictions in Figure 3.17 (C) instead of restrictions in Figure 3.17 (B), the space of solutions corresponds to a unique solution of three colors, which is very simple to find.

3.5.3. Tabu Search Algorithm

Tabu Search is a metaheuristic algorithm, used to solve different kinds of problems, such as graph coloring [**4**]. The idea of this algorithm, when applied to graph coloring problems, is to answer the following question: given a graph *G(V,E)*, where V represent the set of vertices and E the set of edges between vertices, is it possible to feasibly color it with k colors? This algorithm has the following main steps:

(1) It starts by defining an initial solution S to color the graph with a predefined value k of colors, which can be obtained randomly or by using a constructive heuristic, like Greedy or DSATUR [**4**].

(2) The algorithm next proceeds by computing the number of clashes (i.e. two adjacent vertices with the same color) which is represented by function *f(S)*, defined by:

$$f(S) = \sum_{\forall\{u,v\}\in E} g(u,v) \tag{3.13}$$

with

$$g(u,v) = \begin{cases} 1 & \text{if } c(u)=c(v), \\ 0 & \text{otherwise.} \end{cases}$$

where $c(u)$ is the color of vertex u and $c(v)$ is the color of vertex v.

If $g(u,v) = 1$ it means that the color of the two adjacent vertices u and v is the same and therefore a clash occurs. The aim of the algorithm is to eliminate the clashes, i.e., $f(S) = 0$.

If this number is not zero, a new solution S' is obtained, by using the neighbor operator, which is defined as follows: if a vertex v is assigned to a color i, a neighbor operator corresponds to a color change of vertex v to a new color j. Note that to obtain this new solution S' there are some vertices color changes that can not be done. These vertices color changes are registered in a list of forbidden vertices' color changes, called the Tabu list **T**. This list is used to avoid previous undesired and already checked solutions [**7**].

If with this new solution S' condition $f(S') < A(f(S))$ is verified, the best solution S' is found. In this condition, A is an "aspiration level" function that gives the possibility that solutions S' with a superior number of clashes be chosen, with the aim to escape from local minima [**7**]. If $f(S) = 0$ and number of operations (*Niter*) is less than *Nmax* it means that a solution with k colors is found. If the number of operations is *Nmax* the algorithm stops.

(3) If a solution with k colors is found the algorithm starts again with k-1 colors.

Figure 3.18 represents a scheme that allows to understand how the best candidate solutions are selected in Tabu Search algorithm. It starts evaluating each candidate move. If from the current candidate list, the candidate move returns a better evaluation than any other move admissible so far, then is analysed if it is a candidate tabu. If does not represent a move with higher evaluation, is added to the candidate list. When the tabu status is checked, there are two hypotheses: the candidate is tabu and it's aspiration level is evaluated (if satisfy aspiration criteria is an admissible move, if not is added to the

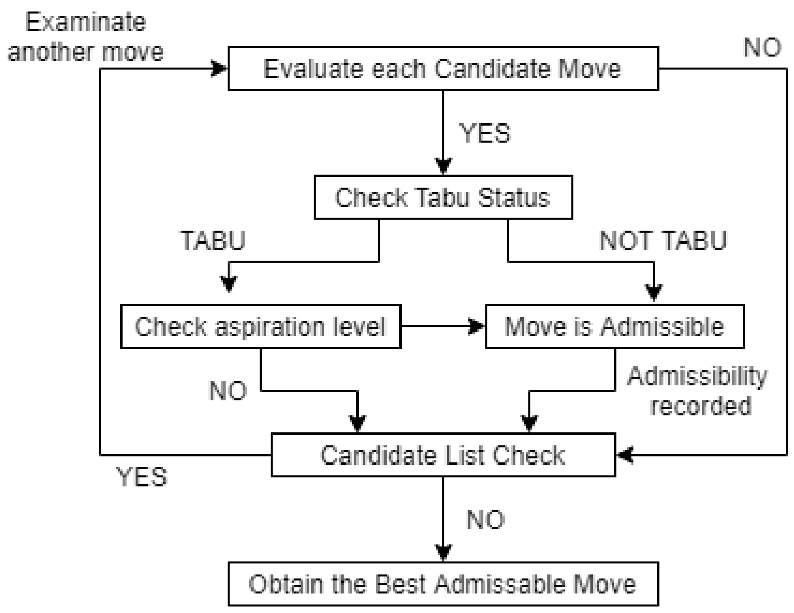


FIGURE 3.18. Selecting the Best Candidate on Tabu Search [**6**]

candidate list) or is not tabu and is designated as best admissable candidate and added to the candidate list. The candidate list is verified in order to understand if any move in this list is better or if the candidate list should be extended. If don't, the best admissible move is obtained.

Figure 3.19 shows the pseudocode of the Tabu Search algorithm.

This algorithm starts by defining an initial solution S and the Tabu list **T** size. While clashes occur, i.e., $f(S) > 0$, a new solution $S´$ is searched, the condition $f(S') < A(f(S))$ is checked and the Tabu list **T** is updated.

```
TABU SEARCH ($S = (V_1 \ldots V_k)$; $T$; Niter = 0)

(1)while $f(S) > 0$ and Niter $<$ Nmax
(2)        generate neighbours $S'$ with move $S \rightarrow S' \notin$ T
                                                    or $f(S') < A(f(S))$
(3)        (stop generation if $f(S') < f(s)$ )
(4)
(5)        $T \leftarrow T'$
(6)        $S \leftarrow S'$
(7)        $Niter \leftarrow Niter + 1$
(8)end
(9)  if $k$ colors guarantees $f(S) = 0$
(10)                      $k \leftarrow k - 1$
(11)     end
```

FIGURE 3.19. Pseudocode for Tabu Search algorithm.

If a solution with k colors ensures that the number of clashes is zero, the algorithm tries a solution with $k-1$ colors (lines 9-10). The algorithm stops when no solution with $k-1$ colors is achieved or the number maximum of iterations is reached.

Figure 3.20 represents the same network of Figure 3.14, but now the coloring is going to be made with the Tabu Search algorithm. The algorithm starts with a random coloring solution (step 1). In this case, this random solution has 4 colors and 2 clashes, so $f(S') = 2$. As explained before, this is an improper coloring, since there are adjacent nodes with the same color. Once the solution has clashes, the algorithm generates a new solution $S\acute{}$. In step 2, the neighbor operator works in the vertex 3 changing the color pink to green. This operation eliminate the clash that occur between vertices 1 and 3 but create a new clash between nodes 3 and 7, keeping the number of clashes to 2. The Tabu list **T** is updated with this move, i.e., vertex 3 changes from pink to green.

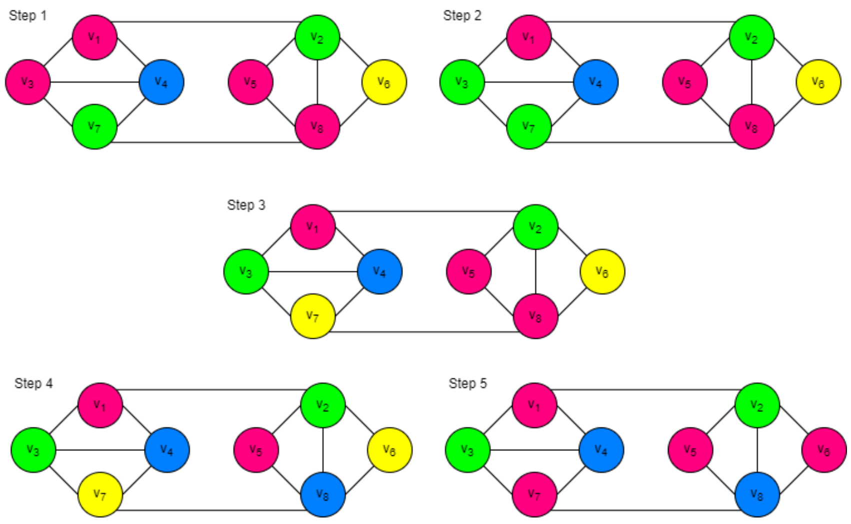


FIGURE 3.20. Tabu Search algorithm example.

In step 3, the clash between vertices 3 and 7 is eliminated and a better solution is found that reduces the number of clashes to one, between adjacent nodes 5 and 8. In this step, the neighbor operator works in vertex 7, coloring it with yellow, which is one of the four initial colors of the solution. The Tabu list **T** is updated again, with the addition of the move corresponding to vertex 7 colored in green. Thus, in step 3 there is a solution with 4 colors and one clash between vertices 5 and 8.

In step 4 occurs the coloring of vertex 8 with blue. Thus, all clashes are eliminated, i.e., $f(S) = 0$ and a complete proper k coloring is found with k=4. At this stage, the algorithm tries the solution k=3 colors, which represents a decrease of variable k. If vertices 6 and 7 are colored with pink or vertices 1 and 5 are colored with yellow, this solution is feasible, with a total of k colors smaller than initially tested, making it a better solution.

Next, (line 9 in the pseudocode) the algorithm tries the solution with k=2 colors, but no solution with proper coloring is found, so, k=3 is the minimum number of colors.

In this dissertation we have used the Tabu Search algorithm implementation given in [**4**] and developed in C++. This implementation has the following main features:

(1) it uses constructive methods for calculate the initial k colors, such as Greedy or DSATUR (by default, the algorithm uses DSATUR).

(2) each time a feasible solution is found, the number of colors decreases in order to find a better solution.

(3) the total number of constraint checks (see definition in 3.5.1) imposed by default is 1×10^8. In our experiments we have used a higher number of constraint checks, 5×10^{11} in order to find the best possible solutions [**4**] for the studied topologies.

3.6. Conclusion

In this chapter, the basic concepts that define and help to understand the graph coloring problem have been described. The concepts regarding the presentation of a graph and the clarification of a correct coloring such as the vertices adjacency or non-adjacency, a complete coloring, clashes, improper coloring, feasible coloring or optimal solution have been defined through the illustrations of real examples. Also, some restrictions imposed on solving this problem are clarified, such as the k minimal or the coloring of adjacent nodes with different colors. The graph coloring problem complexity was highlighted, with some examples, in order to understand the complexity of the problem in what concerns the number of constraint checks needed to obtain the optimal solution.

Finally, the three graph coloring algorithms that are studied and compared in this dissertation were presented and explained: Greedy, exact algorithm based on ILP and Tabu Search algorithms. For each of the algorithms, illustrative examples of their behavior were given, as well as the respective pseudocodes. This study pretends to compare them in what concerns to the number of wavelengths (i.e. number of colors) and the respective computational effort. These results will be analysed in Chapter 4.

CHAPTER 4

Application of Exact and Heuristic Graph Coloring Techniques for WA in Optical Networks

4.1. Introduction

In this chapter, the graph coloring algorithms discussed in Chapter 3, i.e. Greedy, exact algorithm based on ILP and Tabu Search, are applied to small networks, e.g. a mesh network with 8 nodes, and real networks, e.g. COST239, NSFNET, UBN and CONUS network with 30 nodes. Section 4.2 briefly explains the RWA planning tool used in this work to solve the RWA problem. Section 4.3 characterises the parameters of the network physical topology and section 4.4 characterises the same parameters, but applied to the logical topology. In section 4.5, the performance of the exact algorithm based on ILP, discussed in section 3.5.2, is assessed and analysed. We discuss the number of colors predicted and the computational time for achieving an optimal solution for several small networks and compare its results to the ones obtained with the Greedy algorithm. The particular cases of ring networks and bipartite networks are also analysed. In section 4.6, the performance of the Tabu Search algorithm, discussed in section 3.5.3, is studied. First, its performance is studied on random graphs and compared with the Greedy algorithm. Then, the performance of these algorithms considering regular (e.g. rings) and non-regular real networks is evaluated and compared. Finally, in section 4.7, the main conclusions of this chapter are presented.

4.2. RWA Planning Tool

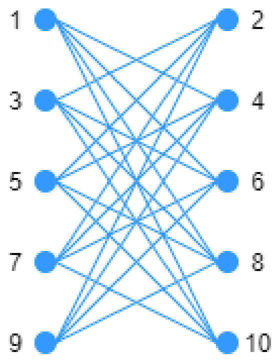
The planning tool used to solve the RWA problem was developed in [**15**] and extended in this work in order to study the performance of the exact algorithm based on ILP and the Tabu Search algorithm as graph coloring WA techniques. The planning tool has the following three main functionalities:

(1) Definition of the physical and logical topologies characterized, respectively, by the adjacency and the traffic matrices.
(2) Routing algorithm based on the Yen´s k-shortest path algorithm.

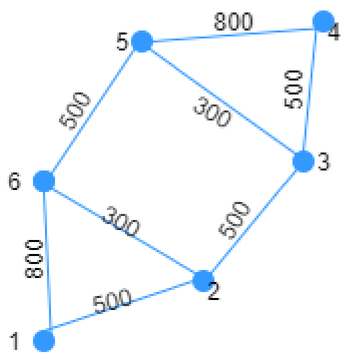
(3) WA algorithms based on graph coloring techniques: Greedy algorithm, exact algorithm based on ILP and Tabu Search algorithm. Note that before using the Graph Coloring algorithms, the path graph $G(W, P)$ must be computed. This graph is obtained from the graph $G(V, E)$ that represents the physical topology. The vertices of $G(W, P)$ represent the optical paths and P represents the set of links between those vertices [**15**]. These links are between one or more vertices (i.e. paths) that share one or more physical links. After obtaining the graph $G(W, P)$, the vertices can be colored considering the Greedy, the ILP and the Tabu Search algorithms explained in Chapter 3. The number of colors obtained corresponds to number of wavelengths needed for solving the RWA problem.

4.3. Parameters of the Network Physical Topology

In this section, some parameters of the physical topology of several networks, in particular the average and variance node degree, are defined and computed. These parameters result from the analysis of the adjacency matrix of the network (see matrix (2.1)). Figures 4.1, 4.2 and 4.3 represent the physical topology of the networks studied, which are classified, respectively, as small networks, ring networks and real networks. The ring networks studied range from 6 to 60 node networks, and the real networks are COST239 [**11**], NSFNET [**12**], UBN [**13**] and CONUS with 30 nodes network [**14**]. Note that in the Figure 4.1 (B) and Figure 4.3 the values on the edges represent the distance, in kilometers, between the nodes, which is the metric used by the routing algorithm, whereas in Figures 4.1 (A) and 4.2, the edges have no values since the metric used, in these cases, is the number of hops.



(A) Bipartite network with 10 nodes.



(B) Partial Mesh network with 6 nodes.

FIGURE 4.1. Physical topology of two small networks

The network COST239, represented in Figure 4.3 (A), is an European transparent optical Network with 11 nodes linked by 26 fibre optic links, that was built with the

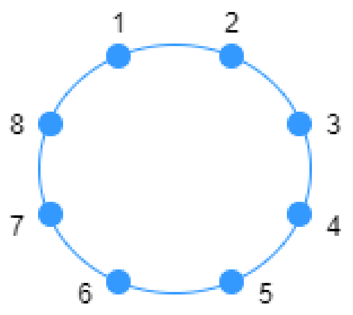


FIGURE 4.2. Physical topology of a regular ring network with 8 nodes.

goal of carrying all international traffic between the main cities of Europe [**11**], [**20**]. The National Science Foundation Network, best known as NSFNET (Figure 4.3 (B)), appeared in 1990 to enhance communications, collaboration and resource sharing in the United States scientific and engineering research community. This network has 14 nodes and 21 links [**12**]. This is a benchmarking network [**21**]. The UBN or US Backbone Network (Figure 4.3 (C)) is a network with 24 nodes and 43 links that appeared between 1997 and 1999 in the USA in order to respond to the fast and massive flow of information in urban locations where distances vary greatly [**13**]. The CONUS network, with 30 nodes and 36 links, represented in Figure 4.3 (D), is a fibre optic network in USA deployed to guarantee a higher protection, since it has some cross-continental paths completely interconnected [**14**]. It was a network developed for use in research on large-scale Dense Wavelength Division Multiplexing (DWDM) networks [**22**].

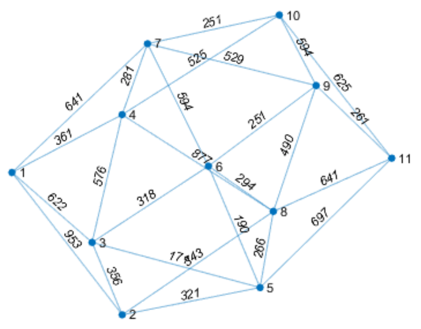
A network physical topology can be characterized by several parameters such as the average node degree and the variance of the node degree, respectively, given by [**23**]:

$$\overline{d} = \frac{\sum_{i=1}^{N} Dg_i}{N} \tag{4.1}$$

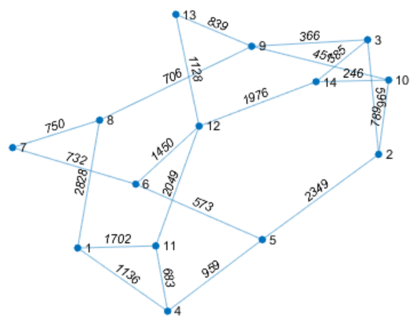
$$\sigma_d^2 = \frac{\sum_{i=1}^{N} (Dg_i - \overline{d})^2}{N-1} \tag{4.2}$$

where Dg_i is the node degree of vertex i and N is the number of vertices (nodes). The variance of the node degree helps to understand how regular the network is from the point of view of the number of connections at each node in the network [**23**].

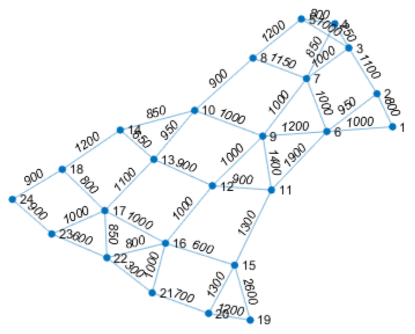
Table 4.1 presents the information regarding the network physical topologies with respect to the number of nodes and links, average and variance of the node degree. It can be observed that regarding ring networks, the number of nodes is equal to the number of links and the average node degree and the variance of the node degree are, respectively,



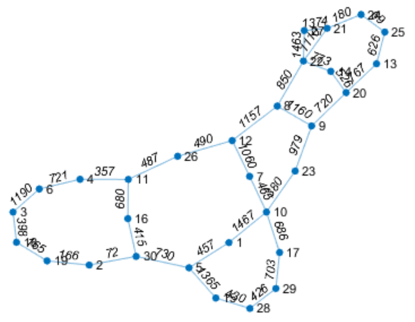
(A) COST239 network.



(B) NSFNET network.



(C) UBN network.



(D) CONUS 30 nodes network.

FIGURE 4.3. Physical topology of the real networks.

TABLE 4.1. Physical topology parameters for some networks.

Network	**Nodes**	**Links**	**Average Nodes Degree**	**Variance Node Degree**
Mesh 6 nodes	6	8	2.7	0.27
Ring 6 nodes	6	6	2.0	0.0
Ring 7 nodes	7	7	2.0	0.0
Ring 10 nodes	10	10	2.0	0.0
Ring 15 nodes	15	15	2.0	0.0
Ring 20 nodes	20	20	2.0	0.0
Ring 45 nodes	45	45	2.0	0.0
COST239	11	26	4.7	0.4
NSFNET	14	21	3.0	0.3
UBN	24	43	3.6	0.9
CONUS	30	36	2.4	0.4

2.0 and 0, which means that all nodes have degree 2, i.e., all nodes have 2 input/output connections. When the variance node degree is null, it means that all nodes have the same number of connections. Higher variances correspond to more irregular networks.

4.4. Parameters of the Network Logical Topology

The logical topology considered for the networks presented in section 4.3 is a full mesh topology, i.e., there is a logical connection between every node in the network. The parameters used to characterize the physical topology in the previous section can also be used to characterize the logical topology. Table 4.2 presents these parameters considering that a full mesh logical topology (with one unit of traffic - in this work we consider that the unit of traffic is one OTU-4) is applied over the physical topologies presented in the previous section. Table 4.2 also presents the number of nodes of the network and the number of paths. In this scenario, the average node degree for a network with a total mesh logical topology is $N-1$. Furthermore, the variance of the node degree is zero for the logical topologies considered, because all nodes have the same number of edges.

TABLE 4.2. Network logical topology parameters.

Network	**Number of nodes**	**Number of paths**	**Average Node Degree**	**Variance Node Degree**
Mesh 6 nodes	6	15	5	0.0
Ring 6 nodes	6	15	5	0.0
Ring 7 nodes	7	21	6	0.0
Ring 10 nodes	10	45	9	0.0
Ring 15 nodes	15	105	14	0.0
Ring 20 nodes	20	190	19	0.0
Ring 45 nodes	45	990	44	0.0
COST239	11	55	10	0.0
NSFNET	14	91	13	0.0
UBN	24	276	23	0.0
CONUS	30	435	29	0.0

For a ring network with a full mesh logical topology, the number of paths (bidirectional paths) is given by [**23**]

$$N_{paths} = \frac{N(N-1)}{2} \tag{4.3}$$

Note that the parameters presented in Table 4.2 result from the analysis of the traffic matrix of the network (see matrix (2.2)). For example, the total number of 1's in each row of the traffic matrix corresponds to the number of connections of each node at the logical level.

4.5. Performance of an Exact Algorithm Based on ILP

In this section, the performance of the exact algorithm based on ILP, presented in subsection 3.5.2, is studied. In subsection 4.5.1, the behaviour of this algorithm considering the different constraints, discussed in section 3.5.2, is studied and analyzed considering the following small networks: bipartite network with 10 nodes, 8-node partial mesh network, rings with 6 and 7 nodes and a small mesh network with 6 nodes. For these networks, it is possible to find the optimal solution using the ILP algorithm. Then, in subsection 4.5.2, more complex networks, rings with higher number of nodes and non-regular real networks, are considered in order to compare the ILP results with the Greedy algorithm results. In subsections 4.5.3 and 4.5.4, the particular cases concerning ring networks and bipartite networks with different dimensions (i.e. number of nodes) are studied with more detail. Note that all results below were obtained by simulations performed on a computer with an Intel XEON-CPU E5-2620v2 2.1 GHz processor and 128 GB of RAM.

4.5.1. Small Networks

In this subsection, the performance of an exact algorithm based on ILP is studied for small networks. Tables 4.3 and 4.4 show the number of paths, number of colors (i.e. number of wavelengths) and computational time for the constraints defined in eqs. (3.7), (3.8), (3.9), (3.10) and (3.11) + (3.12) for different network topologies. In Table 4.3, the 8-node (Figure 3.20) and bipartite with 10 nodes logical topology are presented and, in Table 4.4, the ring physical topology with 6 and 7 nodes and a mesh physical topology with 6 nodes presented in Figure 4.1 (B) are considered.

TABLE 4.3. Results of ILP algorithm for logical topology networks

Network		ILP Algorithm				
Type	Number of paths	Number of colors	Computational times (sec) with the restrictions			
			(3.7) + (3.8)	(3.9)	(3.10)	(3.11) + (3.12)
8 node topology (Figure 3.20)	8	3	0.18	0.09	0.07	0.04
Bipartite 10 nodes	10	2	0.27	0.08	0.07	0.06

TABLE 4.4. Results of ILP Algorithm for physical topology networks

Network		ILP Algorithm				
Type	Number of paths	Number of colors	Computational times (sec) with the restrictions			
			(3.7) + (3.8)	(3.9)	(3.10)	(3.11) + (3.12)
Ring 6 nodes	15	6	766.2	419.1	407.5	166.6
Mesh 6 nodes	15	8	39847.3	16244.0	15983.2	174.9
Ring 7 nodes	21	6	48427.4	17714.8	17416.1	9055.5

As can be observed in Tables 4.3 and 4.4, the computational time predicted by the ILP algorithm decreases as the complexity of the constraints increases, i.e., the restriction defined in eqs. (3.7) + (3.8) leads to much higher computational times than (3.11) +

(3.12). This happens because the space of solutions, initially with a total number of solutions equal to $\frac{n!}{(n-k)!}$ (n is the number of vertices in the path graph and k is the number of colors) decreases considerably with the application of restrictions, and therefore the search time for the optimal solution also decreases [**4**]. When constraint (3.9) is used, the total number of solutions decreases to $k!$, and the most significant decrease in computational time is obtained, i.e., a decrease between 46% (for ring with 6 nodes) and 71% (for bipartite with 10 nodes) in computational time [**4**]. When constraint (3.9) is replaced by constraint (3.10), the difference in computational time is not significant, about 2% for physical topology networks and 21% and 10% for both 8-node and bipartite logical topology networks. Then, by using constraints (3.11) and (3.12), there is again a large decrease in the computing time, i.e., about 42% and 20% for logical topology networks and between 50% and 98% for physical topology networks. With these constraints, for each possible k-coloring, only a unique permutation of the first k permutations is specified [**4**]. In the case of the mesh topology with 6 nodes, the difference between the first restriction and (3.10) is almost 7 hours. When constraint (3.10) is changed to (3.11) + (3.12), the computational time improves again (about 4 hours) because redundant possible solutions are eliminated [**4**]. So, it can be concluded that for this network, that there is an improve of about 11 hours when restriction (3.7)+(3.8) are replaced by restrictions (3.11) + (3.12).

Thus, although the initial constraint (3.7)+(3.8) returns the optimal solution, the introduction of more complex constraints can return optimal solutions much faster. Also, for the networks with more nodes such as the ring with 7 nodes, the use of the complex constraints has a higher impact on the reduction of the computational time.

4.5.2. Comparison of the Greedy Algorithm with the Exact Algorithm

In this subsection, the exact algorithm based on ILP results are compared with the ones obtained with the Greedy algorithm considering the number of colors and computational effort.

Table 4.5 shows the number of paths, the number of colors and computational time obtained with the Greedy and the exact algorithms for the topologies analyzed in the previous sections. The Greedy algorithm used considers three different sorting strategies: descending order, ascending order and random order, whose number of colors is obtained from the average of 10 simulation runs [**15**]. The particular cases of ring networks and bipartite networks are more extensively analyzed in subsections 4.5.3 and 4.5.4, respectively.

TABLE 4.5. Total number of colors and computational effort of Greedy and ILP Algorithms for several network topologies.

Network		Greedy Algorithm				ILP Algorithm	
Type	Number of paths	Number of colors			Computat. Time (sec)	Number of colors	Computat. Time (sec)
		descend. order	ascendent order	rand. order			
8 node Topology	8	3	3	3	0.032	3	0.04
Bipartite 10 nodes	10	5	5	2	0.023	2	0.058
Mesh 6 nodes	15	8	8	8	0.022	8	174.9
Ring 6 nodes	15	6	6	6	0.022	6	166.63
Ring 7 nodes	21	6	7	7.3	0.033	6	9055.5
Ring 10 nodes	45	15	17	15.4	0.15	15	27228.78
COST239	55	8	9	8.3	0.088	8	32402.5
NSFNET	91	24	25	24.1	0.15	24	180076.8
Ring 15 nodes	105	28	34	33.6	0.20	32	180055.7
Ring 20 nodes	190	55	68	59.1	0.57	optimal solution not found	
UBN	276	64	74	65.9	0.50	optimal solution not found	
CONUS 30 nodes	435	123	144	123.9	1.35	optimal solution not found	
Ring 45 nodes	990	253	318	286.6	5.5	optimal solution not found	

From Table 4.5, it is concluded that the Greedy using ascending order tends to return a higher number of colors than the descending one, as the number of paths in the networks increases. The Greedy algorithm using the random order strategy returns, in general, a number of colors between the ones predicted by the descending and ascending strategies, except for the case of the bipartite with 10 nodes network, where the random ordering gives better results than the descending/ascending order. In fact, with the random order the optimum solution is obtained in this scenario as can be seen in Table 4.5 by comparing with the ILP result. This case is studied with more detail in subsection 4.5.4. From Table 4.5, it can be also observed that the Greedy algorithm with descending order and the ILP give the same number of colors for the following networks: 8-node topology, mesh with 6 nodes, rings with 6, 7 and 10 nodes, COST239 and NSFNET. For a ring with 15 nodes, the number of colors given by the ILP does not reach the optimal solution found by the Greedy, because the computational time reaches the computer time limit, which is 18×10^4 seconds.

With the exact algorithm based on ILP, as can be seen in Table 4.5, some results give "optimal solution not found". This is due to either exceeding the limit computational time of the MATLAB implementation or lacking of available RAM memory to process data.

4.5.3. A Particular Case: Ring Networks

Ring networks are an example of a regular network [**24**] and, in this case, an analytical expression for the number of colors (with one unit of traffic per demand) can be found, respectively, for an even and odd, number of nodes [Chapter 8 [**25**]], assuming a shortest path routing:

$$W = \frac{N^2 + 2 \times N}{8} \tag{4.4}$$

$$W = \frac{(N-1)^2 + 2 \times (N-1)}{8} \tag{4.5}$$

Table 4.6. Number of colors obtained by Equations (4.4) and (4.5), Greedy and exact algorithm based on ILP for each ring network.

Nodes	**Equations (4.4) and (4.5)**	**Greedy**	**Exact algorithm based on ILP**
6	6	6	6
10	15	15	15
15	28	28	32
20	55	55	optimal solution not found
45	253	253	optimal solution not found

Table 4.6 shows the number of colors obtained for ring networks with different number of nodes by Equations (4.4) and (4.5), Greedy and exact algorithm based on ILP . From Table 4.6, it can be observed that the results obtained with the exact and the Greedy algorithms are in agreement with the analytical results for rings with 6, 10 and 15 nodes. For rings with 20 and 45 nodes, the exact algorithm based on ILP did not found an optimal solution due to memory constraints.

Although ILP always finds the optimal solution through massive solution testing, it is concluded that when the network has a high number of nodes (> 20) the optimal solution is not found in an acceptable computational time.

4.5.4. Another Particular Case: Bipartite Networks

Bipartite networks, presented in section 3.4, are also an example of a regular network. These networks have always an even number of vertices. In bipartite networks, the node degree is always the same, equal to $\frac{N}{2} - 1$, e.g., for a 10 node bipartite, the node degree is four. So, when using the Greedy algorithm, the coloring is totally dependent on the sorting strategy.

Figure 4.4 shows the four possible colorings of using the Greedy with a random sorting in a 10 node bipartite network.

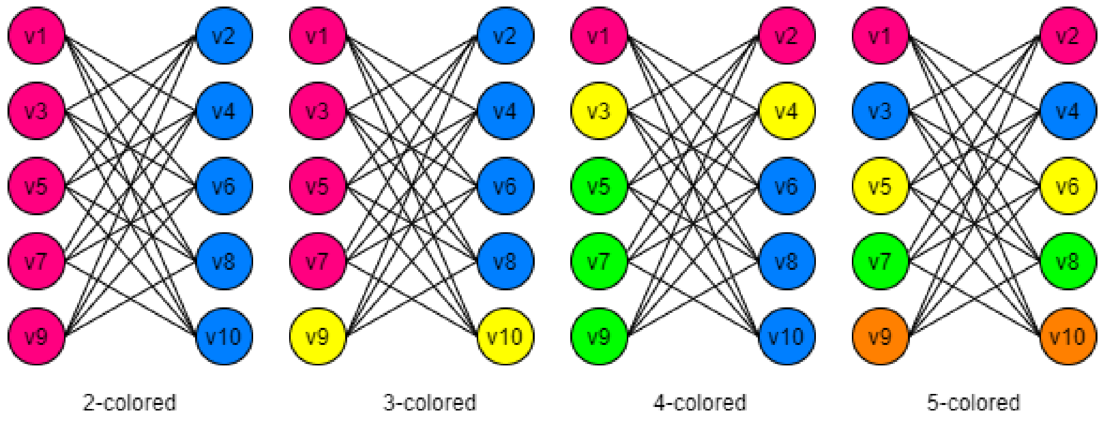


FIGURE 4.4. Different coloring for the bipartite graph with 10 nodes.

A bipartite graph corresponds to a graph $G = (V_1, V_2, E)$ where the vertex sets are defined as $V_1 = \{v_1, v_3, ..., v_{n-1}\}$ and $V_2 = \{v_2, v_4, ..., v_n\}$ and where edges set is defined as $E = \{\{v_i, v_j\} : v_i \in V_1 \wedge v_j \in V_2 \wedge i + 1 \neq j\}$ [**4**]. The last condition in the definition of the set of edges, i.e., $i + 1 \neq j$ guarantees that vertex v_1 does not connect to v_2, v_3 does not connect to v_4, and so on.

The results obtained in Figure 4.4 depend on the order the vertices are colored. For example, the two color case can be obtained when $\pi = (v_1, v_3, ..., v_{n-1}, v_2, v_4, ..., v_n)$ [**4**]. This scenario has a higher probability of occurrence since there are a lot sequences where the odd vertices appear first than the even vertices, and vice-versa. A possible permutation that leads to the 3 colors outcome is $\pi = (v_9, v_{10}, v_6, v_7, v_4, v_1, v_3, v_8, v_5, v_2)$ and a possible permutation for the 4 color case is $\pi = (v_1, v_2, v_3, v_4, v_{10}, v_6, v_7, v_8, v_5, v_9)$. The 5-color scenario appears only when the color order is $\pi = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10})$ or in the reverse order. In this case, the outcome is half the total number of nodes, $\frac{N}{2}$ colors [**4**]. This scenario happens with only 2 possible combinations in a total of 10!, so the probability of occurrence is very low, $\frac{2}{10!}$.

Table 4.7 shows the results of bipartite graphs with 10, 20, 30, 40 and 50 vertices. In the exact algorithm based on ILP case, the optimal result of two colors is always reached, as well as in most of the simulations using the Greedy, corresponding to the minimum value of colors obtained, which is also the most probable outcome. For all the bipartite networks, if the Greedy is sorted from v_1 to v_n or in reverse order, the total number of estimated colors is equal to $\frac{N}{2}$, being the maximum value of colors reached.

Table 4.7. Total number of colors and computational effort of Greedy and ILP algorithms for bipartite graphs.

Network		Greedy Algorithm			Exact Algorithm	
Bipartite	Number of paths	Number of colors: max value	Number of colors: min value	Computat. Time (sec)	Number of colors	Computat. Time (sec)
10 nodes	10	5	2	0.021	2	0.099
20 nodes	20	10	2	0.025	2	8565.3
30 nodes	30	15	2	0.031	2	11232.5
40 nodes	40	20	2	0.033	2	25434.8
50 nodes	50	25	2	0.039	2	30922.6

Regarding the computational time, each algorithm requires more time to obtain a solution as the number of vertices increases in the bipartite network. As expected, the exact algorithm based ILP takes significantly much longer time to provide a solution than the Greedy.

4.6. Performance of the Tabu Search Algorithm

In this section, first, we will study the performance of the Tabu Search algorithm on random graphs. Then, the comparison between the performance of the Tabu Search algorithm with the performance of the other algorithms studied previously for real regular and non-regular network scenarios is presented. Note that the Tabu Search algorithm used in this work is within the software tool developed and provided in [**4**] and uses the DSATUR algorithm as the constructive method to calculate the initial k colors. For the Greedy algorithm, the descending strategy order is considered since it is the order that returns the best results [**15**].

4.6.1. Random Path Graphs

Random graphs, $G_{n,p}$, are graphs with n vertices that are randomly generated by using the parameter p, which corresponds to the probability of two vertices being adjacent [**4**]. Note that each one of these random graphs correspond to a graph path that need to be colored. The parameter p is given by

$$p = \frac{\sum_{i=1}^{n} \frac{Dg_i}{n-1}}{n} \tag{4.6}$$

In the following analysis, we will present results for $n = 20$, 50, 100 and 1000 vertices. For each value of p, 25 random graphs are generated through simulation, and the average value regarding the number of colors of these simulations is analyzed. We also consider a maximum number of constraint checks of 5×10^{11} for the Tabu Search to limit the computational process [**4**].

The random graphs are built by generating random matrices with a size of $n \times n$ and according to the chosen probability p. Pairs of adjacent vertices are randomly generated (using an uniform distribution) in the matrix. In particular, when $p = 1$, it means that the degree of any vertex of the matrix is $n - 1$, i.e., all vertices of the matrix are adjacent to each other. After obtaining the random graphs, the Greedy (using descending order strategy) and the Tabu Search algorithms are applied for graph coloring and comparison purposes. It is possible to change this method to calculate the initial k by using the in-built Greedy algorithm, which always considers a random ordering of the vertices. In this case, the initial k value is always worse than the initial k obtained with the DSATUR.

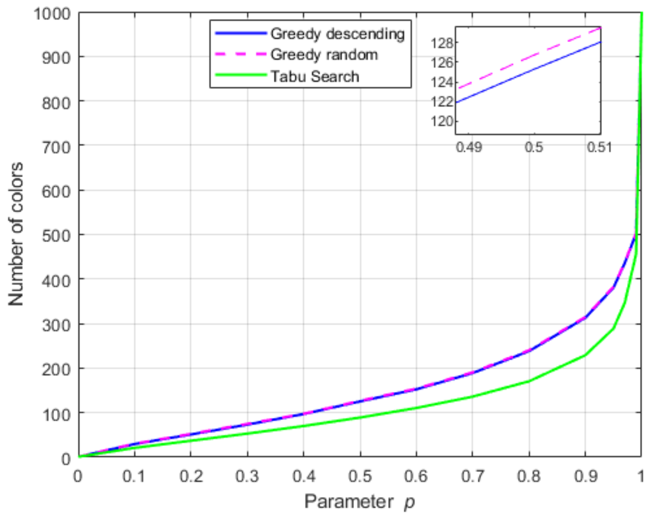


FIGURE 4.5. Number of colors as a function of p for $n = 1000$ calculated using the Greedy (descending and random order) and the Tabu Search algorithms.

Figure 4.5 shows the number of colors as a function of p for $n = 1000$ calculated using the Greedy (with random and descending order strategy) and the Tabu Search algorithms. A very good agreement between these results and the ones presented in Figure 4.6 of [**4**] is found. From these results, we can observe that the Tabu Search returns fewer colors than the Greedy algorithm, and that the difference in the number of colors increases with the increase of the value of p, but the relative percentage increase is always around 40%. For example, for $p = 0.1$, the Tabu Search gives 21 colors and the Greedy 30 colors, which is a difference in 9 colors (43 % increase); for $p = 0.5$, the Tabu Search provides 89 colors and the Greedy gives 125 colors, which is a difference in 36 colors (40 % increase) and for $p = 0.9$, the Tabu Search gives 229 colors and the Greedy gives 313 colors, which is a difference of 84 colors (37 % increase). Although in Figure 4.5 the results obtained with Greedy random and Greedy descending orders seem to be very similar, in the inset of Figure 4.5, it can be seen that the descending order gives always slightly better results (1 color difference).

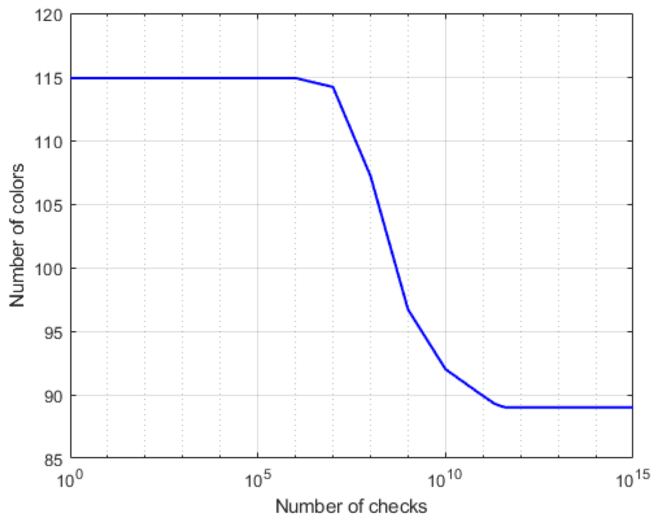


FIGURE 4.6. Number of colors as a function of the number of constraint checks considering the Tabu Search algorithm for $n = 1000$ and $p = 0.5$.

Another important issue to consider when comparing the Tabu Search and the Greedy algorithm is the computation time. In order to compute the time needed for the Tabu Search to provide a graph coloring solution, we first check the minimum number of constraint checks needed to achieve the best solution. In Figure 4.6, the number of colors as a function of the number of constraint checks is represented for $n = 1000$ and $p = 0.5$, considering the generation of 10 random graphs for each number of constraint checks.

As can be observed in Figure 4.6, the number of colors is minimized only when the number of checks is above 10^{11}. From that point on, the number of colors remains practically constant and the minimum number of colors has been reached. Thus, for $n = 1000$, a number of constraint checks of 4×10^{11} is sufficient to ensure that the optimal solution is reached. It was verified that the parameter p does not influence the number of constraint checks required to achieve the minimum number of colors with the Tabu Search, being this number only dependent on the size of the network.

Figure 4.7 shows the evolution of the number of colors for Tabu Search algorithm as the number of constraint checks imposed increases for n = 20, 50,100, 200 and $p = 0.5$. The number of constraint checks in this study varies between 1 and 1×10^{12}.

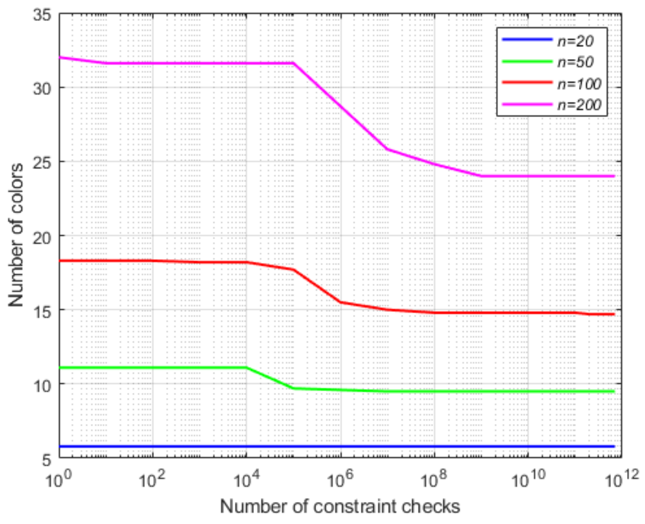
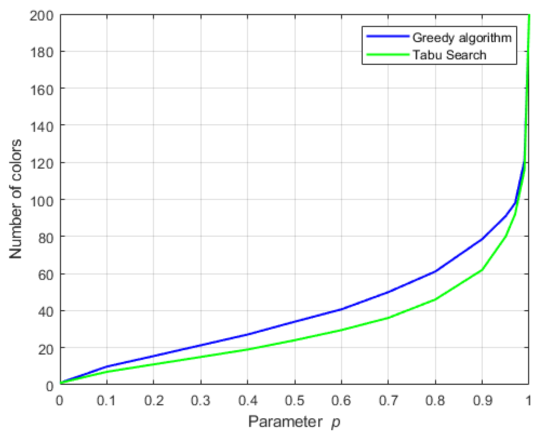


FIGURE 4.7. Number of colors as a function of the number of constraint checks imposed on Tabu Search for $p = 0.5$ and for different values of n.

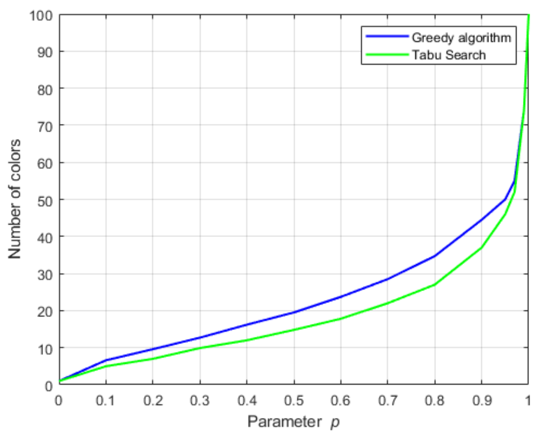
From Figure 4.7, it can be concluded that for $n = 20$, only one constraint check is sufficient to obtain the best solution with the Tabu Search. Likewise for $n = 50$, $n = 100$ and $n = 200$, a number of constraint checks above, respectively, 1×10^7, 1×10^8 and 1×10^9 is sufficient to obtain the best solution. Above these values of constraint checks, the required number of colors increases with the network size, as expected. In what concerns the computational effort, as n decreases, the required number of constraint checks that minimizes the number of colors also decreases, and therefore the computation time for lower n will be lower as well.

After studying the number of constraint checks to use in the Tabu Search algorithm in random graphs, the behavior of the Greedy and Tabu Search algorithms in finding the optimal color solution is studied. Figure 4.8 shows the performance of both Greedy and Tabu Search algorithms for random graphs with 200, 100, 50 and 20 nodes, as a function of p.

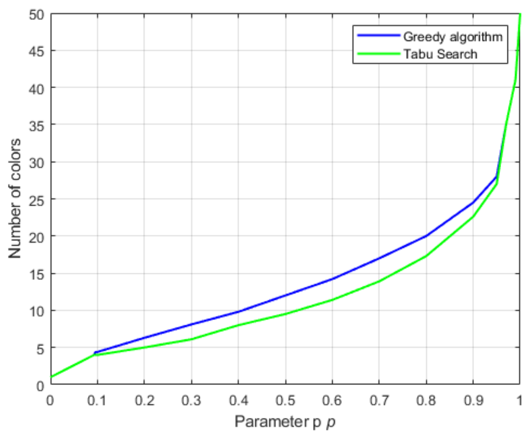
As can be observed in Figure 4.8, the Tabu Search performs better than the Greedy algorithm by predicting less colors and this behavior is more pronounced for increasing values of n and p. For networks with a lower number of nodes, the number of colors is



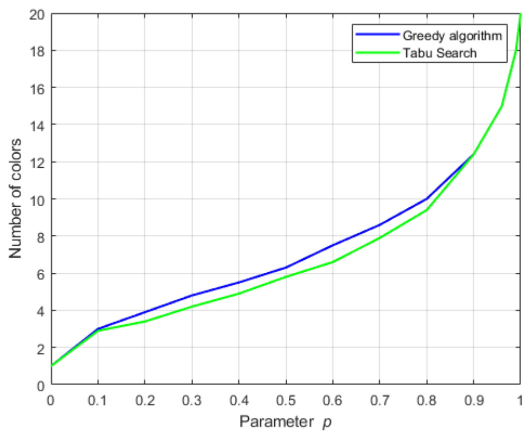
(A) $n = 200$



(B) $n = 100$



(C) $n = 50$



(D) $n = 20$

FIGURE 4.8. Performance of Greedy and Tabu Search for random graphs.

lower than in networks with more vertices. For $p = 0$ and $p = 1$, there are no difference between the algorithms in the number of colors, as these are the cases where no adjacency and full adjacency between nodes occur, respectively. That is, if no nodes are adjacent to others, the same color, independently of the algorithm, can be applied to all nodes. Similarly, if there is adjacency of a node with all nodes in the network, then each node is assigned its own color.

To better understand the conclusions presented above, Figure 4.9 shows the increase in percentage of the number of colors as a function of p predicted by the Greedy algorithm in comparison with the Tabu Search, considering n = 20, 50, 100, 200, 1000. Note that 25 simulations were performed to obtain the results presented.

From Figure 4.9, it can be concluded that the higher the number of vertices in the network, the higher the percentage growth. The maximum value found is around 40%. For $p = 0.1$ and $n = 200$, there is a maximum 40% increase in the number of colors used

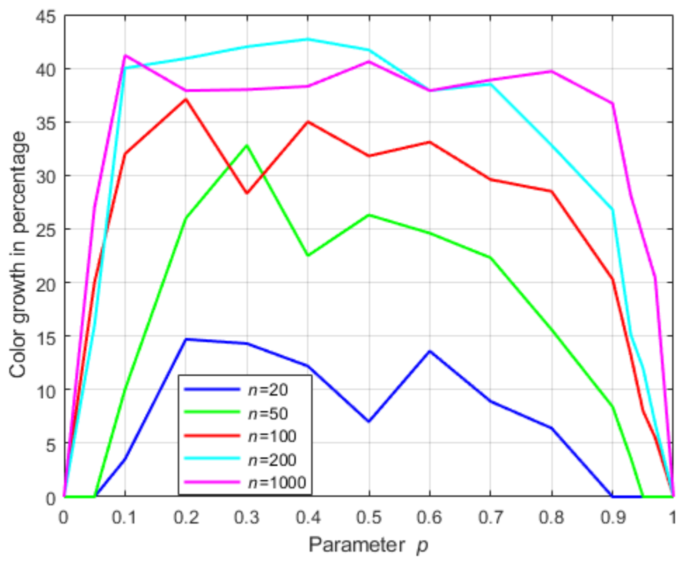


FIGURE 4.9. Percentage of the number of colors increase between Tabu Search and Greedy algorithms for $n = 20, 50, 100, 200$ and 1000 as a function of p.

by the Greedy algorithm in comparison with the number of colors predicted by the Tabu Search, whereas, for $n = 100$, there is only 32% increase, for $n = 50$, there is 10% increase and, for $n = 20$, there is only 3% increase. From these results, it can be concluded that the decrease in the total number of colors attributed by the Tabu Search in comparison with the Greedy is more pronounced in networks with more nodes. For $p = 0.9$ and for $n = 200$, there is only 26% increase in the numbers of colors used by the Greedy algorithm. Similarly, for $n = 100$ and $n = 50$, the difference of total number of colors between both algorithms decreases, respectively, to 20% and 8%. For $n = 20$ and $p = 0.9$, the results predicted by both algorithms is equal. This is because the network has so few nodes that it becomes easier for both algorithms to reach the same solution.

The study of the number of constraint checks needed to minimize the number of colors for each network was developed to save computation time and is shown is Figure 4.10. Note that the Tabu Search line represents the minimum number of constraint checks that allows to minimize the number of colors using the Tabu Search. The number of checks, as verified in Figure 4.10, depends remarkably on the number of vertices n in the network. The study presented has been performed considering 25 simulations for a parameter $p = 0.5$. However, from extensive simulation results, it has been verified that there is no dependence of the number on constraint checks required of the parameter p used. Thus, regardless the value of p, from Figure 4.10, it is possible to obtain an approximation for the required number of constraint checks for any number of vertices n

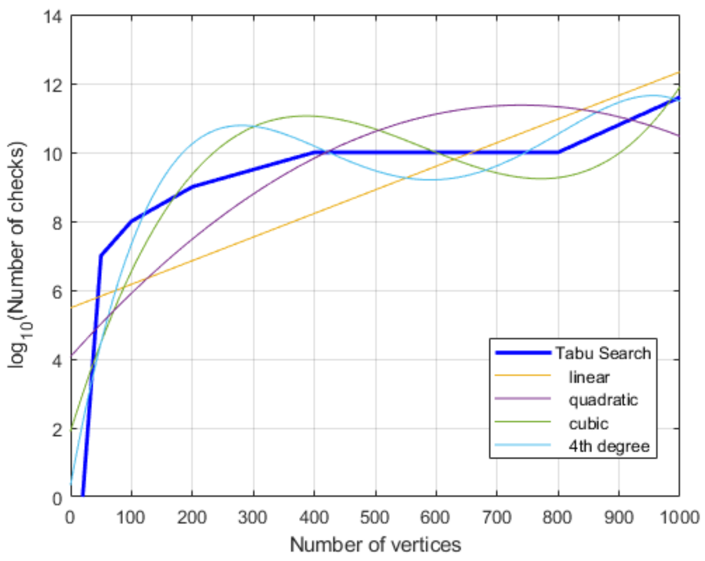


FIGURE 4.10. Number of constraint checks as function of the number of vertices and some fitting curves.

that minimizes the number of colors in a random graph. To obtain such approximation in Figure 4.10, a curve fitting has been performed to the obtained Tabu Search curve, including linear, quadratic, cubic and fourth degree fittings. Note that for proper fitting, the logarithm base 10 of the number of checks has been used.

As can be observed in Figure 4.10, the linear and quadratic fittings, respectively, below $n = 500$ and $n = 300$, predict a number of checks much smaller than the one given by the Tabu Search. The cubic and fourth degree fittings predict a very similar number of constraint checks, but we prefer to use the cubic fitting, because it represents a simpler function. The cubic fitting is given by:

$$6.364 \times 10^{-8} n^3 - 1.105 \times 10^{-4} n^2 + 5.691 \times 10^{-2} n + 1.897 \qquad (4.7)$$

4.6.2. Ring Networks: Comparison Between Tabu Search, Exact and Greedy Algorithms

In this subsection, a comparative study between the Tabu Search, exact algorithm based on ILP and Greedy algorithms is performed considering ring networks. All results obtained in this subsection consider a full mesh logical topology. In the case of the exact algorithm based on ILP, we use the constraints (3.11)+(3.12) to save computational time. Table 4.8 shows the number of colors computed by the Greedy, ILP and Tabu Search algorithms and also the analytical results from eqs. (4.4) and (4.5) that return the minimum number of colors for rings with different number of nodes.

Table 4.8. Number of colors obtained by eqs. (4.4) and (4.5), Greedy, ILP and Tabu Search algorithms for each ring network.

Network	**Eqs. (4.4) and (4.5)**	**Greedy**	**ILP**	**Tabu Search**
6 nodes	6	6	6 (optimal solution)	6
7 nodes	6	6	6 (optimal solution)	6
8 nodes	10	10	10 (optimal solution)	10
9 nodes	10	10	11 (feasible solution)	10
10 nodes	15	15	15 (feasible solution)	15
15 nodes	28	28	32 (feasible solution)	29
20 nodes	55	55	optimal solution not found	55
25 nodes	78	78	optimal solution not found	81
30 nodes	120	120	optimal solution not found	120
35 nodes	153	153	optimal solution not found	157
40 nodes	210	211	optimal solution not found	210
45 nodes	253	253	optimal solution not found	262
50 nodes	325	327	optimal solution not found	325
55 nodes	378	378	optimal solution not found	385
60 nodes	465	469	optimal solution not found	465

Comparing the results obtained by the several algorithms in Table 4.8, it is possible to observe that the analytical results obtained by eqs. (4.4) and (4.5) are in agreement with the ones obtained by the Greedy and Tabu Search algorithms for rings with $N = 6, 7, 8, 9, 10, 20$ and 30 nodes. Regarding the remaining presented networks, it is concluded that for rings with an odd number of vertices, the Greedy returns the optimal solution calculated by the equations, while the Tabu Search does not. This result was not expected since the Tabu Search algorithm is supposed to have a better or at least equal performance to the Greedy algorithm. On the contrary, for networks with an even number of vertices, the Tabu Search provides the optimal number of colors, while the Greedy predicts a slightly higher number of colors. For some networks, the ILP is not able to compute the optimal solution. In the case of Greedy algorithm for regular networks with even number of nodes, the results are equal to those presented in [**15**] and the difference in the number of colors is enhanced as the number of paths increases.

4.6.3. Real Networks: Comparison Between Tabu Search, Exact and Greedy Algorithms

In this subsection, the three algorithms studied in this chapter are applied to the non-regular real networks COST239, NSFNET, UBN and CONUS30. The Greedy algorithm considers the descending order strategy and the exact algorithm based on ILP uses the constraints (3.11)+(3.12). In this subsection, we also study the influence of the logical topology on the number of colors obtained by the Greedy and Tabu Search algorithms, as well as the influence of the average and variance node degree of the path graph $G(W, P)$.

Table 4.9 presents the number of colors and the corresponding simulation times for the networks COST239, NSFNET, UBN and CONUS30, considering a full mesh logical topology, obtained with the Greedy, Tabu Search and ILP algorithms. In the case of the Greedy algorithm, two simulation times are presented: the simulation time measured with our MATLAB script and the simulation time measured by running the software provided in [**4**], in the Microsoft Visual Studio. Note that the Tabu Search algorithm runs also on Microsoft Visual Studio and the ILP runs as a MATLAB script.

Table 4.9. Number of colors and simulation time obtained by Greedy, exact algorithm based on ILP, and Tabu Search for some non-regular networks.

Network		Greedy Algorithm			ILP Algorithm		Tabu Search	
Name	Parameter p	Number of colors	Time (sec)	Time [**4**] (sec)	Number of colors	Time (sec)	Number of colors	Time (sec)
COST239	0.101	8	0.02	0.002	8	3.24×10^4	8	0.008
NSFNET	0.25	24	0.06	0.009	24	1.8×10^5	24	0.015
UBN	0.2250	64	0.07	0.014	optimal solution not found		64	0.096
CONUS30	0.379	123	0.12	0.040	optimal solution not found		123	0.373

From the results presented in Table 4.9, regarding the computational time, it is observed that the simulation time of the Greedy algorithm provided by [**4**] is 4 times faster than the Tabu Search time for the COST239 network and almost 10 times faster than the Tabu Search time for the CONUS30 network. The ILP, as expected, is the algorithm that takes the longest time to run, around 9 hours for the COST239 network and more than 48 hours for the NSFNET network, which can be justified by the higher p and higher number of paths of the NSFNET networks comparing to COST239. Thus, it is verified that the Greedy algorithm provided by [**4**] (which leads to a lower time than the one obtained with the Greedy developed in MATLAB) is faster than the Tabu Search, as expected, since the Tabu Search runs the Greedy algorithm several times, instead of just one.

Also, as observed in Table 4.9, it can be concluded that for Greedy and Tabu Search algorithms, the same number of colors are obtained for all the networks. These results are also verified by the exact algorithm based on ILP in the case of COST239 and NSFNET networks. For the other networks, an optimal solution could not be found due to time limitations. However, in section 4.6.1, considering random path graphs, we have seen that the Tabu Search predicts a lower number of colors than the Greedy algorithm. In order to understand this apparently contradicting behavior, in the following, we are going to study the influence of the traffic pattern in the number of colors obtained by the two algorithms. Therefore, we are going to change the traffic matrix in order to obtain various logical topologies different from the full mesh topology considered for the networks presented in Table 4.9. First, we define a metric called the percentage of network traffic, denoted as N_T. This metric ranges from 0 (no network traffic) to 100% (full mesh topology) and aims to quantify the change of network traffic in a traffic matrix considering only unitary traffic units, for different real networks.

Figure 4.11 shows the numbers of colors as a function of the percentage of network traffic N_T, studied for the non-regular real networks considered in Table 4.9, using the Greedy (considering descending strategy) and Tabu Search algorithms. From Figure 4.11, it can be observed that, when $N_T = 0$, as there is no traffic in the network, no colors are assigned. When $N_T > 0$, the two algorithms give exactly the same number of colors. Thus, it can be concluded that the change of the traffic matrix (i.e. the logical topology) does not produce any differences on the number of colors predicted by both algorithms. The reason for this behavior is going to be detailed next, but relies on the fact that the variance node degree of the path matrix is considerably greater than the corresponding average value.

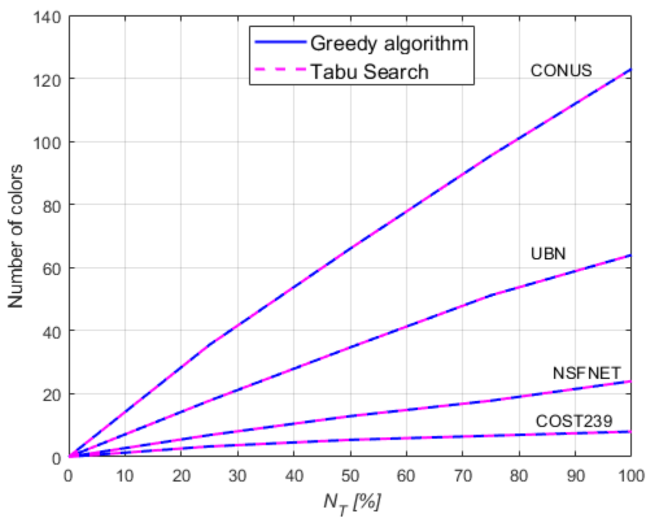


FIGURE 4.11. Number of colors provided by the Greedy and Tabu Search algorithms for non-regular real networks depending on N_T

Next, we are going to investigate the impact of the average and variance node degrees of the path graph $G(W, P)$ in the performance of the Greedy and Tabu Search algorithms, in order to try to explain why the two algorithms give the same number of colors when real networks are considered, independently of the network traffic, and different number of colors with random path graphs.

In Table 4.10, the parameter p, the average and variance node degrees of the path graph, $G(W, P)$ of the COST239 ($n = 55$ and $p = 0.101$), NSFNET ($n = 91$ and $p = 0.25$), UBN ($n = 276$ and $p = 0.225$) and CONUS30 ($n = 435$ and $p = 0.392$) networks and the respective number of colors predicted by the Greedy and Tabu Search algorithms are presented. The same network paramaters are also presented for random path graphs (where the corresponding path matrix is generated with a uniform distribution as in section 4.6.1), with the same number of paths and parameter p of the referred networks.

TABLE 4.10. Number of paths n, parameter p, average node degree, variance node degree and number of colors given by the Greedy with descending order and Tabu Search for the studied real networks and for random path graphs with the same characteristics, n and p, of the real networks.

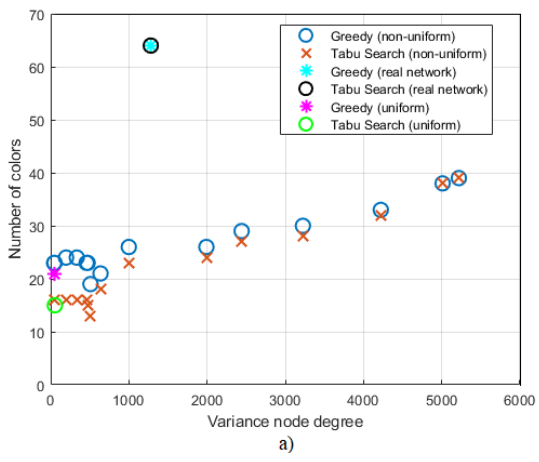
Network	Number of paths	Parameter p	Average node degree	Variance node degree	Greedy descend	Tabu Search	Color Difference
Real networks							
COST239	55	0.101	5.6	14	8	8	0
NSFNET	91	0.25	22.5	121.0	24	24	0
UBN	276	0.225	61.9	1.2840×10^3	64	64	0
CONUS30	435	0.392	164.6	4.3696×10^3	123	123	0
Random path graphs with uniform distribution							
$n = 55$ $p = 0.1$	55	0.101	5.88	4.6	5	4	1
$n = 91$ $p = 0.25$	91	0.25	22.5	14	11	8	3
$n = 276$ $p = 0.225$	276	0.225	61.9	51.9	21	15	5
$n = 435$ $p = 0.392$	435	0.392	170.2	164.5	47	34	13

As can be observed in Table 4.10, the average node degree in real networks and random path graphs is very similar as it depends on the number of paths and on the parameter p that is equal for the real and random networks. However, it can be noticed, that in random path graphs, the variance node degree has a similar magnitude than the average node degree, while, in real networks, the variance node degree is at least one order of magnitude higher than the average node degree. The lower variance found in random path graphs is due to the uniform distribution of 1´s in the path matrix $G(W, P)$ (e.g. for $p = 0.1$ it means that each line of the matrix has 10% ones and 90% zeros on average), while the higher variances found in real networks are due to the non-uniform distribution of 1´s in the path matrix $G(W, P)$. Furthermore, in random path graphs, for lower variance node degrees, the difference in colors between Greedy and Tabu Search is notorious, whereas in real networks both algorithms produce the same number of colors. For example, for $n = 435$ and $p = 0.392$, in Table 4.10, the variance node degree for the CONUS with 30 nodes network is 4369, whereas for the respective random graph, the variance has a much lower value, 164.5. So, this finding can justify the fact that the number of colors computed by the Greedy and Tabu Search algorithms when random graphs are used is different, while the same number of colors is obtained when real networks are considered.

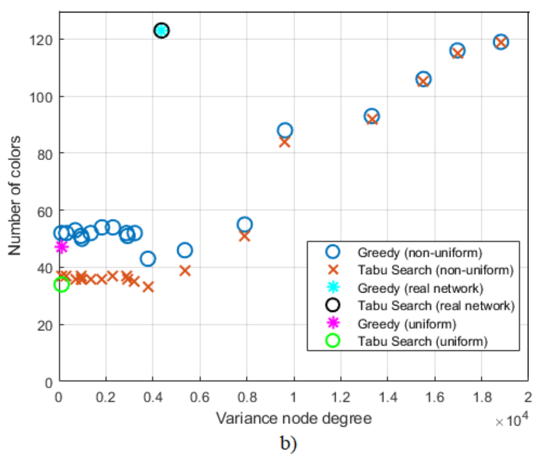
To further confirm this finding we are going to study the evolution of the number of colors as a function of the variance node degree of the path graph, $G(W, P)$, for both Greedy and Tabu Search algorithms.

Figure 4.12 shows the number of colors obtained with the Greedy and Tabu Search algorithms as a function of the variance node degree for random matrices using uniform and non-uniform distributions with the same number of paths n and parameter p of the real networks (A) UBN and (B) CONUS30. In Figure 4.12 (C), the case of the random graph shown in Figure 4.8 with $n = 100$ and $p = 0.5$ is studied also considering uniform and non-uniform distribution. In Figures 4.12 (A) and (B), we also represent the number of colors corresponding to the real cases of UBN and CONUS 30 network. To obtain the results of the Figure 4.12, a script has been developed to change the distribution of the 1's in the random path matrices in order to increase the variance node degree in relation to the average node degree without changing the parameter p of the path matrix. It has been verified that the parameter p of the network is kept constant for any variance node degree considered and that the average node degree is also constant and equal to the ones given in Table 4.10 for the UBN and CONUS30 networks.

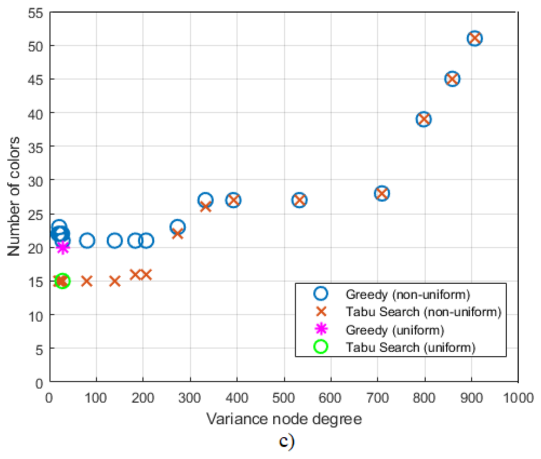
As can be seen in Figure 4.12, for higher variance node degrees, the number of colors given by Greedy and Tabu Search algorithms tends to converge, whereas for lower variances the number of colors produced by these algorithms is different, with Tabu Search algorithm presenting a lower number of colors. This behavior is observed for all the three networks studied. For example, in Figure 4.12 (C), for a low variance around 27, the Greedy algorithm leads to, respectively, 22 and 20 colors, with a non-uniform and a uniform distribution, while the Tabu Search gives 15 colors for both distributions. Likewise, in Figure 4.12 (C), for a high variance of 900, both algorithms produce around 52 colors with a non-uniform distribution. In Figures 4.12 (A) and (B), the number of colors



(A) $n = 276$ and $p = 0.225$



(B) $n = 435$ and $p = 0.392$



(C) $n = 100$ and $p = 0.5$

FIGURE 4.12. Number of colors estimated by the Greedy and Tabu Search algorithms for random graphs using a non-uniform distribution as a function of the variance node degree.

obtained in the UBN and CONUS with 30 nodes networks is also represented for both algorithms. It can also be observed that in these scenarios, there is no difference between the number of colors given by the Tabu Search and the Greedy algorithms, when the number of colors predicted is high, which for the non-uniform distribution, in Figure 4.12, happens, for high variance node degree. In the limit, when the maximum number of colors is used (i.e. all vertices are adjacent) both algorithms must return the same number of colors.

A more conclusive observation can be taken from Figure 4.13, where the number of colors provided by the Greedy and the Tabu Search algorithms for a network with $n = 435$ and $p = 0.392$ is represented as a function of the variance node degree. In Figure 4.13, several cases of the non-uniform distribution that lead to a high number of colors (above

80) have been collected. It can be observed that even for low values of the variance node degree, when a higher number of colors is required, both algorithms return the same number of colors. When the number of colors becomes lower than 80, in most cases, the Tabu Search algorithm gives less colors than the Greedy algorithm. For example, for a variance node degree equal to 5120, the two algorithms return the same number of colors, 140. However, for similar variance node degree, when the Greedy returns 70 colors, the Tabu Search returns only 62.

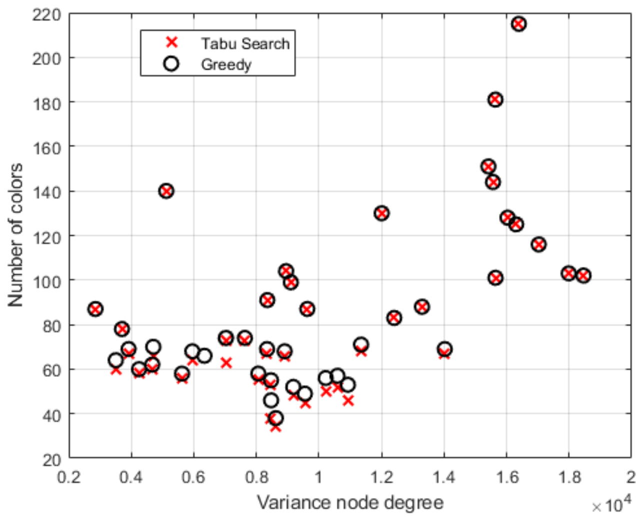


FIGURE 4.13. Number of colors provided by the Greedy and Tabu Search algorithms for a network with $n = 435$ and $p = 0.392$ as a function of the variance node degree.

So, it can be concluded that for the real networks tested, due to these networks resulting logical topology that always requires a high number of colors, the simpler and faster Greedy algorithm with descending order should be used, instead of the Tabu Search, since it it gives the same number of colors as the Tabu Search.

4.7. Conclusion

In this chapter, the performance of an exact algorithm based on ILP and of the Tabu Search algorithms has been studied in detail for wavelength (i.e. color) assignment in regular and non-regular real networks, as well as, in randomly generated path graphs and their performance has been compared with the one given by the Greedy algorithm.

We conclude that the exact algorithm based on ILP, although providing in theory the optimal solution, it requires high computation times. It was observed that this algorithm takes an unfeasible computational time for networks with a number of nodes larger than 20. However, we have also shown that by applying the constraints (3.11) and (3.12) in order to reduce the space solutions found with the initial constraints (3.8) and (3.9), a significant computational time is saved. In particular, for the simple mesh logical topology with 6 nodes network, the algorithm takes less 11 hours when using constraints (3.11) and (3.12). We have also used the Greedy algorithm with descending order in the same networks and have concluded that the Greedy algorithm gives in almost all the network scenarios studied the same results as the exact algorithm. One exception is in Bipartite networks, where the Greedy algorithm does not predict always the optimum number of colors.

Regarding the Tabu Search algorithm, when the path graph is obtained randomly with a uniform distribution, the Tabu Search can lead to a superior performance than the Greedy algorithm. In particular, when $n = 1000$ and $p = 0.5$, the Tabu Search algorithm returns only 89 colors, whereas the Greedy algorithm gives 124 colors. However, when non-regular real networks are considered, both Greedy and Tabu Search algorithms give the same number of colors. We have found that, independently of the variance node degree, when the number of colors required is high, the Greedy and Tabu Search algorithms tend to return the same number of colors. When the number of colors required is low, the Tabu Search outperforms the Greedy. So, we can conclude that in real networks, the resulting logical topology always demands a high number of colors, being the Greedy a more advantageous algorithm, due to its simplicity and fastness, while predicting the same number of colors as the Tabu Search.

CHAPTER 5

Conclusions and Future Work

In this chapter, the main conclusions of this work are presented and some possible future studies are suggested.

5.1. Conclusions

In this work, the behaviour and performance of different graph coloring algorithms for WA in optical networks with static traffic has been studied. The static RWA scenario goal is to minimize the number of colors to attribute to a specific number of optical paths in the network.

In Chapter 2, the basic concepts of optical networks are presented, such as physical and logical topologies, as well as, corresponding adjacent and traffic matrices. Moreover, a brief description of the usual RWA algorithms is presented.

Chapter 3 introduces some graph coloring basics and gives some examples for graph coloring applications, such as constructing schedules, taxi scheduling, and WA, the application that we study in this work. Then, the three graph coloring algorithms studied in this work, Greedy algorithm, exact algorithm based on ILP and Tabu Search algorithm, are explained using pseudocode and several examples of these algorithms are presented and discussed. The ILP formalism used for the exact algorithm is studied and implemented for several constraints that range from simple ones with high computational times to more complex constraints with faster computational times.

In Chapter 4, Greedy, Tabu Search and exact algorithm based on ILP are applied, as WA algorithms, to minimize the number of colors in several scenarios: random path graphs, ring networks and real networks (COST239, NSFNET, UBN, CONUS 30). First, the RWA tool is briefly described and the parameters of the physical and logical topologies of these networks are explained and calculated.

The exact algorithm based on ILP, although leading to optimum solutions when its application is feasible, even with restrictions imposed to save some computational time, takes a huge computational time to return the same number of colors as the Greedy algorithm using the descending order strategy. With a number of nodes larger than 20, it was not possible to find the optimal solution due to the computational time and memory limitations. The bipartite graphs were the only example studied where the number of colors estimated by the exact algorithm can be smaller than the Greedy.

Regarding the Greedy and Tabu Search algorithms, it was concluded that in the random path graphs using a uniform distribution to build the path matrix, the Tabu Search algorithm returned always a lower number of colors than the Greedy. For example, when $n = 1000$ and $p = 0.5$, Tabu Search returns less 35 colors than the Greedy algorithm. However, it was observed that in real networks, where the path matrix shows a non-uniform distribution, both algorithms predict the same number of colors, independently of the network traffic (or logical topology). We have concluded that the number of colors estimated in each scenario determines the behavior of the Tabu Search and Greedy algorithms, independent of the variance node degree of the path graph. For random graphs with low variance node degree, the Tabu Search generally gives less colors than the Greedy, as it happens with random path graphs with uniform distribution. However, even for low variance node degrees, when the number of colors required is high, both Tabu Search and Greedy algorithms give the same number of colors. For high variance node degree, as the number of colors required is always high, the Greedy and Tabu Search predict the same number of colors. As a final conclusion, for real networks, the Greedy with descending order is a simpler and better tool for WA, as it takes a lower computational time than the Tabu Search and estimates the same number of colors.

5.2. Future Work

In the following, possible future studies to continue the work developed are presented:

- Study the performance of evolutionary algorithms, inspired by biological evolution, as graph coloring algorithms [**4**] and compare its performance with the Tabu Search and Greedy performance;
- Further study the statistical properties of the path matrix, in terms of higher order moments such as the skewness of the node degree as a function of the number of colors, considering the Greedy, Tabu Search and evolutionary algorithms;
- Consider traffic demands with more that one traffic unit and assess the performance of the Greedy, Tabu Search and evolutionary algorithms. In this case, each vertex of the path graph can have multiple colors [**4**].

References

[1] Peter J. Winzer, David T. Neilson, and Andrew R. Chraplyvy. Fiber-optic Transmission and Networking: the Previous 20 and the Next 20 years. *Journal Optics Express*, vol. 26, pp. 24190-24239, September 2018.

[2] J. Simmons. Optical Network Design and Planning, 2nd edition. New York, USA: Springer, 2014.

[3] H. Zang, J. P. Jue, and B. Mukherjeey. A Review Of Routing and Wavelength Assignment Approaches for Wavelength-routed Optical WDM Networks. *Optical Networks Magazine*, vol. 1, pp. 47–60, March 2000.

[4] R. M. R. Lewis. A Guide to Graph Coloring: Algorithm and Applications. Switzerland: Springer, 2016.

[5] I. Duarte, L. Cancela and J. Rebola. Graph Coloring Heuristics for Optical Networks Planning. Telecoms Conference (ConfTELE), 2021, pp. 1-6.

[6] S. Wang, A Tabu Search Heuristic for Routing in WDM Networks. Master dissertation in Science. Canada, University of Windsor, 2004.

[7] Hertz, A., de Werra, D. Using Tabu Search Techniques for Graph Coloring. *Journal Computing*, vol. 39.4, pp: 345-351, 1987.

[8] M. Pióro and D. Medhi. Routing, Flow, and Capacity Design in Communication and Computer Networks. Morgan Kaufmann Publishers. Elsevier, 2004.

[9] R. Goścień, M. Klinkowski and K. Walkowiak. A Tabu Search Algorithm for Routing and Spectrum Allocation in Elastic Optical Networks. 16th International Conference on Transparent Optical Networks (ICTON), 2014, pp. 1-4.

[10] C. Dzongang, P. Galinier and S. Pierre. A Tabu Search Heuristic for the Routing and Wavelength Assignment Problem in Optical Networks. *IEEE Communications Letters*, vol. 9, no. 5, pp. 426-428, May 2005.

[11] M. Niksirat, S. Mehdi Hashemi, and M. Ghatee. Branch-and-price Algorithm for Fuzzy Integer Programming Problems with Block Angular Structure. *Fuzzy Sets Syst.*, vol. 296, no. 2, pp. 70–96, August 2016.

[12] T. L. LaQuey. NSFNET. In *The User's Directory of Computer Networks*, pages 247–250. Boston: Digital Press, 1990.

[13] E. Biernacka, J. Domzal, and R. Wójcik. Investigation of Dynamic Routing and Spectrum Allocation Methods in Elastic Optical Networks. *International Journal of Electronics and Telecommunications*, vol. 63, no. 1, pp. 85–92, February 2017.

[14] CONUS WDM network topology, http://monarchna.com/topology.html.

[15] I. Duarte. Exploring Graph Coloring Heuristics for Optical Networks Planning, Master dissertation in Telecommunications and Computer Engineering. Lisbon, ISCTE-IUL, October 2020.

[16] R. Ramaswami, K. Sivarajan, and G. Sasaki. Optical Networks: A Practical Perspective, 3rd edition. USA: Morgan Kaufmann, 2010.

[17] G. Ellina, E. Bouillet, R. Ramamurthy, J.-F. Labourdette, S. Chaudhuri, and K. Bala. Restoration in Layered Architectures with a WDM Mesh Optical Layer. *Annual Review of Communications for the International Engineering Consortium (IEC)*, vol. 55, 2002.

[18] S. Perrin. The Need for Next-Generation ROADM Networks. White Paper in Heavy Reading. vol. 1, pp. 1–15, September 2010.

[19] M. Filer and S. Tibuleac. N-degree ROADM Architecture comparison: Broadcast-and-select versus Route-and-select in 120 Gb/s DP-QPSK Transmission Systems. OFC 2014, pp. 1-3.

[20] M. C. Sinclair. Minimum Cost Topology Optimisation of the Cost 239 European Optical Network. In *Artificial Neural Nets and Genetic Algorithms*, pp. 26-29, Springer, Vienna, 1995.

[21] D. L. Mills and H. Braun. The NSFNET Backbone Network. In *Proceedings of the ACM Workshop on Frontiers in Computer Communications Technology*, SIGCOMM '87, pp. 191–196, New York, USA, 1987.

[22] T. Zami, B. Lavigne, and M. Bertolini. How 64 GBaud Optical Carriers Maximize the Capacity in Core Elastic WDM Networks with Fewer Transponders per Gb/s. *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A20–A32, January 2019.

[23] C. Fenger, E. Limal, and U. Gliese. Statistical Study of the Influence of Topology on Wavelength usage in WDM Networks. In *Optical Fiber Communication Conference. Technical Digest Postconference Edition.*, vol. 1, pages 171–173. Baltimore, Maryland, USA, March 2000.

[24] P. Chentsho, L. Cancela, and J. Pires. A Framework for Analyzing in-band Crosstalk Accumulation in ROADM-based Optical Networks. *Optical Fiber Technology*, vol 57, pp. 2-11, July 2020.

[25] I. Djordjevic and M. Cvijetic. Advanced Optical Communications: Systems and Networks. Norwood, MA: Artech House, 2013.

Appendices

APPENDIX A

Scientific Contributions

Exploring the Tabu Search algorithm as a graph coloring technique for wavelength assignment in optical networks

Inês Gomes[2], Luís Cancela[1,2] and João Rebola[1,2]

[1] *Optical Communications and Photonics Group, Instituto de Telecomunicações, Lisboa, Portugal*

[2] *Department of Science and Information Technology, Instituto Universitário de Lisboa (ISCTE-IUL), Lisboa, Portugal*

ifgsa@iscte-iul.pt, luis.cancela@iscte-iul.pt, joao.rebola@iscte-iul.pt

Keywords: Graph Coloring, Greedy, Optical Networks, Tabu Search, Wavelength Assignment

Abstract: The aim of this work is to study the Tabu Search algorithm as a graph coloring technique for wavelength assignment in optical networks, a crucial function in optical network planning. The performance of the Tabu Search is assessed in terms of the number of wavelengths and computation time and is compared with the one of the most common Greedy algorithm. It is concluded that for real networks with a large number of nodes and a higher variance node degree of the path graph relatively to its average node degree value, the Greedy algorithm is preferable to the Tabu Search algorithm since it returns the same number of colors of Tabu Search, but in a shorter computation time.

1 INTRODUCTION

Routing and Wavelength Assignment (RWA) are fundamental functions to transport data in an efficient way in optical networks (Winzer et al., 2018). Routing is responsible for finding the best path for a given traffic demand, and wavelength assignment (WA) is responsible for choosing an appropriate wavelength in that path to transport the given traffic demand taking into account the wavelength continuity and the distinct wavelength constraints (Simmons, 2014).

Several techniques have been used to solve the WA problem, ranging from exact algorithms to heuristics that typically give a sub-optimal solution to the problem, but in a shorter time, like the First-Fit or the Most Used algorithms (Simmons, 2014), (Zangy et al., 2000). Graph coloring techniques, although applied to a large range of applications, such as constructing schedules, can also be used for WA in optical networks (Simmons, 2014). The most used graph coloring algorithm for WA is the Greedy algorithm (Lewis, 2016). Some studies have, however, used other Graph coloring algorithms for WA, such as the DSATUR and RLF (Duarte et al., 2021), but for the majority of the networks studied the Greedy algorithm performs as well as these algorithms.

In this work we aim to study a more complex and more rigorous graph coloring technique for WA in optical networks, the Tabu Search algorithm. A performance study is made for several network topologies in terms of the number of colors and computation time. Moreover, a detailed comparison with the the Greedy algorithm is performed. The aim of these algorithms, considering a static network scenario, is to find the minimum number of wavelengths that satisfies all the traffic demands, in a feasible and reasonable computational time.

Note that the Tabu Search algorithm is a metaheuristic algorithm used for solving different kinds of problems, such as optimization problems in network design (Pióro and Medhi, 2004). For example, it has been used for solving RWA problems based on Integer Linear Programing (ILP) formalisms (Wang, 2004), (Goścień et al., 2014), (Dzongang et al., 2005). Moreover, in (Hertz and Werra, 1987) this algorithm has been used to solve graph coloring problems. But, to the best of our knowledge, there are no works that have used it as a graph coloring technique for WA in optical networks, as we do in this work.

This paper is organized as follows. In Section 2, the Greedy and Tabu Search graph coloring algorithms are explained and their pseudocodes and illustrative examples provided. In Section 3, the performance of the Tabu Search algorithm in random graphs and its comparison with the Greedy algorithm is studied. In Section 4, the RWA planning tool is briefly described and the performance of both algorithms as graph coloring techniques for WA in optical networks is assessed for several real networks. Finally, in Section 5, the conclusions are drawn.

2 GRAPH COLORING ALGORITHMS

In this section, the Greedy and Tabu Search algorithms, are explained through their respective pseudocode and an illustrative example is given.

2.1 Greedy Algorithm

The Greedy algorithm is probably the most used graph coloring algorithm (Lewis, 2016). It consists in coloring the vertices of a given graph one by one, with some ordering strategy, so that adjacent vertices have different colors (Lewis, 2016).

Figure 1 shows the pseudocode for Greedy algorithm (Lewis, 2016). Initially S that represents the set of colors that are going to be assigned along the Greedy algorithm process is empty and π represents a possible permutation of the graph vertices, e.g. descending, ascending or random order of the vertices. The *for* cycle in line (1) of the pseudocode goes through the set of vertices π and, for each vertex of π tries to find a color class S_j belonging to S to which it can be associated. This process involves checking the color class of the adjacent vertices. If the working vertex is an independent set then a color S_j can be assigned to this vertex. If this is not the case then a new color class must be assigned (lines 7 to 9).

```
GREEDY ($S \leftarrow 0, \pi$)
(1) for $i \leftarrow 1$ to $|\pi|$ do
(2)     for $j \leftarrow 1$ to $|S|$
(3)         if ($S_j \cup \{\pi_i\}$) is an independent set then
(4)             $S_j \leftarrow S_j \cup \{\pi_i\}$
(5)             break
(6)         else $j \leftarrow j+1$
(7)     if $j > |S|$ then
(8)         $S_j \leftarrow \{\pi_i\}$
(9)         $S \leftarrow S \cup S_j$
```

Figure 1: Pseudocode for Greedy algorithm.

Figure 2 represents an example of the operation of the Greedy algorithm, assuming a coloring strategy based on the descending order of degree. The vertex with the highest number of links connected, i.e., the highest vertex degree, is colored first, i.e. v_2 in step 1 is color with green (S_1={green}). In step 2 the algorithm continues the coloring with the following highest vertex degree, v_8, which is adjacent to v_2 so it is assigned to a different color, pink (S_2={pink}). In step 3, since there are four vertices with degree 3, one of them is randomly chosen. We have choose v_1 with the color class S_2 (step 4). This process continues until all vertices have been colored and in the end (step8) we can see that three colors are used.

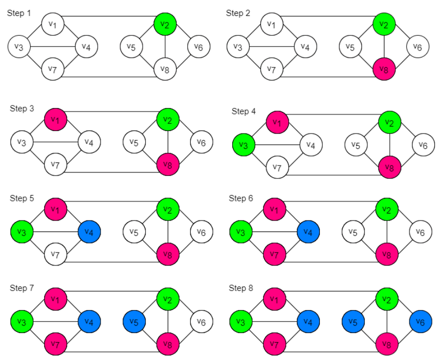


Figure 2: Greedy algorithm example.

2.2 Tabu Search Algorithm

Tabu Search is a metaheuristic algorithm, used to solve different kinds of problems, such as graph coloring (Lewis, 2016). The idea of this algorithm, when applied to graph coloring problems, is to answer the following question: given a graph *G(V,E)*, where V represent the set of vertices and E the set of edges between vertices, is it possible to feasibly color it with k colors? This algorithm has the following main steps:

1. It starts by defining an initial solution S to color the graph with a predefined value k of colors, which can be obtained randomly or by using a constructive heuristic, like Greedy or DSATUR (Lewis, 2016).
2. The algorithm next proceeds by computing the number of clashes (i.e. two adjacent vertices with the same color) which is represented by function *f(S)*, defined by:

$$f(S) = \sum_{\forall\{u,v\}\in E} g(u,v) \tag{1}$$

$$\text{with } g(u,v) = \begin{cases} 1 & \text{if} c(u)=c(v), \\ 0 & \text{otherwise.} \end{cases}$$

where $c(u)$ is the color of vertex u and $c(v)$ is the color of vertex v.

If $g(u,v) = 1$ it means that the color of the two adjacent vertices u and v is the same and therefore a clash occurs. The aim of the algorithm is to eliminate the clashes, i.e., $f(S) = 0$. If the number of clashes is not zero, a new solution $S´$ is obtained, by using the neighbor operator, which is defined as follows: if a vertex v is assigned to a color i, a neighbor operator corresponds to a color change of vertex v to a new color j. Note that to obtain this new solution $S\,'$ there are some vertices color changes that can not be done. These vertices color changes are registered in a list of forbidden vertices color changes, called the Tabu list **T**. This list is used to avoid previous undesired and already checked solutions (Hertz and Werra, 1987). If with this new solution $S\,'$ condition $f(S') < A(f(S))$ is verified, the best solution $S\,'$ is found. In this condition, A is an "aspiration level" function that gives the possibility that solutions $S\,'$ with a superior number of clashes be chosen, with the aim to escape from local minima (Hertz and Werra, 1987). If $f(S) = 0$ and number of operations (*Niter*) is less than *Nmax* it means that a solution with k colors is found. If the number of operations is *Nmax* the algorithm stops.

3. If a solution with k colors is found the algorithm starts again with k-1 colors.

Figure 3 shows the pseudocode of the Tabu Search algorithm, which follows the previous explanation. The pseudocode is initialize by defining an initial solution S and the Tabu list **T** size. While clashes occur, i.e., $f(S) > 0$, a new solution $S\,'$ is searched, the condition $f(S') < A(f(S))$ is checked and the Tabu list **T** is updated.

```
TABU SEARCH ($S = (V_1 \ldots V_k)$; $T$; Niter = 0)
(1) while $f(S) > 0$ and Niter $<$ Nmax
(2)     generate neighbours $S'$ with move $S \to S' \notin$ T
                                        or $f(S') < A.(f(S))$
(3)     (stop generation if $f(S') < f(s)$ )
(4)
(5)     $T \leftarrow T'$
(6)     $S \leftarrow S'$
(7)     $Niter \leftarrow Niter + 1$
(8) end
(9)  if $k$ colors guarantees $f(S) = 0$
(10)                $k \leftarrow k - 1$
(11)  end
```

Figure 3: Pseudocode for Tabu Search algorithm.

If a solution with k colors ensures that the number of clashes is zero, the algorithm tries a solution with $k-1$ colors (lines 9-10). The algorithm stops when no solution with $k-1$ colors is achieved or the number maximum of iterations is reached.

Figure 4 represents the same network of Figure 2, but now the coloring is going to be made with the Tabu Search algorithm. The algorithm starts with a random coloring solution (step 1). In this case, this random solution has 4 colors and 2 clashes, so $f(S') = 2$. Once the solution has clashes, the algorithm generates a new solution $S\,'$. In step 2, the neighbor operator works in the vertex 3 changing the color pink to green. This operation eliminate the clash that occurs between vertices 1 and 3, but a new clash appears between vertices 3 and 7, keeping the number of clashes equal to 2. The Tabu list **T** is updated with this move, i.e., vertex 3 changes from pink to green.

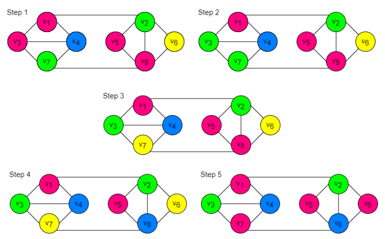


Figure 4: Tabu Search algorithm example.

In step 3, the clash between vertices 3 and 7 is eliminated and a better solution is found that reduces the number of clashes to one, between adjacent vertices 5 and 8. In this step, the neighbor operator works in vertex 7, coloring it with yellow, which is one of the four initial colors of the solution. The Tabu list **T** is updated again, with the addition of the move corresponding to vertex 7 colored in green. Thus, in step 3 there is a solution with 4 colors and one clash between vertices 5 and 8.

In step 4 occurs the coloring of vertex 8 with blue. Thus, all clashes are eliminated, i.e., $f(S) = 0$ and a complete proper k coloring is found with k=4. At this stage, the algorithm tries the solution k=3 colors. If vertices 6 and 7 are colored with pink (step 4) or vertices 1 and 5 are colored with yellow, this solution is feasible, with a total of k colors smaller than initially tested, making it a better solution. Next, the algorithm tries the solution with k=2 colors, but no solution with proper coloring is found, so, k=3 is the minimum number of colors.

3 PERFORMANCE OF THE TABU SEARCH ALGORITHM IN RANDOM GRAPHS

In this section, we study the performance of the Tabu Search algorithm in random graphs, and compare its performance with the one of Greedy algorithm with descending order. We have used the implementation of these algorithms available in (Lewis, 2016). But, first, we analyze the influence of the number of constraint checks (Lewis, 2016) on the accuracy of the Tabu Search algorithm, in order to minimize the respective computation time.

Random graphs, $G_{n,p}$, are graphs with n vertices characterized by the parameter p, which corresponds to the probability of two vertices being adjacent (Lewis, 2016). The parameter p can be given by

$$p = \frac{\sum_{i=1}^{n} \frac{Dg_i}{n-1}}{n} \tag{2}$$

where Dg_i is the degree of vertex i and n is the number of vertices.

Random graphs are built by generating random matrices with a $n \times n$ dimension according to the parameter p. Each matrix position represents a pair of vertices; if its value is one it means that the vertices are adjacent, if its value is zero, the vertices are non-adjacent. These values are randomly generated using a uniform distribution. In particular, when $p = 1$, it means that the degree of any vertex of the matrix is $n - 1$, i.e., all vertices of the matrix are adjacent to each other.

3.1 Influence of the number of constraint checks

In order to compute the time needed for the Tabu Search algorithm to provide a graph coloring solution, we first analyze the minimum number of constraint checks needed to achieve the best solution. Constraint checks are the operations within the Tabu Search algorithm that involve information requests about the graph, such as determining the degree of a vertex, or determining if two vertices are adjacent or not (Lewis, 2016).

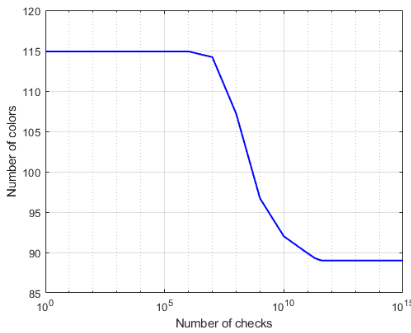


Figure 5: Number of colors as a function of the number of constraint checks considering the Tabu Search algorithm for $n = 1000$ and $p = 0.5$.

In Figure 5, the number of colors as a function of the number of constraint checks is represented for $n = 1000$ and $p = 0.5$, considering the generation of 10 random graphs for each number of constraint checks. As can be observed, the number of colors is minimized only when the number of checks is above 1×10^{11}. From that point on, the number of colors remains practically constant. Thus, for $n = 1000$, a number of constraint checks of 4×10^{11} is sufficient to ensure that the optimal solution is reached. It was confirmed that the parameter p does not influence the number of constraint checks required to achieve the minimum number of colors with the Tabu Search. This number only depends on the size of the graph.

Figure 6 shows the evolution of the number of colors for Tabu Search algorithm as the number of constraint checks increases for $n = 20$, 50, 100 and 200 and $p = 0.5$. The number of constraint checks in this study is between 1 and 1×10^{12}. From Figure 6, it can be concluded that for $n = 20$, only one constraint check is needed to obtain the best solution. Likewise for $n = 50$, $n = 100$ and $n = 200$, a number of constraint checks above, respectively, 1×10^7, 1×10^8 and 1×10^9 is needed to obtain the best solution. Below these values of constraint checks, the required number of colors increases with the graph size, as expected. In what concerns to the computational effort, as n decreases, the required number of constraint checks that minimizes the number of colors also decreases, and therefore the computation time for lower n will be lower as well.

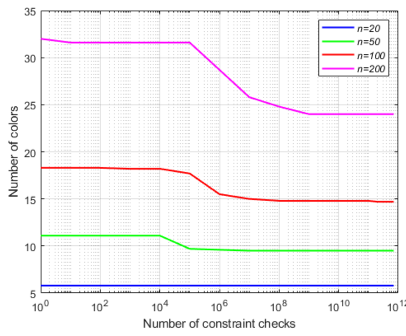


Figure 6: Number of colors as a function of the number of constraint checks imposed on Tabu Search for $p = 0.5$ considering different values of n.

The study of the number of constraint checks needed by the Tabu Search algorithm to minimize the number of colors for each graph was carried out to save computation time and is shown in Figure 7. The number of constraint checks, as observed in Figure 7 for the Tabu Search curve, depends remarkably on the number of vertices, n, in the graph. This study has been performed considering 25 simulations for a parameter $p = 0.5$. However, from extensive simulation results, it has been concluded that there is no dependence between the number of constraint checks required and the parameter p. Thus, regardless the value of p, from Figure 7, it is possible to obtain an approximation for the required number of constraint checks for any number of vertices n that minimizes the number of colors in a random graph. To obtain such approximation in Figure 7, a curve fitting has been performed relatively to the Tabu Search curve, including linear, quadratic, cubic and fourth degree fittings. Note that for proper fitting, the logarithm base 10 of the number of checks has been used.

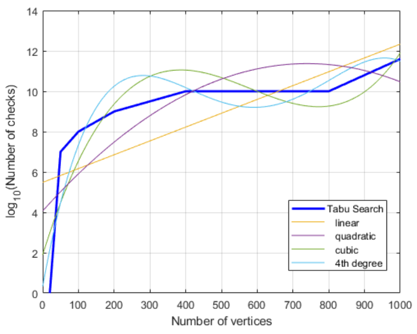


Figure 7: Number of constraint checks as function of the number of vertices and some fitting curves.

As can be observed in Figure 7, the linear and quadratic fittings, respectively, below $n = 500$ and $n = 300$, predict a number of checks much smaller than the one given by the Tabu Search. The cubic and fourth degree fittings predict a very similar number of constraint checks, but we prefer to use the cubic fitting, because it represents a simpler function. The cubic fitting is given by:

$$6.4 \times 10^{-8} n^3 - 1.1 \times 10^{-4} n^2 + 5.7 \times 10^{-2} n + 1.9 \quad (3)$$

3.2 Comparison with the Greedy algorithm

After studying the appropriate number of constraint checks to use in the Tabu Search algorithm, the performance of the Greedy and Tabu Search algorithms in finding the optimal color solution is also studied and compared considering random graphs.

Figure 8 a) shows the number of colors as a function of p for $n = 1000$ calculated using the Greedy (with random and descending order) and the Tabu Search algorithms. A very good agreement between the Tabu Search results and the ones presented in Figure 4.6 of (Lewis, 2016) is found. It can be observed from Figure 8 a) that for $p = 0$ and $p = 1$, there are no difference between the algorithms in the number of colors, as these are the cases where no adjacency and full adjacency between vertices occur, respectively. That is, if the vertices are all non-adjacent, the same color, independently of the algorithm, can be applied to all vertices. Similarly, if all vertices are adjacent between each other, then each vertex is assigned its own color. Also from Figure 8 a), we can observe that the Tabu Search needs fewer colors than the Greedy algorithm, and that the difference in the number of colors increases with p, but the relative percentage increase is always around 40%. For example, for $p = 0.1$, the Tabu Search gives 21 colors and the Greedy 30 colors, which is an increase of 9 colors (43 % increase); for $p = 0.5$, the Tabu Search provides 89 colors and the Greedy gives 125 colors, which is an increase of 36 colors (40 % increase) and for $p = 0.9$, the Tabu Search gives 229 colors and the Greedy gives 313 colors, which is an increase of 84 colors (37 % increase). Although in Figure 8 a) the results obtained with Greedy random and Greedy descending orders seem to be very similar, in the inset of Figure 8 a), it can be seen that the descending order gives always slightly better results (1 color difference) (Duarte, 2020).

Figures 8 b) and c) shows again the performance of both Greedy and Tabu Search algorithms for random graphs, but for 100 and 20 vertices, respectively, as a function of p. As observed in Figures 8 b) and c), it can be concluded that the Tabu Search performs once again better than the Greedy algorithm by predicting less colors and this behavior is more pronounced for increasing values of n and p.

To better understand the conclusions presented in Figure 8, Figure 9 shows the increase in percentage of the number of colors as a function of p predicted by the Greedy algorithm in comparison with the Tabu Search, considering $n = 20, 50, 100, 200, 1000$. Note that 25 simulations were performed to obtain the results presented.

From Figure 9, it can be concluded that the greater the number of vertices in the graph, the greater the percentage growth. The maximum value found is around 40%. For $p = 0.1$ and $n = 200$, there is a maximum 40% increase in the number of colors used by the Greedy algorithm in comparison with the number

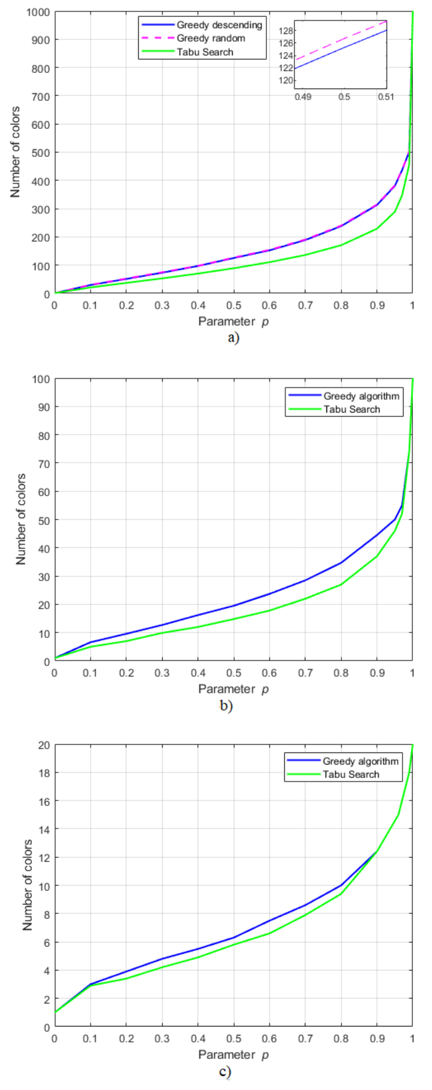


Figure 8: Number of colors as a function of p calculated using the Greedy and Tabu Search algorithms for random graphs for a) $n = 1000$, b) $n = 100$ and c) $n = 20$.

of colors predicted by the Tabu Search, whereas, for $n = 100$, there is only 32% increase, for $n = 50$, there is 10% increase and, for $n = 20$, there is only 3% increase. From these results, it can be concluded that

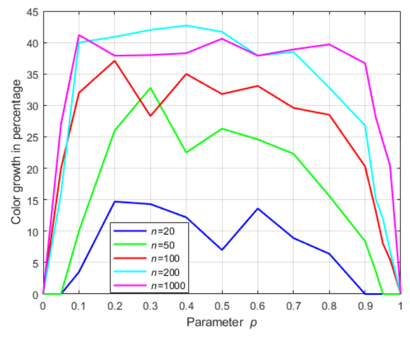


Figure 9: Percentage of the number of colors increase between Tabu Search and Greedy algorithms for $n = 20, n = 50, n = 100, n = 200$ and $n = 1000$ as a function of p.

the decrease in the total number of colors attributed by the Tabu Search in comparison with the Greedy is more pronounced in graphs with more vertices.

4 TABU SEARCH ALGORITHM AS A GRAPH COLORING WA TECHNIQUE

In this section, we assess the performance of the Tabu Search algorithm as a graph coloring WA technique in several real networks. A comparison with the Greedy algorithm is also performed. But, first, we briefly outline the RWA planning tool used, as well as, the network physical and logical topologies studied.

4.1 RWA planning tool

The planning tool used to solve the RWA problem was developed in (Duarte, 2020) and extended in this work in order to study the performance of the Tabu Search algorithm as a graph coloring WA technique. The planning tool has the following three main functionalities:

1. Definition of the physical and logical topologies characterized, respectively, by the adjacency and the traffic matrices.
2. Routing algorithm based on the Yen´s k-shortest path algorithm.
3. WA algorithms based on graph coloring techniques: Greedy and Tabu Search algorithms. Note that before using the Graph Coloring algorithms, the path graph $G(W,P)$ must be computed. This

graph is obtained from the graph $G(V,E)$ that represents the physical topology. The vertices of $G(W,P)$ represent the optical paths and P represents the set of links between those vertices (Duarte, 2020). These links are between one or more vertices (i.e. paths) that share one or more physical links. After obtaining the graph $G(W,P)$, the vertices can be colored considering the Greedy and the Tabu Search algorithm explained in Section 2. The number of colors obtained corresponds to number of wavelengths needed for solving the RWA problem.

4.2 Parameters of the network physical, logical and path topologies

The network physical topologies used in this work are the COST239 (Niksirat et al., 2016), NSFNET (LaQuey, 1990), UBN (Biernacka et al., 2017) and CONUS with 30 nodes (Monarch Network Architects, 1999), which we denominate in this work as real networks.

A network physical topology can be characterized by several parameters such as the average node degree and the variance node degree, respectively, given by (Fenger et al., 2000):

$$\overline{d} = \frac{\sum_{i=1}^{n} Dg_i}{n} \tag{4}$$

$$\sigma_d^2 = \frac{\sum_{i=1}^{n} (Dg_i - \overline{d})^2}{n-1} \tag{5}$$

Table 1: Real networks physical topology parameters.

Network	Nodes	Links	Average Node Degree	Variance Node Degree
COST239	11	26	4.7	0.4
NSFNET	14	21	3.0	0.3
UBN	24	43	3.6	0.9
CONUS	30	36	2.4	0.4

Table 1 shows the information regarding the network physical topologies with respect to the number of nodes and links, average and variance of the node degree. The variance node degree helps to understand how regular the network is from the point of view of the number of links at each node in the network (Fenger et al., 2000). Higher variances correspond to more irregular networks. When the variance node degree is zero, it means that all nodes have the same number of links (Fenger et al., 2000).

The parameters used to characterize the physical topology, i.e. the average and variance node degree, can also be used to characterize the logical topology. Table 2 presents these parameters considering that a full mesh logical topology is applied over the physical topologies. In this scenario, the average node degree is $N-1$. Furthermore, the variance node degree is zero, because all nodes have the same number of links. In table 2 the number of bidirectional paths for a full mesh logical topology is also shown and is given by $\frac{n(n-1)}{2}$.

Table 2: Real networks logical topology parameters.

Network	Number of Paths	Average node degree	Variance node degree
COST239	55	10	0
NSFNET	91	13	0
UBN	276	23	0
CONUS	435	29	0

Also, the average and variance node degree parameters can be evaluated in the context of the path graph $G(W,P)$, as shown in Table 3. The parameter p is also presented in Table 3. A high average value means that on average, one or more links of every path are being used by several different paths. From Table 3 it can be observed that the variance node degree is at least one order of magnitude higher than the average node degree. This means that there are some links belonging to a path (i.e. vertex) that are being used by many other different paths, and also that there are some links belonging to a path that are not being used, or are slightly used by other paths. So, networks with high path variances need a high number of colors, as we will discuss in subsection 4.3.

Table 3: Real networks path topology parameters.

Network	Parameter p	Average Node Degree	Variance Node Degree
COST239	0.101	5.6	14
NSFNET	0.25	22.5	121
UBN	0.2250	61.9	1.2840×10^3
CONUS	0.379	164.6	4.3696×10^3

4.3 Performance analysis

In this subsection, the performance of Greedy and Tabu Search algorithms are assessed and compared when applied to the real networks described in subsection 4.2.

Table 4 presents the number of colors and the corresponding simulation times for the networks COST239, NSFNET, UBN and CONUS with 30 nodes, considering a full mesh logical topology, obtained with the Greedy and Tabu Search algorithms.

From the results presented in Table 4, regarding

Table 4: Number of colors and simulation time obtained by Greedy and Tabu Search for some networks.

Network	Number of colors	Time (sec)	
		Greedy	Tabu
COST239	8	0.002	0.008
NSFNET	24	0.009	0.015
UBN	64	0.014	0096
CONUS	123	0.040	0.373

the simulation time, it is observed that the Greedy algorithm is 4 times faster than the Tabu Search for the COST239 network and almost 10 times faster than the Tabu Search for the CONUS network. Thus, the Greedy algorithm leads to a faster simulation time than the one obtained with the Tabu Search.

Also, as observed in Table 4, the number of colors obtained with the Greedy and Tabu Search algorithms for the networks considered is the same. However, in section 3, considering random graphs, we have seen that the Tabu Search predicts a lower number of colors than the Greedy algorithm. In order to understand this apparently contradicting behavior, in the following, we are going to study the influence of the traffic pattern in the number of colors obtained by the two algorithms. Therefore, we are going to change the traffic matrix in order to obtain various logical topologies different from the full mesh topology considered for the networks presented in Table 4. First, we define a metric called the percentage of network traffic, denoted as N_T. This metric ranges from 0 (no network traffic) to 100% (full mesh topology) and aims to quantify the change of network traffic in a traffic matrix for different networks.

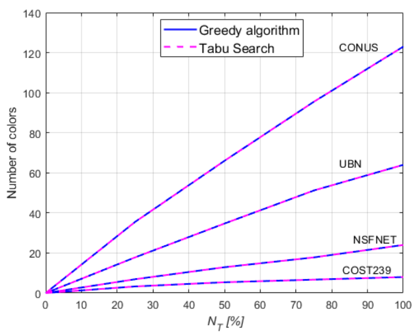


Figure 10: Number of colors provided by the Greedy and Tabu Search algorithms for some networks as function of N_T.

Figure 10 shows the number of colors as a function of the percentage of network traffic N_T, for the networks considered in Table 4, using the Greedy and Tabu Search algorithms. From Figure 10, it can be observed that, when $N_T = 0$, as there is no traffic in the network, no colors are assigned. When $N_T > 0$, the two algorithms give exactly the same number of colors. Thus, it can be concluded that the change of the traffic matrix (i.e. the logical topology) does not produce any differences on the number of colors predicted by both algorithms. The reason for this behavior is going to be detailed next, but relies on the fact that the variance node degree of the path matrix is considerably greater than the corresponding average value.

Next, we are going to investigate the impact of the average and variance node degrees of the path graph $G(W,P)$ in the performance of the Greedy and Tabu Search algorithms, in order to try to explain why the two algorithms give the same number of colors when real networks are considered, independently of the network traffic, and different number of colors with random graphs.

Table 5: Average and variance node degree and number of colors given by the Greedy with descending order and Tabu Search algorithms for real networks and for random path graphs with the same characteristics, n and p, of the real networks.

Network	Average node degree	Variance node degree	Greedy descend	Tabu Search
Real networks				
COST239	5.6	14	8	8
NSFNET	22.5	121	24	24
UBN	61.9	1.28×10^3	64	64
CONUS	164.6	4.37×10^3	123	123
Random path graphs with uniform distribution				
$n = 55$ $p = 0.1$	5.88	5.88	4.6	5
$n = 91$ $p = 0.25$	22.5	14	11	8
$n = 276$ $p = 0.225$	61.9	51.9	21	15
$n = 435$ $p = 0.392$	170.2	164.5	47	34

In Table 5, the average and the variance node degrees of the path graph, $G(W,P)$ of the COST239 ($n = 55$ and $p = 0.101$), NSFNET ($n = 91$ and $p = 0.25$), UBN ($n = 276$ and $p = 0.225$) and CONUS with 30 nodes ($n = 435$ and $p = 0.392$) networks and the respective number of colors predicted by the Greedy and Tabu Search algorithms are presented for a full mesh logical topology, i.e. $N_T = 100\%$. The same network parameters are also presented for random graphs (where the corresponding path matrix is generated with a uniform distribution), with the same number of paths and parameter p of the referred networks.

As can be observed in Table 5, the average node degree of the path graph in real networks and random graphs is very similar as it depends on the number of paths and on the parameter p that is the same for the real and random networks. However, it can be noticed that in random graphs, the variance node degree has a similar magnitude relatively to the average node degree, while, in real networks, the variance node degree is at least one order of magnitude higher than the average node degree. The lower variance found in random graphs is due to the uniform distribution of 1´s in the path matrix, that defines the graph path $G(W,P)$ (e.g. for $p = 0.1$ it means that each line of the matrix has 10% of ones and 90% of zeros on average), while the higher variances found in real networks are due to the non-uniform distribution of 1´s in the path matrix. Furthermore, in random graphs, for lower variance node degrees, the difference in colors between Greedy and Tabu Search is notorious, whereas in real networks both algorithms produce the same number of colors. For example, for $n = 435$ and $p = 0.392$, in Table 5, the variance node degree for the CONUS with 30 nodes network is 4370, whereas for the respective random graph, the variance has a much lower value, 164.5. So, this finding can justify the fact that the number of colors computed by the Greedy and Tabu Search algorithms when random graphs are used is different, while the same number of colors is obtained when real networks are considered.

To further confirm this finding we are going to study the evolution of the number of colors as a function of the variance node degree of the path graph, $G(W,P)$, for both Greedy and Tabu Search algorithms.

Figure 11 shows the number of colors obtained with the Greedy and Tabu Search algorithms as a function of the variance node degree for random matrices using uniform and non-uniform distributions with the same number of paths n and parameter p of the real networks: a) UBN and b) CONUS with 30 nodes. In Figure 11 c), the case of the random graph with $n = 100$ and $p = 0.5$ is also studied considering uniform and non-uniform distributions. In Figures 11 a) and b), we also represent the number of colors corresponding to the real cases of UBN and CONUS with 30 nodes network, respectively.

As can be seen in Figure 11, for higher variance node degrees, the number of colors given by Greedy and Tabu Search algorithms tends to converge, whereas for lower variances the number of colors produced by these algorithms is different, with Tabu Search algorithm presenting a lower number of colors. This behavior is observed for all the three networks studied. For example, in Figure 11 c), for a low variance value of around 27, the Greedy algorithm produces, respectively, 22 and 20 colors, with a non-uniform and a uniform distribution, whereas the

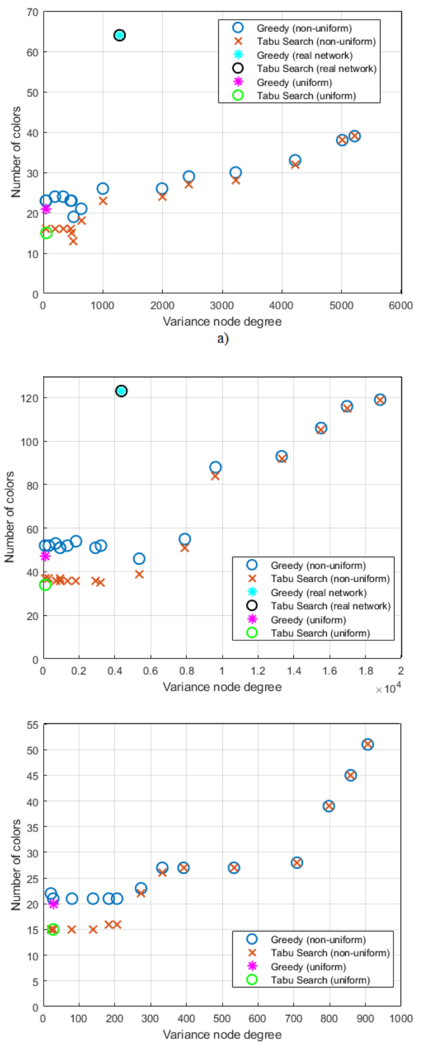


Figure 11: Number of colors obtained with the Greedy and Tabu Search algorithms for random graphs as a function of the variance node degree for a) $n = 276$ and $p = 0.225$, b) $n = 435$ and $p = 0.392$ and c) $n = 100$ and $p = 0.5$.

Tabu Search produces 15 colors for both distributions. Likewise, in Figure 11 c), for a high variance value of 900, both algorithms produce around 52 colors with a non-uniform distribution. In Figures 11 a) and b), the number of colors obtained in the UBN and CONUS with 30 nodes networks is also represented for both algorithms. It can be observed that in these scenarios, there are no difference between the number of colors given by the Tabu Search and the Greedy algorithms, which can be explained by the higher variances values relatively to its average values and also due to the higher number of colors used. In the limit, when the maximum number of colors is used both algorithms must return the same number of colors.

5 CONCLUSIONS

In this work, the performance of a metaheuristic algorithm, the Tabu Search algorithm, has been studied as a graph coloring technique for WA in optical networks, as well as in randomly generated path graphs and his performance has been compared with the one of the Greedy algorithm.

The Tabu Search algorithm, when the path graph is obtained randomly (with a uniform distribution) has been shown to have a superior performance to the Greedy algorithm. In particular, when $n = 1000$ and $p = 0.5$, the Tabu Search algorithm returns only 89 colors, whereas the Greedy algorithm gives 124 colors, which represents a decrease of 35 colors. However, when real networks are considered, both Greedy and Tabu Search algorithms give the same number of colors. We have found that as the variance node degree of the path graph, $G(W,P)$, increases the Greedy and Tabu Search algorithms tend to return the same number of colors, whereas when the variance gets lower this number of colors becomes different. So, we can conclude that in real network scenarios the simplest and faster Greedy algorithm sorted with descending order should be used, instead of the more complex and slower Tabu Search algorithm, since real networks have typically high variance node degree values which causes the Tabu Search and Greedy algorithms to have a similar performance.

ACKNOWLEDGEMENTS

This work was supported under the project of Instituto de Telecomunicacões UIDB/50008/2020.

REFERENCES

Biernacka, E., Domzal, J., and Wójcik, R. (2017). Investigation of dynamic routing and spectrum allocation methods in elastic optical networks. *International Journal of Electronics and Telecommunications*, 63:85–92.

Duarte, I. (2020). Exploring graph coloring heuristics for optical networks planning. Master's thesis, ISCTE - Instituto Universitário de Lisboa.

Duarte, I., Cancela, L., and Rebola, J. (2021). Graph coloring heuristics for optical networks planning. In *Telecoms Conference (ConfTELE)*, pages 1–6.

Dzongang, C., Galinier, P., and Pierre, S. (2005). A tabu search heuristic for the routing and wavelength assignment problem in optical networks. *IEEE Communications Letters*, 9(5):426–428.

Fenger, C., Limal, E., and Gliese, U. (2000). Statistical study of the influence of topology on wavelength usage in WDM networks. In *Optical Fiber Communication Conference*, pages 171–173. Optical Society of America.

Goścień, R., Klinkowski, M., and Walkowiak, K. (2014). A tabu search algorithm for routing and spectrum allocation in elastic optical networks. In *16th International Conference on Transparent Optical Networks (ICTON 2014)*.

Hertz, A. and Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing*, 39:345–351.

LaQuey, T. (1990). NSFNET. In *The User's Directory of Computer Networks*, pages 247–250, Boston. Digital Press.

Lewis, R. (2016). *A Guide to Graph Coloring: Algorithm and Applications*. Springer, Switzerland.

Monarch Network Architects (1999). CONUS WDM network topology. http://monarchna.com/topology.html.

Niksirat, M., Hashemi, S. M., and Ghatee, M. (2016). Branch-and-price algorithm for fuzzy integer programming problems with block angular structure. *Fuzzy Sets and Systems*, 296:70–96.

Pióro, M. and Medhi, D. (2004). *Routing, Flow, and Capacity Design in Communication and Computer Networks*. MORGAN KAUFMANN PUBLISHERS.

Simmons, J. (2014). *Optical Network Design and Planning*. Springer, New York, USA, 2nd edition.

Wang, S. (2004). A tabu search heuristic for routing in WDM networks. Master's thesis, University of Windsor.

Winzer, P. et al. (2018). Fiber-optic transmission and networking: the previous 20 and the next 20 years. *Optics Express*, 26(18):24190–24239.

Zangy, H., Juez, J., and Mukherjeey, B. (2000). A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks. *Optical Networks Magazine*, 1:47–60.