Pontifícia Universidade Católica
do Rio de Janeiro

Mariana Alves Londe

The RSI Allocation Problem: exact and heuristic methods

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em Engenharia de Produção of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia de Produção.

Advisor : Prof. Luciana de Souza Pessôa
Co-advisor: Dr. Carlos Eduardo de Andrade

Rio de Janeiro
April 2021

Pontifícia Universidade Católica
do Rio de Janeiro

Mariana Alves Londe

The RSI Allocation Problem: exact and heuristic methods

Dissertation presented to the Programa de Pós-graduação em Engenharia de Produção of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Engenharia de Produção. Approved by the Examination Committee.

Prof. Luciana de Souza Pessôa
Advisor
Departamento de Engenharia Industrial – PUC-Rio

Dr. Carlos Eduardo de Andrade
Co-advisor
AT&T Labs Research – AT&T

Prof. Luciana Salete Buriol
Universidade Federal do Rio Grande do Sul – UFRGS

Dr. Mauricio Guilherme de Carvalho Resende
Mathematical Optimization and Planning – Amazon.com/MOP

Rio de Janeiro, April 13th, 2021

Mariana Alves Londe

The author graduated *cum laude* in Industrial Engineering at UFRJ (Universidade Federal do Rio de Janeiro) in May 2019. During graduation, the author worked as an intern at the engineering school of UFRJ, focusing on process mapping and optimization.

Bibliographic data

Londe, Mariana Alves

The RSI Allocation Problem: exact and heuristic methods / Mariana Alves Londe; advisor: Luciana de Souza Pessôa; co-advisor: Carlos Eduardo de Andrade. – Rio de janeiro: PUC-Rio, Departamento de Engenharia Industrial, 2021.

v., 126 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Engenharia Industrial, 2021.

Inclui bibliografia

1. Engenharia de Produção – Teses. 2. Root Sequence Index;. 3. Métodos Matemáticos;. 4. Algoritmo Genético com Chaves Aleatórias Viciadas. I. Pessôa, Luciana de Souza. II. Andrade, Carlos Eduardo. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Engenharia Industrial. IV. Título.

CDD: 658.5

Acknowledgments

This work would not have been possible if not for the support of several people. I dedicate this section to all of those who helped and supported me in the last few years, with some in particular deserving special mention.

First, I would like to thank my family, in special my parents Ricardo and Elizabeth, who have always supported and encouraged my interests.

Secondly, I want to thank my advisors, Professor Luciana Pessôa and Carlos Eduardo de Andrade. Without their guidance, lessons, motivation, and ideas, this work would have not been possible.

I want to thank both CNPq and PUC-Rio, for the aids granted, without which this work does not could have been accomplished. I also want to express my gratitude to all professors and staff of the DEI, for all knowledge and support given.

As for professors outside of PUC-Rio, I want to thank professor Pedro Hokama of UNIFEI. His interesting discussions about mathematical models helped enrich this dissertation. Another professor that deserves a mention is professor Andre Salles of UFRJ, for encouraging me to continue my studies towards a master's, and, specifically, for suggesting PUC-Rio for these studies.

Finally, I want to thank all my friends for their support. For my post-graduate colleagues at PUC-Rio, in particular Sofia, Thiago, Leticia, Mara, and Gabriel. And for my oldest friends, Pedro, Anna Leticia, Fernanda, Gabriela, Marcos, Julia, Nathalia, and many others, who provided emotional support during my studies and in these uncertain times. A smaller thank you goes to my cats Selene, Zuri, and Soleil, who did their best in keeping me company during long studying and coding sessions.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Londe, Mariana Alves; Pessôa, Luciana de Souza (Advisor); Andrade, Carlos Eduardo (Co-Advisor). **The RSI Allocation Problem: exact and heuristic methods**. Rio de Janeiro, 2021. 126p. Dissertação de Mestrado – Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro.

Since its introduction, mobile wireless communication has grown and changed substantially. This massive growth leads to different levels of complexity, mainly concerned with the assignment of different parameters to radio or base stations. One parameter is the Root Sequence Index (RSI), related to the Physical Random Access Channel (PRACH) preambles, used to allocate uplink channels between the user equipment and the base station. The assignment of RSIs close-in-range to neighbor antennas may cause collisions, which are responsible for failures on service establishment, and therefore, performance degradation. Such allocation problems can be modeled as Graph Coloring Problems, including several additional constraints. However, few studies focus on RSI allocation and collisions from the optimization perspective. The objective of this study is to develop methods for allocating the RSI, trying to lessen the risk of collision, and obeying other constraints. In this study, both exact and heuristics methods are explored and compared. For this, several mathematical models were made, alongside a biased random key genetic algorithm. The results show that the utilization of an allocation strategy based on neighbor relations is efficient for finding good solutions.

Keywords

Root Sequence Index; Mathematical Models; Biased Random Key Genetic Algorithm.

Resumo

Londe, Mariana Alves; Pessôa, Luciana de Souza; Andrade, Carlos Eduardo. **O Problema de Alocação do RSI: métodos exatos e heurísticos**. Rio de Janeiro, 2021. 126p. Dissertação de Mestrado – Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro.

Desde sua introdução, a comunicação móvel sem fio cresceu e se modificou severamente. Seu crescimento acentuado significa que a alocação de diferentes parâmetros para rádios ou estações-base ganhou diversos graus de complexidade. Um parâmetro é o *Root Sequence Index* (RSI), relacionado com os preâmbulos do *Random Access Channel* (PRACH), usado para alocar canais de upload entre o equipamento do usuário e a estação rádio-base. A alocação de RSIs próximos a radios ou antenas vizinhas pode causar colisões, que são responsáveis por falhas no estabelecimento do serviço de comunicação e, portanto, degradação no desempenho da rede. Em geral, tais problemas de alocação são modelados como um Problema de Coloração de Grafos, incluindo diversas restrições. Contudo, não há estudos que foquem na alocação de RSI e colisões. O objetivo deste estudo é explorar e comparar modelos exatos e heurísticos para esse problema. Para isso, diversos modelos matemáticos foram elaborados, além de um algoritmo genético de chaves aleatórias viciadas. Os resultados apontam que a utilização de uma estratégia baseada nas relações de vizinhança é eficaz para a obtenção de boas soluções.

Palavras-chave

Root Sequence Index; Métodos Matemáticos; Algoritmo Genético com Chaves Aleatórias Viciadas

Table of contents

List of figures

List of tables

List of algorithms

But I have promises to keep;
And miles to go before I sleep.

Robert Frost, *Stopping by Woods on a Snowy Evening.*

1
Introduction

Since its introduction in the late 1970s, mobile wireless communication has grown and changed substantially. The many alterations in technological capacity, alongside new devices such as smartphones and tablets, cause a massive growth of internet use [1]. In 2018, 22 billion devices were connected to networks, with a forecast of 40 billion by 2025 [2].

This growth in connection density, with many devices and requirements, is something that the fifth generation of wireless communication systems, known as 5G, must address [3]. 5G must also consider the advent of the Internet of Things (IoT) [4], in which device-to-device communications (D2D) [5] are a crucial feature. With it, it is expected that as much as 240–500 billion mobile devices or physical entities will be connected to the internet in the future [6].

In this context of massive networks, not only the implementation of these networks [7], but also the allocation of parameters to radios or base-stations, gain several levels of complexity.

One such parameter is the Root Sequence Index (RSI). Related to the random access procedure for the establishment of upload channels between the user and the station, this parameter is used to calculate variables of the Physical Random Access Channel (PRACH). However, if two or more neighbor radios have the same RSI number, then a collision can occur. RSI collisions can cause an increase in connection failures [8].

Figure 1.1 illustrates the effect of incorrect parameter allocation in network traffic. It shows the real amount of msg3 of three different cells or antennas of the same radio station, with each cell in a coverage angle of 120°. On the last two days indicated in the figure, the values of both Physical Cell Identification (PCI) and RSI parameters of the cells were incorrectly allocated. One can note the effect of this on the amount of msg3 messages, a value indicative of network traffic, which is drastically reduced.

Parameter allocation problems in telecommunications are, generally, modeled as graph coloring problems (GCP) [9]. Hale [10] introduces the frequency assignment problem (FAP), studied intensively by authors such as Siddiqi and Sait [11], Marsa-Maestre et al. [12], Zhao et al. [13], Acedo-Hernández et al. [14]. However, there are no studies, as far as the author is aware, that focus on allocation of the RSI and its possible collisions.

In this context, this study focuses on solving the RSI allocation problem. For this problem, both exact and heuristic methods are proposed and compared

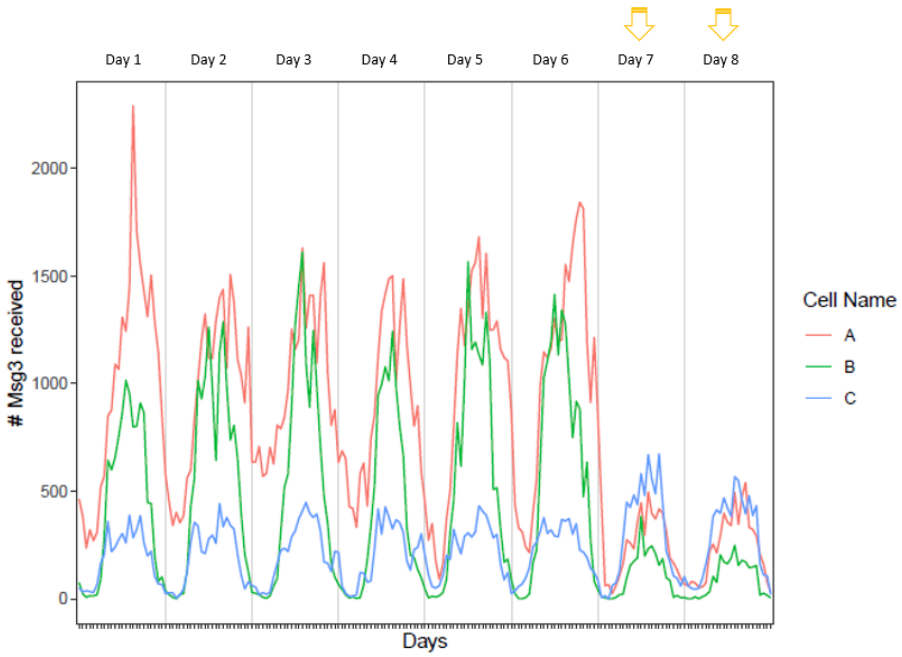


Figure 1.1: Effect of incorrect parameter allocation. Each cell indicates an antenna or cell of the same radio base station. The vertical lines separate different days. Note the last two days, in which the parameters are incorrectly allocated, and the corresponding effect on network traffic.

in their effectiveness. The contribution of this work can be defined as follows:

- Introduction of a novel problem in telecommunications, which will have great significance with the advent of 5G;
- Application of exact and heuristic methods to solve the aforementioned problem, advancing the literature in graph coloring problems and its generalizations.

To attain the aforementioned objectives, we study classical ways of solving similar situations. Graph Coloring Problems are not, usually, tackled with exact formulations [15]. Meanwhile, metaheuristic methods have shown to be effective in solving Graph Coloring Problems [16].

The remainder of this document is structured as follows. In Chapter 2, the problem is defined, alongside the proposed notation and mathematical models. After this, Chapter 3 presents related, pertinent literature. Afterward, Chapter 4 details the proposed heuristic. In Chapter 5 the results of the experiments are listed, alongside a comparison between methods. Finally, Chapter 6 shows the concluding remarks.

2
The Root Sequence Index Problem

This chapter defines the Root Sequence Index (RSI) allocation problem. It starts with detailing the graph notation and concepts, followed by the formal definition of the problem and, finally, the proposed mathematical models.

2.1
Graph Concepts and Notation

Let $G(V, E)$ be a graph, in which V identifies the set of vertices and E, the edges. Vertices $i, j \in V$ are said to be adjacent or neighbors if the edge $(i, j) \in E$. The degree $d(i)$ of a vertex i is the number of edges attached to it. A subgraph H of G is a graph that has all its vertices and all its edges contained in G [17].

The coloring of a graph is formally defined as the assignment of colors to vertices in V that respects the constraints of the problem. The group of possible colors of a vertex can be indicated by $\Lambda(i)$ [18]. The chromatic number $\chi(G)$ is the minimum number for which a coloring exists for a graph G [17].

The discrete number of colors used is called the order of a graph, and the difference between the highest and the smallest color used, the span [10]. The edge span, meanwhile, is the minimum span taken over the edges of the graph [17].

An independent set is a subgroup of vertices so that none are neighbors. It is called maximal if it is not a subset of any other independent set [19]. A complete subgraph of G is called a clique, and a maximum clique is a subgraph that has all vertices connected to each other [20].

2.2
Problem Definition

First, some telecommunications concepts must be clarified. A radio can be defined as an equipment for wireless transmission and/or reception of electromagnetic waves, especially when used to transmit sounds [21]. Meanwhile, an antenna or aerial is a portion of a radio transmitter or station used for radiating waves or receive them from spaces, changing electric currents into radio waves, and vice versa [21]. Antennas are usually made of metal and can have a wide variety of configurations [22]. Finally, a base station is defined as a transmission and reception station in a fixed location, which serves as a bridge between all mobile users in a cell and connects mobile calls to mobile

switching centers. Base stations usually consist of one or more receive/transmit antennas, microwave dish, and electronic circuitry [23]. In this work, we use the words "antenna" and "base station" interchangeably.

The Root Sequence Index (RSI) is related to the random access procedure, being a parameter that can suffer interference when two neighbor antennas have close values. The Physical Random Access Channel (PRACH) procedure occurs when the User Equipment (UE) tries to connect to a network, establish or reestablish a service connection, and synchronize for downlink data transfers [8]. This procedure can be contention or non-contention based.

In contention-based access, the UE utilizes 40 preambles randomly selected to do PRACH procedure [24]. This is the case in which RSI collisions can happen, as the non-contention access does a different, more complex procedure that guarantees it will not occur [25]. The principal aim of PRACH is to synchronize UE with the eNodeB (4G radio station) or the gNodeB (5G radio station). Meanwhile, the RSI is used to calculate the 40 preambles of the contention-based access. If two neighbor antennas have the same RSI values, then the same PRACH values will be obtained, hence collisions may occur [25].

The RSI can have two different sets of values. They are dependent on the frequency used and the cell size. The Long Sequence has four preamble formats, used in macrocells with values between 0 and 839. The Short Sequence, meanwhile, has values between 0 and 139, indicated in nine preamble formats.

Small cells and macrocells use different frequency intervals and have different uses. Macrocells are widely used in the coverage of wide areas, with up to 10 kilometers with antenna heights of 25 meters [26], and use lower frequency ranges [27]. Meanwhile, small cells are used in hotspot or indoor coverage [26, 27] with significantly higher frequencies [1]. Interference between cells of different sizes may happen when they are in the same frequency ranges, however, the instances used in this work do not contemplate this possibility.

To prevent interference between close cells of the same sequence, a minimum distance (*minDist*) between the values of the RSI of neighbors is needed. This minimal distance is calculated with a value related to the division of frequency spectrum in the network, which is the constant 64 for 5G networks, the maximal value of the sequence called L_{RA} or *maxRSI* (the smallest possible value is called *minRSI*), and the cyclic shift value N_{CS} [25]. The cyclic shift originates from network characteristics, therefore the minimal required distance is the same for all antennas in the same network and sequence. The computation of the minimal distance is shown in Equation (2-1).

$$minDist = \frac{64}{L_{RA}/N_{CS}} \tag{2-1}$$

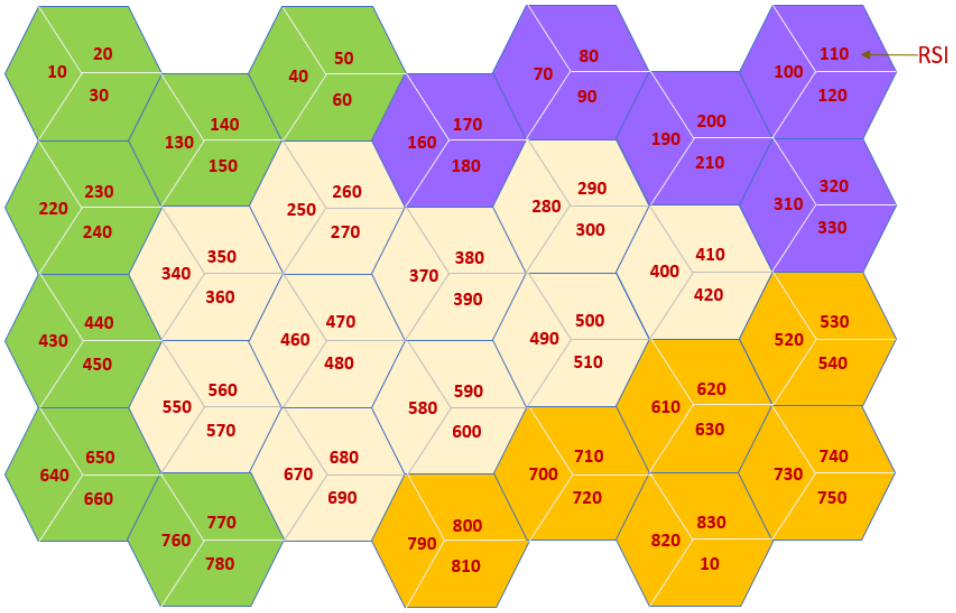


Figure 2.1: Representation of RSI allocation. Each hexagon represents a radio base station with three antennas in coverage angles of 120º each. The edges of each hexagon indicate adjacent antennas, which must have at least a difference of 10 units between their RSI. Hexagons of different colors indicate towers in different regions or markets.

Figure 2.1 represents how a network can be configured, in which the minimum distance between neighbor antennas is 10 units. Note that, although in this figure there are only adjacent antennas inside a hexagon, in real life, the signals overlap between antennas, hence the problem cannot be modeled as a simple planar graph. In fact, this is the most common case in 5G: generally, there is a macro cell for coverage and several small cells for network capacity inside the coverage radius of the macro cell, as can be seen in Figure 2.2. In this example, the small cells can interfere between themselves and between them and the macro cell.

The 5G network is inserted in the context of ultra-dense networks [1]. Ultra-dense networks are the result of capacity enhancements needed for 5G, and are synonym with antennas or radios stationed only few meters away from each other, and thus potentially interfering with each other [28]. This indicates that defining a maximal distance (*maxDist*) limit between the difference of the RSI of neighbors is adequate. Such a constraint allows setting a higher number of antennas in each network so that we would improve coverage and network access.

Figures 2.3a and 2.3b illustrate how these characteristics influence the RSI values for a cell X. In Figure 2.3a, cell N is the only neighbor of X, thus the

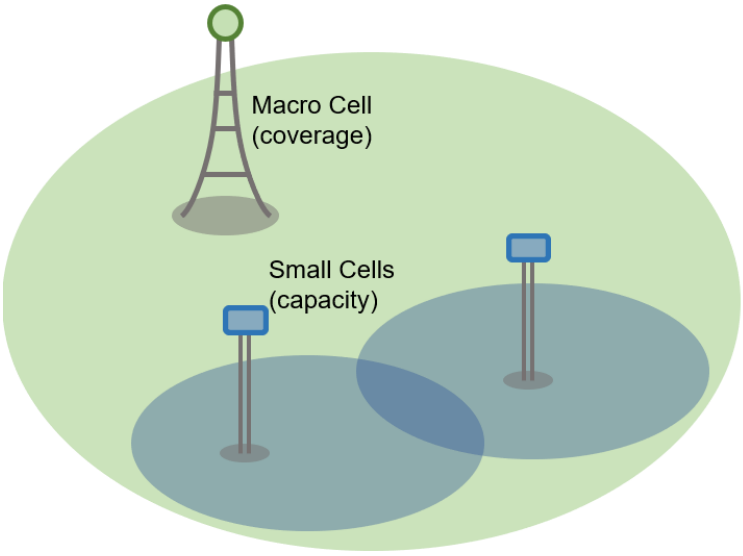
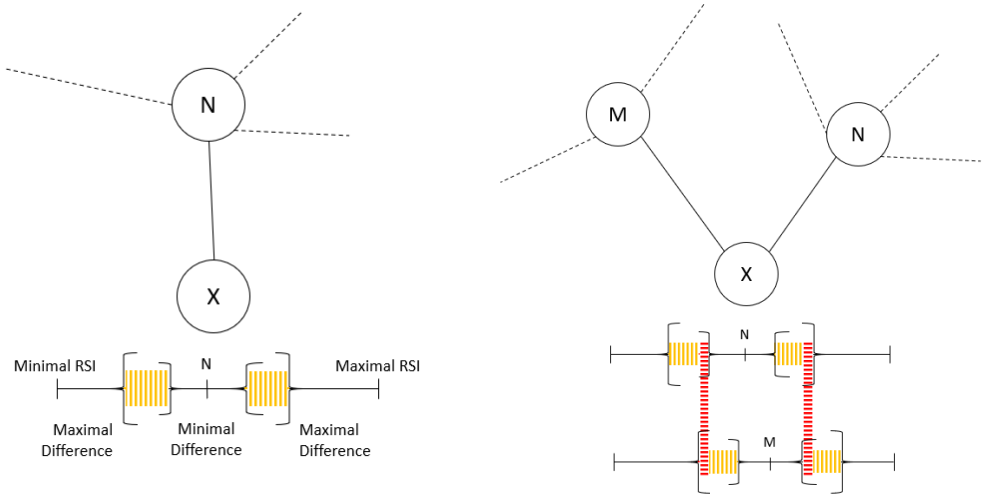


Figure 2.2: Typical implementation of macro and small cells in 4G and 5G. Note that the macrocells cover a big area (in green) that includes small cells of network capacity with a smaller action radius (in blue). Observe the possibility of other macro cells overlapping with either the macro or small cells shown in the figure.

RSI values for X must be inside the intervals indicated by the yellow hatch. These are the values that respect the minimal and maximal difference and are inside the possible values for the RSI. Meanwhile, Figure 2.3b presents a case with more than one adjacent cell. In this case, X's RSI must respect the differences for both neighbors, which decreases the possible RSI values and thereby increases the complexity of allocating the RSI.



(a) Illustration of the RSI for a vertex with only one edge. In the intervals indicated by the vertical yellow hatch, the possible values for the RSI of node X in relation to the RSI of node N.

(b) Illustration of the RSI for a vertex with more than one edge. In the intervals with horizontal red hatch, the possible values for the RSI of vertex X, in relation to the RSI of neighbor nodes N and M.

Figure 2.3: Two representations for allocation of the RSI of a cell X in relation to its neighbor vertices.

We propose two different objectives for the RSI allocation problem. The first objective involves the minimization of the number of changes between new and old network configurations. This aims to increase model realism, as this scenario occurs when new vertices are introduced in the system and the RSI of other antennas should be modified as least as possible, to avoid disruptions and degradation in the service quality. The second objective tries to maximize the smallest difference between the RSI of neighbors. This scenario considers the future need to add new vertices, and thus the necessity of having possible RSI values available for them, which is a consideration needed for the ultra-density expected in these networks, and the expected increase in coverage both in urban and rural areas.

2.3 Proposed Mathematical Models

The RSI Allocation Problem can be modeled with an undirected graph $G = (V, E)$, in which V indicates the set of vertices (antennas) and E, the set of existing edges. There is an edge if the antennas are neighbors and, therefore, an incorrect allocation can cause interference between their RSI. Consider $\Lambda \in \mathbb{N}$ as the possible RSI values. It is considered that the graph has a starting coloration, which indicates a prior configuration, that can be illegal

or infeasible with respect to the constraints, or needs to have new antennas added and be reconfigured to accommodate them.

Purely exact models are less used for solving Graph Coloring and related problems, in comparison to heuristic methods [15]. This is due to the NP-Hard nature of these problems, which leads to high run time and memory requirements. However, they have an important role in proving the optimality of solutions, if any are found.

Here are proposed four different models for the RSI allocation problem. The sets, parameters, and variables can be seen in Table 2.1. These models were presented earlier in Londe et al. [29].

Table 2.1: Sets, Parameters and Variables of the model.

Set	**Definition**
V	Set of Graph Vertices
E	Set of Graph Edges
Λ	Set of Possible RSI values, which are between $minRSI$ and $maxRSI$
Parameter	**Definition**
$value_k$	Numerical Value of RSI of color k
$oldRSI_i$	Old configuration of vertex i
$minRSI$	Minimal value of RSI
$maxRSI$	Maximal value of RSI
$minDist$	Minimal distance between RSI of neighbor vertices
$maxDist$	Maximal distance between RSI of neighbor vertices
$M1, M2$	High values
Variable	**Definition**
$new_{i,k}$	Binary variable indicative of allocation of RSI k to vertex i
rsi_i	Non-negative integer variable indicating value of RSI allocation to vertex i
$y_{i,j}$	Binary variable related to distance between RSI of vertices i and j
$change_i$	Binary variable indicative if there was a change on the RSI of vertex i
min_span	Minimal value of the difference between RSI of two neighbor vertices

Formulations NC, Non-linear Change model, and LC, Linear Change model, have the objective of minimizing the amount of change between new and old network configuration, which means fewer disruptions and performance loss. Following are both formulations. Equations (2-2)–(2-9) compose formulation NC, while (2-10)–(2-20) do the same for LC.

For NC, Objective Function (2-2) minimizes changes in network configuration. Constraint (2-3) guarantees that all vertices will be allocated exactly one new RSI, while Constraint (2-4) indicates the relation between the allocated RSI and its numeric value. Constraint (2-5) guarantees that the new RSI will respect the minimal and maximal distances for all neighbor vertices, using the absolute value of the difference between their RSI. This use of absolute

value is what renders this formulation non-linear. Constraint (2-6) indicates the changes in RSI of the vertices. Finally, Constraints (2-7)–(2-9) define the possible values of the variables.

$$\min \quad \sum_i change_i \tag{2-2}$$

$$\text{s.t.} \sum_k new_{i,k} = 1 \qquad \forall i \in V, \tag{2-3}$$

$$rsi_i = \sum_k new_{i,k} \cdot value_k \qquad \forall i \in V, \tag{2-4}$$

$$minDist \leq |rsi_i - rsi_j| \leq maxDist \qquad \forall (i,j) \in E, \tag{2-5}$$

$$M1 \cdot change_i \geq |rsi_i - oldRSI_i| \qquad \forall i \in V, \tag{2-6}$$

$$change_i \in \{0,1\} \qquad \forall i \in V, \tag{2-7}$$

$$new_{i,k} \in \{0,1\} \qquad \forall i \in V, k \in \Lambda, \tag{2-8}$$

$$rsi_i \in \mathbb{N}. \tag{2-9}$$

For LC, the objective function is described in Equation (2-10). Constraints (2-11) and (2-12) are analogues to Constraints (2-3) and (2-4) of NC. Meanwhile, Constraints (2-13) and (2-14) restrict the maximal and minimal distances using an auxiliary variable to indicate whether the difference is positive or negative. Because of model symmetry, one of the differences must be positive and the other negative. Constraints (2-15) and (2-16) let the model register a change if the new value of the RSI is higher or lower than the old configuration. Finally, Constraints (2-17)–(2-20) define the domain of the variables.

$$\min \quad \sum_i change_i \tag{2-10}$$

$$\text{s.t.} \sum_k new_{i,k} = 1 \qquad \forall i \in V, \tag{2-11}$$

$$rsi_i = \sum_k new_{i,k} \cdot value_k \qquad \forall i \in V, \tag{2-12}$$

$$minDist \cdot (1 - y_{i,j}) - maxDist \cdot y_{i,j} \leq rsi_i - rsi_j \qquad \forall (i,j) \in E, \tag{2-13}$$

$$rsi_i - rsi_j \leq maxDist \cdot (1 - y_{i,j}) - minDist \cdot y_{i,j} \qquad \forall (i,j) \in E, \tag{2-14}$$

$$M1 \cdot change_i \geq rsi_i - oldRSI_i \qquad \forall i \in V, \tag{2-15}$$

$$M1 \cdot change_i \geq oldRSI_i - rsi_i \qquad \forall i \in V, \tag{2-16}$$

$$change_i \in \{0,1\} \qquad \forall i \in V, \tag{2-17}$$

$$new_{i,k} \in \{0,1\} \qquad \forall i \in V, k \in \Lambda, \tag{2-18}$$

$$rsi_i \in \mathbb{N}, \tag{2-19}$$

$$y_{i,j} \in \{0,1\} \qquad \forall i,j \in V. \tag{2-20}$$

Formulations NS, Non-linear Span model, and LS, Linear Span model try to maximize the smallest difference between the RSI of neighbors. This scenario considers the necessity of allocating the highest possible number of antennas in a certain space. For this, it is necessary that there are possible values for the RSI of these vertices, and that means increasing the distance between used RSI. NS is composed by Equations (2-21)–(2-27), and LS, by (2-28)–(2-36).

For NS, Constraints (2-22)–(2-24) and (2-26)–(2-27) are the same as the ones shown in (2-3)–(2-5) and (2-8)–(2-9) of NC, respectively. This makes this formulation equally non-linear. The objective function in Equation (2-21) indicates that we want to maximize the smallest of differences, defined by Constraint (2-25) as the absolute value of the smallest edge span.

$$\max \quad min\_span \tag{2-21}$$

$$\text{s.t.} \sum_k new_{i,k} = 1 \qquad \forall i \in V, \tag{2-22}$$

$$rsi_i = \sum_k new_{i,k} \cdot value_k \qquad \forall i \in V, \tag{2-23}$$

$$minDist \leq |rsi_i - rsi_j| \leq maxDist \qquad \forall (i,j) \in E, \tag{2-24}$$

$$min\_span \leq |rsi_i - rsi_j| \qquad \forall (i,j) \in E, \tag{2-25}$$

$$new_{i,k} \in \{0,1\} \qquad \forall i \in V, k \in \Lambda, \tag{2-26}$$

$$rsi_i \in \mathbb{N}. \tag{2-27}$$

In the case of LS, Constraints (2-29)–(2-32) and (2-34)–(2-36) are the same as portrayed in (2-11)–(2-14) and (2-18)–(2-20) of LC, while the Objective

Function (2-28) is the same as NS. The difference is in Constraint (2-33), which calculates the value of the difference between the RSI of two neighbors using the auxiliary variable. If the difference between the RSI of two vertices is negative, it will be added to a high value that will prevent this negative number from impacting the solution.

$$\max \quad min\_span \tag{2-28}$$

$$\text{s.t.} \sum_k new_{i,k} = 1 \qquad \forall i \in V, \tag{2-29}$$

$$rsi_i = \sum_k new_{i,k} \cdot value_k \qquad \forall i \in V, \tag{2-30}$$

$$minDist \cdot (1 - y_{i,j}) - maxDist \cdot y_{i,j} \leq rsi_i - rsi_j \qquad \forall (i,j) \in E, \tag{2-31}$$

$$rsi_i - rsi_j \leq maxDist \cdot (1 - y_{i,j}) - minDist \cdot y_{i,j} \qquad \forall (i,j) \in E, \tag{2-32}$$

$$min\_span \leq rsi_i - rsi_j + M2 \cdot y_{i,j} \qquad \forall (i,j) \in E, \tag{2-33}$$

$$new_{i,k} \in \{0,1\} \qquad \forall i \in V, k \in \Lambda, \tag{2-34}$$

$$rsi_i \in \mathbb{N}, \tag{2-35}$$

$$y_{i,j} \in \{0,1\} \qquad \forall i,j \in V. \tag{2-36}$$

It is necessary to comment that all four models are standard mathematical models, without any symmetry removing features, nor cutting features, or more sophisticated constraints. Moreover, similar models are implemented in ONAP platform [30] though ONAP Optimization Platform - Self-Organizing Networks (OOF-SON) for small and medium-sized networks.

3
Related Literature

With the RSI allocation problem described, it is logical to search for related literature. As it is a novel question, it means locating similar problems and their solution methods, alongside their similarities and differences to the RSI allocation problem.

The proposed mathematical models shown in Section 2.3 consider a characteristics of a modified graph coloring problem. In fact, many similar studies in parameter allocation for telecommunications problems consider variants of the graph coloring problem [11, 12, 14, 31], meaning that it may be used as a starting point for a review of related literature.

Thus, this chapter starts with a review of related literature to the classical graph coloring problem, and then explores generalizations of it that are closer to the RSI problem.

3.1
Graph Coloring Problem

The graph coloring problem, or vertex coloring problem (GCP), is a classic NP-hard problem known for its difficulty from the computational point of view, and its many real applications [32]. This problem's constraints are defined as such [33]:

$$\begin{aligned} c(i) &\in \Lambda(i) \quad \forall i \in V \\ c(i) &\neq c(j) \quad \forall (i,j) \in E \end{aligned} \tag{3-1}$$

In other words, this problem aims to color all vertices of a graph with possible colors, so that all neighbor vertices do not have the same coloration. In its classical form, the objective function of this problem is to minimize the number of colors needed to do a full coloring of the graph. As such, graphs with a known chromatic number are frequently used to check the strength of models and algorithms.

Randall-Brown [34] develops two algorithms to solve the GCP. The basic algorithm enumerates by backtracking, as starts with a partial solution to the GCP, then introduces each new vertex, checking all possible permutations of the solution in which the coloring is possible. The look-ahead algorithm, on the other hand, determines if a possible coloring would, at some later time, introduce an increase in the number of colors. As it is, the look-ahead has the

aim of reducing the number of backtracks. By testing on random graphs and a real life problem from the Massachusetts Institute of Technology, the look-ahead algorithm is clearly superior to the basic algorithm for certain graphs with high density or bigger graphs.

Leighton [35] introduces the recursive largest first (RLF) algorithm for coloring graphs. This strategy recursively selects nodes for coloring so that the resulting nodes are colorable in as few colors as possible. This is made by starting with assigning color 1 to the node with maximal degree. After coloring a number of nodes, then, select the uncolored node with maximal degree, such as it does not have any colored node as an adjacent. The author also introduces a procedure to generate test graphs with a known chromatic number, which would be called Leighton Graphs in the following works. The author tests many algorithms on a set of Leighton graphs and proves that the RLF algorithm is more efficient than other procedures for coloring graphs, with fewer colors in less time.

Brélaz [36] describes heuristic methods to solve the GCP, alongside introducing the DSatur method. The author compares both the DSatur method with other heuristics, and the combined Randall-Brown with look-ahead algorithm with other exact methods. DSatur starts with ordering of the vertices by decreasing order of degrees. Then, the first vertex receives color 1. A vertex with maximal saturation degree, and maximal degree that is uncolored is then colored with the least used possible color. This step repeats until all vertices are colored. The comparison was made with randomly generated graphs. The results point that the DSatur method is one of the best heuristics, while the addition of the look-ahead method to the Randall-Brown algorithm improves its performance.

Hertz and de Werra [37] introduce the TABUCOL procedure. This is the appliance of a Tabu Search to the GCP. The TABUCOL starts by generating a random coloring. From this coloring, random nodes are chosen, and colored with a random color that respects the problem constraints. From a number of possible changes, the best is picked-up and its change, as a pair (node-color), is added to the Tabu list. However, if a move that is in the Tabu list would cause a change to the objective, so that it would be a set percentage better than the best result, then the move has its Tabu status dropped. The TABUCOL was compared with a simulated annealing approach from the literature, and with a combined approach that mixes TABUCOL with an independent set strategy. This combined approach depended on finding the largest possible independent sets, which are then colored with TABUCOL. The combined technique is shown to have the best results, while the TABUCOL can find a possible coloring in

very short CPU time in comparison with results from literature.

Johnson et al. [38] study simulated annealing approaches to different problems, among which is the GCP. The authors describe and compare three approaches to the GCP, each with different neighborhood and objective function characteristics. The first approach is inspired by the RLF algorithm, and considers that a solution is any partition of a graph, even if the coloring is non-feasible. The amount of non-feasible edges are penalized in the objective function. The second approach only considers feasible solutions in its neighborhood, and uses Kempe chains to obtain new feasible solutions. A Kempe chain is any interconnected component in the join graph created by the union of two independent sets. The third approach focus on minimizing the number of non-feasible edges instead of the chromatic number. This case fixes the number of possible colors, and exchanges the coloring of vertices among those. The authors also implement a branch-and-bound algorithm with certain shortcuts, and a generalization of the RLF algorithm, both for comparison purposes. For the experiments the authors generated random graphs. The results point that for certain instance categories, like smaller or denser graphs, the simulated annealing approaches are competitive in comparison with the other methods. However, the authors note that it is difficult to point which methods are the best, as the different objective functions influence in the final results.

Fleurent and Ferland [39] examine the performance of hybrid genetic algorithms for GCP. Their approach depends of a greedy algorithm that colors the individual vertices, based on the ordering of the nodes given by a chromosome. This algorithm picks the smallest color than can be given, respecting the restrictions, or the least used color if none respect them. Another scheme availed was the string-based encoding, which is well suited to be combined with local and Tabu search. In particular, TABUCOL is noted as an efficient algorithm for graph coloring. The tests were made on random and Leighton graphs. For the first case, the hybrid approach was efficient in finding excellent results, but the Tabu Search was proven to be more effective in a smaller time, and the bigger graphs needed the combined Tabu-genetic approach to be solved. On the Leighton graphs, the hybrid Tabu-genetic algorithms managed to color some instances that the Tabu approach could not.

Costa and Hertz [40] study ant colony algorithms for CGP. The authors defend that the ant colony algorithm is well suited to solve assignment type problems, such as traveling salesman, job shop, and graph coloring problems. Their ANTCOL algorithm, in which each ant colors the entire graph in a constructive way, uses derivatives of the DSatur and RLF algorithm as

constructive methods. The experiments were made with Random graphs, in comparison with the results of other constructive methods. The combination of ANTCOL and RLF is shown to be much more effective than the one with DSatur, even though the algorithms were not able to surpass the best results of literature.

Galinier and Hao [41] design a hybrid evolutionary algorithm for the GCP. Their algorithm applies a local search to the child that results from the crossover on every generation. This local search is an improvement of the TABUCOL algorithm, with this addition being a change in the function of the best result, that indicates if a Tabu move can be accepted or not. For the experiments, the authors used DIMACS benchmarks and compared the hybrid evolutionary algorithm with Tabu Searches and best known records from literature. The hybrid outperforms the simple Tabu Searches in quality and speed of the solution, and is competitive in relation with the best known algorithms from literature.

Kassotakis et al. [42] try a hybrid genetic approach for channel reuse in telecommunication networks, a problem that can be modeled as a GCP. The proposed algorithm represents an individual as a binary matrix of dimension channels×nodes, and is combined with a local search algorithm that works as a local operator. This local search procedure modifies an existing chromosome by examining each node and checking if a change to the first possible channel causes a decrease of at least 25% of the initial fitness value. If no, then the algorithm checks the next possible channel until one is found. The simulations were made on two scenarios, one with static parameters and other, with dynamic ones. The experiments indicate that the hybrid GA reaches better solutions in a lower computational time in comparison with results from literature.

Chiarandini and Stützle [43] apply an Iterated Local Search (ILS) to the GCP. In this ILS, they identify three phases: the initialization phase that consists on an already existing algorithm and gives a feasible coloring; the color number decreasing phase, in which one color is removed from the graph, and the vertices with this color are all reassigned with a heuristic that focus on the vertices with higher degree; and the local search phase, that happens when the coloring returned by the color decreasing phase is not feasible. This local search uses 1-change neighborhood and has two possible strategies: it can choose at random a conflicted vertex and change its color; or it can see all combinations of vertices and colors, and choose the one that reduces the maximal number of conflicts. Those strategies are also enhanced with a Tabu search. The perturbation has three different possibilities, being

the random recoloring of a certain number of nodes, a heuristic recoloring of a certain number of nodes, and a randomly change which can be a Tabu search move of a random search. The authors test the algorithm on large or difficult instances from COLOR02/03/04. The comparison between algorithms shows that the heuristic reconstruction is the best perturbation strategy, with the all-combination analyzing local search being shown to be better than the alternative.

Barbosa et al. [44] introduce two evolutionary formulations of the GCP. The first formulation searches for the chromatic number of a graph as a problem of finding an acyclic orientation of the graph, according to the longest direct, from the highest degree to smallest, path that is shortest among all acyclic orientations. The second formulation is loosely inspired by the DSatur heuristic, and thus permutes the vertices in decreasing order of degrees, that are colored with the an pre-specified ordering of colors. The authors experimented on DIMACS instances, and the results are competitive when compared with other results from literature.

Galinier and Hertz [45] perform a survey of local search methods for GCPs, with the aim of explaining the still popularity of the old TABUCOL algorithm. The authors do this by dividing the search strategies for GCPs into four categories: the legal strategy has a search space with all legal colorings and minimizes the number of used colors; the penalty strategy has the same goal as the former, but its search space contains all colorings possible; the k-fixed partial legal and k-fixed penalty aim to color all vertices with a fixed number of colors k. Between these strategies, and the possibility of more neighborhoods, the authors conclude that the TABUCOL is very easy to implement and offers a compromise between quality and computational effort that the other strategies cannot compare with.

Méndez-Díaz and Zabala [46] propose a branch-and-cut algorithm for the GCP. The algorithm starts with the identification, with a simple heuristic, of a large clique as possible. Then, a coloring is generated using the Dsatur algorithm which gives an upper bound to the instances. After this, the authors relax and strengthen the linear programming constraints with valid inequalities. The branching rule is related to all sub problems generated by a yet uncolored vertex, and the cutting planes are based on the inequalities presented before. The experiments used DIMACS benchmarks and were implemented using C++ and CPLEX6.0. In comparison with Dsatur, the branch-and-cut algorithm managed to obtain optimality certificates faster.

Bui et al. [47] present an ant-colony strategy for GCP. In this strategy, the ants do not color the entire graph, but only a portion of it using local

information, with the sum of all individual colorings being the complete graph. The experiments were implemented in C++ in a Linux environment, with 119 benchmarks from COLOR02/03/04. The algorithm is shown to perform well in comparison with the best results in the literature, and is particularly consistent with a small standard deviation across all its runs.

Méndez-Díaz and Zabala [48] use cutting planes to diminish the number of symmetric solutions and have a more tractable model. The cutting plane strategy is made of a linear relaxation of the different constraints, followed by the introduction of strong valid inequalities, in order to have a smaller solution space. Their model manages to obtain better results than the classical maximum clique size strategy, except for cases made deliberately difficult to solve.

Caramia and Dell'Olmo [49] develop an iterated local search to solve the GCP. The heuristic starts with a greedy phase to generate the so called lowest and highest solutions. The highest can pass by another procedure which is equivalent to a simple local search. If the solution has not improved for a set number of iterations, then a perturbation that can find infeasible solutions, corrected afterwards, is made. The authors experimented with 119 instances from COLOR02/03/04 in a C environment. The algorithm manages to find solutions comparable to the state-of-art and improves the best known results for some.

Dowsland and Thompson [50] focus on an ant colony heuristic to solve the GCP. The heuristic updates the trail between vertices with the sum of the inverse of the total number of colors used on each solution. It also chooses between different colorings by the inverse of the number of uncolored vertices, after a number of chromatic classes are completed, number equivalent to the chromatic number of the graph. In the case of graphs in which the number is unknown, the chromatic number is considered dynamic and equal to the upper bound of the solution. The presented version enhances an already existing one and is shown not to be particularly robust. The experiments were made on the $G_{300,0.5}$ graphs already used in the literature, and point that this version is a dramatic improvement over the original, and that the approach is not competitive on larger graphs.

Malaguti et al. [9] propose a metaheuristic approach that starts with an evolutionary algorithm and ends with a post optimization phase based on the set covering model of the GCP. The evolutionary phase tries to improve the best found solution by performing a tabu search alongside crossover and mutation, with the focus on finding a solution with the known upper bound of a graph. If found, then the algorithm is reiterated with the upper bound

minus one as its new target. If the solution is not proven to be equal to the lower bound of the graph, then the column optimization algorithm is made. The authors note that this approach avoids symmetries in the solution. The algorithms were compared with the state-of-art algorithms on DIMACS instances. The evolutionary algorithm is shown to be very efficient, while the complete two-phase algorithm only performs slightly better in solution quality than the EA.

Mabrouk et al. [51] are interested in a parallel genetic-tabu search algorithm for the GCP. In this case, the tabu search replaces the mutation operator of the GA, while a RLF initializes the number of needed colors. They parallel the algorithm by considering a number of sub-populations, which behave independently. The master processor distributes the best results to the slave ones and is then sent new solutions to evaluate for each interaction. The tests were made on 13 DIMACS benchmarks for the non-parallel version, and those plus three Random graphs for the parallel version. The non-parallel version is shown to be very time consuming, otherwise performing competitively according to other literature results. The parallel version, on the other hand, does not perform well for small graphs, but is very efficient for larger ones, surpassing the non-parallel version.

Lü and Hao [52] develop a memetic algorithm for the GCP. This memetic algorithm combines a genetic algorithm with a tabu search procedure. The tabu search is used to improve an initial population of illegal individuals, minimizing the existing conflicts. Then, after the crossover, the children are also subjected to a tabu search. The tabu search procedure simply changes the color of a conflicting node and blocks the change for groups of ancient color-new color-nodes. The authors also propose a multi-parent crossover operator which builds a the color of a gene by considering the color of all parents. The authors test the algorithm on a set of DIMACS instances, having been implemented in a C language environment. In comparison with the best known results in literature, the memetic algorithm is shown to be highly competitive. The authors also tested the power of the multi-parent crossover and prove that it is more powerful than the classical 2-parent crossover in this context.

Hu et al. [53] studies the maximum differential graph coloring problem. In this, the aim is to maximize the difference along the edges of a colored graph, being thus a version of the classical GCP. This problem is noted as related to that of finding a k-Hamiltonian path, and the authors propose exact and heuristic algorithms for the problem.

Malaguti et al. [54] introduces a branch-and-price algorithm for the GCP. This algorithm starts with a MMT algorithm, that is at the root node of

the branch decision tree and generates both a tight upper bound and a set of columns that are used for the continuous relaxation of the model. Then, the authors generate those columns by relaxing an integrality constraint, obtaining the slave problem. This problem is then executed with a Tabu Search algorithm, in order to obtain a maximal stable set of the graph. If the Tabu Search cannot find a column with negative reduced cost, then the slave problem is tackled with CPLEX. This obtains a lower bound for the master problem. The authors then compare two branching schemes, one that branches on variables and the other, on edges. The experiments were made on DIMACS and show that the proposed algorithm proved optimal solutions to previous open instances and reduced the optimality gap of many others.

Douri and Elbernoussi [55] study a hybrid genetic algorithm based on a local search called DBG to the GCP. The DBG approximates the maximal independent set of the graph which is then used to initialize the number of possible colors. The evaluation of the algorithm was made with the DIMACS benchmarks, and the results were compared with other results of literature. It is shown that the algorithm is competitive in comparison with other approaches.

Guo et al. [20] explore a cluster resource allocation problem, formulated as a dynamic graph coloring problem, to assign resources for a device-to-device network. They propose an algorithm with three parts: first, the graph is decomposed into maximal cliques. Then, the cliques are ordered so that the ones with more inter-relations are first. Finally, the vertices are colored in order of their appearance in each suit. DIMACS graphs were used in the experiments, which show that the algorithm requires less run time than others in the literature.

Shukl and Garg [56] present a list-based solution to the GCP. The algorithm initiates with a vertex-color list with all vertices. The first node is assigned randomly with a possible color, while the other nodes depend on a list of all available colors, which is updated every time a neighbor pair color-node is assigned. The authors do not test the heuristic on instances.

Marappan and Sethumadhavan [57] try genetic and tabu search procedures to solve the GCP. For this, they introduce a new genetic operator called Advanced Local Guided Search. This takes a gene that selected to be crossed-over, goes to each of its conflicting vertices, and re-colors each of them with the smallest valid possibility. Then, the pair (color-node) is added to the tabu list. There are three methods compared for crossover and mutation, first the Single Parent Conflict Gene Extended Crossover (SPCGEX) and Conflict Gene Mutation (CGM), second, the Extended SPCGEX (ESPCGEX) and extended CGM (ECGM), and, thirdly, the Multipoint SPCGEX (MSPCGEX) and Mul-

tipoint CGM (MCGM) . The experiments were made with different categories of benchmark graphs, and show that the algorithm runs well even for small graphs, with the MSPCGEX and MCGM being the most satisfactory.

Sharma and Chaudhari [19] propose a different method for solving the GCP using the maximal independent set. They propose an algorithm that has three phases: create the complementary edge table by exploration, find the maximal independent sets, and, finally, color independent sets, each with a unique color. Implementation was made with Java for experiments on DIMACS instances, and results show that the algorithm always finds the chromatic number of the graphs, no matter the vertices' sequencing.

As it is shown, the GCP's solution methods are often heuristical in nature, as its difficulty is notorious. However, this problem's constraints are not as restricted as the RSI's - the colors of neighbors in the GCP must only be different, while the RSI has a range of possible differences.

3.2 Frequency Assignment Problem

A generalization of the previous problem, the frequency assignment problem (FAP) indicates that the difference between the labels of two adjacent vertices should be higher than the minimum weight of the edge between the two nodes [18]. Therefore, it is closer to the RSI problem than the GCP in nature. The problem can be characterized by those two constraints[18]:

$$\begin{aligned} & f(i) \in \Lambda(i) && \forall i \in V \\ & |f(i) - f(j)| \geq w(i,j) && \forall (i,j) \in E \end{aligned} \tag{3-2}$$

The variable $w(i, j)$ in Constraints 3-2 indicates the weight present on an edge. This is the formulation for the simpler FAP, which restrains the so called collision but not confusion. Collision occurs if two neighbors have the same assigned channel, while confusion happens when two neighbors of the same node have the same assigned channel [8]. Aardal et al. [58] classify FAPs by their objective function and two constraint types: assignment and interference.

This problem and many of its generalizations were introduced by Hale [10]. The author exemplifies different cases to prove certain characteristics of the problem and the relationship between generalized problems, in order to unify the theory of frequency assigned problems.

Cuppini [59] applies a genetic algorithm to the channel assignment problem. In this, the author represents a possible solution as a binary sequence of $m * n$ dimension, in which 1 indicates that the channel can be used for

that node. The algorithm was implemented in many different instances and indicates that the fitness function should be chosen carefully, as it strongly influences the performance of the heuristic.

Dorne and Hao [60] study evolutionary algorithms and their application to FAP. The authors present a hybrid evolutionary approach with a local search which was used with either a population of 1 individual or of 20 individuals. The approach was tested with 18 FAP instances provided by the French National Research Center on Telecommunications. They compared the results with a simulated annealing (SA) and a constraint programming (CP) approach. Both evolutionary algorithms perform better than the SA or CP, while the 20 individuals population is better than the 1 individual one.

Smith et al. [61] affirm that maximal cliques and subgraphs can improve known solutions to the FAP. The authors propose an algorithm that starts with the largest clique of a graph, which is then colored using any heuristic, and then the clique is extended with nodes to create a near clique and, again, is colored by some heuristic. The authors use the Philadelphia problems to compare the performance with existing results and demonstrate that their approach can have better results than the direct application of heuristics.

Chakraborty [62] proposes a heuristic algorithm for the channel assignment problem. The heuristic creates a pool of valid solutions to a given problem. The algorithm creates a matrix $m \times n$, in which each line, indicating the possible colors, is computed one by one. The column order that indicates the nodes is randomized for each line. The first free channel is given a +1, indicating that it is assigned to the node. Then, the other nodes are analyzed and given a 0 if the frequency is assignable, -1 if not, and +9 if unused. This goes to all columns. The experiments were made with several instances from previous works in literature. The algorithm is shown to be very fast and can in most cases reach the known optimum or vert close to it.

Krumke et al. [63] study the channel assignment problem for packet radio networks. They approximate a solution using the distance-2 coloring problem, a generalization of the frequency assignment problem that considers a square edge between two vertices if there is a path between then of at most two edges. This square edge must obey certain restrictions, like a normal edge. The algorithm starts by selecting a random vertex and assigning levels to the other vertices, corresponding to the number of nodes between the selected and the others. The coloring of the resulting tree for a certain level considers the two anterior levels which can be reused.

Alabau et al. [64] present hybrid genetic algorithms to the FAP. Their individual is made of genes coded by integers that indicate the color of each

individual. The authors developed two original crossover operators, one that passes all the good genes to the child and operates on part or all the genes in conflict, and one that performs jumps in the space solution to escape from the uniformity of the population. Two mutation operators were also developed, the first that classifies the genes in decreasing order of their objective function and randomly mutates a number of then, and the second that uses a probabilistic tabu search, in which changes in the tabu list are possible. The authors make experiments with several instances provided by TDF-C2R Broadcasting and Wireless Research Center. The results point that the probabilistic tabu search alongside the first crossover operator is the best choice in comparison with other results in literature.

Kendall and Mohamad [65] propose a hyper-heuristic methodology for the channel assignment problem. This is a knowledge-poor metaheuristic that chooses which heuristic to call at each decision point. The authors compare with the results of previous studies in already existing benchmarks. It is indicated that this approach is better than other existing ones, and that the the acceptance criteria that permits worse solutions is the one with best performance.

Bandh et al. [66] analyze how the proprieties of a colored graph can be used to extend an already existing network with new cells. For this, the authors use a greedy algorithm to create solutions, followed by the extension of the graph, so that it also contains the cells of the neighbors of neighbors. To add new cells, the algorithm proposes that only the neighbors of the new cell can have their colors changed, thus affecting the least the current configuration. This approach has tested in two data sets, one from Vodafone Germany‘s 3G sites, and another artificial, although the results are not compared with other approaches.

Ahmed et al. [67] apply both local search and complete algorithms to primary component carrier selection and Physical Cell Identification (PCI) assignment. The local search algorithms differ in the ordering of the nodes and the selection of them; the ordering can be by the number of conflicts or the strength of the conflicts, while the selection can be random or greedy. The complete algorithms are based on a complete search tree and use global identification to break the symmetry between agents. One of them orders the resources in alphanumerical order, while the other reorder agents during computation to minimize constraints. To evaluate the algorithms, an office Manhattan scenario was developed. The authors concluded that the complete algorithms tend to outperform the local searches when the number of colors is small, and that the number of components is strongly related to the number

of neighbors.

Xu et al. [68] model a PCI configuration scheme based on hypergraph coloring. The authors describe this approach as trying to find the forbidden set, composed by a group of cells that cannot have the same PCI. The proposed model focuses on the inclusion of new nodes by allocating its PCI in accordance with the algorithm. The authors do not test the algorithm on instances.

Sun et al. [69] propose a genetic algorithm to solve the PCI assignment problem. For this, they use as chromosomes a binary matrix $n*m$, with n being the number of nodes and m, the number of colors. In this case, an antenna needs to have a certain number of assigned PCIs. The algorithm was coded with C# and compared with the simulation results of manual and automatic PCI plans. This assignment of PCIs is shown to reduce the interference in the network.

Klincewicz [70] uses GRASP to solve a particular cell site assignment problem. The objective of the problem is to minimize the sum of the weights of nodes with the same colors. The GRASP algorithm starts with the ordering of the nodes by their degree, followed by the random choice of one of the first L nodes from the candidate list. This node is assigned the color that would increase the least the objective function. After constructing a solution, the algorithm then would do a local search, in which the nodes are listed arbitrary, then each node is chosen and has all possible exchanges evaluated. If there is an advantageous exchange, then it is made. A double exchange is also evaluated, in which all pairs of colors are evaluated together. The heuristic was implemented in a C environment and tested on four instances with real data of four US areas. The results presented indicate that the double exchange is better for larger instances, while the single is preferred for smaller ones. The GRASP CPU time also is shown to be always smaller than the CPLEX time.

Ahmed and Tirkkonen [71] discuss the PCI assignment for densely deployed heterogeneous networks. The authors analyze local search methods, each modifying the PCI of one cell, and focused search algorithms, which only allow local moves to the unsatisfied nodes. The proposed algorithms were tested on instances simulated with a plethora of parameters, and the results show that the proposed algorithms can satisfactorily obtain conflict and confusion free colorings.

Pratap et al. [31] propose an algorithm to allocate the PCI of femtocells which tries to satisfy the collision and confusion free assignment, as well as to accommodate the maximum number of femtocells possible. As femtocells are deployed by client requirements, thus having a strong random component, the authors model this problem with random graphs, and as a k-coloring problem.

The k-coloring limits the used colors to set k number, so that two neighbors do not have the same color. The algorithm first creates the femtocellular network topology, then identifies each femtocell as a node, with neighbors connected via edges, as well as second degree neighbors. The resulting graph is used to the PCI allocation. The tests were made with a real dataset from Vodafone Germany 3G sites. The proposed algorithm lets a higher number of femtocells be deployed in comparison with the existing PMM-GRS method, and a lower number of PCI's are used in a lower computational time.

Kowalik and Socała [72] apply the meet-in-the-middle approach to the channel assignment problem and the generalized t-coloring problem. The meet-in-the-middle technique finds partial solutions for all possible halves of the vertex set, then merging the partial solutions in order to solve the full instances. This algorithm is also applied to the generalized T-coloring problem, in order to improve the known upper bound of the problems. The article focuses on the theoretical ramifications of the algorithm, not on an application.

Acedo-Hernández et al. [14] study the PCI allocation problem. The authors model it as a graph partitioning problem, in which the graph is simplified many successive times until the number of vertices is equal to the desired number of subdomains. In this simplest graph, the partitioning is then built. Then, the graph is unfolded until the original one, with each unfolding being succeeded by a local refinement algorithm. The authors test the algorithm with a real instance, composed of 620 nodes and six different configurations to the planning method of the algorithm. The results indicate that the algorithm can eliminate PCI collision and confusion and minimize the collision for other parameters in most cases, in comparison with the current solution.

Zhao et al. [13] introduce a greedy channel-based assignment algorithm to solve the interference in device-to-device communications. After presenting the exact model for the problem, the authors then show an algorithm that assigns a color to an uncolored vertex based on the current coloring of the entire system. The heuristic assigns colors based on the sum of the suffered interference of a device, allocating the color that has been used last in the neighborhood. A tabu search is also made for the robust version of the problem. It is shown that the greedy heuristic has better results and is fairer than the robust, random and classic heuristics for this problem.

Siddiqi and Sait [11] propose a neighborhood search-based heuristic for the fixed spectrum frequency assigned problem. This problem has constraints that specify the needed separation between frequencies in order to minimize interference. The heuristic has two components: a local search, in which only

the frequency of one transmitter is changed and whose positivity is evaluated with a lookup table; and a compound move, which shifts the local search to a new location in the solution space after a certain number of iterations without improvement. The algorithm also uses an archive of solutions which stores the best options found. The authors implemented the code in C++ and R, the first for the local search and the second to maintain the archive and compound move. The comparisons on benchmarks with two existing single solution heuristics show that the proposed one is better in most cases, with its worst result being, in some cases, better than the best reported result for the others.

Marsa-Maestre et al. [12] tackle the channel assignment in Wireless Local Area Network, which is similar to the well-known FAP and GCP. They model the Wi-Fi infrastructure network as a three layered graph and analyze the use of nonlinear negotiation approaches with graph properties. The authors characterize the problem as a negotiation process, in which two protocols interact in order to improve the candidate coloring. The negotiators have different decision mechanisms that can be *hill-climber*, in which the agents only accept a proposal if its utility is at least the same as the anterior; or *annealer*, that follows the simulated annealing strategy. The comparison as made with the other two strategies, the *random* and *Augmented Lagrangian Particle Swarm Optimization*. The experiment, made on a realistic Wi-Fi configuration network, show that there is no difference in performance for simpler graphs, but the *annealer SA* becomes the best choice as the network gets more complex.

As can be seen in this section, the FAP is also frequently solved using heuristics. In fact, genetic algorithms are frequently used, alongside local searches and greedy algorithms. Like the GCP, the FAP's constraints are easier than the RSI's, as the latter problem calls for a more limited interval of possible differences between the color of neighbors in comparison with the former.

3.3 T-Coloring Problem

Introduced by Liu [73], this problem indicates that the difference between the labels of two neighbors *must not* be a value inside of a set called T. As the set can be non-sequential, this can be considered a generalization of the Frequency Assignment Problem. If the possible assigned colors are restricted to a set, this problem becomes a *List T-Coloring* [18]. This problem has been proved as NP-Hard [18].

The aim of this problem is to find a coloring of a graph, such as [18]:

$$\begin{aligned} &\phi(i) \in \Lambda(i) && \forall i \in V \\ &|\phi(i) - \phi(j)| \notin T(i,j) && \forall (i,j) \in E \end{aligned} \tag{3-3}$$

In the constraints above, the set indicated by T is the number of values there are forbidden for the edge span. If this set is composed only by consecutive numbers, then the T-coloring problem (TCP) becomes a FAP. Otherwise, if the set T only comprises of 0, then the problem becomes a classical GCP [18]. Lastly, if the set is composed as shown in Equation 3-4, then the TCP describes correctly the RSI allocation problem. The TCP is, however, a problem rarely studied in literature, whose authors focus more on the mathematical characteristics instead of practical uses and solution methods.

$$T(i) = [0, \ldots, minDist] \cup [maxDist, \ldots, L_{RA}], \ \forall i \in V \tag{3-4}$$

Costa [74] studies a generalization of the T-coloring problem, in which each edge has a minimum difference associated. For this, he adapts both a Tabu search and a simulated annealing procedure. The Tabu search changes the color of conflicting nodes only if the pair color-node is not present in the tabu list, and chooses the best option in the neighborhood. The simulated annealing also changes the color of conflicting nodes, but can accept a worse solution with a decreasing probability. The author compares the results with an exact branch-and-bound algorithm in randomly generated graphs. The branch-and-bound is only applicable in small graphs, while the tabu search is much more efficient than the simulated annealing procedure.

Alon and Zaks [75] study algebraic and probabilistic techniques to solve the T-coloring problem, with a focus on mobile communication networks. The authors focus on the theoretical characteristics of this problem.

Mcdiarmid [76] applies the T-Coloring Problem for a UHF transmitters problem with a constraint matrix model. In particular, the author discusses many different proprieties of this problem, including the upper and lower bounds and the span of the graph. He focus on the mathematical side of this problem, and therefore do not experiment with instances or practical results.

Junosza-Szaniawski and Rzążewski [18] introduce an exact algorithm for the generalized list T-coloring problem. This algorithm uses binary variables to encode partial k-channel assignments. The authors show that the time of this algorithm depends on the maximum forbidden difference of the input graph, as well as present different ways to improve the bounds if the graph has some special propriety.

3.4 Other remarks

Table 3.1 presents the many works contemplated in this Literature Review. As commented in Section 3.1, the Graph Coloring Problem is a famously difficult problem, but whose constraints are markedly different than those of the RSI problem. The Frequency Assignment Problem detailed in Section 3.2 is closer in nature, but it is its generalization, the little explored T-Coloring Problem, that the RSI allocation problem can be classified as.

It is noticeable that the literature studied in this chapter is very disperse. There is a tendency to adapt the problem to the author's circumstances, and thus a certain difficulty in following already known works. Also of note is the tendency of using heuristic methods instead of exact ones. This is due to the difficulty of the problems, considered NP-Hard, and therefore demanding more sophisticated methods or heuristics to cut the time needed to find good solutions.

In fact, it is of note the use of genetic algorithms. They are one of the most frequently used algorithms for graph coloring and correlated problems, and show varying levels of success, in special when in combination with other heuristics in hybrid genetic algorithms.

Authors are also inclined to use their own instances in experiments. This is related to the many adaptations of this problem to real life situations. In fact, many do start studies in this problem to solve to solve real life issues, hence do not see a need to adapt old instances to the current settings.

Author	Year	Problem	Method
Randall-Brown	1972	GCP	Randall-Brown
Leighton	1979	GCP	RLF
Brélaz	1979	GCP	Dsatur
Hale	1980	FAP	N/A
Hertz and de Werra	1987	GCP	TABUCOL
Johnson et al.	1991	GCP	Simulated Annealing
Costa	1993	TCP	Tabu Search; Simulated Annealing
Cuppini	1994	FAP	Genetic Algorithm
Dorne and Hao	1995	FAP	Hybrid Evolutionary Algorithm
Fleurent and Ferland	1996	GCP	Hybrid Genetic Algorithm
Costa and Hertz	1997	GCP	Ant Colony Algorithm
Smith et al.	1998	FAP	Maximal Clique Strategy
Alon and Zaks	1998	TCP	Algebraic and Probability
Galinier and Hao	1999	GCP	Hybrid Evolutionary Algorithm
Kassotakis et al.	2000	GCP	Hybrid Genetic Algorithm
Chakraborty	2001	FAP	Many Solutions Generation
Krumke et al.	2001	FAP	Distance-2 Coloring Strategy
Chiarandini and Stützle	2002	GCP	Iterated Local Search
Alabau et al.	2002	FAP	Hybrid Genetic Algorithm
Mcdiarmid	2003	TCP	Constraint Matrix Model
Barbosa et al.	2004	GCP	Evolutionary Algorithm
Kendall and Mohamad	2004	FAP	Hyper-Heuristic
Galinier and Hertz	2006	GCP	N/A
Méndez-Díaz and Zabala	2006	GCP	Branch-and-Cut
Bui et al.	2008	GCP	Ant Colony Algorithm
Méndez-Díaz and Zabala	2008	GCP	Cutting Planes
Caramia and Dell'Olmo	2008	GCP	Iterated Local Search
Dowsland and Thompson	2008	GCP	Ant Colony Algorithm
Malaguti et al.	2008	GCP	Evolutionary Algorithm, Set Covering
Mabrouk et al.	2009	GCP	Hybrid Genetic Algorithm
Bandh et al.	2009	FAP	Greedy Algorithm
Lü and Hao	2010	GCP	Memetic Algorithm
Hu et al.	2010	GCP	N/A
Ahmed et al.	2010	FAP	Local Search
Malaguti et al.	2011	GCP	Branch-and-Price
Xu et al.	2011	FAP	Hypergraph Coloring
Sun et al.	2012	FAP	Genetic Algorithm
Klincewicz	2012	FAP	GRASP
Junosza-Szaniawski and Rzążewski	2014	TCP	Binary-encoded Assignments
Douri and Elbernoussi	2015	GCP	Hybrid Genetic Algorithm
Ahmed and Tirkkonen	2015	FAP	Local Search
Pratap et al.	2016	FAP	K-Coloring Strategy
Kowalik and Socała	2016	FAP;TCP	Meet-in-the-middle Approach
Guo et al.	2017	GCP	Cluster Allocation
Acedo-Hernández et al.	2017	FAP	Graph Partitioning Strategy
Shukl and Garg	2018	GCP	List-based Algorithm
Marappan and Sethumadhavan	2018	GCP	Hybrid Genetic Algorithm
Zhao et al.	2018	FAP	Greedy Algorithm; Tabu Search
Siddiqi and Sait	2019	FAP	Neighborhood Search-based Algorithm
Marsa-Maestre et al.	2019	FAP	Negotiation Process Algorithm
Sharma and Chaudhari	2020	GCP	Maximal Independent Set Strategy

Table 3.1: Summary of Literature Review

4
Solving the RSI Allocation Problem

This chapter introduces the proposed solution method for the RSI allocation problem. As Chapter 3 shows, the Graph Coloring Problem (GCP) and related problems are all considered NP-Hard [32], so authors tend to use heuristics to find good solutions in tractable computer time and effort. Metaheuristic and heuristic methods are frequently used to solve NP-Hard problems [16]. This is due to the problem's hardness, which can translate into exponential computational times in general. This high complexity holds for the GCP, the Frequency Assignment Problem (FAP), and other graph coloring problems.

As Chapter 3 shows, heuristic methods are frequently found in the literature. In particular, the application of Genetic Algorithms is rather frequent, as detailed by Mostafaie et al [16]. Genetic Algorithms (GA) have certain advantages for the GCP. They can find good, competitive solutions in comparison with other methods, with a trend of quick convergence even in large graphs. On the other hand, they have high exigences of memory and processing [16]. GAs are some of the first population-based algorithms, and are inspired in the Darwinian theory, evoking the evolution of a population and the survival of the fittest individuals [77].

One GA in particular, the Biased Random Key Genetic Algorithm (BRKGA), introduced by Gonçalves and Resende [78], had many successful applications in telecommunications problems [7, 79, 80, 81]. However, it has not yet been used for graph coloring applications.

Thus, this chapter introduces a BRKGA for the RSI allocation problem. First, the algorithm is presented, then the adaptation of it to the RSI problem is detailed.

4.1
Biased Random Key Genetic Algorithm

In the BRKGA, a possible solution is defined by an individual chromosome, a vector composed of genes indicated by *random keys*. The algorithm initiates by creating a population $\mathcal{P}$ of $|\mathcal{P}|$ chromosomes, so that each gene of each individual is initialized with a randomly selected number between 0 and 1.

The *Decoder* procedure obtains the *fitness* of a certain chromosome, which is related to the value of the problem's objective function. At each

generation, the chromosomes are divided into ELITE or NON-ELITE sets, depending on their ranking inside of the population, based on their fitness values.

4.1.1 Evolutionary process

At each generation, three different operators act to create a new population: *Reproduction*, *Mutants Generation*, and *Crossover*. *Reproduction* copies all $|\mathcal{P}_e|$ individuals in the elite set, while *Mutants Generation* deletes the worst individuals $|\mathcal{P}_m|$ and randomly generates the same number for the new population. At last, *Crossover* creates $|\mathcal{P}| - |\mathcal{P}_m| - |\mathcal{P}_e|$ offspring chromosomes to fill the population.

In the classical BRKGA [78], *Crossover* uses two randomly chosen chromosomes, one from the elite set and the other from the non-elite set. A new individual is then generated by the mating of the two parents, in which the choice between genes is biased in favor of the *elite* one.

This study uses the BRKGA framework from Lucena et al. [82] and Andrade et al. [83], in which multiple parents can be used for mating. The algorithm utilizes π_e elite parents and $\pi_t - \pi_e$ parents from the non-elite set, whose genes are selected with a fitness ranking/position-based bias function. Londe et al [84] have shown the improvement in solution quality gained by this approach, in comparison with other methods. As Lü and Hao [52] show, a multi-parent alternative can have better results than the classical one for genetic approaches to graph coloring.

After reproduction, mutants generation, and crossover steps, a new cycle of evaluation-selection-evolution takes place repeatedly until a stopping criterion is met, i.e., the BRKGA runs while a pre-specified number of generations or a certain processing time is not reached in general. However, other stopping criteria can be used.

4.1.2 Auxiliary Methods

Warm-start: Classical BRKGA starts by creating the population with randomly generated individuals. Nonetheless, recent works [79, 85, 86] have shown that the introduction of good solutions on the initial population increases the performance of the algorithm. This procedure, thus, creates one or several initial good solutions – normally based on a greedy algorithm – and introduces them to the initially generated population as an individual.

Shaking: Introduced in Andrade et al. [86], this procedure partially re-initializes the population, so that good genes originated from the convergence will not be lost, as happens in a full resetting. This method involves the application of random modifications to elite chromosomes and the re-initialization of the non-elite ones. Thus, the structure of the elite chromosomes is partially preserved, and the diversity of the non-elite set is guaranteed.

Resetting: If the shaking procedure does not manage to re-introduce adequate genetic diversity, i.e., the algorithm finds itself stuck on a local optimum for a high number of generations, then a full reset may be needed [87, 88]. This procedure re-initializes the full population with randomly generated individuals, losing the benefits of convergence but potentially finding yet unexplored solutions. Resetting prevents premature convergence [89].

Island model: Another tool for preventing premature convergence [87, 89], the island model involves the evolution of parallel populations, which exchange elite individuals after a set amount of generations. This procedure also improves individual variability [83].

Implicit Path-Relinking (IPR): An intensification strategy, the path-relinking procedure explores the neighborhood obtained in the path between two good, distinct solutions [90]. Path-relinking operations normally depend exclusively on the problem in question. However, the BRKGA-MP-IPR framework [83] introduces the implicit path-relinking, which creates the procedure inside the existing BRKGA solution space. In the same framework, the path-relinking is applied in concert with the island model, so that different good solutions from distinct populations in a circular effort from population 1 to 2, from 2 to 3, and from 3 to 1. Those good solutions might be the best ones in the populations or sampled randomly among the elite sets. There is more than one type of implicit path-relinking, as show Andrade et al. [83]. Of relevance to this work is the permutation-based IPR. A permutation-based algorithm considers a permutation of the decision variables by assigning each decision variable one index of the random-key vector and sorting this vector to find a possible solution. The permutation-based IPR, thus, observes if an element is in the same position, in this sorted vector, on both analyzed individuals. If not, then the base solution is changed, so that the element takes the same position as the guide solution [83]. In this case, the distance between two individuals is calculated as the Kendall tau rank

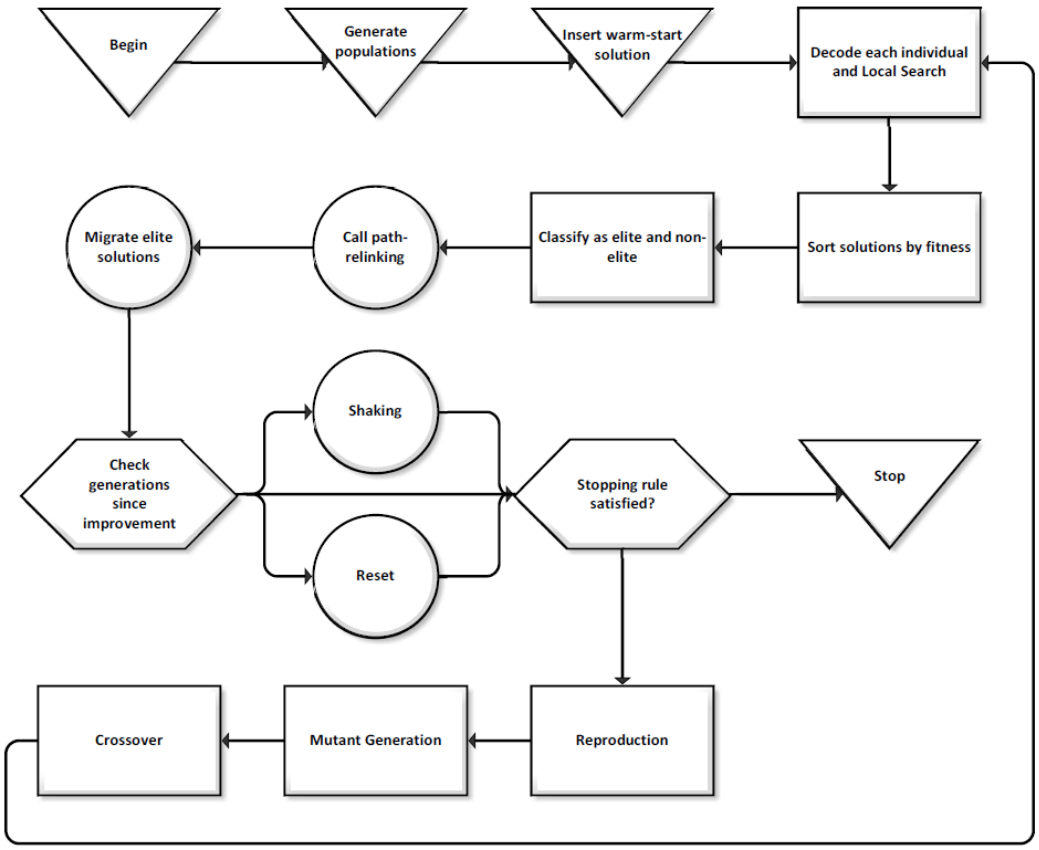


Figure 4.1: Flowchart detailing BRKGA for the RSI allocation problem. Triangles indicate procedures done only once. Circles are procedures that may happen in each generation, based on pre-specified parameters. Finally, squares are procedures that are made in all generations.

distance [91], which considers the number of pairwise disagreements between two ranking lists.

4.2 Solution Procedures using BRKGA

In the next subsections, there are the customized procedures for the RSI problem, whose procedure order may be observed in Figure 4.1 and parameters, in Table 4.1. In it, the algorithm begins by generating p populations of $|\mathcal{P}|$ random individuals. Afterward, the warm-start solution detailed in subsection 4.2.2 is introduced in the populations.

Those individuals pass, then, through the decoding process, obtaining their respective fitness value. These values are used to rank the individuals as elite or non-elite. The amount of elite individuals is given by a percentage $\mathcal{P}_e\%$ of the total population size.

If possible, then the IPR procedure is called. Path-relinking selects chromosomes with minimal distance md, and performs the relinking according

Table 4.1: Main algorithm attributes and control variables.

Parameter	**Description**
p	Number of independent populations
$\|\mathcal{P}\|$	Population(s) size
$\mathcal{P}_e\%$	Percentage of elite individuals
typ	Path-relinking type
md	Minimal distance between chromosomes for IPR
sel	Individual selection for IPR
ps	Path percentage/size
g	Interval for inter-population migrations
I_s	Number of iterations for shake calling
R_m	Reset iteration multiplier
$\mathcal{P}_m\%$	Percentage of mutant individuals
π_e	Number of elite parents in mating
π_t	Number of total parents in mating
$\Phi(r)$	Bias function for mating

to type typ. The elite individuals are selected depending on criteria sel, while ps controls the maximum path size. This maximum path size is computed as a percentage of the chromosome size.

After path-relinking, it is possible to migrate elite individuals between different populations. This happens after g generations without a betterment of the optimal solution. Similarly, if I_s generations have passed without an improvement, then the shaking procedure may be called. Finally, if there are $I_s \cdot R_m$ generations without improvement, then a full reset of the populations is performed.

The algorithm, then, checks if the stopping rule has been satisfied. If not, then the evolutionary process is performed, with reproduction, mutant generation, and crossover phases. In reproduction, all elite chromosomes are copied, while mutant generation deletes a percentage $\mathcal{P}_m\%$ of the worst chromosomes and creates the same amount with randomly generated keys. Finally, crossover performs mating between π_t parents, of which π_e are elite individuals. As a criterion for selecting genes in mating, the algorithm uses the bias function indicated as $\Phi(r)$, which considers the rank r of the selected parents in relation to each other. Afterward, the algorithm goes back to the decoding phase and performs its procedures until the stopping criteria is satisfied.

Six different decoders were customized for this problem, each using a different strategy. These decoders use several ways of interpreting the

chromosome, or even different chromosome characteristics, in order to obtain a possible coloring for the graph.

One may note that all decoders permit illegal colorings, i.e. colorings in which the minimum-maximum difference is not respected. These conflicting edges are penalized in the optimal function by a constant factor C. In addition, a procedure for correcting the illegal colorings can be performed as a part of all decoders. This procedure is detailed in Subsection 4.2.10.

4.2.1 Fitness Value Calculation

The decoders obtain the coloring cost separately, as shown in Algorithm 1. In it, lines 3–5 detail the procedure to identify the number of conflicting edges, by checking if the minimum-maximum difference is not respected. Then, between lines 6–16 the coloring cost is calculated. In the first case, the number of changes in configuration is obtained by comparing the old and new RSI configuration of each vertex (Lines 6–10). For the second case, the cost is calculated with the minimal edge span in Lines 12–16. It is of note that the minimal edge span *has to* be the value of a legally colored edge.

After the cost is first calculated, two procedures can be performed, the correction procedure, and the local search. These procedures are detailed in Subsections 4.2.10 and 4.2.11.

4.2.2 Warm Start

For the RSI allocation problem, the warm start procedure colors the vertices with the smallest possible color, considering the minimum and maximum difference between the RSI of vertices. This procedure colors the vertices in non-increasing degree order, i.e., the vertices with higher degrees are colored first. The solution generated by this procedure may be infeasible.

This procedure uses the initial coloration for both objectives. The initial coloration is more important for the minimize changes objective, but it is also considered for the maximize minimal span case.

Algorithm 1: Obtaining the coloring cost.

Input: Vector $RSI \in \mathbb{Z}^n$ where n is the number vertices; type of objective.
Output: Total coloring cost.

```
1  Initialize NumChanges, NumConflicts as zero;
2  Initialize MinEdgeSpan as MaxDist;
3  foreach (i, j) ∈ E do
4      if (MaxDist < |RSI_i − RSI_j|) or (MinDist > |RSI_i − RSI_j|) then
5          NumConflicts ← NumConflicts + 1;
6  if Objective is "minimize Changes" then
7      foreach i ∈ V do
8          if RSI_i ≠ OldRSI_i then
9              NumChanges ← NumChanges + 1;
10     ColoringCost ← NumChanges + C · NumConflicts;
11 else                          // Objective is maximize minimal edge span
12     foreach (i, j) ∈ E do
13         if MaxDist ≥ |RSI_i − RSI_j| ≥ MinDist then
14             if |RSI_i − RSI_j| ≤ MinEdgeSpan then
15                 MinEdgeSpan ← |RSI_i − RSI_j|;
16     ColoringCost ← MinEdgeSpan − C · NumConflicts ;
17 if NumConflicts ≥ 0 then
18     ColoringCost ← Correction(RSI);
19 ColoringCost ← Local Search(RSI, ColoringCost);
20 return ColoringCost
```

4.2.3 Logic Direct decoder (LD)

The *Logic Direct* decoder (LD) is named as such due to its use of logical variables for the relationship between the values of the RSI of neighbor nodes. These boolean variables indicate, in the end, whether a vertex can or cannot be colored with a given RSI. This decoder interprets the chromosome as the order in which vertices are colored, being thus n-sized, with n indicating the number of vertices in the instance.

Algorithm 2 shows the procedure of this decoder. The decoder starts by assigning the old configuration to the first vertex in the non-increasing order. All other vertices pass by a check, seeing if the previous configuration is possible with aid of the logical relation in Line 9. If the old configuration is not possible (Line 12), then all possible colors are checked in increasing order, with the loop in Line 16. If the vertex could not be colored with any of the other possible RSI values, the vertex is allocated its previous configuration regardless of its impossibility.

Algorithm 2: Coloring the graph – decoder LD.

Input: Chromosome/vector $v \in [0,1]^n$ where n is the number of vertices.
Output: Total coloring cost.

```
1  Let γ to be a permutation of vertices induced by the non-decreasing order
     of corresponding keys in v;
2  Initialize RSI as zero-vector;
3  foreach i ∈ γ in the given order do
4      if i is first in order then
5          RSI_i ← OldRSI_i;
6      else
7          CandidateColor ← OldRSI_i;
8          Possible ← true;
9          foreach j adjacent to i and RSI_j ≠ 0 do
10             if (|RSI_j − CandidateColor| > MaxDist) or
                 (|RSI_j − CandidateColor| < MinDist) then
11                 Possible ← false;
12         if Possible then
13             RSI_i ← CandidateColor;
14         else
15             CandidateColor ← minRSI;
16             while CandidateColor ≤ maxRSI do
17                 foreach j adjacent to i and RSI_j ≠ 0 do
18                     if (|RSI_j − CandidateColor| > MaxDist) or
                         (|RSI_j − CandidateColor| < MinDist) then
19                         Possible ← false;
20                 if Possible then
21                     RSI_i ← CandidateColor;
22                     break;
23                 else
24                     CandidateColor ← CandidateColor + 1;
25     if RSI_i = 0 then
26         RSI_i ← OldRSI_i;
27 ColoringCost ← CalculateCost(RSI);
28 return ColoringCost
```

4.2.4 Logic Indirect Decoder (LI)

Similar to the Logic Direct Decoder, the Logic Indirect Decoder also considers the logical constraints. However, in this variant, indirect neighbors (i.e., neighbors of neighbors) are also taken into consideration. This is the only difference between these two logic decoders.

The relationship between neighbors of neighbors can be defined as follows: Let $i, k \in V$ be two vertices so that $(i,j),(j,k) \in E$ exist for at

least one vertex $j \in V$. The values for RSI_i and RSI_k must obey the relation shown in either Equations (4-1a) and (4-1b), or (4-2), which depends of the values of MinDist and MaxDist.

$$2 \cdot \mathsf{MaxDist} \geq |\mathsf{RSI}_i - \mathsf{RSI}_k| \geq 2 \cdot \mathsf{MinDist} \tag{4-1a}$$

$$\mathsf{MaxDist} - \mathsf{MinDist} \geq |\mathsf{RSI}_i - \mathsf{RSI}_k| \tag{4-1b}$$

$$2 \cdot \mathsf{MaxDist} \geq |\mathsf{RSI}_i - \mathsf{RSI}_k| \tag{4-2}$$

To better illustrate this relationship, Figures 4.2–4.4 show the process to obtain these mathematical relations for the case shown in Equations (4-1a) and (4-1b). First, as shown in Figure 4.2, the possible values for X's RSI in relation to A, and Y's in relation to X.

The objective is to obtain Y's possible RSI in relation to A, which will be done by moving the center of Y's range to each of the diamonds named α, δ, η, and ζ in X's range, with Y's ranges being indicated in a blue left-inclined hatch. This is shown, for each diamond, in Figure 4.3. The last range of this figure indicates the union of the four previous ranges.

Finally, this last object of Figure 4.3 is explored in Figure 4.4, which shows the relations between the blue zones and the center of the interval. As such, Y's RSI in relation to A's must be in the blue left-inclined hatch, which, when considering the hole of the absolute operator, means one of two zones.

For Equation 4-2, the difference between the maximal and minimal distance or the RSI of neighbors must be higher than twice the minimal distance. This would cause the two intervals to intercept each other, and thus only the maximal distance would be a restriction.

Algorithm 3 presents the pseudo-code for this decoder. It starts similarly to the previous decoder, with the addition of the indirect neighbors check in Lines 12–18. These lines guarantee that the color of an indirect neighbor will not influence negatively in the legality of the vertex.

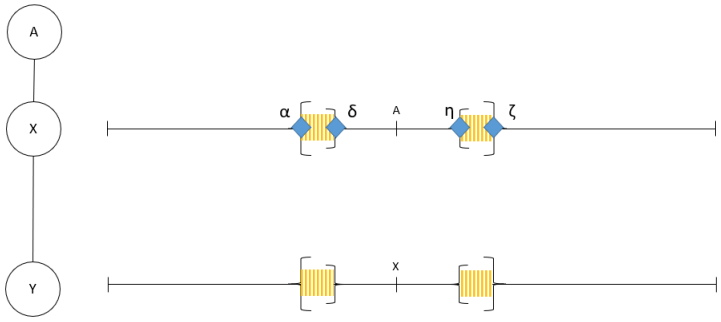


Figure 4.2: Illustration of the relation between neighbors. In this, the upper range shows X's possible RSI values in relation to node A, and the lower, Y's in relation to X.

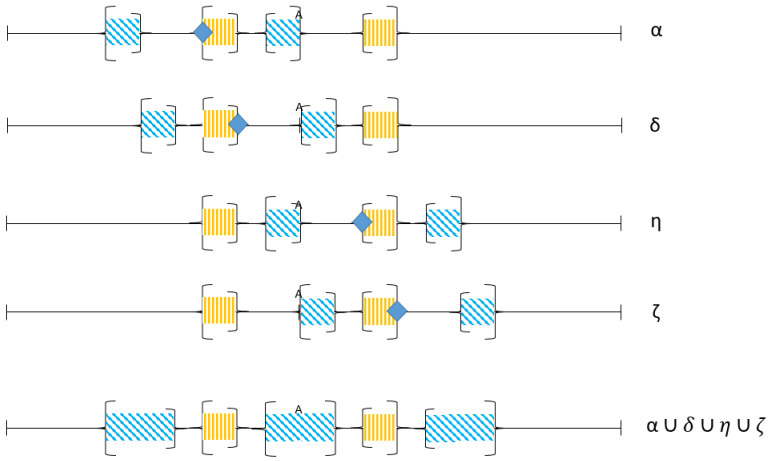


Figure 4.3: The possible mathematical relation between Y's and A's RSI obtained by adding Y's range to A's RSI in turn. This is observed by the blue left-inclined hatch. The lowest object is the union of the four previous ones.

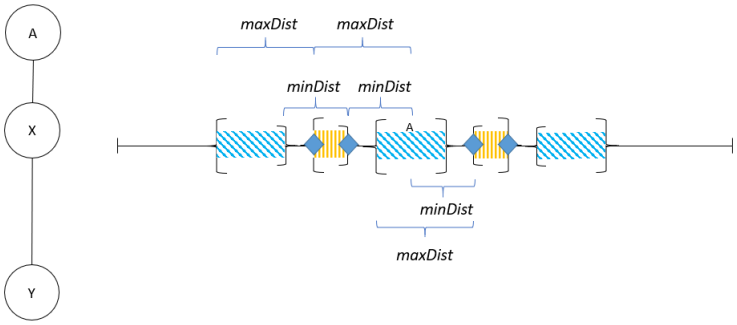


Figure 4.4: Close up of the lowest range of Figure 4.3. The mathematical relation between the center of the range and the blue left-inclined ranges can be observed.

Algorithm 3: Coloring the graph – decoder LI.

Input: Chromosome/vector $v \in [0,1]^n$ where n is the number of vertices.
Output: Total coloring cost.

```
1  Let γ to be a permutation of vertices induced by the non-decreasing order of corresponding keys
     in v;
2  Initialize RSI as zero-vector;
3  foreach i ∈ γ in the given order do
4      if i is first in order then
5          RSI_i ← OldRSI_i;
6      else
7          CandidateColor ← OldRSI_i;
8          Possible ← true;
9          foreach j adjacent to i and RSI_j ≠ 0 do
10             if (|RSI_j − CandidateColor| > MaxDist) or (|RSI_j − CandidateColor| < MinDist) then
11                 Possible ← false;
12         foreach j adjacent to i and RSI_j = 0 do
13             foreach k adjacent to j and RSI_k ≠ 0 do
14                 if |RSI_k − CandidateColor| > 2 · MaxDist then
15                     Possible ← false;
16                 if 2 · MinDist > MaxDist − MinDist then
17                     if 2 · MinDist > |RSI_k − CandidateColor| and
                         |RSI_k − CandidateColor| > MaxDist − MinDist then
18                         Possible ← false;
19         if Possible then
20             RSI_i ← CandidateColor;
21         else
22             CandidateColor ← minRSI;
23             while CandidateColor ≤ maxRSI do
24                 foreach j adjacent to i and RSI_j ≠ 0 do
25                     if (|RSI_j − CandidateColor| > MaxDist) or
                         (|RSI_j − CandidateColor| < MinDist) then
26                         Possible ← false;
27                 foreach j adjacent to i and RSI_j = 0 do
28                     foreach k adjacent to j and RSI_k ≠ 0 do
29                         if |RSI_k − CandidateColor| > 2 · MaxDist then
30                             Possible ← false;
31                         if 2 · MinDist > MaxDist − MinDist then
32                             if 2 · MinDist > |RSI_k − CandidateColor| and
                                 |RSI_k − CandidateColor| > MaxDist − MinDist then
33                                 Possible ← false;
34                 if Possible then
35                     RSI_i ← CandidateColor;
36                     break;
37                 else
38                     CandidateColor ← CandidateColor + 1;
39     if RSI_i = 0 then
40         RSI_i ← OldRSI_i;
41 ColoringCost ← CalculateCost(RSI);
42 return ColoringCost
```

4.2.5 Simple Coloring Decoder (SC)

The Simple Coloring Decoder also uses an n-sized chromosome, but instead of ordering the vertices, this decoder uses directly the value of each gene. The possible RSI values are normalized so that each gene correspond

Algorithm 4: Coloring the graph – decoder SC.

Input: Chromosome/vector $v \in [0,1]^n$ where n is the number of vertices.
Output: Total coloring cost.

```
1 Let τ to be the gene value of vertices induced by the value of corresponding
    keys in v;
2 Initialize RSI as zero-vector;
3 foreach i ∈ τ do
4     RSI_i ← τ_i · (maxRSI − minRSI) + minRSI;
5 ColoringCost ← CalculateCost(RSI);
6 return ColoringCost
```

directly to a RSI, which is then allocated to its vertex. This decoder is presented in Algorithm 4.

This decoder has the characteristic of generating many infeasible solutions, in comparison with its counterparts, due to the strong random component in its generation.

4.2.6 Ordered Restricted Coloring Decoder (OR)

The Ordered Restricted Coloring Decoder uses a 2n-sized chromosome, in which n refers to the number of vertices in the instance. In this case, genes between $[0, n-1]$ are used to extract an ordering of vertices and genes $[n, 2n-1]$ correspond to a position inside the set of allowed RSI of a vertex. These allowed RSI are obtained by the neighborhood relationships between colored and uncolored neighbors of a vertex, and they are updated each time a vertex is colored.

Algorithm 5 presents this approach. In Lines 5–14, either the vertex receives the color indicated by gene value, or, if the list of allowed RSI is empty, the color of its previous configuration. Then, as shown in Lines 15–28, the list of allowed RSI of neighbor vertices is updated.

4.2.7 Color Ordered by Degrees Decoder (KD)

Differing from the previous decoders, the Color Ordered by Degrees Decoder uses a $k+1$-sized chromosome, with k indicating the number of possible colors. This decoder gives an ordering of colors. The vertices are ordered by decreasing values of degrees. This is so that vertices with the potential to create more conflicts (i.e., more difficult vertices) are colored preferably before the others.

Algorithm 5: Coloring the graph – decoder OR.

Input: Chromosome/vector $v \in [0,1]^{2n}$ where n is the number of vertices.
Output: Total coloring cost.

```
1  Let γ to be a permutation of vertices induced by the non-decreasing order
     of corresponding [0, n − 1] keys in v;
2  Let τ to be the gene value of vertices induced by the value of corresponding
     [n, 2n − 1] keys in v;
3  Initialize RSI as zero-vector;
4  Initialize AllowedRSI with all values from minRSI to maxRSI for each vertex;
5  foreach i ∈ γ in the given order do
6      if i is first in order then
7          RSI_i ← OldRSI_i;
8      else
9          if AllowedRSI_i ≠ Empty then
10             position ← τ_i · size of AllowedRSI_i;
11             RSI_i ← AllowedRSI_{i,position}
12         else
13             RSI_i ← OldRSI_i;
14     AllowedRSI_i ← Empty;
15     foreach j adjacent to i do
16         if AllowedRSI_j ≠ Empty then
17             foreach k ∈ AllowedRSI_j do
18                 if |k − RSI_i| < MinDist or |k − RSI_i| > MaxDist then
19                     Remove k from AllowedRSI_j;
20         foreach l adjacent to j do
21             if AllowedRSI_l ≠ Empty then
22                 foreach k ∈ AllowedRSI_l do
23                     if 2 · MinDist ≤ MaxDist − MinDist then
24                         if |k − RSI_i > 2 · MaxDist| then
25                             Remove k from AllowedRSI_l;
26                     else
27                         if |k − RSI_i| > 2 · MaxDist or
                             ((|k − RSI_i| < 2 · MinDist) and
                             (|k − RSI_i| > MaxDist − MinDist)) then
28                             Remove k from AllowedRSI_l;
29 ColoringCost ← CalculateCost(RSI);
30 return ColoringCost
```

The Welsh-Powell [92] algorithm for graph coloring is similar to this decoder. This algorithm consists of the iterative coloring of non-adjacent vertices, whose degrees are non-increasing, with the same colors. Note that the Welsh-Powell algorithm is used for the classical GCP to find an upper bound for the chromatic number of graphs. Moreover, its strategy depends

heavily on a GCP-specific characteristic, that of differing values for adjacent vertices. However, this characteristic is not useful for generalizations of the GCP, such as the RSI allocation problem.

Algorithm 6 presents this decoder. By following the order of colors and the order of degrees, the decoder checks each vertex to verify the feasibility of using the color as in the previous configuration (Lines 4–13). Afterward, all still uncolored vertices have the other possible colors checked as possibilities, as show Lines 14–23. If a vertex is still uncolored, then it is assigned its old configuration, as presented in Lines 24–26.

4.2.8 Color and Vertex Ordered Decoder (KC)

Similar to the previous decoder, the Color and Vertex Ordered Decoder uses both orders of colors and vertices to obtain a coloring for a graph. However, instead of obtaining the order of vertex from the network configuration, this decoder obtains it from the chromosome. The first part of the chromosome $[0, k + 1]$ has the order of the colors, while the second part $[k + 2, k + n + 1]$ becomes the order of the vertices. Note that k indicates the number of colors, and n indicates the number of vertices in the instance.

Algorithm 7 presents this decoder. With the order of colors and the order of degrees, the decoder starts by checking, in order, each vertex with the color as the previous configuration (Lines 4–13). Afterward, all still uncolored vertices have the other possible colors checked as possibilities, as show Lines 14–23. If a vertex is still uncolored, then it is assigned its old configuration, as presented in Lines 24–26.

4.2.9 Shaking and reset

As the shaking procedure is correlated with chromosome representation, the different interpretations of the developed decoders must be considered in this approach. Thus, two types of shaking procedures were implemented in this work. Algorithm 8 brings the shaking procedure applied with decoders LD, LI, KD, KC, and the first n genes of OR. Algorithm 9 details the shaking procedure applied with decoders SC and the last n genes of OR.

Both reset and the shaking procedures are called if a given number of iterations has happened without any improvement in the best solution. The amount of iterations for reset is a multiple of the number of iterations until shaking, i.e., shaking always occurs first and more frequently than reset.

Algorithm 6: Coloring the graph – decoder KD.

Input: Chromosome/vector $v \in [0,1]^{k+1}$ where $k+1$ is the number of possible colors.
Output: Total coloring cost.

```
1  Let κ be a permutation of colors induced by the non-decreasing order of
     corresponding keys in v;
2  Let δ to be a permutation of vertices induced by the non-increasing order of
     corresponding degrees;
3  Initialize RSI as zero-vector;
4  foreach k ∈ κ in the given order do
5      foreach i ∈ δ in the given order do
6          if OldRSI_i = k and RSI_i = 0 then
7              Possible ← true;
8              foreach j adjacent to i do
9                  if RSI_j ≠ 0 then
10                     if |RSI_j − k| > MaxDist or |RSI_j − k < MinDist| then
11                         Possible ← false;
12             if Possible then
13                 RSI_i ← k;
14 foreach k ∈ κ in the given order do
15     foreach i ∈ δ in the given order do
16         if OldRSI_i ≠ k and RSI_i = 0 then
17             Possible ← true;
18             foreach j adjacent to i do
19                 if RSI_j ≠ 0 then
20                     if |RSI_j − k| > MaxDist or |RSI_j − k < MinDist| then
21                         Possible ← false;
22             if Possible then
23                 RSI_i ← k;
24 foreach i ∈ δ in the given order do
25     if RSI_i = 0 then
26         RSI_i ← OldRSI_i;
27 ColoringCost ← CalculateCost(RSI);
28 return ColoringCost
```

4.2.10 Correction Procedure

The corrective procedure only happens if there are conflicts, i.e., the coloring is illegal for a certain solution. This procedure was inspired by the one in Krumke et al. [63] and interprets the chromosome as a tree-like structure. The procedure visits the adjacent nodes of all already visited vertices and changes the coloring of these neighbors if the edge is in conflict. All vertices

Algorithm 7: Coloring the graph – decoder KC.

Input: Chromosome/vector $v \in [0,1]^{k+1+n}$ where $k+1$ is the number of possible colors and n, the number of vertices.
Output: Total coloring cost.

```
1  Let κ to be a permutation of colors induced by the non-decreasing order of
     corresponding [0, k+1] keys in v;
2  Let γ to be a permutation of vertices induced by the non-decreasing order
     of corresponding [k+2, k+1+N] keys in v;
3  Initialize RSI as zero-vector;
4  foreach k ∈ κ in the given order do
5      foreach i ∈ γ in the given order do
6          if OldRSI_i = k and RSI_i = 0 then
7              Possible ← true;
8              foreach j adjacent to i do
9                  if RSI_j ≠ 0 then
10                     if |RSI_j − k| > MaxDist or |RSI_j − k < MinDist| then
11                         Possible ← false;
12             if Possible then
13                 RSI_i ← k;
14 foreach k ∈ κ in the given order do
15     foreach i ∈ γ in the given order do
16         if OldRSI_i ≠ k and RSI_i = 0 then
17             Possible ← true;
18             foreach j adjacent to i do
19                 if RSI_j ≠ 0 then
20                     if |RSI_j − k| > MaxDist or |RSI_j − k < MinDist| then
21                         Possible ← false;
22             if Possible then
23                 RSI_i ← k;
24 foreach i ∈ γ in the given order do
25     if RSI_i = 0 then
26         RSI_i ← OldRSI_i;
27 ColoringCost ← CalculateCost(RSI);
28 return ColoringCost
```

are visited, and coming back to an already visited vertex is not allowed. The correction procedure ends when all vertices have been visited.

This procedure is effective in finding feasible solutions from infeasible ones, but it may still result in infeasible solutions.

Algorithm 8: Shaking for permutation genes.

```
foreach Chromosome ∈ Elite Set do
    NumberGenes ← ShakingStrenght · n · Random;
    for 0 ≤ i ≤ NumberGenes do
        target ← Random · n;
        Chromosome[target] ← Chromosome[target + 1];
        Chromosome[target + 1] ← Chromosome[target];
Reset all non-elite chromosomes;
```

Algorithm 9: Shaking for non-permutation genes.

```
foreach Chromosome ∈ Elite Set do
    NumberGenes ← ShakingStrenght · n · Random;
    for 0 ≤ i ≤ NumberGenes do
        target ← Random · n;
        Chromosome[target] ← 1 − Chromosome[target];
Reset all non-elite chromosomes;
```

4.2.11 Local Search

The local search procedure occurs at the end of every decoding and tries to find better solutions in the 1-color exchange solution space. The local search for both objectives can be based on either Best Improvement (BI) or First Improvement (FI) strategies. In addition, local search BI may be performed on only part of the solution space. This characteristic is defined by a percentile LS% of the available vertices in each instance.

For the minimizing changes objective, the local search procedure changes the new color of a vertex to its old coloring and then recalculates the related coloring cost. If a vertex already has its old configuration, then the procedure skips it.

The maximize minimal span objective has a different procedure. For each vertex, it identifies its smallest legal edge. Then, the adjacent vertex related to this edge has its color changed to the best available value, which would be the one that increases the most the edge value without violating the maximal possible difference of the instance. In the case of the vertex having no legal edges, then it is skipped by the procedure. This case is illustrated in Figure 4.6.

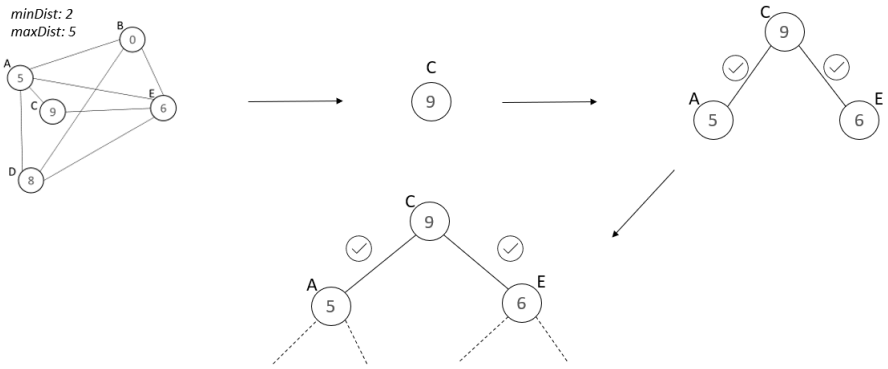


Figure 4.5: Representation of the Correction Procedure. In it, the procedure starts at vertex C, then visiting its neighbors A and E. As the edges (C,A) and (C,E) are legal, then the procedure goes to the not visited neighbors of both A and E in turn. It is of note that there is an edge (A,E) in the graph - and that it would be recolored if analyzed, but in this case, by starting with C, the procedure ignores that edge.

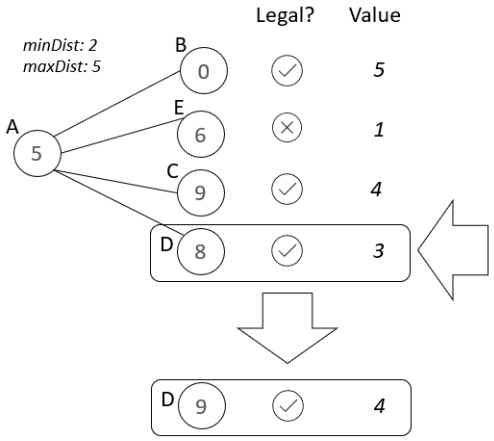


Figure 4.6: Representation of Local Search for the maximizing minimal span objective. In this, the smallest legal edge of vertex A corresponds to the edge (A,D). Thus, vertex D has its color changed, so that the difference is the maximum possible while considering instance maximal edge and possible color values. Note that edge (A,E) is the smallest of A, but it has a smaller value than the minimal possible and, thus, it is ignored by the procedure.

5
Results and Discussion

This chapter shows the results of this work. First, the characteristics of the used instances are detailed. Then, the results of the mathematical models proposed in Chapter 2 are explained. Afterward, the results of the BRKGA described in Chapter 4 are discussed.

5.1
Instances

The developed methods are applied on 139 novel instances based on real scenarios of a global telecommunications company. They are divided into two groups which indicate whether the RSI is in the long or short sequence, with 82 long sequence instances and 57 short sequence instances. Each instance has a number of vertices, the minimum and maximum distances, the old configuration of each node, and the existing edges. As for nomenclature, the instances are named by instance group, followed by the number of vertices, minimal distance, and maximal distance between the RSI of neighbors. For example, instance `long_n0030_r030_150` is a 30-vertex-sized instance of the long sequence group, with a minimal distance of 30 and maximal distance of 150. The instances are described in Appendix A.

Tables 5.1 and 5.2 summarize statistical characteristics of those instances. In Table 5.1, the number of vertices is explored, with "Mean" being the average number of vertices in the instances of the sequence, while "σ" is the standard deviation of the vertices, "Median" is the median number of vertices, and "Min" and "Max," the the minimal and maximal amount of vertices among the instances of the sequence. Table 5.2 uses the same statistical characteristics in relation to the density of the graphs. The density of a graph is defined as shown by Equation 5-1, in which $|E|$ is the number of edges and $|V|$, the number of vertices.

$$Density = \frac{2 \cdot |E|}{|V| \cdot (|V| - 1)} \tag{5-1}$$

As can be seen in Table 5.1, the instances of different groups clearly differ in size, with long sequence instances being larger and more distributed regarding the number of vertices in comparison with the short sequence ones.

On the other hand, Table 5.2 indicates that the sequences have similar characteristics, as they have close numbers in all metrics noted in the table.

Table 5.1: Comparison between instances of the short and long sequence, in vertex characteristics.

		Vertices				
Sequence	# inst.	Mean	σ	Median	Min	Max
Long	82	497.44	763.21	188	30	4125
Short	57	228.37	261.64	95	30	1348

Table 5.2: Comparison between instances of the short and long sequence, in density characteristics.

		Density				
Sequence	# inst.	Mean	σ	Median	Min	Max
Long	82	0.10	0.089	0.073	0.005	0.36
Short	57	0.12	0.092	0.093	0.012	0.36

As density is calculated by the relation between vertex and edge amounts, this indicates that the instances are similarly constructed. This may be related to the fact that, as the antennas are distributed on the earth's surface, they represent almost-plane graphs. They are not completely plane graphs, as the intersection of the different antennas is a characteristic of this problem and non-compatible with the definition of a plane graph.

Note that the amount of possible integer solutions for each instance is calculated by n^k (with n as vertex number and k the number of possible colors). This amount considers both feasible and infeasible solutions.

The amount of optimal solutions for each instance is influenced by the objective function analyzed, as the presence of symmetrical solutions cannot be ignored. The minimizing changes objective will have a number of optimal solutions related to the number of vertices with non-zero RSI as old configuration. Meanwhile, the maximizing minimal edge span objective has a dependency on the number of edges. This second objective has, thus, a higher amount of possible symmetrical optimal solutions in comparison to the first one.

5.2 Computational Environment and parameters

The experiments were conducted on a cluster of identical machines, each with processor Intel Xeon E5-2650 CPU 2.0 GHz (12 cores / 24 threads) and 128 GB of RAM, running CentOS Linux 6.9. For the mathematical models, solver IBM ILOG CPLEX 12.10 was used, while BRKGA-MP-IPR

was programmed on C++, both on a maximum of one hour, four threads, and 100 GB of RAM.

As BRKGA-MP-IPR has a higher amount of parameters for tuning than classical BRKGA, the application of design of experiments for tuning each parameter is too complex and time-consuming. Therefore, the *iterated racing* procedure was used to perform the parameter tuning. This method consists of three steps: the sampling of configurations from a particular distribution, the selection of the best ones using a statistic method, and the refining of the sampling distribution.

The `irace` package [93] was used for the tuning, with a budget of 3,000 experiments over 30 of the available instances, with 15 of each instance group. Each experiment, i.e. BRKGA-MP-IPR run, was limited to 1,800 seconds. To tune the BRKGA-MP-IPR parameters, the following ranges were used:

- population size $|\mathcal{P}| \in [100, 500]$;
- percentage of elite individuals $\mathcal{P}_e\% \in [0.1, 0.5]$;
- percentage of mutants introduced at each generation $\mathcal{P}_m\% \in [0.1, 0.5]$;
- number of elite individuals and total number of parents for mating procedure $(\pi_e, \pi_t) \in \{(1,3),(2,3),(2,6),(3,6),(4,6),(3,10),(5,10),(7,10)\}$;
- bias function for mating $\Phi(r) \in \{1/\log(r+1), r^{-1}, r^{-2}, r^{-3}, e^{-r}, 1/\pi_t\}$;
- number of independent populations $p \in [1, 3]$;
- minimum distance between chromosomes for path-relinking $md \in [0.0, 0.3]$;
- individual selection $sel \in \{\text{RE}, \text{BS}\}$, where RE indicates random elite individuals, and BS indicates best solutions from different populations;
- path percentage/size $ps\% \in [0.01, 1.00]$;
- interval for exchange individuals between populations $g \in [50, 500]$;
- shake interval without an improvement in the population's best solution (population stall) $I_s \in [20, 100]$;
- reset multiplier $R_m \in [1.1, 2.0]$;
- local search type $\text{LS} \in \{\text{NLS}, \text{FI}, \text{BI}\}$, in which NLS indicates no local search, FI means first improvement strategy, and BI means best improvement strategy;
- percentage of neighbors the local search $\text{LS}\% \in [0.1, 1.0]$, which indicates the percentage of the available vertices in each instance to be explored in the local search BI;

– correction procedure CR $\in \{TRUE, FALSE\}$, in which TRUE indicates that the correction procedure is performed inside the decoder, and FALSE, that it is never performed.

We use "permutation" path relinking type (*typ*) since almost all decoders are permutation-based. The maximum IPR time is given by the remaining time when it is called.

Tables 5.3 and 5.4 show the suggestions made by `irace` that were used in the experiments. Those suggestions are the first line of the tables shown in Appendix B. We picked the values of the first line of each table, rounded up to two digits for real values, and round to the next multiple of ten, in the case of integer values.

Table 5.3: Best parameter configurations suggested by `irace` for decoders of the minimizing change objective.

	BRKGA						IPR				Shaking		Procedures		
	$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
LD	772	0.39	0.14	3,10	r^{-2}	2	0.12	RE	0.86	242	180	3.34	NLS	0.46	FALSE
LI	963	0.36	0.14	3,10	r^{-2}	1	0.08	BS	0.41	223	123	3.88	NLS	0.11	TRUE
OR	277	0.42	0.19	3,6	e^{-r}	1	0.27	BS	0.45	475	164	1.67	NLS	0.49	TRUE
SC	1752	0.37	0.14	7,10	r^{-1}	1	0.06	BS	0.94	325	276	4.00	FI	0.62	FALSE
KD	609	0.41	0.20	3,10	r^{-2}	1	0.04	BS	0.07	229	198	2.84	FI	0.51	TRUE
KC	1647	0.40	0.12	3,10	r^{-2}	1	0.15	BS	0.41	176	153	2.91	NLS	0.38	FALSE

Table 5.4: Best parameter configurations suggested by `irace` for decoders of the maximizing minimal span objective.

	BRKGA						IPR				Shaking		Procedures		
	$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
LD	1010	0.38	0.10	5,10	r^{-2}	1	0.02	BS	0.30	107	270	2.93	FI	0.97	TRUE
LI	1010	0.38	0.10	5,10	r^{-2}	1	0.02	BS	0.30	107	270	2.93	FI	0.96	TRUE
OR	619	0.35	0.12	3,10	e^{-r}	1	0.06	BS	0.88	398	167	3.64	FI	0.39	TRUE
SC	4902	0.27	0.13	2,6	e^{-r}	2	0.02	RE	0.38	307	144	4.99	NLS	0.45	FALSE
KD	1268	0.13	0.31	1,3	e^{-r}	2	0.18	BS	0.45	463	128	1.31	FI	0.86	TRUE
KC	2304	0.34	0.23	5,10	e^{-r}	1	0.01	RE	0.33	441	123	1.65	FI	0.51	TRUE

5.3 Results of the proposed mathematical models

The results of the mathematical models are separated by objective and sequence and compared among themselves. The preliminary results were also shown in Londe et al. [29].

Recall that formulations NC and NS are composed of non-linear restrictions, while LC and LS are linear. NC and LC try to minimize the number of changes in RSI allocation, and thus their results are compared in Subsection 5.3.1, whilst NS and LS maximize the smallest edge span and are shown in Subsection 5.3.2. More detailed results can be seen in Appendix D.

To clarify what a expected result should be, Figure 5.1 portrays the optimal solution for instance `long_n0030_r030_150`. In this case, the objective was to minimize RSI changes.

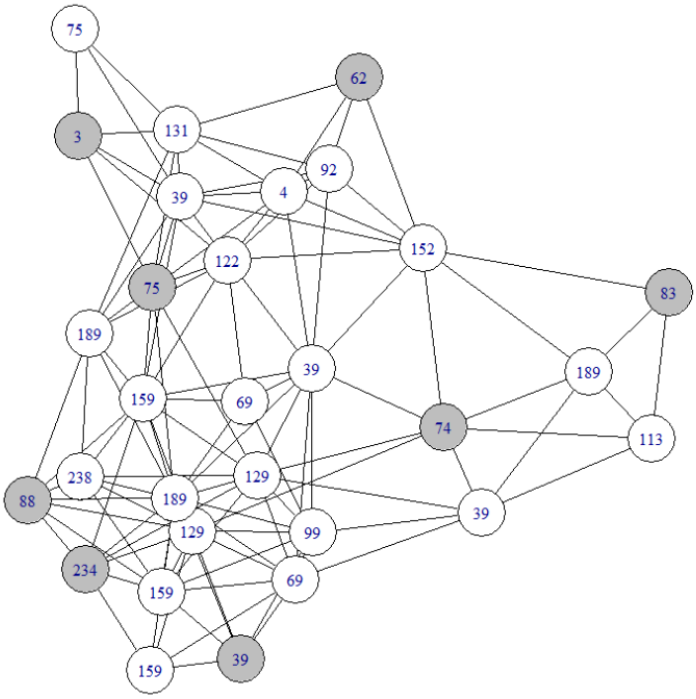


Figure 5.1: Optimal solution for instance `long_n0030_r030_150`. RSI values are indicated on each node, and a gray coloring implies that the configuration of the vertex was not changed.

According to our experiments, 40 instances (28%) had no feasible solution found by, at least, one of the formulations. Of these 40 instances, 17 (12%) we did not find at least one feasible solution with any of the proposed exact models, which is an indicator of the hardness of this problem. These instances with no solutions can be seen in Appendix D and are, thus, not considered in the comparison, with eight in the short sequence and nine in the long sequence.

5.3.1 Objective: minimize changes

In this subsection, the results for the mathematical models regarding the minimizing change objective are introduced. This subsection shows the results separately for the short and long sequence instance groups.

5.3.1.1 Short Sequence

For the 49 short sequence instances that have at least one feasible result found, a summary of the results can be seen in Table 5.5.

Table 5.5: Summary of results for the short sequence, in the minimizing change objective function. We consider 49 instances that had at least one feasible result.

	Unsolved		Optima		Non-optima	
	#	Avg. vertex	#	Avg. time(s)	#	Avg.GAP(%)
LC	0	NA	10	1900	39	22.98
NC	4	479.00	6	822	39	25.49

Comparing both models, it is observable that models LC and NC have quite similar performances among the different instances. One may note that the set of 39 non-optima instances differ between the two models – in fact, only 15 instances had an optimum found in this case. It can be seen that the LC model found feasible solutions for all 49 possible instances, while NC had a slightly worse performance, with only 45 instances. LC also had a higher amount of optima found but with approximately the double average time than that of NC - something surprising when considering the linear nature of LC.

In fact, model NC found the optimal solutions to the easiest instances and did not find the optima for the hardest ones due to the time constraint of one hour. Meanwhile, LC found more optimal solutions close to the time limit, thus increasing its average time for the optimal solutions.

Both models have similar performances for the non-optima solutions, with 39 instances and a close average GAP. This indicates that the solution quality of both, for the non-optima, is quite close, with LC solutions being slightly better.

5.3.1.2 Long Sequence

For the 73 short sequence instances that had at least one result feasible found, a summary of the results can be seen in Table 5.6.

Table 5.6: Summary of results for the long sequence, in the minimizing change objective function. We consider 73 instances that had at least one feasible result.

	Unsolved		Optima		Non-optima	
	#	Avg. vertex	#	Avg. time(s)	#	Avg.GAP(%)
LC	10	609.40	11	538	52	23.55
NC	10	465.00	5	362	58	34.40

The results indicate that model LC managed to find optimal results for more than the double of instances that NC. In fact, this behavior happened with higher average times – as did also for the short sequence ones. Again, this probably is a reflection of the higher times needed to find the solution for more difficult instances.

This result is corroborated by the fact that, for the same amount of unsolved instances, the average number of vertices for LC was considerably higher than NC. This measure is related to instance size and can be considered indicative of the instance difficulty.

Finally, model LC had a significantly smaller average GAP among the non-optimal solutions than NC. This is another indication of LC's higher solution quality in relation to its counterpart NC.

Also, note that the models performed similarly in the long sequence in comparison with the short sequence. The percentile of non-optimal instances is close for both LC (71% in the long sequence versus 79% in the short sequence) and NC (79% in both sequences).

The average time to reach optimal solutions is smaller for both models in the long sequence, in comparison with the short sequence. That is curious, as the long sequence appears to need more computational time and effort to find better solutions, and, yet, the optima were found in less average time. The need for more processing time in the long sequence mentioned would be related to the increase in both variable and restraint amounts, in comparison with the short sequence.

5.3.2 Objective: maximize minimum edge span

In this subsection, the results for the mathematical models regarding the maximizing minimal edge span objective are introduced. This subsection shows the results separately for the short and long sequence instance groups.

5.3.2.1 Short Sequence

Table 5.7 presents the results for the maximizing minimal span objective of the short sequence instances.

Table 5.7: Summary of results for the short sequence, in the maximizing minimal edge span objective function. We consider 49 instances that had at least one feasible result.

	Unsolved		Optima		Non-optima	
	#	Avg. vertex	#	Avg. time(s)	#	Avg.GAP(%)
LS	0	NA	4	957	45	274.16
NS	5	354.60	2	1524	42	265.66

LS and NS results show that those models have similar difficulties in finding optimal solutions for this objective and sequence. In fact, NS difficulties are of note, as it has a considerably worse performance in comparison with its counterpart NC for the other objective. Similar to its counterpart LC in the short sequence, LS also found feasible solutions for all 49 possible instances. LS found also higher amounts of optimal solutions, with a smaller average time, and similar GAP for the non-optimal solutions.

5.3.2.2 Long Sequence

Table 5.8 presents the results for the maximizing minimal span objective of the long sequence instances.

Table 5.8: Summary of results for the long sequence, in the maximizing minimal edge span objective function. We consider 73 instances that had at least one feasible result.

	Unsolved		Optima		Non-optima	
	#	Avg. vertex	#	Avg. time(s)	#	Avg.GAP(%)
LS	5	973.80	6	1124	62	587.97
NS	11	589.91	0	NA	62	1004.25

On this sequence and objective, the linear model LS appears to have a better performance when compared with its counterpart NS. In fact, not only was LS the only one that found optimal solutions in this case, but also a smaller by half average GAP for the non-optima, lower number of unsolved instances, and higher number of vertices on those unsolved instances, which indicates bigger and harder instances.

Comparing the results from both sequences on this objective, it can be seen that the long sequence, again, has harder instances for the models to solve. This is indicated by the higher percentage of unsolved instances and non-optimal instances, on both decoders, for the long sequence (93% unsolved and non-optima for LS, 100% for NS) in comparison with the short sequence (91% unsolved and non-optima for LS, 96% for NS).

5.4 BRKGA results

The BRKGA results are presented and discussed in this section. They are separated by objective and sequence and are all compared with existing CPLEX results.

It must be commented that the tuning procedure results shown in Section 5.2 indicated, in some cases, that the lack of a corrective operation in the decoding procedure is the best configuration. As such, those decoders have been explored for both cases, and the decoders without the correction procedure have a “no” following the decoder name. For example, “KCno” is the decoder KC without the correction procedure, while “KC” includes the procedure.

Also note that the analysis considers, in most part, the runs with feasible final results of the decoders. Because of it, we also perform an analysis of the feasible and infeasible solutions. The effect of the Implicit Path-Relinking (IPR) procedure was also considered and is detailed in Appendix C.

To compare the algorithms, the results are analyzed regarding the solution quality and computational effort. For solution quality, the classical Relative Percentage Deviation (RPD) and associated averages are used as defined in Andrade et al. [86], which are reproduced here. Let $\mathcal{I}$ be a set of instances. Let $\mathcal{A}$ be the set of algorithms, and assume that set $\mathcal{R}_A$ enumerates the independent runs with feasible final results for algorithm $A \in \mathcal{A}$. C_{ir}^{A} is defined as the best cost obtained by algorithm A in instance i on run r, and C_i^{best} as the best cost found across all algorithms for instance i. The Relative Percentage Deviation (RPD) from the best solution cost of instance i is defined as

$$RPD_{ir}^{A} = \frac{C_{ir}^{A} - C_i^{best}}{C_i^{best}} \times 100, \quad \forall A \in \mathcal{A},\ i \in \mathcal{I},\ \text{and } r \in \mathcal{R}_A. \tag{5-2}$$

5.4.1 Objective: minimize changes

The BRKGA results for the minimize changes objective are presented in the next subsections. Recall that this objective focuses on minimizing the changes in configuration of vertices in regards to an old configuration of the network.

5.4.1.1 Short Sequence

For the 57 short sequence instances, Table 5.9 indicates the percentage of infeasible and feasible final results obtained by each decoder, alongside the average amount of conflicting edges per infeasible instance.

In it, it is possible to observe that decoders LD, LDno, and LI are the only ones with less than 10% of runs indicating an infeasible final result. These decoders appear to quickly converge to feasible solutions in comparison with other decoders.

On the other side of the spectrum, decoders KC and KCno suffer from a very high number of infeasible solutions. This indicates that one hour is not enough for these algorithms to converge to feasible solutions in the great majority of the runs.

At last, decoder KCno finds more infeasible solutions than its counterpart KC, and a higher number of average conflicts. The same holds true between SCno and SC. LDno has also a higher number of average conflicts than LD, with a close percentage of feasible runs. These are effects of the correction procedure, which was absent in decoders KCno, SCno, and LDno.

Table 5.9: Comparison between infeasible and feasible runs for all decoders, in the short sequence. Consider a total of 570 runs for each decoder in this case.

Algorithm	Avg. conflicts	% Runs infeasible	% Runs feasible
KC	57.93	99.5	0.5
KCno	58.35	99.8	0.2
KD	16.84	84	16
LD	4.94	5	95
LDno	6.62	4	96
LI	4.87	8	92
OR	31.10	95	5
SC	9.29	79	21
SCno	9.87	82	18

From here on, we only consider the results for feasible solutions. For those, the boxplot of RPD for each algorithm is in Figure 5.2. The statistical results of each boxplot, meanwhile, are in Table 5.10.

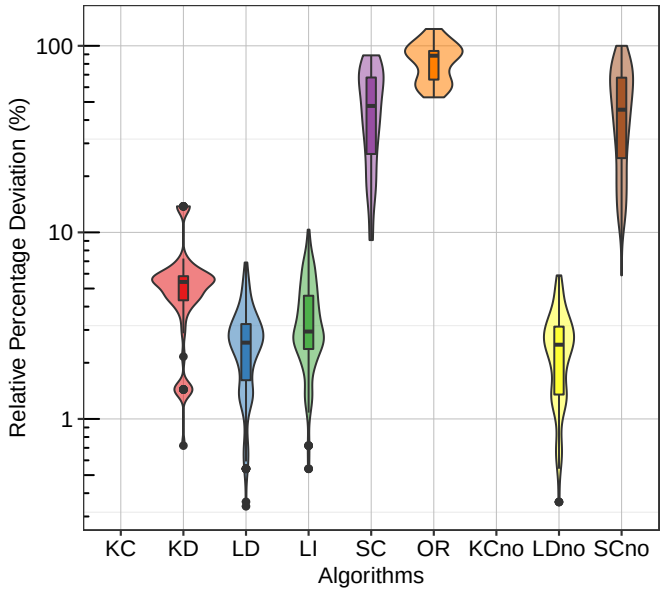


Figure 5.2: Distribution of relative percentage deviations for each algorithm considering instances of short sequence. Note that, since the data is log-transformed before plot, the shown statistics (median, quartiles, and others) are from the transformed data rather than the actual data. Also, note that, for the same reason, zero deviations are not shown, although the algorithms have reached them.

Note that the lack of distribution in the boxplot for decoders KC and KCno is correct. Both decoders have the least amount of runs with feasible final solutions, as it can be seen in Table 5.9. However, all these runs with

feasible solutions, in this case, found the optimum or best solution of the instances. Decoders KC and KCno, thus, have difficulties in converging to feasible solutions, but when they manage to converge they quickly reach the best available solutions. In this way, the RPD for those decoders stays at zero, and, thus, they have no observable figure in the boxplot.

When analyzing the statistical results of the distribution, it is of note that LD and LDno have a median of 0.00. This indicates that at least 50% of their runs result in the best solutions. They also have similar mean and standard deviation (σ), and the lowest values of maximal RPD. Decoder LI also deserves a mention, as it ranks third in these characteristics if KC and KCno cases are ignored. In this case, decoders LD, LDno, and LI also have the highest percentage of feasible runs, and KC and KCno, the lowest.

Finally, the Wilcoxon rank test was made to check possible differences between distributions of RPD. This test indicated that decoder KD has a consistently better performance than the others, while decoders LD, LDno, and LI have a good performance overall. Decoders SCno and SC, LDno and LD, and KCno and KC do not present statistical differences.

Table 5.10: Comparison between decoders, considering RDP statistical criteria.

Decoder	Median	Mean	σ	Max
KC	0.00	0.00	0.00	0.00
KD	4.32	3.62	3.51	13.79
LD	0.00	1.18	1.55	6.90
LI	1.13	1.85	2.21	10.34
SC	47.62	47.42	23.63	88.89
OR	88.53	83.69	19.32	123.08
LDno	0.00	1.09	1.443	5.88
KCno	0.00	0.00	0.00	0.00
SCno	45.46	47.35	24.90	100.00

Table 5.11: Algorithm's performance for instances of short sequence, in the minimizing changes objective. Note that one instance did not have any feasible solutions in any algorithm, and thus it was excluded from this analysis.

Algorithm	Known Optima (15 instances)					Unknown Optima (41 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	σ	# Best	% Best	% Run	%	σ
KC	1	6.67	100.00	—	—	0	0.00	—	—	—
KCno	1	6.67	100.00	—	—	0	0.00	—	—	—
KD	2	13.33	100.00	3.27	1.65	1	2.44	13.89	5.38	2.96
LD	12	80.00	62.42	3.27	1.65	30	73.17	36.92	2.45	1.12
LDno	11	73.33	61.74	3.17	1.60	32	78.05	37.53	2.21	1.04
LI	9	60.00	59.59	5.90	1.72	20	48.78	29.10	3.30	1.68
OR	0	0.00	0.00	86.85	19.04	0	0.00	0.00	64.71	4.43
SC	0	0.00	0.00	38.13	21.88	0	0.00	0.00	61.25	19.07
SCno	0	0.00	0.00	39.33	21.91	0	0.00	0.00	63.85	22.69

Table 5.11 shows the algorithm performance for the instances of the short sequence group. It is of note that instance `short_n0463_r015_139` did not have any feasible solutions on both CPLEX and BRKGA and, thus, was not considered in this analysis. As this table confirms, decoders LD, LDno, and LI have the best performance among all decoders, considering the used criteria. They are also the decoders with the best performance in the previous analysis.

Note that the addition of the correction procedure does not appear to have a powerful effect, in this case at least, on the performance of the decoders. The results from pairs LD and LDno, KC and KCno, and SC and SCno do not differ substantially among themselves.

At last, Figure 5.3 presents the running time for each algorithm, alongside the cumulative probability of finding either best or optimal solutions. It is possible to see that all decoders have better performances than CPLEX, with decoders LD, LDno, and LI being the best. However, in the 200 seconds mark, decoder KD appears to present the best performance, when ignoring the KC and KCno results. Also, note that decoders KC and KCno have a very quick performance - they reach the best solutions much faster than the others, but also for much fewer instances and number of runs.

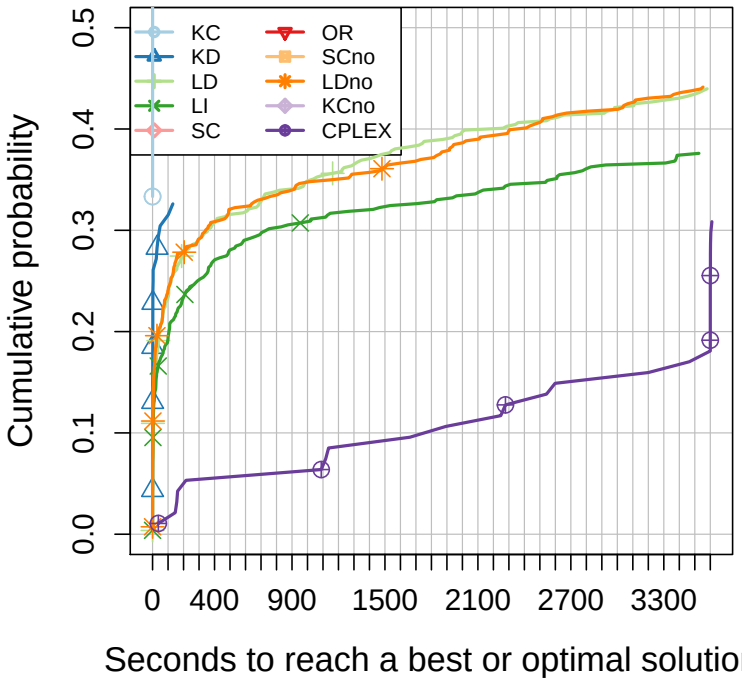


Figure 5.3: Running time empirical distributions to the best or optimal solution values for short sequence instances. The identification marks correspond to 1% of the points plotted for each algorithm.

5.4.1.2 Long Sequence

For the 82 long sequence instances, Table 5.12 indicates the percentage of infeasible and feasible final results obtained by each decoder, alongside the average amount of conflicting edges per infeasible instance.

One can note the similarities between this table and Table 5.9. The decoders appear to have similar behaviors for both long and short sequences in this objective, but it is worth highlighting some differences.

The amount of average conflicting edges is lower for all decoders except SC and OR. This is surprising, as the long sequence instances have higher numbers of nodes and colors, thus needing bigger chromosomes and higher processing times - logically, the decoders should have less time to converge.

Among the best decoders are, again, LD, LDno, and LI. They are the only ones with less than 10% of runs indicating an infeasible final result, which confirms their tendency of quickly converging to feasible solutions. In addition, KC and KCno trends of higher infeasible final solutions are also confirmed, though in smaller proportions.

For the long sequence, the pairs KC and KCno, and LD and LDno behave similarly, as they did in the short sequence. However, SC and SCno have a very different average number of conflicts. This may be an effect of SCno's lack of

Table 5.12: Comparison between infeasible and feasible runs for all decoders, in the long sequence. Consider a total of 820 runs for each decoder in this case.

Algorithm	Avg. conflicts	% Runs infeasible	% Runs feasible
KC	29.59	87	13
KCno	27.08	86	14
KD	5.32	24	76
LD	1.00	2	98
LDno	1.33	2	98
LI	1.06	4	96
OR	208.69	99	1
SC	58.92	76	24
SCno	7.96	79	21

a corrective procedure, as it lets the algorithm run more generations in the given time, thus letting it converge more frequently to feasible solutions.

Figure 5.4 depicts the RPD boxplots for runs with feasible solutions only. The statistical results of each boxplot, meanwhile, are in Table 5.13. Differently from the short sequence results, decoders KC and KCno have only a regular performance for the long sequence. This can be derived from the fact that those decoders depend heavily on the number of possible colors, and thus with a higher number of colors, their performance suffers. Another possibility is the higher percentage of feasible runs – on the short sequence case, all 0.2% of feasible runs found optima, which did not happen for the approximately 13% of long sequence. At the same time, they have median values smaller than one, indicating that most feasible runs have good RPD results.

Considering the statistical results of the distributions, decoders LD and LDno have, again, a median of 0.00. This result indicates that at least 50% of their results found the best solutions. They also have similar means and standard deviation (σ), and the lowest values of maximal RPD. The LI decoder also deserves a mention, as it ranks third in these characteristics. Decoders LD, LDno, and LI also had the highest percentage of feasible runs, in this case. Decoder LD has a better performance than LDno in all criteria - the addition of the correction procedure used in LD appears to lead to better results;

A comparison with the Wilcoxon rank test was also performed. In it, decoders LD and LDno have similar deviations, and they are shown to be the best decoders in this case. Meanwhile, KC is indicated as better than LI – possibly related to the median values, and KD is better than KCno, while KCno and KC have the same performance.

Table 5.14 shows the algorithm performance for the instances of the long sequence. In this case, all instances had a feasible solution in, at least, one

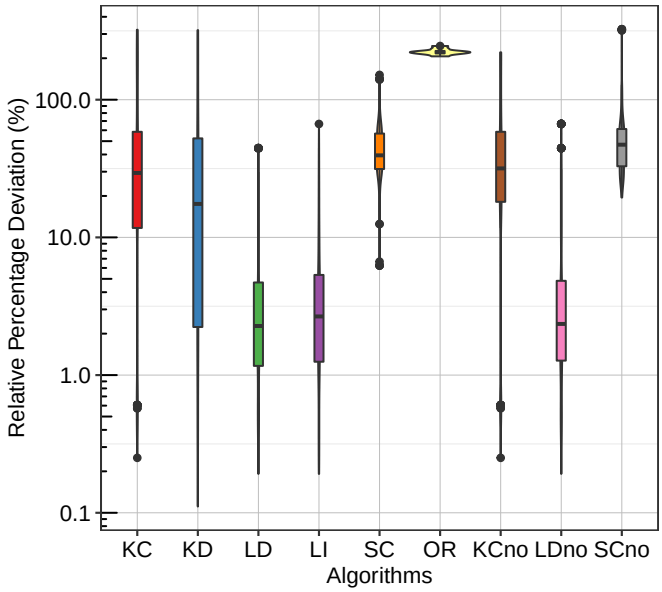


Figure 5.4: Distribution of relative percentage deviations for each algorithm considering instances of long sequence. Note that, since the data is log-transformed before plot, the shown statistics (median, quartiles, and others) are from the transformed data rather than the actual data. Also, note that, for the same reason, zero deviations are not shown, although the algorithms have reached them.

Table 5.13: Comparison between decoders, considering RDP statistical criteria.

Decoder	Median	Mean	σ	Max
KC	0.61	37.38	71.27	322.22
KD	0.98	31.10	72.64	320.00
LD	0.00	2.57	6.69	44.44
LI	0.56	3.18	7.19	66.67
SC	38.12	42.78	26.90	151.43
OR	222.10	223.20	11.50	220.78
LDno	0.00	3.68	10.23	66.67
KCno	11.71	38.08	62.92	220.78
SCno	45.86	54.61	52.84	325.00

Table 5.14: Algorithm's performance on instances of long sequence, in the minimizing changes objective.

Algorithm	Known Optima (21 instances)					Unknown Optima (61 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	σ	# Best	% Best	% Run	%	σ
KC	0	0.00	0.00	23.04	56.43	3	4.92	27.54	62.05	84.09
KCno	0	0.00	0.00	20.56	25.05	4	6.56	25.30	61.03	77.15
KD	6	28.57	26.47	91.20	121.83	22	36.07	32.30	26.70	50.59
LD	12	57.14	38.27	6.82	11.99	39	63.93	39.67	3.95	6.15
LDno	13	61.90	40.00	13.99	21.01	41	67.21	40.37	3.83	6.17
LI	11	52.38	25.26	7.78	12.05	28	45.90	30.35	5.09	6.55
OR	0	0.00	—	—	—	0	0.00	0.00	223.23	11.50
SC	1	4.76	3.45	41.82	20.19	0	0.00	0.00	50.89	27.70
SCno	0	0.00	0.00	75.68	76.85	0	0.00	0.00	49.28	24.18

algorithm or mathematical model. Again, decoders LD, LDno, and LI have a better performance than all other decoders. it is worth noting the performance of KD, as it has also a regular performance for instances without an optimum.

The correction procedure appears to not have any significant effect on algorithm performance. While decoder SC has a slightly better performance than its counterpart for known optima instances, both pairs LD and LDno, and KC and KCno have very similar performances according to the observed criteria for all instances.

Finally, Figure 5.5 shows the cumulative probability of finding the best or optimal solutions for each algorithm. In this case, all decoders have a better performance than CPLEX except SC, whose performance is lacking in this case. Again, decoders LD and LDno had the best performance, with KD and LI in third and fourth places, respectively. LDno had a slightly better performance than LD at all moments. This can be justified as an effect of the addition of the correction procedure, which makes LD slower. Thus, LD would need more time to reach the same solution as LDno.

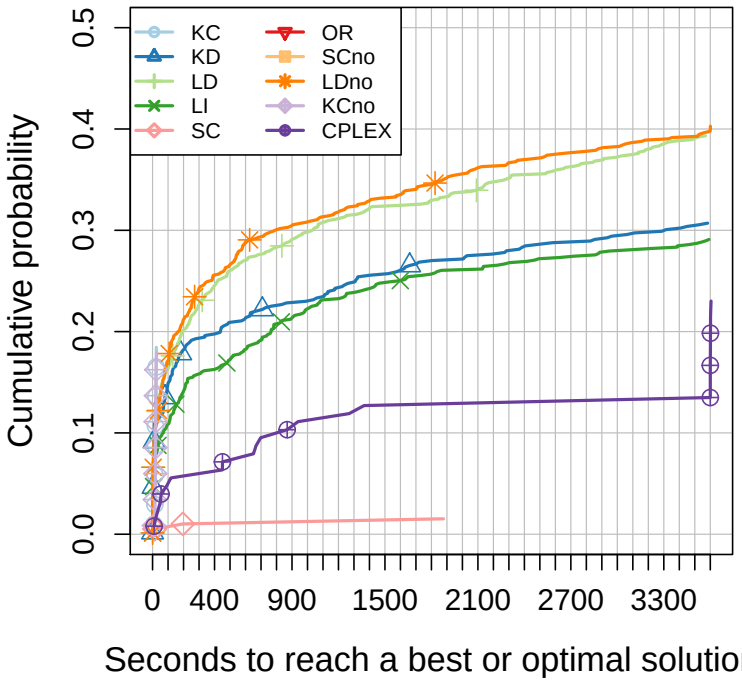


Figure 5.5: Running time empirical distributions to the best or optimal solution values for long sequence instances. The identification marks correspond to 1% of the points plotted for each algorithm.

5.4.2 Objective: maximize minimum edge span

The BRKGA results for the maximize minimal edge span objective are presented in the next subsections. Recall that this objective focuses on maximizing the minimal difference between the RSI of neighbor vertices.

5.4.2.1 Short Sequence

For the 57 short sequence instances, Table 5.15 indicates the percentage of infeasible and feasible final results each decoder had, alongside the average amount of conflicting edges per infeasible instance.

The numbers show that, except for decoders LD and LI, all others had a higher percentage of infeasible runs than feasible. In fact, this can be seen as an effect of the difficulty of this objective – the exact models had also more difficulty in finding good solutions for this objective in comparison with the other one. On the other hand, decoders LD and LI had, again, the best numbers of average conflicts and percentage of feasible runs. The same behavior was observed regarding the minimizing changes objective function.

In contrast, decoders KC and OR had very high amounts of no-feasible runs ($\geq$95%). They also had a higher amount of average conflicts in those

Table 5.15: Comparison between infeasible and feasible runs for all decoders, in the short sequence. Consider a total of 570 runs for each decoder in this case.

Algorithm	Avg. conflicts	% Runs infeasible	% Runs feasible
KC	55.83	99.8	0.2
KD	16.75	86	14
LD	2.04	45	54
LI	2.25	47	53
OR	22.74	95	5
SC	9.89	80	20
SCno	9.56	84	16

infeasible solutions, in comparison with the other decoders.

Considering the pair SC and SCno, the correction procedure does not appear to have a significant effect in obtaining more feasible solutions, as shown by the close values in both average conflicting edges and percentage of feasible runs.

From now on, the analysis will only consider the runs with feasible final results. For those, the boxplot of RPD for each algorithm is in Figure 5.6. The statistical results of each boxplot, meanwhile, are in Table 5.16.

Again, decoder KC had a similar performance to the other short sequence case, finding the optimum or best solutions for all runs with feasible final results. As such, it does not have a visible performance in the boxplot of Figure 5.6. One must note that, again, KC only had 0.2% of all runs analyzed and, thus, its results tend to be somewhat skewed.

As Table 5.16 shows, all decoders except OR had a median of 0.00, which indicates that at least half of their feasible runs found the best available results for the short sequence instances. Also, again excluding OR, all decoders had a mean RPD below 10%. With these criteria, LD, LI, and KD had the best performances, with KD having the best results among them.

Comparison with the Wilcoxon rank test is in Table 5.17, in which a value of less than 0.05 indicates that the column algorithm has significantly better deviations than the row algorithm. As expected, decoder OR was indicated as worse than all others. Meanwhile, decoder KD had, again, better deviations than all other decoders except KC, whose showing with almost all zeros would have skewed the results.

Table 5.18 shows the algorithm performance for the instances of the short sequence. In this case, all instances had one feasible solution in, at least, one algorithm. For this objective and sequence, decoders KC and KD appear to have a bad performance, while LD, LI, SC, and SCno have the best results in

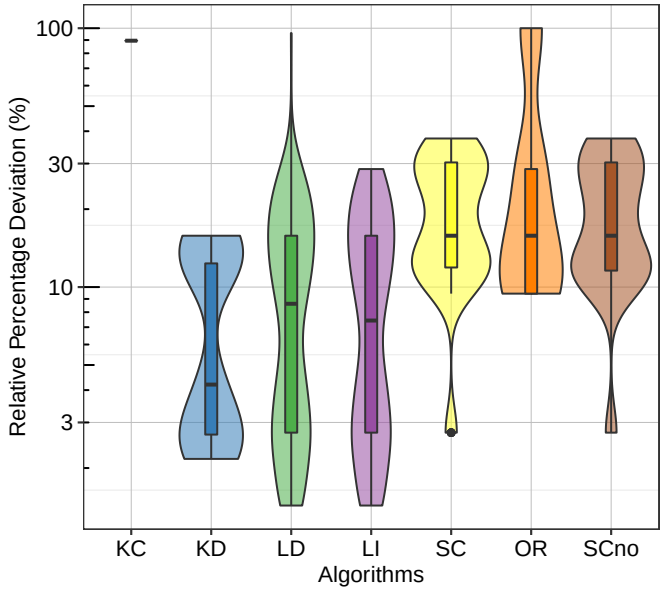


Figure 5.6: Distribution of relative percentage deviations for each algorithm considering instances of short sequence. Note that, since the data is log-transformed before plot, the shown statistics (median, quartiles, and others) are from the transformed data rather than the actual data. Also, note that, for the same reason, zero deviations are not shown, although the algorithms have reached them.

Table 5.16: Comparison between decoders, considering RDP statistical criteria.

Decoder	Median	Mean	σ	Max
KC	0.00	0.66	7.67	89.47
KD	0.00	1.70	4.14	15.79
LD	0.00	4.17	7.92	95.74
LI	0.00	4.01	7.06	28.57
SC	0.00	7.66	11.49	37.50
OR	10.06	22.42	31.59	100.00
SCno	0.00	7.16	211.23	37.50

Table 5.17: p-values from pairwise Wilcoxon rank-sum test (with Bonferroni p-value correction) of RPDs.

	KC	KD	LD	LI	SC	OR
KD	$\ll 0.05$	—	—	—	—	—
LD	$\ll 0.05$	$\ll 0.05$	—	—	—	—
LI	$\ll 0.05$	$\ll 0.05$	1.00	—	—	—
SC	$\ll 0.05$	$\ll 0.05$	1.00	1.00	—	—
OR	$\ll 0.05$	$\ll 0.05$	$\ll 0.05$	$\ll 0.05$	$\ll 0.05$	—
SCno	$\ll 0.05$	$\ll 0.05$	1.00	1.00	1.00	$\ll 0.05$

Table 5.18: Algorithm's performance on instances of short sequence, in the maximizing minimal span objective.

Algorithm	Known Optima (5 instances)					Unknown Optima (51 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	σ	# Best	% Best	% Run	%	σ
KC	0	0.00	—	—	—	0	0.00	0.00	672.73	—
KD	0	0.00	—	—	—	9	17.65	100.00	—	—
LD	2	40.00	100.00	—	—	31	60.78	99.66	718.18	—
LI	2	40.00	100.00	—	—	29	56.86	100.00	—	—
OR	1	20.00	100.00	—	—	5	9.80	73.91	637.47	309.03
SC	3	60.00	65.52	6.00	2.11	10	19.61	91.57	6.67	0.00
SCno	2	40.00	55.00	5.00	0.00	12	23.53	90.41	6.67	0.00

comparison with others.

The correction procedure has an improving effect on SC performance when comparing with SCno performance. SCno had a higher number of best solutions for instances with unknown optima, and a lower percentage of proportional difference for the ones with known optima, but in all other criteria SC had better results.

Figure 5.7 presents the cumulative probability of finding the best or optimal solutions in a certain time for each algorithm. In this case, all decoders had a better performance than CPLEX. The decoders also have, in general, good performances, reaching >80% chance in less than one hour. In special, decoders LD, LI, and KD have very good performances, eventually reaching 100% probability.

On the other hand, decoders SC and SCno have similar performance, with SC being consistently better than its counterpart. Meanwhile, OR is the worst among all decoders, but also manages to reach almost 80% cumulative percentage in one hour.

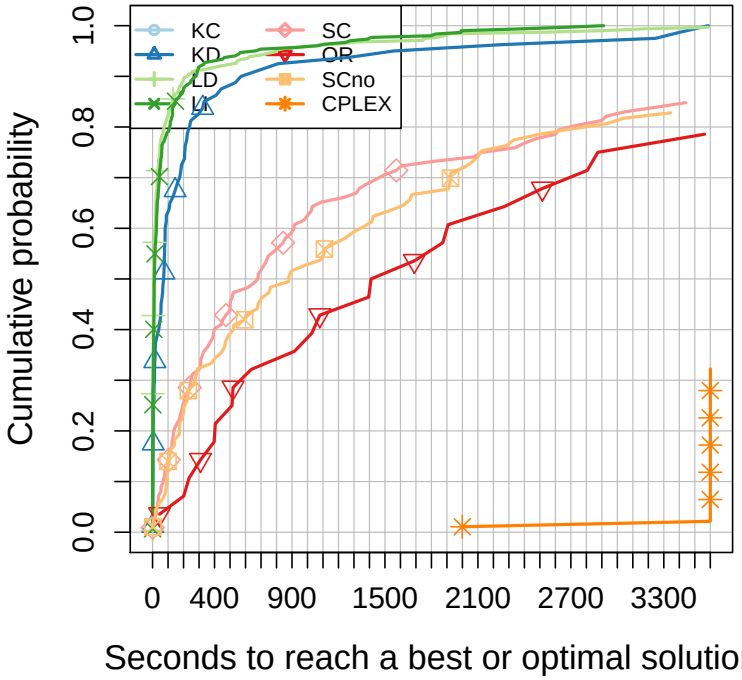


Figure 5.7: Running time empirical distributions to the best or optimal solution values for short sequence instances. The identification marks correspond to 1% of the points plotted for each algorithm.

5.4.2.2 Long Sequence

For the 82 long sequence instances, Table 5.19 indicates the percentage of infeasible and feasible final results each decoder had, alongside the average amount of conflicting edges per infeasible instance.

In this case, all decoders had a higher percentage of infeasible runs than feasible. Although it is not the highest in total among all sequences and objectives, it is still a high percentage that indicates the difficulty to approach this objective. In this case, decoders LD and LI had the best performances, with KD in a close third place.

In this case, decoder OR had the worst performance among all decoders, with 99% of runs with infeasible final results. It also has a very high amount of average conflicting edges – by far, the highest among all decoders, objectives, and sequences.

While comparing decoders SC and SCno, one can observe a similar phenomenon as in the long sequence of minimizing changes objective function. That is, decoder SC presents a much higher number of conflicts than SCno.

From now on, the analysis will only consider the runs with feasible final results. For those, the boxplot of RPD for each algorithm is in Figure 5.8. The statistical results of each boxplot, meanwhile, are in Table 5.20. In this

Table 5.19: Comparison between infeasible and feasible runs for all decoders, in the long sequence. Consider a total of 820 runs for each decoder in this case.

Algorithm	Avg. conflicts	% Runs infeasible	% Runs feasible
KC	30.82	84	16
KD	3.94	64	36
LD	1.24	59	41
LI	1.52	58	42
OR	261.15	99	1
SC	61.75	78	22
SCno	7.19	80	20

Table 5.20: Comparison between decoders, considering RDP statistical criteria.

Decoder	Median	Mean	σ	Max
KC	5.73	18.37	31.43	100.00
KD	3.95	15.50	31.32	100.00
LD	0.00	9.30	24.49	100.00
LI	0.00	8.69	23.14	100.00
SC	5.73	9.58	17.73	100.00
OR	0.00	4.34	7.11	17.96
SCno	5.69	8.02	16.09	100.00

case, decoders LD and LI appear to have the best performances, while OR is the most concentrated of all decoders. Note that, although its results are concentrated above the results of the other decoders, decoder OR does have the smallest maximum RPD.

Truly, decoder OR has the best statistical RPD results, as shown in Table 5.20. LD, LI, and OR had the median of 0.00, but OR had the smallest maximum, smallest mean, and smallest deviation. Also, it is of note that the addition of a correction procedure did not cause a significant change in the statistics of SC and SCno.

Comparison with the Wilcoxon rank test is in Table 5.21, in which a value of less than 0.05 indicates that the column algorithm has significantly better deviations than the row algorithm. As it can be seen, OR, LD, and LI have similar deviations among themselves.

Table 5.22 shows the algorithm performance for the instances of the short sequence. In this case, instances `long_n0259_r030_510` and `long_n0323_r040_640` did not have any feasible result on any algorithm or mathematical model and were, thus, excluded from this analysis.

In this case, decoders LD and LI are, again, the best decoders for both known and unknown optima. On the other hand, OR had bad performances

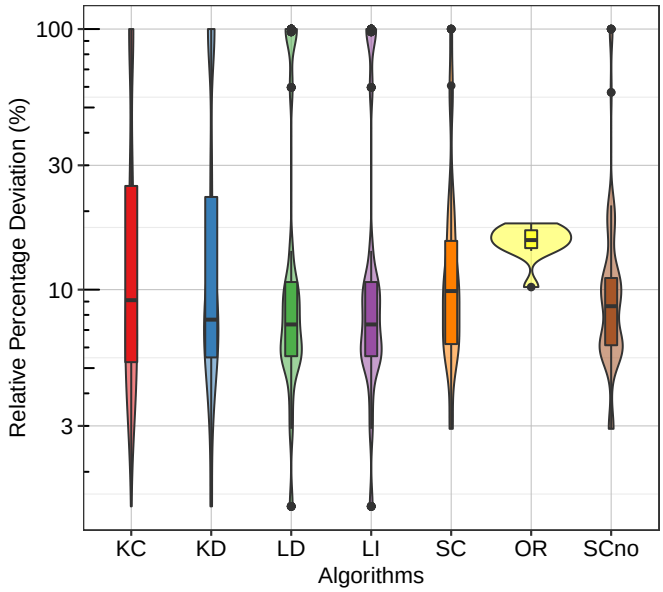


Figure 5.8: Distribution of relative percentage deviations for each algorithm considering instances of long sequence. Note that, since the data is log-transformed before plot, the shown statistics (median, quartiles, and others) are from the transformed data rather than the actual data. Also, note that, for the same reason, zero deviations are not shown, although the algorithms have reached them.

Table 5.21: Pvalues from pairwise Wilcoxon rank-sum test (with Bonferroni Pvalue correction) of RPDs.

	KC	KD	LD	LI	SC	OR
KD	0.02	—	—	—	—	—
LD	$\ll 0.05$	$\ll 0.05$	—	—	—	—
LI	$\ll 0.05$	$\ll 0.05$	1.00	—	—	—
SC	0.11	1.00	$\ll 0.05$	$\ll 0.05$	—	—
OR	0.003	0.44	1.00	1.00	0.38	—
SCno	1.00	$\ll 0.05$	$\ll 0.05$	1.00	0.60	$\ll 0.05$

Table 5.22: Algorithm's performance on instances of long sequence, in the maximizing minimal span objective. Note that two instances did not have any feasible solutions in any algorithm, and thus they were excluded from this analysis.

Algorithm	Known Optima (10 instances)					Unknown Optima (70 instances)				
	Optima			Prop. diff.		Best			Prop. diff.	
	# Opt	% Opt	% Run	%	σ	# Best	% Best	% Run	%	σ
KC	2	20.00	55.00	3.33	0.00	6	8.57	39.13	187.99	431.87
KD	3	30.00	75.00	6.00	1.41	11	15.71	35.43	149.12	299.50
LD	5	50.00	98.04	50.00	—	28	40.00	89.79	533.53	289.16
LI	5	50.00	100.00	—	—	29	41.43	91.47	478.17	248.88
OR	0	0.00	—	—	—	2	2.86	27.27	16.25	8.02
SC	3	30.00	100.00	—	—	22	31.43	69.14	20.91	48.70
SCno	2	20.00	100.00	—	—	18	25.71	61.74	11.32	28.06

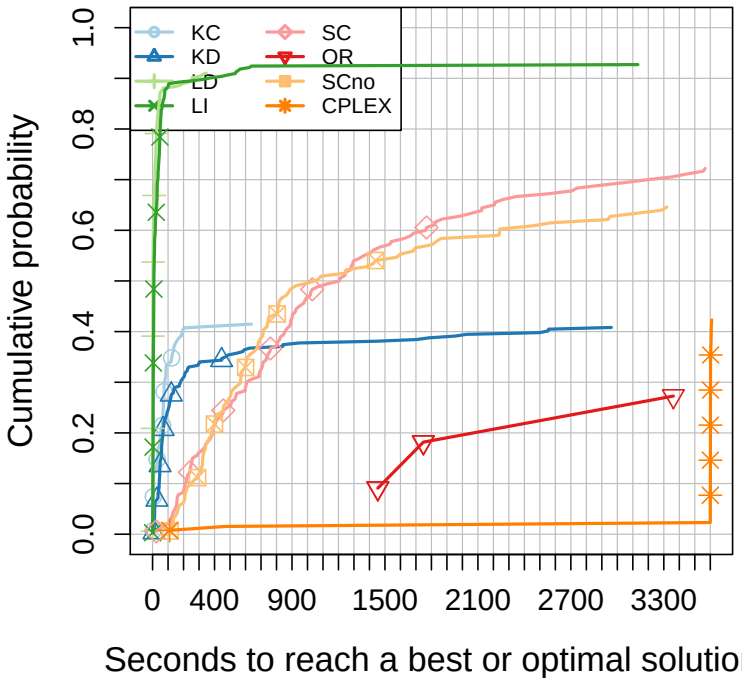


Figure 5.9: Running time empirical distributions to the best or optimal solution values for long sequence instances. The identification marks correspond to 1% of the points plotted for each algorithm.

on both. This contrast regarding the RPD distribution statistics may be attributed both to the very high number of runs without feasible final solutions (only 1% of runs were feasible), and the fact that it has the absolute lowest deviation among unknown optima instances. Consequently, it does not reach optimal or best results often, but has a high amount of good solutions, thus ending with good statistics of the RPD.

About the effect of the corrective procedure, decoders SC and SCno have good performances for both known and unknown optima instances, and they are, again, very similar.

Finally, Figure 5.9 presents the cumulative probability of finding the best or optimal solutions in a certain time for each algorithm. For the long sequence instances, all decoders had a better performance than CPLEX before the one-hour mark. Close to it, both OR and KD have lower cumulative percentages. As for the best decoders, LD and LI have very close performances, being the best decoders in this case and reaching more than 90% cumulative percentage in less than 900 seconds.

Considering the effects of the correction procedure, both SC and SCno have close performances, but in this case decoder SC reaches better cumulative percentages after the 1,400 seconds mark and ends with an almost 10% higher cumulative probability after one hour.

5.5 General comments

Regarding the mathematical models, one may note that the linear models had a smaller number of instances with no results for both types of sequences and objectives functions, in comparison with the non-linear models. Furthermore, those instances solved by the linear models have higher numbers of vertices, which indicates more difficulty and a greater number of variables. In addition, for all models, the number of unsolved long sequence instances is higher than the amount for the short sequence for both objective functions. Again, this is indicative of the impact of more variables and restrictions for the long sequence in comparison with the short sequence, pointing out to a higher level of complexity.

Regarding the BRKGA decoders, first, in relation to the number of conflicts, the behavior of the decoders in relation to long and short sequences is similar in both objectives. In both cases, the short sequence has a higher percentage of infeasible runs, but a much lower amount of conflicts in those infeasible runs.

Meanwhile, for a same objective, the amounts of runs feasible and infeasible are close for instances on both sized-sequence groups. This indicates that the objective function is more influential in the hardness of converging to feasible solutions than the instances' characteristics.

In general, the change from short to long sequence means an increase in conflict amount for decoders OR and SC, and a reduction in all other decoders. This is true for both objective functions.

About the corrective procedure, it appears to not have a significant impact on solution quality – except for decoder SC in the long sequence. In both objectives, decoder SCno has a much smaller number of average conflicts than its counterpart. This can be explained by the fact that decoder SC procedure has a severe increase in time requirements because of the correction procedure. In this way, the SCno manages to run more generations and thus converge to better solutions.

Now, going to special remarks about specific decoders, decoder KC has a strange performance in both cases. It has a very high percentage of infeasible solutions, and the feasible solutions tend to reach the optimal costs. This is especially true for the short sequence cases, as all or almost all feasible solutions are optimal. That can be caused by one of two alternatives: first, the decoder will only converge to feasible solutions by happenstance, and always will hit the best solution. This option could also explain why the time to reach optimal solutions, in this decoder (as shown in the performance profile plots) is always

low. Another reason may be that this decoder needs more time to converge to feasible solutions when it does not in the starting minutes. That could explain the high amount of infeasible solutions and the good performance in the feasible ones, and also be related to the fact that decoder KC uses a $n + k + 1$-sized chromosome, which would be negatively affected by any increase in size and possible color amount of the instances.

Meanwhile, the performance of decoder OR is poor in all four cases. Particularly in the long sequence-maximizing minimal span case, decoder OR has surprisingly good statistics, comparable to decoder LD, but still holds its place as the worst decoder.

Finally, among the best decoders are LD and LI. Both manage to converge to feasible solutions quickly and have the best performances in all cases - with LD being still better than LI in all cases.

6
Conclusions

This study presented the RSI allocation problem, whose main application is for 4G and 5G radio access networks. This problem is an application of the T-coloring problem, which is a generalization of the classical graph coloring problem and is considered NP-hard.

As Chapter 3 shows, problems such as the RSI allocation problem are usually solved by heuristic methods. In fact, the graph coloring problem and its generalizations are notoriously difficult to be solved optimally in reasonable computational times. Thus, this work introduces, alongside different mathematical models, heuristic methods to solve the RSI allocation problem.

As genetic algorithms are frequently presented in the literature to solve similar problems, the heuristic chosen was the Biased Random Key Genetic Algorithm (BRKGA). This type of genetic algorithm has been successfully used in many telecommunications problems, but never for graph coloring problems or similar. The customization of the BRKGA included six different decoders, thus exploring several coloring strategies, as detailed in Chapter 4.

The mathematical models and BRKGA were explored in real-life derived instances. For the mathematical models, the linear models had better performances than their non-linear counterparts. Also, all mathematical models had more difficulty with the long sequence instances in comparison with the short sequence ones, something related to the higher needed amounts of constraints and variables.

Differently, the BRKGA decoders had similar behaviors in both short and long RSI sequences. In fact, for them, the objective function appears to have a more significant influence on the solving difficulty. However, the short sequence instances tend to have higher percentages of infeasible runs, and an increase in conflict amount for all decoders except OR and SC. The corrective procedure, built to solve these issues, does not appear to have a significant impact on solution quality, except for the specific case of decoder SC in the long sequence instances. In it, the procedure does the opposite of what was expected, increasing conflict amounts by severely increasing time requirements.

Among decoding strategies, the Logic Direct Decoder (LD) has shown the best performance in all objectives and instance groups. In fact, it has not only high chances of having a feasible final solution, but those solutions do have better quality than all other decoders. This indicates that the strategy of considering vertex order and only neighbor relations has merit, something

that may be used in future studies.

For future works, we can improve the mathematical models by extending them with symmetry-breaking constraints, in order to reduce the pool of analyzed solutions. Another suggestion is the extension of the problem to a stochastic environment, in which the antennas would have the possibility of being deactivated, thus changing the network design. It is also interesting the possibility of diversification in both difference values for each edge and the RSI possible for each vertex. Another possibility includes the study of strategies similar to the used by the decoder LD, as this procedure showed itself to be effective and thus should be explored in similar problems. Finally, one may consider developing coloring strategies that do not use the old configuration for the maximize minimal span objective.

7
References

[1] GUPTA, A.; JHA, R. K.. **A survey of 5G network: Architecture and emerging technologies**. IEEE, 3:1206–1232, 2015. 1, 2.2, 2.2

[2] STRATEGY ANALYTICS. **Internet of things now numbers 22 billion devices but where is the revenue?**, 2019. Access: 16 April 2020. 1

[3] OSSEIRAN, A.; BOCCARDI, F.; BRAUN, V.; KUSUME, K.; MARSCH, P.; MATERNIA, M.; QUESETH, O.; SCHELLMANN, M.; SCHOTTEN, H.; TAOKA, H.; TULLBERG, H.; UUSITALO, M. A.; TIMUS, B. ; FALLGREN, M.. **Scenarios for 5G mobile and wireless communications: the vision of the metis project**. IEEE communications magazine, 52(5):26–35, 2014. 1

[4] PALATTELLA, M. R.; DOHLER, M.; GRIECO, A.; RIZZO, G.; TORSNER, J.; ENGEL, T. ; LADID, L.. **Internet of things in the 5G era: Enablers, architecture, and business models**. IEEE Journal on Selected Areas in Communications, 34(3):510–527, 2016. 1

[5] SHAMGANTH, K.; SIBLEY, M. J. N.. **A survey on relay selection in cooperative device-to-device (D2D) communication for 5G cellular networks**. In: 2017 INTERNATIONAL CONFERENCE ON ENERGY, COMMUNICATION, DATA ANALYTICS AND SOFT COMPUTING (ICECDS), p. 42–46, 2017. 1

[6] LIU, X.; WU, S.; GUO, Y. ; CHEN, C.. **The demand and development of internet of things for 5G:a survey**. In: 2018 IEEE INTERNATIONAL CONFERENCE ON CONSUMER ELECTRONICS-TAIWAN (ICCE-TW), p. 1–2, 2018. 1

[7] ANDRADE, C. E.; RESENDE, M. G. C.; ZHANG, W.; SINHA, R. K.; REICHMANN, K. C.; DOVERSPIKE, R. D. ; MIYAZAWA, F. K.. **A biased random-key genetic algorithm for wireless backhaul network design**. Applied Soft Computing, 33:150–169, 2015. 1, 4

[8] VERÍSSIMO, R.; VIEIRA, P.; RODRIGUES, A. ; QUELUZ, M. P.. **PCI and RSI conflict detection in a real LTE network using supervised learning**. URSI Radio Science Bulletin, 364:11–19, 2018. 1, 2.2, 3.2

[9] MALAGUTI, E.; MONACI, M. ; TOTH, P.. **A metaheuristic approach for the vertex coloring problem**. INFORMS Journal on Computing, 20(2):302–316, 2008. 1, 3.1

[10] HALE, W. K.. **Frequency assignment: Theory and applications**. Proceedings of the IEEE, 68(12):1497–1514, 1980. 1, 2.1, 3.2

[11] SIDDIQI, U. F.; SAIT, S. M.. **A neighborhood search-based heuristic for the fixed spectrum frequency assignment problem**. Arabian Journal for Science and Engineering, 44(4):2985–2994, 2019. 1, 3, 3.2

[12] MARSA-MAESTRE, I.; DE LA HOZ, E.; GIMENEZ-GUZMAN, J. M.; ORDEN, D. ; KLEIN, M.. **Nonlinear negotiation approaches for complex-network optimization: A study inspired by wi-fi channel assignment**. Group Decision and Negotiation, 28(1):175–196, 2019. 1, 3, 3.2

[13] ZHAO, L.; WANG, H. ; ZHONG, X.. **Interference graph based channel assignment algorithm for D2D cellular networks**. IEEE Access, 6:3270–3279, 2018. 1, 3.2

[14] ACEDO-HERNÁNDEZ, R.; TORIL, M.; LUNA-RAMÍREZ, S. ; UBEDA, C.. **A PCI planning algorithm for jointly reducing reference signal collisions in LTE uplink and downlink**. Computer Networks, 119:112–123, 2017. 1, 3, 3.2

[15] MALAGUTI, E.; TOTH, P.. **A survey on vertex coloring problems**. International Transactions in Operational Research, 17(1):1–34, 2010. 1, 2.3

[16] MOSTAFAIE, T.; KHIYABANI, F. M. ; NAVIMIPOUR, N. J.. **A systematic study on meta-heuristic approaches for solving the graph coloring problem**. Computers & Operations Research, 120:104850, 2020. 1, 4

[17] MURPHEY, R. A.; PARDALOS, P. M. ; RESENDE, M. G.. **Frequency assignment problems**. In: HANDBOOK OF COMBINATORIAL OPTIMIZATION, p. 295–377. Springer, 1999. 2.1

[18] JUNOSZA-SZANIAWSKI, K.; RZĄŻEWSKI, P.. **An exact algorithm for the generalized list t-coloring problem**. Discrete Mathematics and Theoretical Computer Science, 16(3):77–95, 2014. 2.1, 3.2, 3.3, 3.3, 3.3

[19] SHARMA, P. C.; CHAUDHARI, N. S.. **A tree based novel approach for graph coloring problem using maximal independent set**. Wireless Personal Communications, 110(3):1143–1155, 2020. 2.1, 3.1

[20] GUO, J.; MOALIC, L.; MARTIN, J. N. ; CAMINADA, A.. **Cluster resource assignment algorithm for device-to-device networks based on graph coloring**. In: 2017 13TH INTERNATIONAL WIRELESS COMMUNICATIONS AND MOBILE COMPUTING CONFERENCE (IWCMC), p. 1700–1705. IEEE, 2017. 2.1, 3.1

[21] GRAF, R. F.. **Modern dictionary of electronics**. Newnes, 1999. 2.2

[22] ENCYCLOPEDIA BRITANNICA. **Antenna**. 2.2

[23] CANADIAN RADIO-TELEVISION AND TELECOMMUNICATIONS COMMISSION. **Telecommunications glossary**. 2.2

[24] HOLMA, H.; TOSKALA, A.. **WCDMA for UMTS: HSPA Evolution and LTE, Fourth Edition**. Wiley Publishing, New York, 2017. 2.2

[25] COX, C.. **An Introduction to LTE, LTE-advanced, SAE, VoLTE and 4G Mobile Communications, Second Edition**. Wiley Publishing, New York, 2014. 2.2

[26] CHEN, S.; ZHAO, J.. **The requirements, challenges, and technologies for 5G of terrestrial mobile telecommunication**. IEEE Communications Magazine, 52(5):36–43, 2014. 2.2

[27] ALSHARIF, M. H.; NORDIN, R.; SHAKIR, M. M. ; RAMLY, A. M.. **Small cells integration with the macro-cell under LTE cellular networks and potential extension for 5G**. Journal of Electrical Engineering & Technology, 14(6):2455–2465, 2019. 2.2

[28] HAO, P.; YAN, X.; RUYUE, Y.-N. ; YUAN, Y.. **Ultra dense network: Challenges enabling technologies and new trends**. China Communications, 13(2):30–40, 2016. 2.2

[29] LONDE, M. A.; PESSOA, L. ; ANDRADE, C. E.. **Modelos exatos para alocação do root sequence index**. In: ANAIS DO LII SIMPóSIO BRASILEIRO DE PESQUISA OPERACIONAL. SBPO, 2020. 2.3, 5.3

[30] ONAP. **Open network automation platform**. 2.3

[31] PRATAP, A.; MISRA, R. ; GUPTA, U.. **Randomized graph coloring algorithm for physical cell ID assignment in LTE-a femtocellular networks**. Wireless Personal Communications, 91(3):1213–1235, 2016. 3, 3.2

[32] GAREY, M. R.; JOHNSON, D. S.. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W.H.Freeman&Co, New York, 1979. 3.1, 4

[33] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. ; STEIN, C.. **Algoritmos: Teoria e Prática**. Editora Campus, 2002. 3.1

[34] RANDALL-BROWN, J.. **Chromatic scheduling and the chromatic number problem**. Management Science, 19(4-part-1):456–463, 1972. 3.1

[35] LEIGHTON, F. T.. **A graph coloring algorithm for large scheduling problems**. Journal of research of the national bureau of standards, 84(6):489–506, 1979. 3.1

[36] BRÉLAZ, D.. **New methods to color the vertices of a graph**. Communications of the ACM, 22(4):251–256, 1979. 3.1

[37] HERTZ, A.; DE WERRA, D.. **Using tabu search techniques for graph coloring**. Computing, 39(4):345–351, 1987. 3.1

[38] JOHNSON, D. S.; ARAGON, C. R.; MCGEOCH, L. A. ; SCHEVON, C.. **Optimization by simulated annealing: an experimental evaluation; part II, graph coloring and number partitioning**. Operations research, 39(3):378–406, 1991. 3.1

[39] FLEURENT, C.; FERLAND, J. A.. **Genetic and hybrid algorithms for graph coloring**. Annals of Operations Research, 63:437–461, 1996. 3.1

[40] COSTA, D.; HERTZ, A.. **Ants can colour graphs**. Journal of the Operational Research Society, 48(3):295–305, 1997. 3.1

[41] GALINIER, P.; HAO, J. K.. **Hybrid evolutionary algorithms for graph coloring**. Journal of Combinatorial Optimization, 3(4):379–397, 1999. 3.1

[42] KASSOTAKIS, I. E.; MARKAKI, M. E. ; VASILAKOS, A. V.. **A hybrid genetic approach for channel reuse in multiple access telecommunication networks**. IEEE Journal on selected areas in communications, 18(2):234–243, 2000. 3.1

[43] CHIARANDINI, M.; STÜTZLE, T.. **An application of iterated local search to graph coloring problem**. In: PROCEEDINGS OF THE COMPUTATIONAL SYMPOSIUM ON GRAPH COLORING AND ITS GENERALIZATIONS, p. 112–125, 2002. 3.1

[44] BARBOSA, V. C.; ASSIS, C. A. G. ; NASCIMENTO, J. O.. **Two novel evolutionary formulations of the graph coloring problem**. Journal of Combinatorial Optimization, 8(1):41–63, 2004. 3.1

[45] GALINIER, P.; HERTZ, A.. **A survey of local search methods for graph coloring**. Computers & Operations Research, 33(9):2547–2562, 2006. 3.1

[46] MÉNDEZ-DÍAZ, I.; ZABALA, P.. **A branch-and-cut algorithm for graph coloring**. Discrete Applied Mathematics, 154(5):826–847, 2006. 3.1

[47] BUI, T. N.; NGUYEN, T. H.; PATEL, C. M. ; PHAN, K.-A. T.. **An ant-based algorithm for coloring graphs**. Discrete Applied Mathematics, 156:190–200, 2008. 3.1

[48] MÉNDEZ-DÍAZ, I.; ZABALA, P.. **A cutting plane algorithm for graph coloring**. Discrete Applied Mathematics, 156(2):159–179, 2008. 3.1

[49] CARAMIA, M.; DELL'OLMO, P.. **Coloring graphs by iterated local search traversing feasible and infeasible solutions**. Discrete Applied Mathematics, 156(2):201–217, 2008. 3.1

[50] DOWSLAND, K. A.; THOMPSON, J. M.. **An improved ant colony optimisation heuristic for graph colouring**. Discrete Applied Mathematics, 156(3):313–324, 2008. 3.1

[51] MABROUK, B. B.; HASNI, H. ; MAHJOUB, Z.. **On a parallel genetic–tabu search based algorithm for solving the graph colouring problem**. European Journal of Operational Research, 197(3):1192–1201, 2009. 3.1

[52] LÜ, Z.; HAO, J.-K.. **A memetic algorithm for graph coloring**. European Journal of Operational Research, 203(1):241–250, 2010. 3.1, 4.1.1

[53] HU, Y.; KOBOUROV, S. ; VEERAMONI, S.. **On maximum differential graph coloring**. In: INTERNATIONAL SYMPOSIUM ON GRAPH DRAWING, p. 274–286. Springer, 2010. 3.1

[54] MALAGUTI, E.; MONACI, M. ; TOTH, P.. **An exact approach for the vertex coloring problem**. Discrete Optimization, 8(2):174–190, 2011. 3.1

[55] DOUIRI, S. M.; ELBERNOUSSI, S.. **Solving the graph coloring problem via hybrid genetic algorithms**. Journal of King Saud University - Engineering Sciences, 27(1):114–118, 2015. 3.1

[56] SHUKL, A. N.; GARG, M. L.. **A list based approach to solve graph coloring problem**. In: PROCEEDINGS OF THE 2018 INTERNATIONAL CONFERENCE ON SYSTEM MODELING AND ADVANCEMENT IN RESEARCH TRENDS (SMART), p. 265–267. IEEE, 2018. 3.1

[57] MARAPPAN, R.; SETHUMADHAVAN, G.. **Solution to graph coloring using genetic and tabu search procedures**. Arabian Journal for Science and Engineering, 43(2):525–542, 2018. 3.1

[58] AARDAL, K. I.; VAN HOESEL, S. P.; KOSTER, A. M.; MANNINO, C. ; SASSANO, A.. **Models and solution techniques for frequency assignment problems**. Annals of Operations Research, 153(1):79–129, 2007. 3.2

[59] CUPPINI, M.. **A genetic algorithm for channel assignment problems**. European Transactions on Telecommunications, 5(2):285–294, 1994. 3.2

[60] DORNE, R.; HAO, J.-K.. **An evolutionary approach for frequency assignment in cellular radio networks**. In: PROCEEDINGS OF 1995 IEEE INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION, volumen 2, p. 539–544. IEEE, 1995. 3.2

[61] SMITH, D.; HURLEY, S. ; THIEL, S.. **Improving heuristics for the frequency assignment problem**. European Journal of Operational Research, 107(1):76–86, 1998. 3.2

[62] CHAKRABORTY, G.. **An efficient heuristic algorithm for channel assignment problem in cellular radio networks**. IEEE transactions on vehicular technology, 50(6):1528–1539, 2001. 3.2

[63] KRUMKE, S. O.; MARATHE, M. V. ; RAVI, S. S.. **Models and approximation algorithms for channel assignment in radio networks**. Wireless Networks, 7(6):575–584, 2001. 3.2, 4.2.10

[64] ALABAU, M.; IDOUMGHAR, L. ; SCHOTT, R.. **New hybrid genetic algorithms for the frequency assignment problem**. IEEE transactions on broadcasting, 48(1):27–34, 2002. 3.2

[65] KENDAIL, G.; MOHAMAD, M.. **Channel assignment optimisation using a hyper-heuristic**. In: IEEE CONFERENCE ON CYBERNETICS AND INTELLIGENT SYSTEMS, 2004, p. 791–796. IEEE, 2004. 3.2

[66] BANDH, T.; CARLE, G. ; SANNECK, H.. **Graph coloring based physical-cell-ID assignment for LTE networks**. In: PROCEEDINGS OF THE 2009 INTERNATIONAL CONFERENCE ON WIRELESS COMMUNICATIONS AND MOBILE COMPUTING: CONNECTING THE WORLD WIRELESSLY, p. 116–120, 2009. 3.2

[67] AHMED, F.; TIRKKONEN, O.; PELTOMÄKI, M.; KOLJONEN, J. M.; YU, C. H. ; ALAVA, M.. **Distributed graph coloring for self-organization in LTE networks**. Journal of Electrical and Computer Engineering, 2010, 2010. 3.2

[68] XU, H.; ZHOU, X. W. ; LI, Y.. **Model of hypergraph colouring for self-configuration in LTE networks**. In: 2011 INTERNATIONAL CONFERENCE ON INFORMATION MANAGEMENT, INNOVATION MANAGEMENT AND INDUSTRIAL ENGINEERING, volumen 1, p. 393–396. IEEE, 2011. 3.2

[69] SUN, H.; LI, N.; CHEN, Y.; DONG, J.; LIU, N.; HAN, Y. ; LIU, W.. **A method of PCI planning in LTE based on genetic algorithm**. Progress In Electromagnetics Research, 1575:1575–1578, 2012. 3.2

[70] KLINCEWICZ, J. G.. **Using GRASP to solve the generalised graph colouring problem with application to cell site assignment**. International Journal of Mobile Network Design and Innovation, 4(3):148–156, 2012. 3.2

[71] AHMED, F.; TIRKKONEN, O.. **Self organized physical cell ID assignment in multi-operator heterogeneous networks**. In: 2015 IEEE 81ST VEHICULAR TECHNOLOGY CONFERENCE (VTC SPRING), p. 1–5. IEEE, 2015. 3.2

[72] KOWALIK, ; SOCAŁA, A.. **Assigning channels via the meet-in-the-middle approach**. Algorithmica, 74(4):1435–1452, 2016. 3.2

[73] LIU, D. D.-F.. **T-colorings of graphs**. Discrete Mathematics, 101(1-3):203–212, 1992. 3.3

[74] COSTA, D.. **On the use of some known methods fort-colorings of graphs**. Annals of Operations Research, 41(4):343–358, 1993. 3.3

[75] ALON, N.; ZAKS, A.. **T-choosability in graphs**. Discrete applied mathematics, 82(1-3):1–13, 1998. 3.3

[76] MCDIARMID, C.. **Discrete mathematics and radio channel assignment**. In: RECENT ADVANCES IN ALGORITHMS AND COMBINATORICS, p. 27–63. Springer, 2003. 3.3

[77] MIRJALILI, S.. **Genetic algorithm**. In: EVOLUTIONARY ALGORITHMS AND NEURAL NETWORKS, p. 43–55. Springer, 2019. 4

[78] GONÇALVES, J. F.; RESENDE, M. G. C.. **Biased random-key genetic algorithms for combinatorial optimization**. Journal of Heuristics, 17(5):487–525, 2011. 4, 4.1.1

[79] ANDRADE, C. E.; MIYAZAWA, F. K. ; RESENDE, M. G. C.. **Evolutionary algorithm for the k-interconnected multi-depot multi-traveling salesmen problem**. In: PROCEEDINGS OF THE 15TH ANNUAL CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, GECCO'13, p. 463–470, New York, NY, USA, 2013. ACM. 4, 4.1.2

[80] PRASETYO, H.; FAUZA, G.; AMER, Y. ; LEE, S. H.. **Survey on applications of biased-random key genetic algorithms for solving optimization problems**. In: INDUSTRIAL ENGINEERING AND ENGINEERING MANAGEMENT (IEEM), 2015 IEEE INTERNATIONAL CONFERENCE ON, p. 863–870. IEEE, 2015. 4

[81] DE FARIA, H., J.; RESENDE, M. ; ERNST, D.. **A biased random key genetic algorithm applied to the electric distribution network reconfiguration problem**. Journal of Heuristics, 23(6):533–550, 2017. 4

[82] LUCENA, M. L.; ANDRADE, C. E.; RESENDE, M. G. ; MIYAZAWA, F. K.. **Some extensions of biased random-key genetic algorithms**. In: PROCEEDINGS OF THE 46TH BRAZILIAN SYMPOSIUM OF OPERATIONAL RESEARCH, XLVI SBPO, p. 2469–2480, 2014. 4.1.1

[83] ANDRADE, C. E.; TOSO, R. F.; GONÇALVES, J. F. ; RESENDE, M. G. C.. **The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications**. European Journal of Operational Research, 2021. 4.1.1, 4.1.2, C

[84] LONDE, M. A.; ANDRADE, C. E. ; PESSOA, L. S.. **An evolutionary approach for the p-next center problem**. Expert Systems with Applications, 2021. 4.1.1

[85] ANDRADE, C. E.; TOSO, R. F.; RESENDE, M. G. ; MIYAZAWA, F. K.. **Biased random-key genetic algorithms for the winner determi-**

nation problem in combinatorial auctions. Evolutionary computation, 23(2):279–307, 2015. 4.1.2

[86] ANDRADE, C. E.; SILVA, T. ; PESSOA, L. S.. **Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm**. Expert Systems with Applications, 128:67–80, Aug 2019. 4.1.2, 5.4

[87] WHITLEY, D.; RANA, S. ; HECKENDORN, R. B.. **The island model genetic algorithm: On separability, population size and convergence**. Journal of computing and information technology, 7(1):33–47, 1999. 4.1.2

[88] TOSO, R. F.; RESENDE, M. G.. **A C++ application programming interface for biased random-key genetic algorithms**. Optimization Methods and Software, 30(1):81–93, 2015. 4.1.2

[89] PANDEY, H. M.; CHAUDHARY, A. ; MEHROTRA, D.. **A comparative review of approaches to prevent premature convergence in ga**. Applied Soft Computing, 24:1047–1077, 2014. 4.1.2

[90] RIBEIRO, C. C.; RESENDE, M. G.. **Path-relinking intensification methods for stochastic local search algorithms**. Journal of heuristics, 18(2):193–214, 2012. 4.1.2

[91] KENDALL, M. G.. **A new measure of rank correlation**. Biometrika, 30(1/2):81–93, 1938. 4.1.2

[92] WELSH, D. J.; POWELL, M. B.. **An upper bound for the chromatic number of a graph and its application to timetabling problems**. The Computer Journal, 10(1):85–86, 1967. 4.2.7

[93] LÓPEZ-IBÁÑEZ, M.; DUBOIS-LACOSTE, J.; CÁCERES, L. P.; BIRATTARI, M. ; STÜTZLE, T.. **The irace package: Iterated racing for automatic algorithm configuration**. Operations Research Perspectives, 3:43–58, 2016. 5.2

A
Instances

Table A.1: Description of instances used for experiments. "Sequence" is indicative of maximal and minimal allowed RSI. "Vertices" is the number of antennas or nodes the instance has, while "*minDist*" and "*maxDist*" are the minimal and maximal distance requirements.

Instance Name	Sequence	Vertices	*minDist*	*maxDist*
long_n0030_r030_150	long	30	30	150
long_n0033_r020_160	long	33	20	160
long_n0038_r020_190	long	38	20	190
long_n0039_r020_150	long	39	20	150
long_n0040_r040_200	long	40	40	200
long_n0043_r020_170	long	43	20	170
long_n0045_r030_180	long	45	30	180
long_n0047_r020_180	long	47	20	180
long_n0051_r030_250	long	51	30	250
long_n0052_r020_100	long	52	20	100
long_n0059_r020_230	long	59	20	230
long_n0060_r030_240	long	60	30	240
long_n0061_r030_300	long	61	30	300
long_n0062_r020_310	long	62	20	310
long_n0065_r020_260	long	65	20	260
long_n0066_r030_330	long	66	30	330
long_n0067_r040_260	long	67	40	260
long_n0068_r020_130	long	68	20	130
long_n0069_r030_340	long	69	30	340
long_n0070_r040_280	long	70	40	280
long_n0073_r040_360	long	73	40	360
long_n0077_r040_380	long	77	40	380
long_n0081_r030_400	long	81	30	400
long_n0082_r040_320	long	82	40	320
long_n0083_r020_160	long	83	20	160
long_n0083_r040_330	long	83	40	330
long_n0084_r040_420	long	84	40	420
long_n0093_r030_180	long	93	30	180

Continue on next page...

Table A.1 (continued).

Instance Name	**Sequence**	**Vertices**	$minDist$	$maxDist$
long_n0094_r040_470	long	94	40	470
long_n0095_r020_190	long	95	20	190
long_n0109_r030_540	long	109	30	540
long_n0114_r030_450	long	114	30	450
long_n0120_r030_240	long	120	30	240
long_n0122_r040_610	long	122	40	610
long_n0126_r020_250	long	126	20	250
long_n0129_r030_250	long	129	30	250
long_n0132_r020_260	long	132	20	260
long_n0135_r030_270	long	135	30	270
long_n0159_r040_310	long	159	40	310
long_n0162_r020_320	long	162	20	320
long_n0174_r030_340	long	174	30	340
long_n0202_r040_400	long	202	40	400
long_n0210_r040_420	long	210	40	420
long_n0238_r040_470	long	238	40	470
long_n0258_r030_510	long	258	30	510
long_n0259_r030_510	long	259	30	510
long_n0282_r040_560	long	282	40	560
long_n0284_r040_838	long	284	40	838
long_n0297_r030_590	long	297	30	590
long_n0323_r040_640	long	323	40	640
long_n0327_r040_650	long	327	40	650
long_n0340_r040_680	long	340	40	680
long_n0360_r040_720	long	360	40	720
long_n0365_r040_730	long	365	40	730
long_n0374_r040_740	long	374	40	740
long_n0376_r040_750	long	376	40	750
long_n0391_r040_838	long	391	40	838
long_n0405_r040_838	long	405	40	838
long_n0461_r040_838	long	461	40	838
long_n0463_r040_838	long	463	40	838
long_n0479_r040_838	long	479	40	838
long_n0499_r040_838	long	499	40	838
long_n0544_r040_838	long	544	40	838

Continue on next page...

Table A.1 (continued).

Instance Name	**Sequence**	**Vertices**	$minDist$	$maxDist$
long_n0554_r040_838	long	554	40	838
long_n0596_r040_838	long	596	40	838
long_n0600_r030_838	long	600	30	838
long_n0647_r040_838	long	647	40	838
long_n0698_r040_838	long	698	40	838
long_n0728_r040_838	long	728	40	838
long_n0759_r040_838	long	759	40	838
long_n1026_r040_838	long	1026	40	838
long_n1026_r030_838	long	1026	30	838
long_n1348_r030_838	long	1348	30	838
long_n1569_r040_838	long	1569	40	838
long_n1580_r040_838	long	1580	40	838
long_n1892_r030_838	long	1892	30	838
long_n2036_r030_838	long	2036	30	838
long_n2124_r040_838	long	2124	40	838
long_n2284_r030_838	long	2284	30	838
long_n2977_r030_838	long	2977	30	838
long_n3092_r040_838	long	3092	40	838
long_n4125_r030_838	long	4125	30	838
short_n0030_r010_060	short	30	10	60
short_n0033_r010_080	short	33	10	80
short_n0038_r010_095	short	38	10	95
short_n0039_r010_075	short	39	10	75
short_n0040_r015_080	short	40	15	80
short_n0043_r010_085	short	43	10	85
short_n0045_r015_090	short	45	15	90
short_n0047_r010_090	short	47	10	90
short_n0051_r015_125	short	51	15	125
short_n0052_r010_050	short	52	10	50
short_n0059_r010_115	short	59	10	115
short_n0060_r015_120	short	60	15	120
short_n0061_r015_139	short	61	15	139
short_n0062_r010_139	short	62	10	139
short_n0065_r010_139	short	65	10	139
short_n0066_r015_139	short	66	15	139

Continue on next page...

Table A.1 (continued).

Instance Name	**Sequence**	**Vertices**	$minDist$	$maxDist$
short_n0067_r010_065	short	67	10	65
short_n0068_r020_135	short	68	20	135
short_n0069_r010_135	short	69	10	135
short_n0070_r020_139	short	70	20	139
short_n0073_r015_139	short	73	15	139
short_n0077_r015_139	short	77	15	139
short_n0081_r010_139	short	81	10	139
short_n0082_r010_080	short	82	10	80
short_n0083_r010_080	short	83	10	80
short_n0084_r010_139	short	84	10	139
short_n0093_r015_090	short	93	15	90
short_n0094_r010_090	short	94	10	90
short_n0095_r015_139	short	95	15	139
short_n0120_r015_120	short	120	15	120
short_n0126_r015_139	short	126	15	139
short_n0129_r015_125	short	129	15	125
short_n0132_r010_130	short	132	10	130
short_n0135_r015_135	short	135	15	135
short_n0159_r015_139	short	159	15	139
short_n0174_r015_139	short	174	15	139
short_n0202_r015_139	short	202	15	139
short_n0210_r015_139	short	210	15	139
short_n0238_r010_139	short	238	10	139
short_n0259_r010_139	short	259	10	139
short_n0282_r010_139	short	282	10	139
short_n0284_r015_139	short	284	15	139
short_n0327_r010_139	short	327	10	139
short_n0340_r015_139	short	340	15	139
short_n0360_r010_139	short	360	10	139
short_n0365_r010_139	short	365	10	139
short_n0391_r010_139	short	391	10	139
short_n0405_r010_139	short	405	10	139
short_n0461_r010_139	short	461	10	139
short_n0463_r015_139	short	463	15	139
short_n0554_r010_139	short	554	10	139

Continue on next page...

Table A.1 (continued).

Instance Name	**Sequence**	**Vertices**	$minDist$	$maxDist$
short_n0596_r010_139	short	596	10	139
short_n0647_r010_139	short	647	10	139
short_n0698_r010_139	short	698	10	139
short_n0759_r010_139	short	759	10	139
short_n1026_r010_139	short	1026	10	139
short_n1348_r010_139	short	1348	10	139

B
Tuning results

Table B.1: Best parameter configurations suggested by `irace` for decoder LD in the minimize change objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e\%$	$\mathcal{P}_m\%$	π_e,π_t	Φ	p	md	sel	$ps\%$	I_{ipr}	I_s	R_m	LS	LS%	CR
772	0.39	0.14	3,10	r^{-2}	2	0.12	RE	0.86	242	180	3.34	NLS	0.46	FALSE
2036	0.24	0.17	3,10	r^{-2}	1	0.004	BS	0.57	89	143	4.19	NLS	0.32	TRUE
1847	0.29	0.17	3,10	r^{-2}	2	0.16	RE	0.53	101	206	3.34	NLS	0.34	TRUE
2808	0.23	0.11	3,10	r^{-2}	1	0.05	BS	0.18	68	119	3.04	NLS	0.57	TRUE
992	0.36	0.11	3,10	r^{-2}	1	0.0006	BS	0.77	327	134	2.89	NLS	0.56	TRUE

Table B.2: Best parameter configurations suggested by `irace` for decoder LI in the minimize change objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e\%$	$\mathcal{P}_m\%$	π_e,π_t	Φ	p	md	sel	$ps\%$	I_{ipr}	I_s	R_m	LS	LS%	CR
963	0.36	0.14	3,10	r^{-2}	1	0.08	BS	0.41	223	123	3.88	NLS	0.11	TRUE
1063	0.26	0.18	3,10	r^{-2}	1	0.13	BS	0.47	280	156	3.28	NLS	0.37	TRUE
1172	0.35	0.15	3,10	r^{-2}	1	0.03	RE	0.19	313	162	2.35	NLS	0.12	TRUE
899	0.18	0.22	3,10	r^{-2}	1	0.15	RE	0.60	167	161	1.65	NLS	0.18	FALSE
1597	0.31	0.12	3,10	r^{-2}	1	0.06	BS	0.22	331	157	2.88	NLS	0.45	TRUE

Table B.3: Best parameter configurations suggested by `irace` for decoder OR in the minimize change objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e\%$	$\mathcal{P}_m\%$	π_e,π_t	Φ	p	md	sel	$ps\%$	I_{ipr}	I_s	R_m	LS	LS%	CR
277	0.42	0.19	3,6	e^{-r}	1	0.27	BS	0.45	475	164	1.67	NLS	0.49	TRUE
393	0.23	0.17	3,6	e^{-r}	1	0.29	RE	0.49	435	188	3.13	BI	0.72	TRUE
1010	0.38	0.10	5,10	r^{-2}	1	0.02	BS	0.30	107	270	2.93	FI	0.97	TRUE
781	0.27	0.16	5,10	r^{-2}	2	0.004	BS	0.09	219	287	3.75	BI	0.91	TRUE
209	0.46	0.11	3,6	e^{-r}	2	0.29	BS	0.82	307	171	1.91	NLS	0.59	TRUE

Table B.4: Best parameter configurations suggested by `irace` for decoder SC in the minimize change objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
1752	0.37	0.14	7,10	r^{-1}	1	0.06	BS	0.94	325	276	4.00	FI	0.62	FALSE
2093	0.34	0.12	7,10	r^{-1}	1	0.07	BS	0.61	234	295	4.26	FI	0.73	FALSE
1904	0.31	0.17	7,10	r^{-1}	1	0.11	BS	0.81	212	251	4.73	FI	0.52	FALSE
2080	0.18	0.21	7,10	r^{-1}	1	0.09	BS	0.88	283	268	4.33	FI	0.94	FALSE
2842	0.33	0.16	7,10	r^{-1}	1	0.05	BS	0.77	204	257	4.74	FI	0.72	FALSE

Table B.5: Best parameter configurations suggested by `irace` for decoder KD in the minimize change objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
609	0.41	0.20	3,10	r^{-2}	1	0.04	BS	0.07	229	198	2.84	FI	0.51	TRUE
1166	0.39	0.10	3,10	r^{-2}	1	0.08	RE	0.74	479	285	4.55	NLS	0.43	TRUE
735	0.35	0.11	3,10	r^{-2}	1	0.09	BS	0.09	277	238	3.38	FI	0.63	TRUE
733	0.36	0.18	3,10	r^{-2}	1	0.09	BS	0.02	125	267	2.25	FI	0.45	TRUE
437	0.18	0.14	5,10	r^{-1}	2	0.24	BS	0.53	66	227	2.08	FI	0.49	TRUE

Table B.6: Best parameter configurations suggested by `irace` for decoder KC in the minimize change objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
1647	0.40	0.12	3,10	r^{-2}	1	0.15	BS	0.41	176	153	2.91	NLS	0.38	FALSE
963	0.36	0.14	3,10	r^{-2}	1	0.08	BS	0.41	223	123	3.88	NLS	0.11	TRUE
1402	0.17	0.23	4,6	r^{-2}	1	0.05	BS	0.37	345	172	4.89	FI	0.73	FALSE
1469	0.25	0.17	7,10	r^{-1}	2	0.21	BS	0.28	296	160	4.50	FI	0.85	FALSE
1584	0.22	0.16	1,3	e^{-r}	1	0.20	BS	0.18	383	177	1.17	NLS	0.91	TRUE

Table B.7: Best parameter configurations suggested by `irace` for decoder LD in the maximize minimal span objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
1010	0.38	0.10	5,10	r^{-2}	1	0.02	BS	0.30	107	270	2.93	FI	0.97	TRUE
4004	0.39	0.15	7,10	r^{-2}	3	0.08	BS	0.67	236	110	3.49	FI	0.37	FALSE
3840	0.43	0.12	3,6	r^{-2}	2	0.13	BS	0.25	98	129	4.23	FI	0.28	TRUE
2924	0.34	0.45	3,6	e^{-r}	1	0.02	RE	0.47	122	226	2.28	FI	0.69	FALSE
3858	0.36	0.14	1,3	r^{-2}	2	0.13	BS	0.03	187	168	2.71	FI	0.20	TRUE

Table B.8: Best parameter configurations suggested by `irace` for decoder LI in the maximize minimal span objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
1010	0.38	0.10	5,10	r^{-2}	1	0.02	BS	0.30	107	270	2.93	FI	0.96	TRUE
4048	0.12	0.21	1,3	r^{-2}	1	0.08	BS	0.54	354	151	1.31	FI	0.44	TRUE
4772	0.19	0.25	2,3	e^{-r}	3	0.29	BS	0.69	473	145	3.37	FI	0.42	TRUE
935	0.35	0.17	2,3	r^{-1}	3	0.18	RE	0.95	409	151	2.37	FI	0.91	FALSE
3131	0.28	0.38	3,10	r^{-2}	2	0.05	RE	0.23	223	272	4.91	FI	0.72	FALSE

Table B.9: Best parameter configurations suggested by `irace` for decoder OR in the maximize minimal span objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
619	0.35	0.12	3,10	e^{-r}	1	0.06	BS	0.88	398	167	3.64	FI	0.39	TRUE
173	0.31	0.10	3,6	e^{-r}	1	0.14	RE	0.48	446	212	4.19	FI	0.67	TRUE
355	0.29	0.12	3,6	e^{-r}	1	0.17	RE	0.76	459	200	2.79	FI	0.72	FALSE
787	0.22	0.16	5,10	r^{-2}	1	0.08	RE	0.31	157	238	2.34	FI	0.91	TRUE
601	0.45	0.10	5,10	e^{-r}	1	0.07	BS	0.49	50	254	1.69	FI	0.51	TRUE

Table B.10: Best parameter configurations suggested by `irace` for decoder SC in the maximize minimal span objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
4902	0.27	0.13	2,6	e^{-r}	2	0.02	RE	0.38	307	144	4.99	NLS	0.45	FALSE
4711	0.28	0.15	2,6	e^{-r}	1	0.15	BS	0.13	177	150	4.56	NLS	0.35	TRUE
1936	0.22	0.12	4,6	r^{-1}	2	0.005	RE	0.24	287	258	2.85	NLS	0.69	FALSE
2331	0.14	0.23	4,6	r^{-1}	1	0.02	RE	0.31	52	193	2.97	NLS	0.91	FALSE
3209	0.23	0.13	2,6	e^{-r}	2	0.09	BS	0.75	145	256	4.51	NLS	0.48	FALSE

Table B.11: Best parameter configurations suggested by `irace` for decoder KD in the maximize minimal span objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
1268	0.13	0.31	1,3	e^{-r}	2	0.18	BS	0.45	463	128	1.31	FI	0.86	TRUE
944	0.16	0.27	1,3	e^{-r}	1	0.16	BS	0.66	347	261	1.10	FI	0.95	TRUE
1642	0.15	0.31	1,3	e^{-r}	1	0.15	BS	0.58	338	113	2.11	FI	0.59	TRUE
3180	0.29	0.16	2,6	e^{-r}	2	0.19	BS	0.54	431	152	2.01	FI	0.84	TRUE
4226	0.20	0.43	5,10	r^{-2}	1	0.09	BS	0.46	364	101	2.95	FI	0.23	TRUE

Table B.12: Best parameter configurations suggested by `irace` for decoder KC in the maximize minimal span objective.

BRKGA						IPR				Shaking		Procedures		
$\|\mathcal{P}\|$	$\mathcal{P}_e$%	$\mathcal{P}_m$%	π_e,π_t	Φ	p	md	sel	ps%	I_{ipr}	I_s	R_m	LS	LS%	CR
2304	0.34	0.23	5,10	e^{-r}	1	0.01	RE	0.33	441	123	1.65	FI	0.51	TRUE
2210	0.37	0.10	2,3	e^{-r}	1	0.25	BS	0.26	353	107	2.49	FI	0.37	FALSE
2202	0.37	0.12	2,3	e^{-r}	1	0.22	BS	0.29	458	129	3.97	FI	0.56	FALSE
2159	0.38	0.24	5,10	e^{-r}	1	0.01	RE	0.15	392	135	2.05	FI	0.60	TRUE
1384	0.39	0.17	2,3	e^{-r}	1	0.20	BS	0.43	398	128	2.92	FI	0.49	FALSE

C
Implicit Path-Relinking Results

The Implicit Path-Relinking procedure was recently introduced by Andrade et al. [83]. The newness of this procedure means that a more detailed analysis of its behavior and results becomes crucial, in order to clarify its potency and effectiveness in different cases.

Thus, Tables C.1 and C.2 were made. Both detail the IPR role for the improvement of the BRKGA procedure, separated by decoder. Column "Avg. Calls" details the amount of times that the IPR procedure was called on average per instance, while "Elite" indicates the times an individual of the elite set was improved. Finally, column "Best" indicates how many times the best individual was improved by IPR. In parenthesis is the proportion of improvement calls in relation to the number of average calls.

For both objectives, it can be seen that decoders SC and SCno have the most IPR calls and more improvements both in the elite and best individuals. That can be related to the simplicity of these decoders, which lets them run more generations and, thus, have more chances of calling the IPR procedure. Moreover, these tables indicate how many times the improvement happened - not the quality of this improvement. Finally, decoders SC and SCno both use larger populations than that of the other decoders, as showed Appendix B. Larger amounts of individuals mean that the possibility of the Path-relinking procedure of finding distant enough individuals to compare is higher in comparison with the other decoders, further justifying IPR effects.

Something else that is interesting to note is that the IPR effect for decoders SC and SCno is, proportionally and in the minimize change objective, similar to the other decoders. In fact, in this comparison, decoders LD and LDno do have a proportionally higher number of Elite improvements, while OR has a more frequent Best solution improvement. For decoders LD and LDno, this effect may be a derivative of the quick convergence to good, viable solutions, in which the IPR acts as an explorer of non-viable solutions that may lead to better results, thus improving in, at least, the elite set. Meanwhile, the OR decoder has, amongst all decoders, the worst results. IPR may, thus, manage to improve the best solution so often because the initial solutions are not good in the first place, but the exploration of the intermediate solutions may manage to improve their quality significantly by exploring new solution spaces.

For the maximize minimal span, the OR decoder still has a higher percentage of best individual improvement in comparison with the other decoders. This may be

justified the same as the other objective. However, in this case, SC, SCno, and KD have drastically higher amounts of elite improvements. KD's case may be justified by its higher concentration of final results. If this tendency of less varied results was consistent amongst all generations of the algorithm run, then the IPR would give a much-needed shake-like effect, which would let the algorithm explore new solution spaces. Meanwhile, SC and SCno is a similar justification as the OR one, but with one addendum: as SC and SCno manage to find regular solutions more often, the IPR does not manage to strongly improve those solutions, but only finds improvements relative to the elite set, i.e., the effect is not as powerful as it is on OR individuals.

Table C.1: IPR effects by decoder in average per instance. This table refers to the minimize changes objective.

Decoder	Avg. Calls	Improvement	
		Elite(%)	Best(%)
KC	17.9	1.19(0.07)	0.12(0.006)
KCno	17.9	1.29(0.07)	0.18(0.010)
KD	76.1	7.38(0.10)	3.71(0.05)
LD	255.0	60.70(0.24)	10.90(0.04)
LDno	267.0	64.30(0.24)	11.70(0.04)
LI	223.0	14.60(0.07)	11.60(0.05)
OR	105.0	14.70(0.14)	55.60(0.53)
SC	1168.0	104.00(0.09)	133.00(0.11)
SCno	1400.0	118.00(0.08)	162.00(0.12)

Table C.2: IPR effects by decoder in average per instance. This table refers to the maximize minimal span objective.

Decoder	Avg. Calls	Improvement	
		Elite(%)	Best(%)
KC	5.2	1.00(0.19)	0.03(0.01)
KD	2.9	0.98(0.33)	0.26(0.09)
LD	295.0	9.22(0.03)	4.83(0.02)
LI	117.0	3.16(0.03)	1.56(0.01)
OR	31.2	2.12(0.07)	7.65(0.24)
SC	565.0	220.00(0.40)	24.4(0.04)
SCno	587.0	231.00(0.39)	28.1(0.05)

D
Detailed Results - Mathematical Models

D.1
Unsolvable Instances

Table D.1: Listing of instances that at least one model could not find a solution on the maximal time of 3,600 seconds wall-clock. A **"x"** indicates that the model did not find any legal solution for the respective instance.

Instance Name	NC	NS	LC	LS
long_n0065_r020_260	x			
long_n0077_r040_380		x		
long_n0084_r040_420			x	
long_n0114_r030_450	x	x	x	
long_n0122_r040_610	x	x	x	
long_n0129_r030_250	x	x		
long_n0162_r020_320	x	x	x	
long_n0238_r040_470	x		x	
long_n0258_r030_510			x	x
long_n0259_r030_510	x	x		x
long_n0282_r040_560	x	x	x	x
long_n0297_r030_590		x	x	
long_n0323_r040_640	x	x	x	x
long_n0499_r040_838	x		x	x
long_n0698_r040_838		x		
long_n1026_r030_838	x	x		
long_n1348_r030_838	x	x	x	x
long_n1569_r040_838		x		x
long_n1580_r040_838	x	x	x	x
long_n1892_r030_838	x	x	x	x
long_n2036_r030_838	x	x	x	
long_n2124_r040_838	x	x	x	x
long_n2284_r030_838			x	x
long_n2977_r030_838	x	x	x	x
long_n3092_r040_838	x	x	x	x
long_n4125_r030_838	x	x	x	x

Continue on next page...

Table D.1 (continued).

Instance Name	NC	NS	LC	LS
short_n0126_r015_139		x		
short_n0210_r015_139	x	x	x	x
short_n0259_r010_139	x	x	x	x
short_n0282_r010_139	x	x		
short_n0327_r010_139		x		
short_n0340_r015_139	x	x		
short_n0360_r010_139	x	x	x	x
short_n0463_r015_139	x	x	x	x
short_n0554_r010_139	x	x	x	x
short_n0596_r010_139	x			
short_n0698_r010_139	x	x		
short_n0759_r010_139	x	x	x	x
short_n1026_r010_139	x	x	x	x
short_n1348_r010_139	x	x	x	x
Total	31	33	27	22
	22.3%	23.7%	19.4%	15.8%

D.2 Results by model

Table D.2: Results for model NC in 3,600 seconds wall-clock. "GAP%" indicates the best difference between lower and upper bounds. "# feasible" is the amount of found legal solutions, and "Time", the time in seconds to obtain an optimal solution. If not found, then it is equal to the maximal time.

Instance Name	GAP%	# feasible	Time
long_n0030_r030_150	9.09	7	3600.00
long_n0033_r020_160	14.00	8	3600.00
long_n0038_r020_190	0.00	5	651.28
long_n0039_r020_150	7.14	10	3600.00
long_n0040_r040_200	0.00	1	117.89
long_n0043_r020_170	29.03	6	3600.00
long_n0045_r030_180	0.00	6	938.75
long_n0047_r020_180	3.45	13	3600.00
long_n0051_r030_250	30.26	4	3600.00

Continue on next page...

Table D.2 (continued).

Instance Name	GAP%	# feasible	Time
long_n0052_r020_100	0.00	3	50.61
long_n0059_r020_230	36.69	3	3600.00
long_n0060_r030_240	31.56	10	3600.00
long_n0061_r030_300	29.26	1	3600.00
long_n0062_r020_310	11.17	3	3600.00
long_n0066_r030_330	23.49	13	3600.00
long_n0067_r040_260	13.33	5	3600.00
long_n0068_r020_130	0.00	5	54.89
long_n0069_r030_340	35.00	5	3600.00
long_n0070_r040_280	13.36	11	3600.00
long_n0073_r040_360	35.12	7	3600.00
long_n0077_r040_380	46.63	2	3600.00
long_n0081_r030_400	29.12	8	3600.00
long_n0082_r040_320	26.45	5	3600.00
long_n0083_r020_160	26.08	12	3600.00
long_n0083_r040_330	20.65	5	3600.00
long_n0084_r040_420	42.10	7	3600.00
long_n0093_r030_180	34.50	6	3600.00
long_n0094_r040_470	37.16	10	3600.00
long_n0095_r020_190	21.43	18	3600.00
long_n0109_r030_540	52.48	16	3600.00
long_n0120_r030_240	31.07	13	3600.00
long_n0126_r020_250	38.46	7	3600.00
long_n0132_r020_260	40.81	1	3600.00
long_n0135_r030_270	48.24	4	3600.00
long_n0159_r040_310	11.11	16	3600.00
long_n0174_r030_340	34.81	22	3600.00
long_n0202_r040_400	38.48	22	3600.00
long_n0210_r040_420	70.04	17	3600.00
long_n0258_r030_510	42.56	13	3600.00
long_n0284_r040_838	6.33	22	3600.00
long_n0297_r030_590	49.35	18	3600.00
long_n0327_r040_650	25.64	16	3600.00
long_n0340_r040_680	52.15	4	3600.00
long_n0360_r040_720	26.98	66	3600.00

Continue on next page...

Table D.2 (continued).

Instance Name	GAP%	# feasible	Time
long_n0365_r040_730	30.81	30	3600.00
long_n0374_r040_740	43.36	5	3600.00
long_n0376_r040_750	40.36	6	3600.00
long_n0391_r040_838	26.47	34	3600.00
long_n0405_r040_838	33.03	42	3600.00
long_n0461_r040_838	14.10	12	3600.00
long_n0463_r040_838	34.90	25	3600.00
long_n0479_r040_838	38.25	7	3600.00
long_n0544_r040_838	38.72	12	3600.00
long_n0554_r040_838	42.09	27	3600.00
long_n0596_r040_838	50.77	4	3600.00
long_n0600_r030_838	40.23	2	3600.00
long_n0647_r040_838	59.49	3	3600.00
long_n0698_r040_838	60.29	8	3600.00
long_n0728_r040_838	48.16	3	3600.00
long_n0759_r040_838	45.54	33	3600.00
long_n1026_r040_838	55.65	4	3600.00
long_n1569_r040_838	58.24	2	3600.00
long_n2284_r030_838	60.64	2	3600.00
short_n0030_r010_060	14.29	7	3600.00
short_n0033_r010_080	14.18	14	3600.00
short_n0038_r010_095	0.00	5	161.73
short_n0039_r010_075	9.09	14	3600.00
short_n0040_r015_080	0.00	11	156.36
short_n0043_r010_085	21.98	11	3600.00
short_n0045_r015_090	0.00	13	1088.77
short_n0047_r010_090	0.00	16	1136.09
short_n0051_r015_125	34.29	7	3600.00
short_n0052_r010_050	0.00	14	2247.36
short_n0059_r010_115	28.95	10	3600.00
short_n0060_r015_120	25.71	17	3600.00
short_n0061_r015_139	26.32	7	3600.00
short_n0062_r010_139	18.18	9	3600.00
short_n0065_r010_139	27.24	18	3600.00
short_n0066_r015_139	36.35	8	3600.00

Continue on next page...

Table D.2 (continued).

Instance Name	GAP%	# feasible	Time
short_n0067_r010_065	10.00	8	3600.00
short_n0068_r020_135	0.00	6	146.3
short_n0069_r010_135	36.26	8	3600.00
short_n0070_r020_139	8.33	20	3600.00
short_n0073_r015_139	32.07	10	3600.00
short_n0077_r015_139	39.22	9	3600.00
short_n0081_r010_139	8.76	33	3600.00
short_n0082_r010_080	9.52	10	3600.00
short_n0083_r010_080	11.90	8	3600.00
short_n0084_r010_139	43.10	14	3600.00
short_n0093_r015_090	16.55	19	3600.00
short_n0094_r010_090	37.43	22	3600.00
short_n0095_r015_139	4.26	11	3600.00
short_n0120_r015_120	41.57	30	3600.00
short_n0126_r015_139	41.25	4	3600.00
short_n0129_r015_125	43.16	6	3600.00
short_n0132_r010_130	43.12	18	3600.00
short_n0135_r015_135	37.09	6	3600.00
short_n0159_r015_139	5.13	12	3600.00
short_n0174_r015_139	27.87	9	3600.00
short_n0202_r015_139	23.91	17	3600.00
short_n0238_r010_139	27.21	39	3600.00
short_n0284_r015_139	28.63	40	3600.00
short_n0327_r010_139	28.80	50	3600.00
short_n0365_r010_139	12.82	63	3600.00
short_n0391_r010_139	12.94	89	3600.00
short_n0405_r010_139	45.55	34	3600.00
short_n0461_r010_139	16.59	66	3600.00
short_n0647_r010_139	44.60	38	3600.00
Average	27.68	13	3298.03

Table D.3: Results for model NS in 3,600 seconds wall-clock. "GAP%" indicates the best difference between lower and upper bounds. "# feasible" is the amount of found legal solutions, and "Time", the time in seconds to obtain an optimal solution. If not found, then it is equal to the maximal time.

Instance Name	GAP%	# feasible	Time
long_n0030_r030_150	143.33	1	3600.00
long_n0033_r020_160	142.42	3	3600.00
long_n0038_r020_190	313.04	4	3600.00
long_n0039_r020_150	134.36	2	3600.00
long_n0040_r040_200	98.58	3	3600.00
long_n0043_r020_170	304.76	2	3600.00
long_n0045_r030_180	200.00	1	3600.00
long_n0047_r020_180	309.09	3	3600.00
long_n0051_r030_250	168.82	1	3600.00
long_n0052_r020_100	150.00	1	3600.00
long_n0059_r020_230	397.83	4	3600.00
long_n0060_r030_240	300.00	1	3600.00
long_n0061_r030_300	354.55	4	3600.00
long_n0062_r020_310	559.09	3	3600.00
long_n0065_r020_260	471.43	2	3600.00
long_n0066_r030_330	358.33	7	3600.00
long_n0067_r040_260	202.33	3	3600.00
long_n0068_r020_130	106.35	3	3600.00
long_n0069_r030_340	466.67	1	3600.00
long_n0070_r040_280	204.35	6	3600.00
long_n0073_r040_360	670.99	6	3600.00
long_n0081_r030_400	506.06	4	3600.00
long_n0082_r040_320	250.00	1	3600.00
long_n0083_r020_160	700.00	1	3600.00
long_n0083_r040_330	607.32	2	3600.00
long_n0084_r040_420	882.81	3	3600.00
long_n0093_r030_180	200.00	1	3600.00
long_n0094_r040_470	814.89	8	3600.00
long_n0095_r020_190	608.33	4	3600.00
long_n0109_r030_540	1447.83	4	3600.00
long_n0120_r030_240	700.00	1	3600.00
long_n0126_r020_250	468.18	3	3600.00

Continue on next page...

Table D.3 (continued).

Instance Name	GAP%	# feasible	Time
long_n0132_r020_260	766.67	1	3600.00
long_n0135_r030_270	600.00	1	3600.00
long_n0159_r040_310	206.82	7	3600.00
long_n0174_r030_340	1000.00	1	3600.00
long_n0202_r040_400	376.19	5	3600.00
long_n0210_r040_420	950.00	3	3600.00
long_n0238_r040_470	1046.34	2	3600.00
long_n0258_r030_510	1600.00	3	3600.00
long_n0284_r040_838	897.62	5	3600.00
long_n0327_r040_650	1296.67	3	3600.00
long_n0340_r040_680	1558.54	4	3600.00
long_n0360_r040_720	1600.00	3	3600.00
long_n0365_r040_730	1680.49	4	3600.00
long_n0374_r040_740	1750.00	3	3600.00
long_n0376_r040_750	1536.00	3	3600.00
long_n0391_r040_838	1733.12	3	3600.00
long_n0405_r040_838	1757.77	4	3600.00
long_n0461_r040_838	1296.67	1	3600.00
long_n0463_r040_838	1296.67	1	3600.00
long_n0479_r040_838	1695.71	3	3600.00
long_n0499_r040_838	1915.00	3	3600.00
long_n0544_r040_838	1575.00	3	3600.00
long_n0554_r040_838	1895.00	3	3600.00
long_n0596_r040_838	1945.00	3	3600.00
long_n0600_r030_838	2510.00	3	3600.00
long_n0647_r040_838	1945.00	3	3600.00
long_n0728_r040_838	1695.00	3	3600.00
long_n0759_r040_838	1895.00	3	3600.00
long_n1026_r040_838	6307.92	2	3600.00
long_n2284_r030_838	2693.33	3	3600.00
short_n0030_r010_060	50.00	2	3600.00
short_n0033_r010_080	142.42	2	3600.00
short_n0038_r010_095	115.91	2	3600.00
short_n0039_r010_075	87.50	1	3600.00
short_n0040_r015_080	66.67	2	3600.00

Continue on next page...

Table D.3 (continued).

Instance Name	GAP%	# feasible	Time
short_n0043_r010_085	180.00	1	3600.00
short_n0045_r015_090	0.00	3	1998.92
short_n0047_r010_090	104.55	3	3600.00
short_n0051_r015_125	164.76	1	3600.00
short_n0052_r010_050	20.00	1	3600.00
short_n0059_r010_115	248.48	3	3600.00
short_n0060_r015_120	166.67	3	3600.00
short_n0061_r015_139	131.67	3	3600.00
short_n0062_r010_139	360.00	2	3600.00
short_n0065_r010_139	304.55	2	3600.00
short_n0066_r015_139	195.56	3	3600.00
short_n0067_r010_065	62.50	1	3600.00
short_n0068_r020_135	104.55	3	3600.00
short_n0069_r010_135	261.04	2	3600.00
short_n0070_r020_139	0.00	4	1049.73
short_n0073_r015_139	133.61	5	3600.00
short_n0077_r015_139	208.89	3	3600.00
short_n0081_r010_139	527.27	3	3600.00
short_n0082_r010_080	100.00	2	3600.00
short_n0083_r010_080	166.67	2	3600.00
short_n0084_r010_139	256.41	5	3600.00
short_n0093_r015_090	100.00	3	3600.00
short_n0094_r010_090	172.73	3	3600.00
short_n0095_r015_139	143.86	5	3600.00
short_n0120_r015_120	300.00	3	3600.00
short_n0129_r015_125	177.78	1	3600.00
short_n0132_r010_130	456.00	1	3600.00
short_n0135_r015_135	318.75	2	3600.00
short_n0159_r015_139	170.18	5	3600.00
short_n0174_r015_139	273.33	1	3600.00
short_n0202_r015_139	263.40	3	3600.00
short_n0238_r010_139	522.94	3	3600.00
short_n0284_r015_139	133.61	3	3600.00
short_n0365_r010_139	709.68	2	3600.00
short_n0391_r010_139	730.00	2	3600.00

Continue on next page...

Table D.3 (continued).

Instance Name	GAP%	# feasible	Time
short_n0405_r010_139	590.00	1	3600.00
short_n0461_r010_139	590.00	2	3600.00
short_n0596_r010_139	755.77	1	3600.00
short_n0647_r010_139	590.00	1	3600.00
Average	692.65	2	3561.85

Table D.4: Results for model LC in 3,600 seconds wall-clock. "GAP%" indicates the best difference between lower and upper bounds. "# feasible" is the amount of found legal solutions, and "Time", the time in seconds to obtain an optimal solution. If not found, then it is equal to the maximal time.

Instance Name	GAP%	# feasible	Time
long_n0030_r030_150	0.00	5	697.81
long_n0033_r020_160	0.00	8	450.77
long_n0038_r020_190	0.00	7	91.32
long_n0039_r020_150	0.00	8	868.91
long_n0040_r040_200	0.00	3	20.65
long_n0043_r020_170	10.71	8	3600.00
long_n0045_r030_180	0.00	15	451.04
long_n0047_r020_180	3.45	8	3600.00
long_n0051_r030_250	26.81	7	3600.00
long_n0052_r020_100	0.00	4	9.81
long_n0059_r020_230	26.97	5	3600.00
long_n0060_r030_240	14.27	12	3600.00
long_n0061_r030_300	25.00	12	3600.00
long_n0062_r020_310	0.00	5	1268.66
long_n0065_r020_260	10.56	6	3600.00
long_n0066_r030_330	8.53	10	3600.00
long_n0067_r040_260	0.00	3	667.79
long_n0068_r020_130	0.00	3	35.07
long_n0069_r030_340	32.35	7	3600.00
long_n0070_r040_280	2.13	10	3600.00
long_n0073_r040_360	30.79	6	3600.00
long_n0077_r040_380	37.87	7	3600.00
long_n0081_r030_400	26.18	21	3600.00

Continue on next page...

Table D.4 (continued).

Instance Name	GAP%	# feasible	Time
long_n0082_r040_320	12.66	14	3600.00
long_n0083_r020_160	12.43	12	3600.00
long_n0083_r040_330	6.81	12	3600.00
long_n0093_r030_180	22.71	12	3600.00
long_n0094_r040_470	31.37	7	3600.00
long_n0095_r020_190	19.27	13	3600.00
long_n0109_r030_540	31.84	5	3600.00
long_n0120_r030_240	20.82	11	3600.00
long_n0126_r020_250	18.31	9	3600.00
long_n0129_r030_250	23.39	1	3600.00
long_n0132_r020_260	38.46	2	3600.00
long_n0135_r030_270	38.79	7	3600.00
long_n0159_r040_310	0.00	19	1365.41
long_n0174_r030_340	18.30	18	3600.00
long_n0202_r040_400	15.19	17	3600.00
long_n0210_r040_420	38.78	21	3600.00
long_n0259_r030_510	37.23	19	3600.00
long_n0284_r040_838	5.19	8	3600.00
long_n0327_r040_650	21.98	42	3600.00
long_n0340_r040_680	9.20	36	3600.00
long_n0360_r040_720	19.46	54	3600.00
long_n0365_r040_730	13.25	22	3600.00
long_n0374_r040_740	26.30	37	3600.00
long_n0376_r040_750	25.98	36	3600.00
long_n0391_r040_838	14.45	44	3600.00
long_n0405_r040_838	11.49	35	3600.00
long_n0461_r040_838	18.74	39	3600.00
long_n0463_r040_838	21.89	42	3600.00
long_n0479_r040_838	39.61	22	3600.00
long_n0544_r040_838	35.05	42	3600.00
long_n0554_r040_838	24.65	20	3600.00
long_n0596_r040_838	22.44	48	3600.00
long_n0600_r030_838	32.52	31	3600.00
long_n0647_r040_838	18.03	65	3600.00
long_n0698_r040_838	18.43	47	3600.00

Continue on next page...

Table D.4 (continued).

Instance Name	GAP%	# feasible	Time
long_n0728_r040_838	35.85	38	3600.00
long_n0759_r040_838	49.00	3	3600.00
long_n1026_r030_838	18.46	22	3600.00
long_n1026_r040_838	47.29	3	3600.00
long_n1569_r040_838	53.52	11	3600.00
short_n0030_r010_060	0.00	5	2542.87
short_n0033_r010_080	0.00	8	2598.13
short_n0038_r010_095	0.00	7	37.14
short_n0039_r010_075	9.09	9	3600.00
short_n0040_r015_080	0.00	7	1117.59
short_n0043_r010_085	18.40	10	3600.00
short_n0045_r015_090	0.00	12	3199.85
short_n0047_r010_090	0.00	13	215.67
short_n0051_r015_125	30.94	8	3600.00
short_n0052_r010_050	0.00	8	1661.3
short_n0059_r010_115	25.70	13	3600.00
short_n0060_r015_120	24.18	15	3600.00
short_n0061_r015_139	27.13	20	3600.00
short_n0062_r010_139	20.59	4	3600.00
short_n0065_r010_139	26.47	12	3600.00
short_n0066_r015_139	27.91	19	3600.00
short_n0067_r010_065	10.34	8	3600.00
short_n0068_r020_135	0.00	10	1892.98
short_n0069_r010_135	28.95	14	3600.00
short_n0070_r020_139	8.33	13	3600.00
short_n0073_r015_139	21.74	19	3600.00
short_n0077_r015_139	31.34	17	3600.00
short_n0081_r010_139	8.57	22	3600.00
short_n0082_r010_080	2.94	19	3600.00
short_n0083_r010_080	5.12	11	3600.00
short_n0084_r010_139	17.91	31	3600.00
short_n0093_r015_090	19.64	22	3600.00
short_n0094_r010_090	32.56	17	3600.00
short_n0095_r015_139	0.00	19	3464.23
short_n0120_r015_120	38.89	21	3600.00

Continue on next page...

Table D.4 (continued).

Instance Name	GAP%	# feasible	Time
short_n0126_r015_139	39.02	11	3600.00
short_n0129_r015_125	38.19	10	3600.00
short_n0132_r010_130	39.80	6	3600.00
short_n0135_r015_135	38.03	12	3600.00
short_n0159_r015_139	0.00	14	2277.9
short_n0174_r015_139	14.34	36	3600.00
short_n0202_r015_139	18.51	20	3600.00
short_n0238_r010_139	22.05	48	3600.00
short_n0282_r010_139	26.59	24	3600.00
short_n0284_r015_139	26.06	49	3600.00
short_n0327_r010_139	17.65	61	3600.00
short_n0340_r015_139	33.51	11	3600.00
short_n0365_r010_139	6.12	65	3600.00
short_n0391_r010_139	13.56	55	3600.00
short_n0405_r010_139	34.07	43	3600.00
short_n0461_r010_139	5.21	41	3600.00
short_n0596_r010_139	28.37	95	3600.00
short_n0647_r010_139	34.88	100	3600.00
short_n0698_r010_139	23.62	104	3600.00
Average	18.94	19	3149.25

Table D.5: Results for model LS in 3,600 seconds wall-clock. "GAP%" indicates the best difference between lower and upper bounds. "# feasible" is the amount of found legal solutions, and "Time", the time in seconds to obtain an optimal solution. If not found, then it is equal to the maximal time.

Instance Name	GAP%	# feasible	Time
long_n0030_r030_150	0.00	1	473.24
long_n0033_r020_160	137.48	5	3600.00
long_n0038_r020_190	106.52	5	3600.00
long_n0039_r020_150	138.10	4	3600.00
long_n0040_r040_200	0.00	6	110.1
long_n0043_r020_170	169.84	2	3600.00
long_n0045_r030_180	0.00	1	1458.29
long_n0047_r020_180	104.55	3	3600.00

Continue on next page...

Table D.5 (continued).

Instance Name	GAP%	# feasible	Time
long_n0051_r030_250	168.82	1	3600.00
long_n0052_r020_100	0.00	3	489.72
long_n0059_r020_230	150.00	4	3600.00
long_n0060_r030_240	166.67	1	3600.00
long_n0061_r030_300	127.27	4	3600.00
long_n0062_r020_310	369.70	5	3600.00
long_n0065_r020_260	312.70	2	3600.00
long_n0066_r030_330	129.17	7	3600.00
long_n0067_r040_260	0.00	2	2977.31
long_n0068_r020_130	32.15	3	3600.00
long_n0069_r030_340	183.33	1	3600.00
long_n0070_r040_280	0.00	5	1240.2
long_n0073_r040_360	121.11	6	3600.00
long_n0077_r040_380	200.00	3	3600.00
long_n0081_r030_400	304.04	6	3600.00
long_n0082_r040_320	166.67	1	3600.00
long_n0083_r020_160	166.67	3	3600.00
long_n0083_r040_330	101.22	2	3600.00
long_n0084_r040_420	233.33	3	3600.00
long_n0093_r030_180	50.00	1	3600.00
long_n0094_r040_470	233.33	8	3600.00
long_n0095_r020_190	134.57	10	3600.00
long_n0109_r030_540	650.00	7	3600.00
long_n0114_r030_450	650.00	1	3600.00
long_n0120_r030_240	220.00	3	3600.00
long_n0122_r040_610	598.33	1	3600.00
long_n0126_r020_250	220.51	8	3600.00
long_n0129_r030_250	254.84	2	3600.00
long_n0132_r020_260	276.81	4	3600.00
long_n0135_r030_270	300.00	1	3600.00
long_n0159_r040_310	76.14	7	3600.00
long_n0162_r020_320	700.00	1	3600.00
long_n0174_r030_340	233.33	5	3600.00
long_n0202_r040_400	166.67	12	3600.00
long_n0210_r040_420	412.20	4	3600.00

Continue on next page...

Table D.5 (continued).

Instance Name	GAP%	# feasible	Time
long_n0238_r040_470	446.51	4	3600.00
long_n0284_r040_838	173.86	50	3600.00
long_n0297_r030_590	883.33	1	3600.00
long_n0327_r040_650	673.81	5	3600.00
long_n0340_r040_680	593.88	11	3600.00
long_n0360_r040_720	495.00	3	3600.00
long_n0365_r040_730	1028.42	6	3600.00
long_n0374_r040_740	825.00	3	3600.00
long_n0376_r040_750	814.63	4	3600.00
long_n0391_r040_838	1221.85	10	3600.00
long_n0405_r040_838	981.12	11	3600.00
long_n0461_r040_838	831.11	8	3600.00
long_n0463_r040_838	1189.29	8	3600.00
long_n0479_r040_838	1470.00	3	3600.00
long_n0544_r040_838	1431.71	4	3600.00
long_n0554_r040_838	1347.39	3	3600.00
long_n0596_r040_838	1360.47	6	3600.00
long_n0600_r030_838	1296.67	1	3600.00
long_n0647_r040_838	1395.24	5	3600.00
long_n0698_r040_838	1360.47	6	3600.00
long_n0728_r040_838	1470.00	3	3600.00
long_n0759_r040_838	1470.00	1	3600.00
long_n1026_r030_838	1293.33	1	3600.00
long_n1026_r040_838	945.00	3	3600.00
long_n2036_r030_838	2690.00	3	3600.00
short_n0030_r010_060	20.00	1	3600.00
short_n0033_r010_080	45.45	2	3600.00
short_n0038_r010_095	43.94	2	3600.00
short_n0039_r010_075	50.00	1	3600.00
short_n0040_r015_080	0.00	3	102.2
short_n0043_r010_085	70.00	1	3600.00
short_n0045_r015_090	50.00	1	3600.00
short_n0047_r010_090	36.36	2	3600.00
short_n0051_r015_125	177.78	1	3600.00
short_n0052_r010_050	20.00	1	3600.00

Continue on next page...

Table D.5 (continued).

Instance Name	GAP%	# feasible	Time
short_n0059_r010_115	248.48	2	3600.00
short_n0060_r015_120	100.00	3	3600.00
short_n0061_r015_139	131.67	1	3600.00
short_n0062_r010_139	360.00	1	3600.00
short_n0065_r010_139	305.41	2	3600.00
short_n0066_r015_139	126.67	1	3600.00
short_n0067_r010_065	0.00	1	2018.22
short_n0068_r020_135	0.00	2	895.68
short_n0069_r010_135	309.09	2	3600.00
short_n0070_r020_139	0.00	3	812.09
short_n0073_r015_139	170.59	3	3600.00
short_n0077_r015_139	206.67	1	3600.00
short_n0081_r010_139	321.21	4	3600.00
short_n0082_r010_080	166.67	1	3600.00
short_n0083_r010_080	290.00	3	3600.00
short_n0084_r010_139	384.48	6	3600.00
short_n0093_r015_090	50.00	1	3600.00
short_n0094_r010_090	104.55	4	3600.00
short_n0095_r015_139	143.86	5	3600.00
short_n0120_r015_120	100.00	3	3600.00
short_n0126_r015_139	186.67	1	3600.00
short_n0129_r015_125	161.11	1	3600.00
short_n0132_r010_130	490.91	4	3600.00
short_n0135_r015_135	200.00	1	3600.00
short_n0159_r015_139	143.86	5	3600.00
short_n0174_r015_139	346.67	1	3600.00
short_n0202_r015_139	172.55	3	3600.00
short_n0238_r010_139	434.62	6	3600.00
short_n0282_r010_139	363.33	1	3600.00
short_n0284_r015_139	172.55	5	3600.00
short_n0327_r010_139	589.88	3	3600.00
short_n0340_r015_139	286.67	1	3600.00
short_n0365_r010_139	527.27	2	3600.00
short_n0391_r010_139	813.45	1	3600.00
short_n0405_r010_139	680.00	1	3600.00

Continue on next page...

Table D.5 (continued).

Instance Name	**GAP%**	**# feasible**	**Time**
short_n0461_r010_139	590.00	1	3600.00
short_n0596_r010_139	595.00	1	3600.00
short_n0647_r010_139	790.00	1	3600.00
short_n0698_r010_139	760.00	3	3600.00
Average	417.02	4	3383.28