sid.inpe.br/mtc-m21c/2020/12.03.19.26-TDI

METHODS FOR OVERLAPPING CLUSTERING OPTIMIZATION PROBLEMS

Guilherme Oliveira Chagas

Doctorate Thesis of the Graduate Course in Applied Computing, guided by Drs. Rafael Duarte Coelho dos Santos and Luiz Antonio Nogueira Lorena, approved in November 30, 2020.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/43MTEC5>

INPE
São José dos Campos
2020

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GBDIR)
Serviço de Informação e Documentação (SESID)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21c/2020/12.03.19.26-TDI

METHODS FOR OVERLAPPING CLUSTERING OPTIMIZATION PROBLEMS

Guilherme Oliveira Chagas

Doctorate Thesis of the Graduate Course in Applied Computing, guided by Drs. Rafael Duarte Coelho dos Santos and Luiz Antonio Nogueira Lorena, approved in November 30, 2020.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34R/43MTEC5>

INPE
São José dos Campos
2020

Cataloging in Publication Data

Chagas, Guilherme Oliveira.

C346m Methods for overlapping clustering optimization problems / Guilherme Oliveira Chagas. – São José dos Campos : INPE, 2020.

xxvi + 147 p. ; (sid.inpe.br/mtc-m21c/2020/12.03.19.26-TDI)

Thesis (Doctorate in Applied Computing) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2020.

Guiding : Drs. Rafael Duarte Coelho dos Santos and Luiz Antonio Nogueira Lorena.

1. Overlapping clustering. 2. Overlap control. 3. Community detection. 4. Multiple assignment. 5. Hybrid heuristic. I.Title.

CDU 004.023

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DEFESA FINAL DE TESE DE GUILHERME OLIVEIRA CHAGAS

No dia 30 de novembro de 2020, às 10h, por videoconferência, o(a) aluno(a) mencionado(a) acima defendeu seu trabalho final (apresentação oral seguida de arguição) perante uma Banca Examinadora, cujos membros estão listados abaixo. O(A) aluno(a) foi APROVADO(A) pela Banca Examinadora por unanimidade, em cumprimento ao requisito exigido para obtenção do Título de Doutor em Computação Aplicada. O trabalho precisa da incorporação das correções sugeridas pela Banca Examinadora e revisão final pelo(s) orientador(es).

Título: "METHODS FOR OVERLAPPING CLUSTERING OPTIMIZATION PROBLEMS"

Eu, Thales Sehn Korting, como Presidente da Banca Examinadora, assino esta ATA em nome de todos os membros.

Membros da Banca

Dr. Thales Sehn Korting, Presidente INPE.
Dr. Luiz Antônio Nogueira Lorena, Orientador(a) INPE
Dr. Rafael Duarte Coelho dos Santos, Orientador(a) INPE
Dr. Gilberto Ribeiro de Queiroz, Membro da Banca, INPE
Dr. Antonio Augusto Chaves, Convidado(a) UNIFESP
Dr. Leandro Callegari Coelho, Convidado(a) UNIV. LAVAL

Documento assinado eletronicamente por **Thales Sehn Korting**, **Pesquisador**, em 01/12/2020, às 11:17 (horário oficial de Brasília), com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.mctic.gov.br/verifica.html, informando o código verificador **6164069** e o código CRC **6B322097**.

Referência: Processo nº 01340.008250/2020-51 SEI nº 6164069

"Nobody exists on purpose. Nobody belongs anywhere. Everybody's gonna die. Come watch TV."

Morty Smith
from *"Rick and Morty"*

To my parents ***Zaqueu*** *and* ***Irene****, my brother* ***Gabriel*** *and my wife* ***Daniele***

ACKNOWLEDGEMENTS

In portuguese.

Acima de tudo, gostaria de agradecer às pessoas mais importantes da minha vida, meus pais. Obrigado, Irene e Zaqueu, sem o sacrifício de vocês, eu não teria terminado nem a pré-escola.

Ao meu amado irmão Gabriel, por todo apoio e incentivo. A minha esposa Daniele, que torna a vida até que legal de se viver. Seu apoio foi e é essencial durante todo esse processo.

Aos meus orientadores, professores Luiz Lorena e Rafael Santos, por toda ajuda, ensinamentos e paciência. Também, aos professores Gilberto Ribeiro e Leandro Coelho, que, além de me ensinarem muita coisa, foram amigos que fiz durante essa jornada.

Aos meus amigos de longa data, em especial, Cláudio, Dérik, Francisco, Igor, Luiz Fernando, Matheus Lima, Matheus Luís, Marcello, Sarah, Rennan e Richard.

Aos meus amigos do INPE, Adriano e Felipe. Também, a todos os funcionários do INPE, em especial a Glória.

Às bandas Blink-182, CBJR, Forfun, Fresno, LP, NOFX, Rancore, R.Sigma e SOAD que, junto com o café, foram os combustíveis para fazer este trabalho.

Ao Zack Snyder, meu deus que criou o milagre cinematográfico chamado Batman V Superman.

Às políticas públicas de universilização do acesso à educação superior, que possibilitaram que um filho de um encanador e de uma faxineira cursasse uma graduação, um mestrado e um doutorado (e ainda ter ido para a gringa). Apesar de você, amanhã há de ser outro dia.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES).

ABSTRACT

Clustering problems arise from several areas of science. Approaches and algorithms are as varied as the applications. The goal of clustering is to partition a set of elements into disjoint subsets, also known as clusters, according to a similarity metric's values. In many real-world applications, however, vertices can belong to more than one cluster, i.e., clusters may overlap. Identifying such overlapping clusters is usually a less studied problem and a more challenging task than finding non-overlapping ones. Thus, in this work, overlapping clustering problems from four different contexts are explored. First, it is introduced the overlapping cluster editing, a new relaxation of the cluster editing problem. Three hybrid heuristics were developed to generate solutions for this problem, which are composed of coupling metaheuristics and mixed-integer linear programs. The second work introduces a hybrid heuristic for the overlapping community detection problem, where the objective is to identify overlapping clusters from an input network. This is achieved by solving a mixed-integer linear program using, as input, a heterogeneous set of clusters generated by two state-of-the-art overlapping community detection algorithms. In the third work, the p-median problem with overlap control is introduced. This problem is a variation of the well-known p-median problem, where the objective is to select p facilities vertices whereas the sum of the distances from each client vertex to its nearest facility is minimized. In the p-median problem with overlap control, the number of vertices shared between facilities can be managed from a user-defined parameter. A parallel branch-and-price method was developed to solve this problem. In the fourth work, a parallel adaptive large neighborhood search metaheuristic was proposed to solve some facility location problems with multiple assignments. Several tests results in all problems show that all proposed methods can generate good-quality overlapping clustering solutions.

Keywords: Overlapping clustering. Overlap control. Community detection. Multiple assignment. Hybrid heuristic. Branch-and-price

MÉTODOS PARA PROBLEMAS DE OTIMIZAÇÃO DE AGRUPAMENTOS COM SOBREPOSIÇÃO

RESUMO

Problemas de agrupamento são encontrados em várias áreas da ciência e abordagens e algoritmos são tão variados quanto as aplicações. O objetivo em um problema de agrupamento é particionar um conjunto de elementos em subconjuntos disjuntos, também conhecidos como *clusters*. Entretanto, em muitas aplicações de problemas reais, elementos podem pertencer a mais de um *cluster*, isto é, os *clusters* podem se sobrepor. Identificar tais *clusters* sobrepostos é, em geral, um problema menos estudado e mais difícil que o problema original. Então, neste trabalho, problemas de agrupamento com sobreposição, de quatro contextos diferentes, são explorados. No primeiro contexto, é introduzido o problema de edição de *clusters* com sobreposição, uma nova relaxação do problema de edição de *clusters*. Três heurísticas híbridas foram desenvolvidas para gerar soluçoes para o problema proposto, as quais são combinações de meta-heurísticas e problemas lineares inteiros mistos. Introduz-se, no segundo trabalho, uma heurística híbrida para o problema de detecção de comunidades com sobreposição. Essa heurística híbrida é composta de um problema linear inteiro misto que recebe, como entrada, um conjunto de *clusters* gerado por duas heurísticas no estado da arte de detecção de comunidades. No terceiro contexto, o problema de p-medianas com controle de sobreposição é introduzido. Esse problema é uma variação do problema de p-medianas. No problema de p-medianas, o número de vértices compartilhados entre as facilidades pode ser controlado por um parâmetro de entrada. Um algoritmo paralelo de *branch-and-price* foi implementado para resolver esse problema. No quarto contexto, uma meta-heurística *Adaptive Large Neighborhood Search* paralela foi aplicada a três problemas de localização de facilidades com multi-designação. Vários testes foram realizados em todos os quatro contextos e os métodos propostos puderam gerar boas soluções de agrupamento com sobreposição.

Palavras-chave: Agrupamento com sobreposição. Controle de sobreposição. Detecção de comunidades. Multi-designação. Heurística híbrida. Branch-and-price.

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

ALNS	–	Adaptive large neighborhood search metaheuristic
BIMM	–	Biclustering multiple median algorithm
B&P	–	Branch-and-price algorithm
BRKGA	–	Biased random-key genetic algorithm metaheuristic
CG	–	Column generation algorithm
CLP	–	Coverage location problem
CMPMP	–	Multiple assignment CPMP
CPMP	–	Capacitated PMP
CR	–	Cluster refinement local search methods
GA	–	Genetic algorithm metaheuristic
GMAP	–	Generalized multi-assignment problem
GNMI	–	Generalized normalized mutual information
HH	–	Hybrid heuristic
HH-CR	–	Hybrid heuristic with CR
HHM1	–	Hybrid heuristic composed of the metaheuristics and M1
HHM2	–	Hybrid heuristic composed of the metaheuristics and M2
HHM3	–	Hybrid heuristic composed of the metaheuristics and M3
LFM	–	Local fitness maximization algorithm
LP	–	Linear program
ILP	–	Integer linear program
LECM	–	Local expansion conductance minimization
M1	–	First MILP for the overlapping cluster editing problem
M2	–	Second MILP for the overlapping cluster editing problem
M3	–	Third MILP for the overlapping cluster editing problem
MCNFP	–	Minimum-cost network flow problem
MILP	–	Mixed-integer linear program
MP	–	Master problem
MPCP	–	Multiple assignment PCP
MPMP	–	Multiple assignment PMP
NISE	–	Neighborhood-inflated seed expansion algorithm
OCC	–	Overlap control constraint
OCDP	–	Overlapping community detection problem
OCEP	–	Overlapping cluster editing problem
OCM	–	Overlapping clustering model
PALNS	–	Parallel ALNS metaheuristic
PCP	–	p-center problem
PMP	–	p-median problem
PMPOC	–	PMP with overlap control
RMP	–	Restricted master problem
SA	–	Simulated annealing metaheuristic

CONTENTS

Page

1 INTRODUCTION

Clustering is one of the best-known problems in data mining and has applications in several science areas such as bioinformatics, computer vision, multimedia data analysis, facility location problems, data compression, marketing, pattern recognition, community detection and machine learning (BEN-DOR et al., 1999; BANSAL et al., 2004; SHAMIR et al., 2004; DEMAINE et al., 2006; BÖCKER et al., 2009; AGGARWAL, 2013). The goal of clustering is to partition a set of elements into subsets, also known as clusters, according to the values of a given metric. Thus, elements in the same subset are more similar to each other than elements belonging to different subsets (SHAMIR et al., 2004; CHAGAS et al., 2019).

Due to the variety of applications of clustering problems, a vast number of clustering algorithms have been proposed over the years, and there is no single method or technique suitable to all contexts (XU; WUNSCH II, 2005; LI et al., 2017). However, graph theory is a widespread approach used to model these problems and to obtain reasonable good quality solutions (SHAMIR et al., 2004; GUO et al., 2009). Schaeffer (2007) presented a survey of graph clustering and cited several areas, from bioinformatics to stock market, in which graph theory is used for data clustering. Given an unweighted graph, a clustering problem can be modeled by considering elements as vertices and the edges between them are based on a measure of similarity. If the similarity value between two elements is larger (or smaller) than a threshold, then the vertices that represent these two elements are adjacent (CHAGAS et al., 2019).

However, many real-world clustering problems are characterized by overlapping clusters, that is, clusters that are non-disjoint. For instance, in online social networks users are naturally assigned to multiple cluster memberships (XIE et al., 2013). In a biological context, proteins may belong to several protein complexes (PALLA et al., 2005a). In information retrieval and text mining, documents, articles and web pages are classified to one or more categories (BONCHI et al., 2011; BONCHI et al., 2013). Applications of overlapping clustering can also be found in distributed computing (ANDERSEN et al., 2012) and distributed model transformations (BENELALLAM et al., 2016). Maiza et al. (2016) and Pérez-Suárez et al. (2013) cite other areas where the overlapping clustering is important, such as image and video processing (CHAGAS et al., 2019).

In general, overlapping clustering problems are not as explored as the non-overlapping ones and there are some unexplored topics in literature (XIE et al.,

2013; KHANMOHAMMADI et al., 2017; CHAGAS et al., 2019). This work studies some of these subjects which are organized as follows. In Chapter 2, it is proposed the overlapping cluster editing problem, a variation of the cluster editing where the goal is to partition a graph, by editing edges, into *maximal cliques* that are not necessarily disjoint. In addition, we also present three slightly different versions of a hybrid heuristic to solve this problem. Each hybrid heuristic is based on coupling two metaheuristics that, together, generate a set of clusters; and one of three mixed-integer linear programming models, also introduced in this work, that uses these clusters as input. Tests results show that the all proposed hybrid heuristic versions are able to generate good-quality overlapping cluster editing solutions. In particular, one version of the hybrid heuristic achieved, at a low computational cost, the best results in 51 of 112 randomly-generated graphs. Although the other two hybrid heuristic versions have harder to solve models, they obtained reasonable results in medium-sized randomly-generated graphs. In addition, the hybrid heuristic achieved good results identifying labeled overlapping clusters in a supervised data set experiment. Furthermore, we also show that, with our new problem definition, clustering a vertex in more than one cluster can reduce the edges editing cost.

In Chapter 3, it is introduced a hybrid heuristic for detecting overlapping clusters in networks. An overlapping clustering is generated through the solving of a mixed-integer linear program using, as input, a heterogeneous set of good-quality clusters. This set is produced by two state-of-the-art overlapping community detection algorithms. In addition, some local search methods for conductance minimization are used to improve the quality of the clustering generate by our hybrid heuristic. Test results in artificial and real-world graphs show that our approach is able to detect overlapping clusters with better overall conductance than methods in the state-of-the-art.

Chapter 4 introduces the p-median problem with overlap control, which, from a user-defined parameter, can manage the number of vertices shared between facilities. Furthermore, a parallel branch-and-price algorithm is developed to solve this problem. Through a series of computational experiments, we shown that our approach can generate good quality solutions at reasonable execution time.

In Chapter 5, it is presented the capacitated multiple p-median problem and the multiple p-center problem, two extensions of two classical facility location problems where every client must be served by at least mc facilities. We proposed an efficient parallel adaptive large neighborhood search to solve both problems. We also applied

our method in the multiple p-median problem. Several experimental tests show that the metaheuristic performed consistently. Considering all problems, the proposed method found the best known solutions in 76% of the instances.

Concluding remarks and considerations on future work are presented in Chapter 6.

2 A HYBRID HEURISTIC FOR THE OVERLAPPING CLUSTER EDITING PROBLEM[1]

This chapter is divided as follows. An introduction is presented in Section 2.1. Some related work are described in Section 2.2. The mathematical background and the definitions of the cluster editing problem and of the proposed overlapping cluster editing problem are presented in Section 2.3. The proposed hybrid heuristic is detailed in Section 2.4. Section 2.5 shows the tests results. Our concluding remarks and considerations on future work are presented in Section 2.6.

2.1 Introduction

A cluster can be interpreted as vertices that are highly connected, that is, a *dense subgraph* or even a *complete subgraph* (*clique*). A graph is complete if each pair of vertices is adjacent. In this sense, partitioning the vertices of an input graph into a disjoint union of cliques, by adding and deleting edges, can be considered as clustering. Finding the minimum number of edges edition (addition and deletion) is a well-known combinatorial optimization problem referred to as the *cluster editing problem* (SHAMIR et al., 2004). In particular, this problem belongs to the class of *edge modification problems* (NATANZON et al., 2001; SHAMIR et al., 2004). As Fellows et al. (2009), Fellows et al. (2011) state, the cluster editing problem is, probably, the most studied edge modification problem and it has applications, mainly, in bioinformatics, specially in gene expression (BEN-DOR et al., 1999; CHESLER et al., 2005; JIANG; PEI, 2009). This problem has also application in clustering entity names (BANSAL et al., 2004; CHIERICHETTI et al., 2014) and has been used as inspiration for clustering algorithms (BÖCKER; BAUMBACH, 2013).

Modifying the edges set of an input graph so that it becomes a vertex-disjoint union of cliques by the minimum number of edges edition is a NP-hard problem. The NP-hardness of the cluster editing problem was proved, independently, by Delvaux and Horsten (2004), Shamir et al. (2004) and Bansal et al. (2004). Then, several heuristics (WITTKOP et al., 2007; BASTOS et al., 2016), exact methods (BÖCKER et al., 2011; BÖCKER; BAUMBACH, 2013; LORENA et al., 2018) and theoretical studies (KOMUSIEWICZ; UHLMANN, 2012; DAMASCHKE; MOGREN, 2014) regarding this problem are found in the literature.

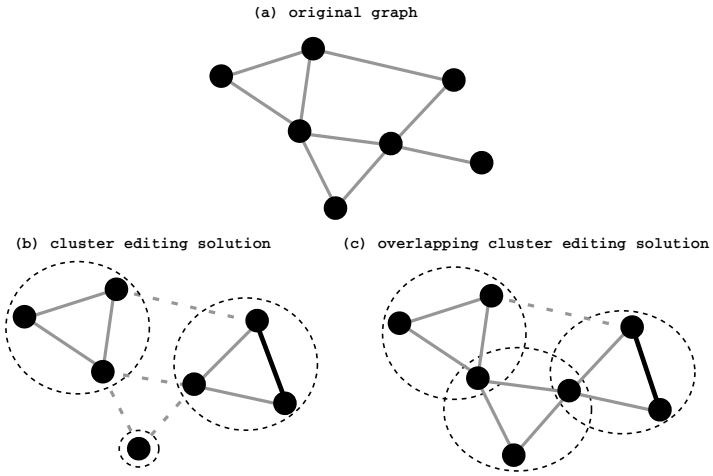
However, the definition of the cluster editing is unable to model these problems where clusters may overlap and, for this reason, has been criticized in the literature

[1]This chapter is an adapted version of the paper: Chagas et al. (2019).

(DEHNE et al., 2006; FELLOWS et al., 2011; MAIZA et al., 2016). Based on this, it is necessary to relax the definition of the cluster editing in a manner which allows clusters to share vertices. Thus, we introduce in this work the *overlapping cluster editing problem* (OCEP), where the aim is to partition a graph, by the smallest possible number of edges' addition and deletion, into *maximal cliques* that are not necessarily disjoint. A clique is maximal if it is not strictly contained in a larger clique. In our problem definition, there is no limit either on the number of clusters that each vertex is contained in or on the number of vertices that each cluster intersects with other clusters.

Figure 2.1 shows an example of cluster editing and overlapping cluster editing solutions of a same graph. In this figure, removed edges are represented by dotted lines and added edges are represented by thick ones. The cost of the cluster editing solution is five, since four edges were removed and one was added. Analogously, the cost of the overlapping cluster editing solution is two.

Figure 2.1 - An example of cluster editing and overlapping clustering editing solutions of a same input graph. In this figure, removed edges are represented by dotted lines and added edges are represented by bold ones. The original graph is represented by Figure 1*a*. A cluster editing solution, of cost five, is depicted by Figure 1*b*. Figure 1*c* shows an overlapping cluster editing solution of cost two.



SOURCE: Produced by the author.

As the cluster editing is a NP-hard problem, exact methods are only practical in instances with few vertices. For instances with a large number of vertices, heuristics are commonly used to generate solutions at a low computational execution

time. However, the solution optimality is not guaranteed, hence, *hybrid heuristics*, also known as *matheuristics* (MANIEZZO et al., 2009), are alternatives to produce good-quality solutions with reasonable computation cost. Hybrid heuristics are composed by coupling exact methods and metaheuristics. Basically, in most of matheuristics' implementations, the metaheuristic acts by defining the boundaries of the solution space and the exploration of this space itself is done by the exact method (MANIEZZO et al., 2009). Matheuristics have been used successfully in current combinatorial optimization research (see, e.g., (PEREIRA et al., 2015; WANG et al., 2017)). To address the proposed overlapping cluster editing problem, we also introduce a hybrid heuristic in this chapter.

Our hybrid heuristic is based on coupling two metaheuristics to one of three *mixed-integer linear program* (MILP), which are also introduced in this work. These two metaheuristics are simultaneously used to generate a set of clusters through the resolution of the cluster editing problem. Thereafter, one of three MILP is solved by using this cluster set as input. The objective with these metaheuristics is to provide a size-limited but diversified set of good quality clusters in order to limit the exploration of solution space while solving one of our mixed-integer linear programs. An overlapping cluster editing solution is obtained as result.

2.2 Related work

In this section, some literature review and related works are presented. Overlapping clustering problems approaches and cluster editing relaxations, where overlapping is allowed, are described in Subsection 2.2.1. Subsection 2.2.2 presents a briefly overview of metaheuristics applied to clustering problems.

2.2.1 Overlapping clustering

The literature regarding overlapping clustering approaches is ample (LEVIN, 2015; BEN N'CIR et al., 2015; BAADEL et al., 2016). As Ben N'Cir et al. (2015) state, the majority of these approaches are extensions of non-overlapping clustering ones. Overlapping clustering methods are found in the context of the classical disjoint clustering approaches such as hierarchical clustering, partitional clustering, generative clustering and graph-based clustering (BEN N'CIR et al., 2015). Some examples of overlapping clustering methods based on each of these approaches are shown in (BAADEL et al., 2015; BERTRAND; JANOWITZ, 2003; FU; BANERJEE, 2008; PÉREZ-SUÁREZ et al., 2013), respectively.

Graph-based clustering methods are also proposed for community detection (LEVIN, 2015; BEN N'CIR et al., 2015). Xie et al. (2013) categorized overlapping methods for this task in five main classes: clique percolation, line graph/link partitioning, local expansion/optimization, fuzzy detection and agent-based/dynamical algorithms.

As Xie et al. (2013) explain, clique percolation methods, e.g. (KUMPULA et al., 2008), are characterized by the identification of complete *connected* subgraphs, where a graph is connected if there is a finite sequence of edges connecting every pair of vertices. In line graph methods, e.g. (AHN et al., 2010), clusters are formed considering edges instead of vertices (XIE et al., 2013). Algorithms based on local expansion, e.g. (LANCICHINETTI et al., 2009), starts by growing a cluster from a seed vertex considering its *neighborhood* or some related metric (XIE et al., 2013). The neighborhood of a vertex is composed by all its adjacent vertices. Xie et al. (2013) categorized algorithms as fuzzy, e.g. (NEPUSZ et al., 2008), when the clustering performed by these algorithms is not crisp, i.e., vertices have a degree of membership related to each cluster. Agent-based and dynamical methods, e.g. (XIE; SZYMANSKI, 2012), are characterized by dynamically associating a set of labels, which represents the clusters, at each vertex considering some metric (XIE et al., 2013).

In spite of all these overlapping clustering approaches, only few studies regarding the overlapping concept as a relaxed constraint of the cluster editing problem can be found (FELLOWS et al., 2009; FELLOWS et al., 2011). Indeed, a small number of relaxations of cluster editing are found in literature, some examples are refs. (GUO et al., 2010; GUO et al., 2011; HEGGERNES et al., 2010; LIU et al., 2012). To the best of our knowledge, the only studies that consider overlapping clustering as a variation of cluster editing are the works of Barthélemy and Brucker (2001), Damaschke (2010), Fellows et al. (2009), Fellows et al. (2011) and Bonchi et al. (2011), Bonchi et al. (2013).

Barthélemy and Brucker (2001) proposed the *w-Zahn clustering problem*, which is a generalized version of the Zahn problem (ZAHN, 1964). In the w-Zahn clustering problem one must partition a graph, by editing edges, into maximal cliques so that each pair of cliques share at most $w - 1$ vertices. Note that, when $w = 1$, we have the cluster editing problem. The difference between the w-Zahn clustering and the overlapping cluster editing, proposed in this work, is that in our definition the number of vertices that each pair of cliques share is not limited.

The relaxed version of cluster editing presented by Damaschke (2010) was the *o-twin graph editing problem*. In that problem the goal is to generate, by editing edges,

an *o-twin graph* from an input graph. A twin graph, also known as *critical clique graph*, is a graph where each vertex represent a critical clique from the original graph (FELLOWS et al., 2009; FELLOWS et al., 2011). A clique is critical when all vertices belonging to it have the same neighborhood. Two vertices in the twin graph are adjacent if and only if edges exist between the corresponding critical cliques in the input graph (DAMASCHKE, 2010). Thus, an o-twin graph is a twin graph with at most o edges.

Fellows et al. (2011) introduced two cluster editing relaxations: the o-vertex-overlap and the o-edge-overlap problems. In both one must partitioning the graph, by edges edition, into maximal cliques. In addition, each vertex, in the o-vertex-overlap problem, or edge, in the o-edge-overlap problem, are contained in at most o maximal cliques. Fellows et al. (2011) proved that these problems are NP-hard when $o \geq 1$. Note that, when $o = 1$, we have the cluster editing problem. As the variation of cluster editing proposed by Barthélemy and Brucker (2001) the relaxation proposed by Fellows et al. (2011) is also different from ours. This is because in the overlapping cluster editing there is no limit in the number of clusters that each vertex can belongs to.

Bonchi et al. (2013) presented the *overlapping correlation clustering*, a variation of the *correlation clustering* problem (BANSAL et al., 2004) where overlapping clusters are possible. In this variation every edge is associated with a weight in the real interval $[0, 1]$ instead of being weighted either positive or negative. Furthermore, the authors utilized two measures to evaluate the similarity between the set of clusters that each pair of vertices are contained, namely the Jaccard coefficient and a set-intersection indicator function (BONCHI et al., 2011; BONCHI et al., 2013). Then, the objective in the overlapping correlation clustering problem is to minimize, for each pair of vertices, the absolute difference between the edge weight and the value of the similarity measure of the set of clusters that each vertex belongs to. Bonchi et al. (2013) showed that this is a NP-hard problem and presented a local-search algorithm to solve it.

Bonchi et al. (2013) also proposed a local-search heuristic for the overlapping correlation clustering. Canisius et al. (2016) used a modification of this algorithm to detect sets of mutually exclusive cancer cells genes.

2.2.2 Metaheuristics in clustering problems

Metaheuristics have been applied, with success, to several problems in the literature (ALJARAH et al., 2018; FARRIS et al., 2018; FARRIS et al., 2019; HEIDARI et al., 2017; HEIDARI; PAHLAVANI, 2017; MAFARJA et al., 2018; MAFARJA et al., 2019). In the context of clustering problems, they also have been largely utilized (DAS et al., 2009). Several review papers regarding the use of metaheuristics in clustering problems can be found in the literature. For instance, José-García and Gómez-Flores (2016) presented a survey of nature-inspired metaheuristics for automatic clustering. In this clustering problem, the number of clusters is not known beforehand. Nanda and Panda (2014) also surveyed the use of nature-inspired metaheuristics but in the partitional clustering problem. In addition, a review of *Particle Swarm Optimization* (PSO) metaheuristics applied to clustering problems was presented by Rana et al. (2011). A survey of evolutionary metaheuristics for overlapping clustering and partitional clustering problems was realized by Hruschka et al. (2009).

One can also find metaheuristics applied in the community detection scope. For example, Atay et al. (2017) proposed and compared six metaheuristics for this task. Qu (2013) proposed a hybrid PSO with *Extremal Optimization* (EO) for finding community in networks. Furthermore, considering the overlapping community detection problem, Imane and Nadjet (2016) presented a Hybrid Bat algorithm with Tabu search for this problem.

In the context of cluster editing problem, Bastos et al. (2016) proposed two heuristics for this problem. The first one is a *Iterated Local Seach* (ILS) metaheuristic and the second one is a *Greedy Randomized Adaptive Search Procedure* (GRASP) metaheuristic. Both are coupled to an exact method based on *Set Paritioning problem* to generate cluster editing solutions (BASTOS et al., 2016). In addition, Filho et al. (2012) presented a hybrid metaheuristic for the Bicluster Editing Problem. This problem consists of editing edges of an input bipartite graph so that this graph becomes a disjoint union of complete bipartite subgraphs (FILHO et al., 2012). The proposed hybrid metaheuristic is formed by the hybridization of a GRASP and a *Variable Neighborhood Search* metaheuristics.

As far as we know, the only study that applies a metaheuristic in the context of an overlap relaxation of the cluster editing problem is the work of Andrade et al. (2014). These authors presented an heuristic for the overlapping correlation clustering based on Biased Random-Key Genetic Algorithm (BRKGA) metaheuristic (GONÇALVES; RESENDE, 2011). The authors achieved good results in the com-

parsion of their heuristic's results with the Bonchi et al. (2013) algorithm.

2.3 Mathematical notation and problem definition

Let $G = (V, E)$ be a simple, undirected and unweighted graph, where V is the set of vertices, E is the set of edges, $n = |V|$ and $m = |E|$. Two vertices $v, u \in V$ are *adjacent* if and only if $\{v, u\} \in E$. A graph G is *complete* if and only if $\forall v \in V$ and $\forall u \in V$, where $v \neq u$, $\{v, u\} \in E$. In complete a graph, $m = \frac{n \cdot (n-1)}{2}$. A subgraph of G *induced* by a subset of vertices $U \subseteq V$ is a graph $G_U = (U, E_U)$, where $\forall v \in U$ and $\forall u \in U$, $\{v, u\} \in E_U$ if and only if $\{v, u\} \in E$. A subset of vertices $U \subseteq V$ is a *clique* if the subgraph of G induced by U, G_U, is complete. In addition, a clique is *maximal* if it is not strictly contained in a larger clique.

Two sets A and B are *disjoint* sets if $A \cap B = \emptyset$. The *symmetric difference* of two sets A and B is given by $A \Delta B = \{(A - B) \cup (B - A)\}$. The *Jaccard coefficient*, between two sets A e B, is defined by $J(A, B) = |A \cap B|/|A \cup B|$. Then, two sets A and B are equal when $J(A, B) = 1$. If $J(A, B) = 0$, then A and B have no elements in common, i.e., $A \cap B = \emptyset$.

A graph G is a *cluster graph* if G is a disjoint union of cliques (SHAMIR et al., 2004). In this work, however, a cluster C is a vertex subset of G, that is, $C \subseteq V$. Note that a cluster is not necessarily a disjoint clique. A clustering is a vertex partitioning $\mathcal{C} = \{C_1, C_2, \ldots, C_l\}$ such that, for $1 \leq i \leq l$, $C_i \subseteq V$, $C_i \neq \emptyset$, and $\bigcup_{i=1}^{l} C_i = V$. A *clustering* $\mathcal{C}$ is disjoint if and only if $\forall C_i \in \mathcal{C}$ and $\forall C_j \in \mathcal{C}$, with $C_i \neq C_j$, $C_i \cap C_j = \emptyset$. $\mathcal{C}$ is an *overlapping clustering* if $\exists C_i \in \mathcal{C}$ and $\exists C_j \in \mathcal{C}$, with $C_i \neq C_j$, such that $C_i \cap C_j \neq \emptyset$. Given a vertex $v \in V$ and a clustering $\mathcal{C}$, the clusters set containing the vertex v is defined by $\ell_{\mathcal{C}}(v) = \{C_i \mid v \in C_i, C_i \in \mathcal{C}\}$. In addition, $\forall v \in V$, $|\ell_{\mathcal{C}}(v)| = 1$ if $\mathcal{C}$ is a disjoint clustering and $\mathcal{C}$ is an overlapping clustering if $\exists v \in V$ such that $|\ell_{\mathcal{C}}(v)| > 1$.

Let E_V be the set of all possible edges of a graph $G = (V, E)$. The cluster editing problem aims at finding an edge subset F, such that $F \subseteq E_V$, so that the graph $G' = (V, E \Delta F)$ is a disjoint union of cliques. The subset F is denominated as *edge edition set*. In the minimization version of the cluster editing problem, it is necessary to find the smallest edge edition set. Given a graph G and a clustering $\mathcal{C}$, the cost of a cluster editing solution is computed as presented by Equation (2.1) (according to Charikar et al. (2005)).

$$K_{ce}(G, \mathcal{C}) = \sum_{i<j,\ \{i,j\}\in E} x_{ij} + \sum_{i<j,\ \{i,j\}\notin E} (1 - x_{ij}), \tag{2.1}$$

where

$$x_{ij} = \begin{cases} 0, & \text{if } \ell_{\mathcal{C}}(i) = \ell_{\mathcal{C}}(j), \\ 1, & \text{if } \ell_{\mathcal{C}}(i) \neq \ell_{\mathcal{C}}(j). \end{cases}$$

In other words, variables x_{ij}, for $1 \leq i < j \leq n$, are equal to one when vertices i and j belong to different clusters. The variables x_{ij} are equal to zero when i and j belong to the same cluster.

In the overlapping cluster editing problem we have to find an edge edition set F such that the vertices of the input graph G are partitioned into maximal cliques. Note that, unlike the cluster editing problem, in the overlapping cluster editing problem cliques are not necessarily disjoint. In other words, cliques can share vertices. Then, to compute the overlapping cluster editing solution cost the value of the x_{ij} variables of Equation (2.1) need to be modified as follows:

$$x_{ij} = \begin{cases} 0, & \text{if } \ell_C(i) \cap \ell_C(j) \neq \emptyset, \\ 1, & \text{if } \ell_C(i) \cap \ell_C(j) = \emptyset. \end{cases}$$

In other words, variables x_{ij}, for $1 \leq i < j \leq n$, are equal to one when vertices i and j have no clusters in common. The variables x_{ij} are equal to zero when i and j belong to at least one same cluster.

2.4 Hybrid heuristic

The hybrid heuristic proposed in this work can be divided into three steps. First, the metaheuristics *Biased Random-Key Genetic Algorithm* (BRKGA) (GONÇALVES; RESENDE, 2011) and *Simulated Annealing* (SA) (KIRKPATRICK et al., 1983) are used to generate, together, a set of diverse cluster editing solutions of the input graph. Subsequently, all clusters belonging to the solutions set are used as input by CPLEX (IBM Corporation, 2017) to solve one of three MILP, which are described in Subsection 2.4.2. The main reason for using the clusters set from metaheuristics is to provide a good-quality and diversified input to solve one of the three proposed MILP. An overlapping cluster editing solution is obtained with the resolution of these models.

In our overlapping cluster editing problem definition, graph vertices are partitioned into maximal cliques that are not necessarily disjoint. Since the cost of a solution of this problem is computed by the overlapping version of Equation (2.1), costs of cliques that are not maximal, i.e., cliques that are strictly contained in a larger clique, are ignored. Based on this, only costs of maximal cliques are considered. For example, consider a graph with vertices i, j and k, where $(i, j) \in E$ and $(j, k) \in E$ and $(i, k) \notin E$. Consider, also, a clustering $\mathcal{C} = \{C_1, C_2, C_3\}$, where $C_1 = \{i, j\}$, $C_2 = \{i\}$ and $C_3 = \{k\}$. As one can see, $C_2 \subset C_1$. The overlapping cluster editing cost of this solution is one, as $\ell(i) \cap \ell(k) = \emptyset$ and, therefore, the edge (i, k) should be removed. Note that, although $i \in C_2$ and $j \in C_1$, the edge (i, j) does not need to be removed, because i is also contained in C_1. Furthermore, the removal of edge (i, k) is performed only once, even though vertex i belongs to C_1 and to C_2. Thus, clusters that are contained in larger clusters are ignored because of the problem definition. Then, solutions produced by our hybrid heuristic naturally address this issue.

The cluster editing solutions are obtained through the BRKGA and SA metaheuristics' execution. For this, a number h_{sol} of solutions is passed as parameter to the hybrid heuristic. Then, $h_{sol}/2$ solutions are selected from BRKGA execution and $h_{sol}/2$ solutions are selected from SA execution. This ratio between the number of metaheuristics' solutions were empirically defined considering the tests presented in Subsection 2.5.1.

A pseudocode for the hybrid heuristic is shown in Algorithm 1 and figure 2.2 depicts the execution of the proposed method. The BRKGA and SA metaheuristics are executed at lines 2 and 3. These metaheuristics generate a set of cluster editing solutions that is stored in variable "*sol_set*". Then, at line 4, the clusters set is formed from *sol_set* and stored in variable "*clusters*". Subsequently, at line 5, CPLEX solves one of the three proposed MILP using the set of clusters and the graph as input. The resulting overlapping clustering solution of this resolution is stored in variable "*ovlp_sol*". Finally, at line 6, the overlapping cluster editing solution cost is computed by the overlapping version of Equation (2.1).

The remainder of this section is divided as follows. In the next subsection, details about the BRKGA and SA implementations are shown. We present the three proposed MILP in Subsection 2.4.2.

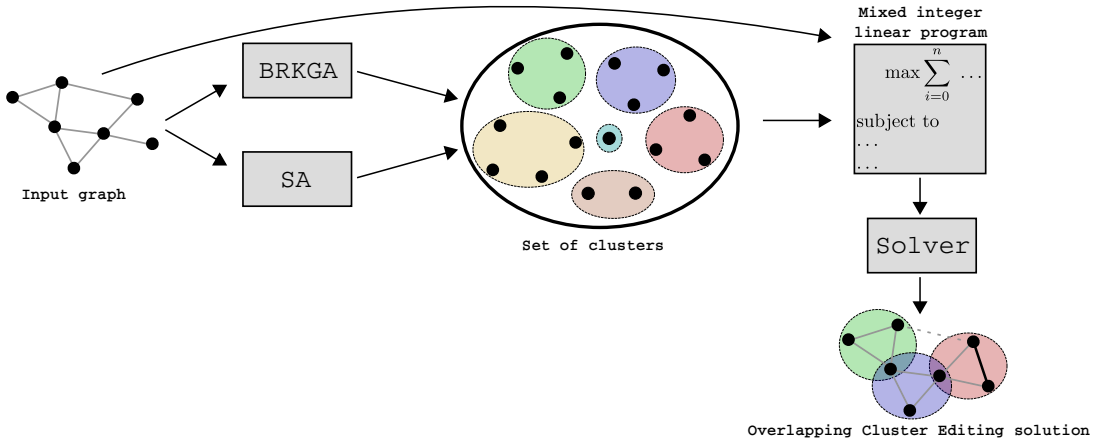
Algoritmo 1: Hybrid heuristic.

input : graph $G = (V, E)$; MILP *model*; BRKGA number of generations $gen_{\max}$; BRKGA population size p; BRKGA elite population size p_e; BRKGA mutant population size p_m; BRKGA elite allele inheritance probability ρ_e; SA initial temperature t_i; SA final temperature t_f; SA cooling rate α; SA Metropolis algorithm step size $sa_{\max}$.

output: overlapping cluster editing solution *ovlp_sol*;

```
1 begin
2     sol_set ← brkga(G, gen_max, p, p_e, p_m, ρ_e);
3     sol_set ← sol_set ∪ sa(G, t_i, t_f, α, sa_max);
4     clusters ← get_clusters(sol_set);
5     ovlp_sol ← cplex_solve(G, model, clusters);
      // computed by the overlapping version of Equation (2.1)
6     ovlp_sol.compute_ovlp_clstring_cost();
7     return ovlp_sol;
8 end
```

Figure 2.2 - Overall execution of the proposed hybrid heuristics.



SOURCE: Produced by the author.

2.4.1 Metaheuristics

The BRKGA (GONÇALVES; RESENDE, 2011) and SA (KIRKPATRICK et al., 1983) metaheuristics were implemented to produce, together, a set of clusters that is used as input to solve one of the three models presented in the Subsection 2.4.2. Considering that solving a MILP is computational expensive, the main idea of using these metaheuristics was to generate a size-limited but diversified set of good quality clusters. In order to guide these metaheuristics to produce such set, both were designed to produce cluster editing solutions. Then, the objective function used in

both metaheuristics is the non-overlapping version of Equation (2.1).

As Glover and Kochenberger (2003) explain, we can classify metaheuristics as population-based and single-solution-based. Thus we decided to implement one metaheuristic of each category to try to diversify the clusters set. In addition, the BRKGA metaheuristic was implemented because it is relatively recent one and was successfully used in a variation of the overlapping cluster editing problem (ANDRADE et al., 2014). The SA metaheuristic was used because it is a classic and well-know metaheuristic. BRKGA and SA implementation details are described in Subsections 2.4.1.1 and 2.4.1.2 respectively. Furthermore, in Subsetcion 2.5.1, tests results are presented showing the advantages of using BRKGA and SA to produce models' input clusters.

2.4.1.1 Biased Random-Key Genetic Algorithm

A Genetic Algorithm (GA) (HOLLAND, 1975) mimics the process of natural selection by performing mating, mutation and selection over a population of individuals. Each individual is represented by a *chromosome* which is a string of *alleles* that represents a solution of the problem in concern (GONÇALVES; RESENDE, 2011). The GA execution is divided into generations. At each generation, a distinct population is created by the survival of the fittest individuals, by combining two or more individuals for producing offspring and by mutation. The fitness of a individual is the cost of the corresponding solution.

Proposed by Gonçalves and Resende (2011), the BRKGA metaheuristic is a GA where the chromosomes are arrays of random values in the real interval $[0, 1]$. Adopting these random-key-based chromosomes allows almost all BRKGA's steps be problem-independent. The only exception is the decoding step, which translates a chromosome to a valid problem solution. Another difference between the BRKGA and a regular GA is the crossover procedure. In the BRKGA, an offspring will be always generated by the crossover of a fittest chromosome and a non-fittest chromosome. This step is biased, i.e., an offspring is more likely to inherit a fittest parent allele. In addition, another difference is that at each BRKGA generation, completely new random individuals, called mutants, are created replacing the worst individuals.

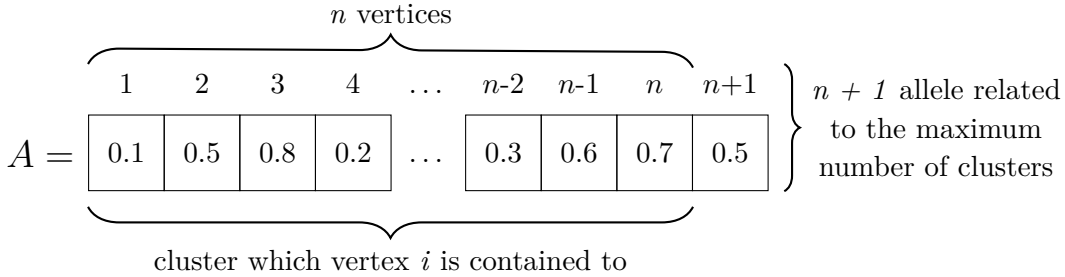
In this work we used a label-based integer encoding scheme (HRUSCHKA et al., 2009) for representing a clustering as a decoded chromosome. Each chromosome in our BRKGA population has $n + 1$ alleles, where $n = |V|$. The last position $(n + 1)$ of each chromosome represents the maximum number of clusters that the decoded

solution has. The n first chromosome positions represent the cluster in which each vertex belongs. In other words, the i-th position, with $1 \leq i \leq n$, represents the cluster that vertex i belongs to in the decoded solution.

The chromosome decoding step starts at allele $n + 1$ to determine the maximum number of clusters of the solution. For this, an upper bound, defined a priori, is utilized for the maximum number of clusters (max_{clst}). Let A be an array with $n+1$ positions which represents a BRKGA chromosome. The number of clusters in a decoded solution is given by $l = \lceil max_{clst} \times A[n+1] \rceil$. In all tests carried out in this chapter we used $max_{clst} = 200$. This value was empirically defined, since the instances' sizes used in this work are between 21 and 1000 vertices. Subsequently, the first n alleles are decoded to determine which of l clusters each vertex will belong to. This decoding is executed regarding l and is given by $k = \lceil l \times A[i] \rceil$, with $1 \leq i \leq n$ and $1 \leq k \leq l$, where C_k is the cluster which has vertex i. The BRKGA decoding step was parallelized.

An example of an array that represents a BRKGA chromosome is depicted by Figure 2.3. For the sake of the example, consider that $max_{clst} = 20$. Then, the maximum number of clusters is computed by $l = \lceil max_{clst} \times A[n+1] \rceil = \lceil 20 \times 0.5 \rceil = 10$ and the cluster that each vertex belongs to is given by $\lceil l \times A[1] \rceil = \lceil 10 \times 0.1 \rceil = 1$, $\lceil l \times A[2] \rceil = \lceil 10 \times 0.5 \rceil = 5$, $\lceil l \times A[3] \rceil = \lceil 10 \times 0.8 \rceil = 8$ and so on.

Figure 2.3 - An example of an array that represents a BRKGA chromosome.



SOURCE: Produced by the author.

As mentioned before, $\frac{h_{sol}}{2}$ cluster editing solutions are selected from BRKGA execution. In order to obtain these $\frac{h_{sol}}{2}$, at every $\frac{gen_{\max}}{h_{sol}/2}$ generations, where $gen_{\max}$ is the BRKGA maximum number of generations, a chromosome is randomly chosen from population and decoded. This decoded solution is then stored in the cluster editing solutions set.

The BRKGA parameters values used in this work are shown in Table 2.1. Specifically, CALIBRA software (ADENSO-DÍAZ; LAGUNA, 2006) was used to obtain these values. The CALIBRA tests were performed in 13 instances of Bastos et al. (2016) with sizes ranging between 25 vertices and 100 vertices. In Table 2.1 are presented the number of generations (gen), the size of population (p), the elite population proportion (p_e), the mutant population proportion (p_m) and the elite allele inheritance probability (ρ_e).

Table 2.1 - BRKGA parameters values used in the experimental tests carried out in this work. These values were obtained by CALIBRA software (ADENSO-DÍAZ; LAGUNA, 2006).

Parameter	gen	p	p_e	p_m	ρ_e
Value	696	820	$0.19 \cdot p$	$0.23 \cdot p$	60%

2.4.1.2 Simulated Annealing

Developed by Kirkpatrick et al. (1983), the SA metaheuristic tries to simulate the process of physical annealing of a metal. In this process, a metal is heated to its melting point and then it is slowly cooled until it reaches the solid state again. With this technique the resulting metal structure will be crystalline and without imperfections (KIRKPATRICK et al., 1983). Kirkpatrick et al. (1983) then attempted to apply this physical concept to solve optimization problems.

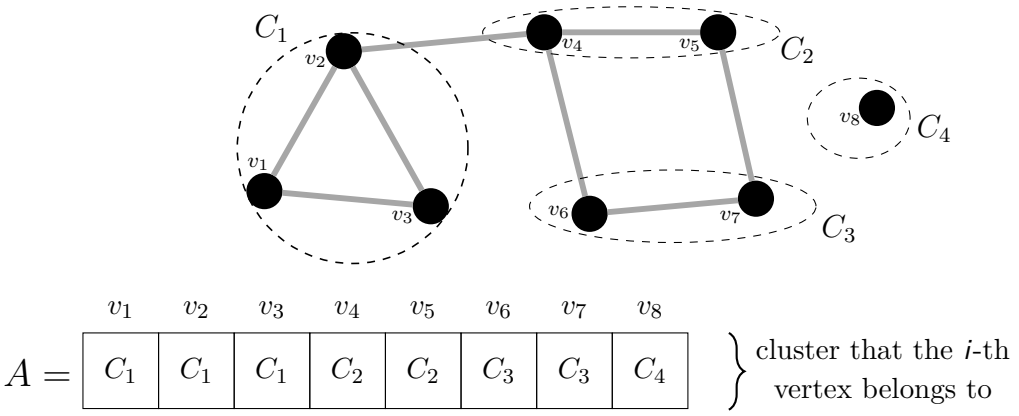
The SA starts from a random solution and with a high initial temperature. In this context, high temperature means higher probability of moving to worse solutions as the solution space is explored by the Metropolis et al. (1953) algorithm. At each SA iteration a neighbor solution to the current one is generated and the temperature is cooled by a constant rate. If a neighbor solution is better than the current one, SA moves to this new solution. The execution stops when the minimum temperature is met.

Following our BRKGA implementation, we also used a label-based integer encoding (HRUSCHKA et al., 2009) for representing a clustering solution in the SA metaheuristic. However, since SA is a single-based solution metaheuristic, there is no need to store the number of clusters in a SA solution encoding. We defined instead a independent value $l \in [1, max_{clst}]$ in which each vertex is assigned to a cluster label in the range $[1, l]$. Then, a SA solution is represented by means of an array A with n positions where each position i of A, with $1 \leq i \leq n$, represents the cluster that the

ith vertex belongs to. The SA initial solution is randomly generated by assigning to each position of array A a random value in the integer interval $[1, l]$.

An example of a SA clustering solution is depicted by Figure 2.4. In this example, the clustering solution composed by clusters $C_1 = \{v_1, v_2, v_3\}$, $C_2 = \{v_4, v_5\}$, $C_3 = \{v_6, v_7\}$ and $C_4 = \{v_8\}$ is represented by array A.

Figure 2.4 - An example of a SA clustering solution array.



SOURCE: Produced by the author.

In order to generate diversified neighbor solutions, four neighborhood functions with different probabilities were utilized in the SA metaheuristic. The greater the perturbation generated by a neighborhood function, the less likely it is to be used. These functions are described below:

- *Random clusters change*: a random number of vertices, in the interval $[1, \frac{n}{10}]$, are selected to change their cluster index. The cluster that each selected vertex belongs to is randomly changed by another one. This function has a 25% chance to be used;
- *Clusters swap between two random vertices*: two distinct vertices are randomly selected to have their cluster index swapped. This function has a probability of 70% to be utilized;
- *Clusters rotation*: two indices $i \in \mathbb{N}$ and $j \in \mathbb{N}$, with $1 \leq i \leq n-2$, and $\lceil 0, 1 \cdot n \rceil \leq j \leq n$, are randomly chosen. Then, the clusters indices of vertices belonging to the range $[i, j]$ are shifted one position. In other words, for $i \leq k < j$, $A[k+1]$ receives the cluster index of $A[k]$ and, for $k = j$,

$A[i]$ receives the cluster index of $A[k]$. This function has a probability of 4% to be chosen;

- *Random change of the clusters number*: in this function a new maximum number of clusters l_{new} is randomly chosen in the range $[1, max_{clst}]$. In case of $l_{new} < l$, all vertices cluster indices that are greater than l_{new} are fixed to the new range $[1, l_{new}]$. In case of $l_{new} > l$, then some random vertices have their cluster indices changed to the new range $[1, l_{new}]$ by the random clusters change function. The random change of the clusters number function has a probability of 1% to be used.

Similar to the BRKGA metaheuristic, $\frac{h_{sol}}{2}$ cluster editing solutions are also selected from SA execution. In order to obtain these solutions, a neighbor solution of the current solution is selected at every sa_{iter} iterations of the Metropolis et al. (1953) algorithm. The sa_{iter} values are calculated by Equation (2.2), where α is the SA cooling rate, $sa_{\max}$ is the step size Metropolis algorithm and t_{init} and t_{final} are, respectively, the SA initial and final temperatures.

$$sa_{iter} = \frac{\log_{\alpha}^{\frac{t_{final}}{t_{init}}} \cdot sa_{\max}}{\frac{h_{sol}}{2}}. \tag{2.2}$$

The SA parameters values used in this work are presented in Table 2.2. The CALIBRA software (ADENSO-DÍAZ; LAGUNA, 2006) was used as in BRKGA metaheuristic to obtain these values. In Table 2.2 are presented the values of the initial temperature (t_{init}), final temperature (t_{final}), the step size of Metropolis algorithm ($sa_{\max}$) and the cooling rate (α).

Table 2.2 - SA parameters values used in the experimental tests carried out in this work. These values were obtained by CALIBRA software (ADENSO-DÍAZ; LAGUNA, 2006).

Parameter	t_{init}	t_{final}	$sa_{\max}$	α
Value	750	10^{-6}	750	0.98

2.4.2 Mixed-integer linear programming models

In this section the three proposed MILP for the overlapping cluster editing problem are introduced. Our objective with these models was to produce overlapping clus-

tering solutions through slightly different strategies. The first MILP (M1) selects a fixed number of clusters considering its quality coefficient based on the overlapping cluster editing problem. However, other coefficients can be used. Although the models were proposed for the overlapping cluster editing, they can also be applied in others overlapping clustering problems. The second MILP (M2) finds an overlapping clustering through the generation of up to q different set covers. The third MILP (M3) is a variation of M2 one which an exactly number of covers are selected. In M3 we associated a quality coefficient to each cluster as in M1.

Despite the fact that the three MILP are different, they were developed with some common aspects. For instance, all models were designed considering the Jaccard coefficient between each pair of clusters from the input set. The reason was to use this coefficient to control, by an input model parameter, the overlapping between clusters. This is because, depending on the instance context, it may be better to use clusters with more overlap or less overlap. In addition, the objective functions of the three MILP are *max-min* functions, which the aim is to minimize the difference of the Jaccard coefficient between clusters and the overlapping parameter. Furthermore, another models' key feature is to ensure that each graph vertex is covered by at least one cluster. The M1, M2 and M3 models are presented in the Subsections 2.4.2.1, 2.4.2.2 and 2.4.2.3, respectively.

2.4.2.1 M1

The first MILP proposed in this work is shown in Equations (2.3a) to (2.3e). Given a cluster set $S = \{C_1, C_2, \ldots, C_N\}$ of the vertices of an input graph $G = (V, E)$, in the proposed model, the objective is to produce an overlapping clustering $\mathcal{C} \subseteq S$, where $|\mathcal{C}| = r$ and $\bigcup_{C \in \mathcal{C}} C = V$. The $\mathcal{C}$ set is composed by r clusters with the best costs and, depending on the established criteria, have more or less overlaps with each other cluster belonging to $\mathcal{C}$.

$$\max \sum_{i=1}^{N} (d_i \cdot y_i - u_i) \tag{2.3a}$$

subject to

$$\sum_{j=1}^{N} \Big| J(C_i, C_j) - z \Big| \cdot (y_i + y_j - 1) \leq u_i, \;\; i = 1, 2, \ldots, N, \tag{2.3b}$$

$$\sum_{i=1}^{N} y_i = r, \tag{2.3c}$$

$$\sum_{i=1}^{N} a_{ji} \cdot y_i \geq 1, \ j = 1, 2, \ldots, n, \tag{2.3d}$$

$$y_i \in \{0, 1\}, u_i \in \mathbb{R}, \ i = 1, 2, \ldots, N. \tag{2.3e}$$

In M1, with the binary variables y_i, for $1 \leq i \leq N$, it is defined which C_i clusters belong, or not, to the solution. Also, there is a cost d_i associated with each cluster C_i that represents how good this cluster is. The d_i values are given by the Equation (2.4).

$$d_i = \frac{E_{C_i}^{in}}{E_{C_i}^{\max}} - \frac{E_{C_i}^{out}}{|C_i| \cdot (|V| - |C_i|)}. \tag{2.4}$$

In Equation (2.4), $E_{C_i}^{\max}$ is the maximum number of edges between C_i vertices, that is, $E_{C_i}^{\max} = \frac{|C_i| \cdot (|C_i|-1)}{2}$. In addition, $E_{C_i}^{in}$ is the number of edges that connect vertices belonging to C_i and $E_{C_i}^{out}$ is the number of edges connecting a vertex from C_i and a vertex that does not belong to C_i. Moreover, the maximum number of edges between vertices from C_i and vertices not belonging to it is given by $|C_i| \cdot (|V| - |C_i|)$. This equation presents values in the real interval $[-1, 1]$. When cluster C_i is the best possible, i.e., a clique with no out-edges, then $d_i = 1$. When cluster C_i has no in-edges and its vertices are adjacent to all other vertices that does not belong to C_i, then $d_i = -1$.

Since the objective function (2.3a) must be maximized, the lowest values of the real variables u_i are obtained. This is because the u_i variables, in this function, have negative coefficients. With these variables, clusters with the smallest differences between the Jaccard coefficient, related to the other clusters, and the overlapping control parameter z are selected. Constraint (2.3b) controls, with $z \in [0, 1]$, the overlaps between clusters. The closer z parameter value is to one, the greater the overlaps between clusters. The closer z parameter value is to zero, the smaller the overlaps between the clusters. The reason is that the overlap between a pair of clusters C_i e C_j is quantified by means of the Jaccard coefficient. Therefore, if clusters C_i e C_j have maximum overlap, that is, $C_i = C_j$, then $J(C_i, C_j) = 1$. If clusters C_i and C_j have no overlap, that is, $C_i \cap C_j = \emptyset$, then $J(C_i, C_j) = 0$. Thus, if $z = 1$, variable u_i will have the lowest value when $J(C_i, C_j) = 1$. On the other hand, if $z = 0$, variable u_i will have the lowest value when $J(C_i, C_j) = 0$.

In the constraint (2.3c) is ensured that exactly r clusters are selected. It is guaranteed

by constraint (2.3d) that each graph vertex belongs to at least one cluster. In this constraint, for $1 \leq i \leq N$ and $1 \leq j \leq n$, $a_{ji} = 1$ if vertex j belongs to cluster C_i and $a_{ji} = 0$ otherwise. In addition, constraint (2.3e) defines variables y_i as binaries and u_i as reals ones.

2.4.2.2 M2

The second proposed MILP is presented in Equations (2.5a) to (2.5h). In this model, given a cluster set $S = \{C_1, C_2, \ldots, C_N\}$ of the n vertices of an input graph, the objective is to produce an overlapping clustering through the generation of up to q set covers, where the set is V. Each set cover is created by selecting a cluster subset $\mathcal{C} \subseteq S$, where $\bigcup_{C \in \mathcal{C}} C = V$. The maximum number of covers (q) is defined a priori. Note that, in this second model, the clusters' costs are not considered, the purpose is to generate overlapping clustering disregarding clusters costs.

$$\max \sum_{j=1}^{q} y_j - \sum_{i=1}^{N} u_i \tag{2.5a}$$

subject to

$$\sum_{j=1}^{q} y_j \geq 1, \tag{2.5b}$$

$$\sum_{j=1}^{q} x_{ij} \leq e, \ i = 1, 2, \ldots, N, \tag{2.5c}$$

$$\sum_{i=1}^{N} a_{ik} \cdot x_{ij} \geq y_j, \ k = 1, 2, \ldots, n, \ j = 1, 2, \ldots, q, \tag{2.5d}$$

$$\sum_{j=1}^{N} \Big| J(C_i, C_j) - z \Big| \cdot (x_{ik} + x_{jk} - 1) \leq u_i, \ i = 1, \ldots, N, \ k = 1, \ldots, q, \tag{2.5e}$$

$$\sum_{i=1}^{N} (x_{ij} - x_{ik}) + N \cdot l_j \geq y_j, \ k = j + 1, \ldots, q, \ j = 1, \ldots, q, \tag{2.5f}$$

$$\sum_{i=1}^{N} (x_{ik} - x_{ij}) + N \cdot l_j \leq N - y_j, \ k = j + 1, \ldots, q, \ j = 1, \ldots, q, \tag{2.5g}$$

$$l_j, x_{ij}, y_j \in \{0, 1\}, u_i \in \mathbb{R}, \ i = 1, 2, \ldots, N, \ j = 1, 2, \ldots, q. \tag{2.5h}$$

In the objective function (2.5a) the binary variables y_j, with $1 \leq j \leq q$, controls which of the q covers belongs to the final overlapping clustering solution. In addition, by means of the binary variables x_{ij}, with $1 \leq i \leq N$ and $1 \leq j \leq q$, the set covers

are created, where $x_{ij} = 1$ if and only if the cluster C_i belongs to the cover j and $x_{ij} = 0$ otherwise. The real variables u_i are used in this model as in M1, presented in Subsection 2.4.2.1. Since the objective function (2.5a) is a maximization, the smallest values of the variables u_i are obtained.

Constraint (2.5b) ensures that at least one cover is selected. With the constraint (2.5c) it is established that clusters belongs up to e covers. When $e = 1$, then the covers are disjoint. Constraint (2.5d) guarantees the vertex-set cover, that is, it is guaranteed that in each selected cover $\mathcal{C}_j$ (clustering) each vertex k, with $1 \leq k \leq n$, belongs to at least one cluster $C_i \in \mathcal{C}_j$. In this constraint, $a_{ik} = 1$ if and only if the vertex k belongs to the cluster C_i and $a_{ik} = 0$ otherwise. The constraint (2.5e) is similar to constraint (2.3b) of the first model. The only difference between these constraints is that the constraint (2.5e) is considered for all N clusters of each q covers and the constraint (2.3b) is considered only for the N clusters.

With the constraints (2.5f) and (2.5g) it is defined that the selected covers must be different from each other. In this sense, two covers are different if exists at least one cluster that belongs to one of these covers and it does not belongs to the another cover. In other words, for each pair of covers $\mathcal{C}_j$ and $\mathcal{C}_k$, with $1 \leq j, k \leq q$ and $j \neq k$, $\mathcal{C}_j \Delta \mathcal{C}_k \neq \emptyset$. Constraints (2.5f) and (2.5g) are useful when $e > 1$. In particular, these constraints are the linearization of the constraint $\sum_{i=1}^{N} |x_{ij} - x_{ik}| \geq y_j$, for all $1 \leq j \leq q$ and $j < k \leq q$. Variables l_j, x_{ij} and y_j are set as binary and variables u_i are set as real in the constraint (2.5h).

2.4.2.3 M3

Model (2.6) presents the third mixed-integer linear program, introduced in this work, for the overlapping cluster editing problem. This model is a modification of M2, presented in the Subsection 2.4.2.2. Given a cluster set $S = \{C_1, C_2, \ldots, C_N\}$ of the n vertices of an input graph, the objective of the M3, defined by Equations (2.6a) to (2.6h), is to produce an overlapping clustering by generating exactly c set covers, where the set is V. These covers are created as in the second model but also considering the costs d_i, with $1 \leq i \leq N$, of clusters C_i. The d_i cost represents how good the cluster C_i is and it is calculated by Equation (2.4).

$$\max \sum_{i=1}^{N} \sum_{j=1}^{q} (d_i \cdot y_j - u_i) \tag{2.6a}$$

subject to

$$\sum_{j=1}^{q} y_j = c, \tag{2.6b}$$

$$\sum_{j=1}^{q} x_{ij} \leq e,\ i = 1, 2, \ldots, N, \tag{2.6c}$$

$$\sum_{i=1}^{N} a_{ik} \cdot x_{ij} \geq y_j,\ k = 1, 2, \ldots, n,\ j = 1, 2, \ldots, q, \tag{2.6d}$$

$$\sum_{j=1}^{N} \left| J(C_i, C_j) - z \right| \cdot (x_{ik} + x_{jk} - 1) \leq u_i,\ i = 1, \ldots, N,\ k = 1, \ldots, q, \tag{2.6e}$$

$$\sum_{i=1}^{N} (x_{ij} - x_{ik}) + N \cdot l_j \geq y_j,\ k = j + 1, \ldots, q,\ j = 1, \ldots, q, \tag{2.6f}$$

$$\sum_{i=1}^{N} (x_{ik} - x_{ij}) + N \cdot l_j \leq N - y_j,\ k = j + 1, \ldots, q,\ j = 1, \ldots, q, \tag{2.6g}$$

$$l_j, x_{ij}, y_j \in \{0, 1\}, u_i \in \mathbb{R},\ i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, q. \tag{2.6h}$$

As mentioned, this third mixed-integer linear program is a modification of M2. Then, the constraints (2.6c), (2.6d), (2.6e), (2.6f), (2.6g) and (2.6h) are the same constraints as M2 constraints (2.5c), (2.5d), (2.5e), (2.5f), (2.5g) and (2.5h). These models differ only in objective functions (2.5a) and (2.6a) and in the first constraints (2.5b) and (2.6b).

The objective function (2.6a) is a maximization of the c set covers with clusters that have the best costs and the lowest values of variables u_i. Constraint (2.6b) guarantees that exactly c set covers are selected in the final solution.

2.5 Experimental results and analysis

In this section results of the hybrid heuristic tests are presented. All implementations were written in *C++* language. For the resolution of models we used the IBM® ILOG® CPLEX® 12.8 (IBM Corporation, 2018). All the computational tests were executed on a computer with Intel® Xeon® E5-2687W v2 CPU 3.40GHz × 8 processor with 25MiB *cache* memory and 62GiB of RAM. The operating system installed on this machine is Ubuntu 14.04.1 64 bits with kernel 3.19.0-32-generic. In addition, all CPLEX (IBM Corporation, 2018) executions were limited to 3 hours.

Two sets of instances were used to evaluate the hybrid heuristic. These sets are detailed below:

- *Random graphs*. Proposed by Bastos et al. (2016), this set consists of 112 randomly generated graphs. These instances have sizes ranging between 21 vertices to 1000 vertices with different levels of difficult in the context of the cluster editing problem. The difficulty of an instance is related to its density and to number of edges edition necessary to partitioning the graph into a disjoint union of cliques. Sparse and dense graphs are easier to partition into disjoint cliques than graphs with density close to 0.6 (BASTOS et al., 2016). One can obtain all the 112 graphs at <http://www2.ic.uff.br/~lbastos/>;
- *LF benchmark graphs*. Set of 30 graphs, with ground truth overlapping clustering solutions, that were generated by Lancichinetti and Fortunato (2009b) algorithm. Five graphs, ranging from sparse to dense, of each value $n = \{25, 50, 100, 200, 500, 1000\}$ of vertices were generated. These instances can be obtained at <http://www.lac.inpe.br/~rafael.santos/OCI/>.

With the LF benchmark graphs the main objective is to verify if the hybrid heuristic is able to reproduce the original overlapping clustering. For this reason, we used the *FBCubed* (AMIGÓ et al., 2009) metric to evaluate the hybrid heuristic solutions in relation to the ground truth solution. The *FBCubed* metric, with values ranging in the real interval $[0, 1]$, is a supervised measure for evaluating overlapping clusterings. The closer to one is the *FBCubed* value, the better is the overlapping clustering relative to the ground truth. The closer to zero, the worse the clustering relative to the ground truth.

Three versions of the hybrid heuristic were evaluated on the tests performed in this work. Each hybrid heuristic version is composed of the BRKGA and SA metaheuristics and one of the three proposed mixed-integer linear programming models. For simplicity, in this chapter we use HHM1 to refer to the hybrid heuristic version formed by the metaheuristics and M1, HHM2 to refer to the hybrid heuristic version composed by the metaheuristics and M2 and HHM3 to refer to the hybrid heuristic version formed by the metaheuristics and M3. The results presented in this section were obtained from one execution of each hybrid heuristic variation.

Table 2.3 shows the models' parameters values used in all tests carried out in this chapter. In this table, *cc_sol* is the number of cluster editing solutions obtained through the metaheuristics executions. Since the resolution of M2 and M3 have a greater number of variables than M1, they are harder to solve. Then, a smaller number of cluster editing solutions were used to solve M2 and M3 than M1. In

addition, we utilized, for both HHM2 and HHM3, a small maximum number of set covers. We utilized $q = 5$ in order to try to make these hybrid heuristic versions generate competitive solutions. Large q values implies in a larger number of variables that increases the models' solving times. Another reason was that a large number of covers can result in a larger number of clusters and then lead to a poor overlapping cluster editing solution. We used $e = 1$ to avoid that a same cluster belongs to more than one set cover. Furthermore, we utilized $c = 1$ in HHM3 to ensure that just one set cover is selected among the q possibilities.

Table 2.3 - Parameters values, of the three proposed models, used in all tests performed in this work.

	HHM1		**HHM2**			**HHM3**			
Parameter	r^*	*cc_sol*	*e*	*q*	*cc_sol*	*c*	*e*	*q*	*cc_sol*
Value	-	100	1	5	10	1	1	5	10

* It was used, in each instance, the average number of clusters of the cluster editing solutions set as r values.

Furthermore, in tests performed in this work each hybrid heuristic variation (HHM1, HHM2 and HHM3) were used with two overlapping control parameters values: $z = 0$ for minimum overlap and $z = 1$ for maximum overlap. The reason was to evaluate how overlapping clusters affect solutions costs. Specifically, the metaheuristics' solutions costs were computed by the non-overlapping version of the Equation 2.1 and the hybrid heuristics solutions costs were calculated by the overlapping version of the Equation 2.1.

The following subsections are organized as follows. In Subsection 2.5.1, an evaluation of the effect of using both metaheuristics in the quality of models' solutions is presented. A summary and an analysis of the results of the tests realized with the random graphs instances and with the LF benchmark graphs are shown in Subsections 2.5.2 and 2.5.3, respectively. In addition, detailed results of each method in each instance are presented in Appendix A.

2.5.1 Analysis of BRKGA and SA influence on models' results

We present, in this subsection, the results of tests conducted aiming to show whether the option of using both metaheuristics for generation of input clusters is more suitable or not than using only one. Since the goal was to evaluate the metaheuristics' influence on models' solutions, we selected 12 random graphs, three of each value $n = \{25, 50, 100, 200\}$, where our methods achieved the best solutions costs. These instances were chosen because it could be possible to show whether the hybrid

heuristics' best costs could be improved or not by using just BRKGA or just SA.

In order to evaluate the metaheuristics' influence, the creation of input clusters was tested using only the BRKGA, only the SA and using both metaheuristics for each of the three models. Then, in addition to the three standard hybrid heuristic variations (HHM1, HHM2 and HHM3) which use BRKGA and SA, six more versions were tested using only one of the two metaheuristics. A summary of the results, obtained in tests with the 12 random graphs, is shown in Table 2.4.

In Table 2.4, each column presents the number of best overlapping cluster editing costs obtained by a particular hybrid heuristic version in comparison with its two related versions. For example, the second column shows the number of best solutions obtained by the resolution of M1 using just BRKGA in comparison with M1 using just SA and the standard hybrid heuristic using both metaheuristics. Results obtained from utilizing minimum and maximum overlapping control parameters for each hybrid heuristic are also shown. The detailed results of each instance are presented in Table A.1.

Table 2.4 - Results summary of tests performed on 12 random graphs instances for evaluate the metaheuristics' influence on each models' solutions.

	M1			**M2**			**M3**		
	BRKGA	**SA**	**both**	**BRKGA**	**SA**	**both**	**BRKGA**	**SA**	**both**
# best costs ($z = 0$)	3	1	**8**	1	2	**9**	2	2	**9**
# best costs ($z = 1$)	0	1	**11**	1	1	**10**	1	2	**9**
total	3	2	**19**	2	3	**19**	3	4	**18**

From Table 2.4, it can be observed that, in all cases, the largest number of total best results were obtained when the input clusters were generated by both metaheuristics. This can also be noted with different overlapping control parameters. The HHM1, HHM2 and HHM3 with BRKGA and SA achieved the largest number of best costs either with $z = 0$ and with $z = 1$. This is because a largest number of clusters are generate when using the two metaheuristics than when using only one. Therefore, the clusters' diversity is improved. With a clusters set more diverse, the resolution of the models can produce better solutions.

We also investigate if there is a statistical significant difference between the results of using both metaheuristics and using just BRKGA or just SA. It was considered for comparison results between each of the three approaches of the three hybrid heuristics. Then, results of the standard HHM1 were compared with M1 using only

BRKGA and with M1 using only SA. The same comparison was realized between M2 versions and between M3 versions.

Since we are comparing three algorithms at each time, traditional non-parametric methods such as Wilcoxon signed-rank test could not be applied (GARCÍA et al., 2010). Based on this, as Calvo and Santafé (2016) suggest, it is necessary to apply an *omnibus* test to identify if at least one the methods presented statistical different results in comparison with the others methods' results. If a significant difference is detected, a *post-hoc* test comparing with a control method is realized (GARCÍA et al., 2010). The objective is to show whether the control method obtained statistical significant different results regarding the others methods' results or not. In order to perform these statistical comparisons, we used the *scmamp* R package (CALVO; SANTAFÉ, 2016).

As García et al. (2010) state, the Quade test (QUADE, 1979) is more suitable for comparisons of up to five algorithms. In this way, we utilized this *omnibus* test in each of the M1, M2 and M3 versions. Then, it was obtained the following p-values:

- M1 versions: p-value $= 8.5 \cdot 10^{-7}$;
- M2 versions: p-value $= 1.9 \cdot 10^{-4}$;
- M3 versions: p-value $= 6.3 \cdot 10^{-4}$.

All the obtained p-values are less than the significance level of 0.05. Thus, it suggests that at least one version of each hybrid heuristic performed differently than the rest (CALVO; SANTAFÉ, 2016). Then, we conducted a *post-hoc* test using the stardand HHM1, HHM2 and HHM3 as control methods. For this test, we use the Quade test with correction of p-values by the Finner's method. This correction method is the default method of the *post-hoc* test in the *scmamp* package and it is a robust corrector (GARCÍA et al., 2010; CALVO; SANTAFÉ, 2016).

Table 2.5 shows the corrected p-values obtained from the *post-hoc* test. These p-values are related to each corresponding standard hybrid heuristic version.

As one can see, all corrected p-values are smaller than 0.05. Therefore, this indicated that all HHM1, HHM2 and HHM3 with both metaheuristics are statistically significant different from the versions using only BRKGA or only SA. Indeed, these results corroborate the results presented in Table 2.4, which the standard HHM1, HHM2 and HHM3 achieved better results. Then, it can be conclude that using BRKGA

Table 2.5 - Corrected p-values from *post-hoc* test with control. The HHM1, HHM2 and HHM3 with both metaheuristics were utilized as control methods. These p-values were obtained using Quade test and Finner's correction.

	M1			**M2**			**M3**		
	BRKGA	**SA**	**both**	**BRKGA**	**SA**	**both**	**BRKGA**	**SA**	**both**
***p*-values**	$6.4 \cdot 10^{-4}$	$6.6 \cdot 10^{-6}$	-	$6.5 \cdot 10^{-4}$	$2.4 \cdot 10^{-3}$	-	$8.4 \cdot 10^{-4}$	$1.3 \cdot 10^{-2}$	-

and SA for generating input clusters for all the three models is a better choice than use just BRKGA or just SA.

2.5.2 Tests with random graphs

Table 2.6 shows a summary of the solutions costs obtained by each of hybrid heuristic variation and of BRKGA and SA metaheuristics. The values presented in this table are the number of best solutions costs obtained by each algorithm regarding the corresponding Equation 2.1 version. In this table, the 112 random graphs were divided in five sets according to the number of vertices. These numbers of vertices and the numbers of instances belonging to each instance set are presented, respectively, on the first and the second columns of Table 2.6.

Table 2.6 - Results summary of tests performed on the 112 random graphs instances. The number of best costs solutions is shown for each of the hybrid heuristic variation (HHM1, HHM2 and HHM3) and for BRKGA and SA metaheuristics.

		number of best solutions costs							
				HHM1		**HHM2**		**HHM3**	
n	**#**	**BRKGA**	**SA**	$z = 0$	$z = 1$	$z = 0$	$z = 1$	$z = 0$	$z = 1$
$[21, 25]$	25	10	**17**	0	10	1	3	1	2
$[45, 50]$	21	1	1	0	**16**	0	3	0	2
$[98, 100]$	21	0	2	0	**11**	3	1	1	1
200	20	1	2	0	**10**	2	5	1	5
500	20	2	4	0	4	1	**7**	0	2
1000	5	1	**3**	0	0	0	0	0	1
total	112	15	29	0	**51**	7	19	3	13

The average computational cost, in seconds, of each hybrid heuristic variation and of BRKGA and SA metaheuristics are presented in Table 2.7. Analogously to Table 2.6, the 112 random graphs were divided in five sets according to the number of vertices. The average execution time of the hybrid heuristic variations showed in this table were computed considering the sum of each metaheuristics' execution time and the CPLEX (IBM Corporation, 2017) execution time to solve the models. For individual times, see Appendix A.

Table 2.7 - Results summary of tests performed on the 112 random graphs. The average computational time, in seconds, is shown for each of the hybrid heuristic variation (HHM1, HHM2 and HHM3) and for BRKGA and SA metaheuristics. The hybrid heuristic variations' execution time is composed by the metaheuristics and the CPLEX times.

		average time (s)							
				HHM1		**HHM2**		**HHM3**	
n	#	**BRKGA**	**SA**	$z = 0$	$z = 1$	$z = 0$	$z = 1$	$z = 0$	$z = 1$
[21, 25]	25	0.9	0.5	1.6	1.8	2.6	3.9	1.9	2.9
[45, 50]	21	1.2	1.4	2.9	3.8	7.9	10.0	4.7	8.4
[98, 100]	21	1.9	5.1	7.7	10.9	35.6	101.0	19.2	99.3
200	20	3.4	18.7	23.7	35.3	936.8	4018.5	1331.5	3804.9
500	20	10.4	109.8	692.8	169,3	10185.6	9869.0	9354.1	9497.8
1000	5	30.5	426.9	3164.8	541.4	9173.4	9204.6	9118.1	9134.4

Analyzing results presented in Table 2.9, it can observed that each hybrid heuristic version achieved better overlapping cluster editing costs when $z = 1$ rather than $z = 0$. In particular, HHM1, HHM2 and HHM3 obtained, respectively, 51, 19 and 13 better costs with $z = 1$ and 0, 7 and 3 better costs with $z = 0$. Using $z = 1$ ensures that the three MILP are solved by trying to select clusters that have more overlap between each other. Based on this, the number of vertices belonging to more than one cluster, in a solution generated with $z = 1$, is greater than those generated with $z = 0$. Thus, the number of inter-clusters edges is smaller, resulting in a better overlapping cluster editing cost. On the other hand, when $z = 0$, clusters with less overlap between each other are selected. Therefore, the number of inter-clusters edges is greater, resulting in a worse overlapping cluster editing cost.

It was observed that the HHM1 was the hybrid heuristic version that achieved the better overlapping cluster editing costs. Indeed, considering all 112 random graphs, the HHM1 obtained better results in 51 instances when compared with the others hybrid heuristic versions and the metaheuristics. All these HHM1 better costs were achieved with $z = 1$. One reason was that, in the first MILP, it is considered exactly r clusters. Then, using $z = 0$, the solution is worsened, especially in this model, because the r clusters with less overlap are selected. As M2 and M3 consider coverings instead of a fixed number of clusters, the difference between results with $z = 0$ and with $z = 1$ is more subtle.

In addition, one can see that the HHM1 presented a better performance in small and medium-size instances. As the average number of clusters of metaheuristics' solutions was used as the value of the parameter r, this may have affected the solutions quality generated by HHM1. This is because, in early metaheuristics' executions, the solution space is huge in larger instances. Then, poor solutions with too many

clusters are generated while metaheuristics do not start converging. Thus, bad solutions with many clusters influence the r parameter and, consequently, influence overlapping clustering cost of HHM1 solutions.

It also can be noted that the HHM2 and the HHM3 generated comparable results. Indeed, as can be observed in tables of Appendix A, these hybrid heuristic versions obtained similar overlapping cluster editing costs and similar computing times. One reason for this is the fact that M3 is a variation of M2. Futhermore, in order to reduce the computation cost for solving the third model, we utilized, as shown in Table 2.3, $c = 1$. Then, M2 and M3 differ only in the objective function.

Although HHM2 and HHM3 did not performed well as the HHM1 version, it can be considered that they presented reasonable overlapping cluster editing results. This is because, as both HMM2 and HHM3 models are harder to solve, they used a significantly smaller number of input clusters than HHM1, as shown in Table 2.3. Consequently, less cluster variety contributed negatively to the quality of solutions generated by HHM2 and HHM3. However, even with a limited number of clusters, HHM2 and HHM3 presented, considering the results with $z = 0$ and $z = 1$, best costs in 26 and 16 instances, respectively. Indeed, the second hybrid heuristic variation achieved the highest number of best costs in random graphs with 500 vertices.

Since the BRKGA and SA metaheuristics were only used to generate solutions that clusters are used as input to solve the proposed models, it also can be considered that these metaheuristics obtained good results regarding the costs of the cluster editing problem and the computational time. As can be seen in Table 2.9, BRKGA and SA achieved best results in 15 and 29 instances, respectively. In addition, as presented in Table 2.8, from 40 known optimal costs, the BRKGA and SA metaheuristics presented 12 and 25 optimal costs, respectively. Table 2.8 shows the number of optimal solutions costs obtained by each method. The cluster editing optimal values were obtained by the resolution of the Charikar, Guruswami and Wirth (CHARIKAR et al., 2005) linear integer programming model. Since we utilized $3h$ as time limit for CPLEX (IBM Corporation, 2017) executions, we could only obtain optimal solutions in 40 instances with up to 100 vertices.

Also, it can be seen from Table 2.8 that, in 18 of the 40 instances in which optimal cluster editing costs are known, the hybrid heuristic versions achieved better costs than the optimal ones. This is because, in an overlapping clustering, vertices can belong to more than one cluster. Hence, there are fewer inter-cluster edges. Therefore, allowing clusters to overlap may be a less costly alternative to the cluster editing

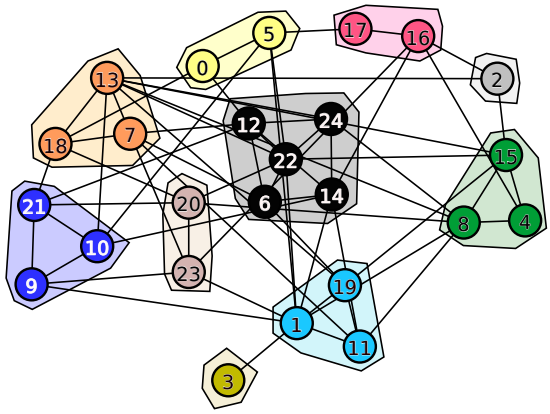
Table 2.8 - Number of optimal solutions costs of each algorithm in the 40 random graphs where the optimal cluster editing cost is known. In some instances, hybrid heuristics' overlapping cluster editing solutions achieved better results than the optimal ones.

		number of optimal (or better) solutions costs							
				HHM1		HHM2		HHM3	
n	# opt	BRKGA	SA	$z = 0$	$z = 1$	$z = 0$	$z = 1$	$z = 0$	$z = 1$
$[21, 100]$	40	12	25	0	21*	1	3**	1	2***

* 15 overlapping cluster editing costs better than optimal cluster editing costs.
** 2 overlapping cluster editing costs better than optimal cluster editing costs.
*** 1 overlapping cluster editing cost better than optimal cluster editing cost.

problem. For instance, an optimal cluster editing solution and an overlapping cluster editing solution, generated by HHM1, of the *cmpr*_101_5_25 graph are depicted, respectively, by Figures 2.5 and 2.6. As presented in Table A.2 of Appendix A, the optimal cluster editing cost of instance *cmpr*_101_5_25 is 44 and the cost of the overlapping cluster editing solution generated by HHM1 is 42.

Figure 2.5 - An optimal cluster editing solution, with cost of 44, of the instance *cmpr*_101_5_25. This solution was obtained from the resolution of the Charikar, Guruswami and Wirth (CHARIKAR et al., 2005) model. This image was generated using the Gephi software (<https://gephi.org/>).

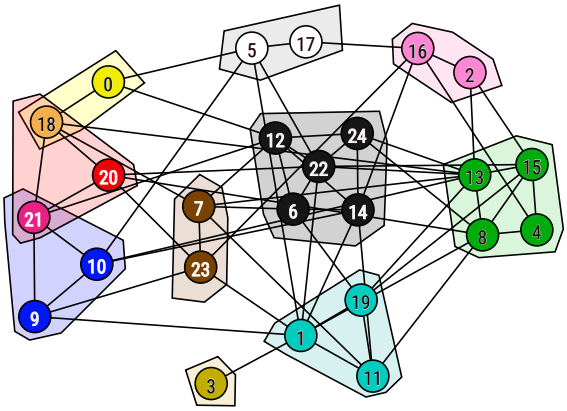


SOURCE: Produced by the author.

In the solution showed in Figure 2.6, vertices 18 and 21 are contained, each one, in two clusters. Thus, edges that would be inter-clusters in a disjoint clustering are intra-cluster, such as the edge $(18, 21)$. In a non-overlapping cluster editing solution, such as the solution presented in Figure 2.5, vertex 21 belongs only to the clique composed by vertices 9 and 10 and it is not contained in a cluster alongside vertex 18. Then, the edge $(18, 21)$ is a inter-cluster edge and, therefore, this edge is computed

as an edge that must be removed.

Figure 2.6 - An overlapping cluster editing solution, with cost of 42, of the instance *cmpr*_101_5_25. Vertices 18 and 21 belong, each one, to two clusters. This image was generated using the Gephi software (<https://gephi.org/>).



SOURCE: Produced by the author.

Analyzing the execution time of the hybrid heuristics variations, one can see that the HHM1 spent less computational cost to generate the solutions than the HHM2 and HHM3 in almost all random graphs instances. As mentioned previously, M1 is easier to solve than M2 and M3 because the first model has a smaller number of variables. Then, even though the HHM1 used a greater number of cluster editing solutions from the metaheuristics than the HHM2 and HHM3, the CPLEX (IBM Corporation, 2017) can solve M1 faster than M2 and M3.

2.5.3 Tests with LF benchmark graphs

Table 2.9 shows the number of best solutions costs obtained by the three hybrid heuristic versions and the metaheuristics in the 30 LF benchmark graphs. The solutions costs were computed considering the non-overlapping version of Equation 2.1 for metaheuristics and the overlapping version of Equation 2.1 for the hybrid heuristic. The 30 graphs were divided into six sets according to their number of vertices. Each instance set has five graphs with density ranging from sparse to dense.

The average computational time, in seconds, of the three hybrid heuristic variation and the two metaheuristics is presented in Table 2.10. The average execution time of the hybrid heuristic variations showed in this table were computed considering the execution time spent by metaheuristics and CPLEX (IBM Corporation, 2017).

Table 2.9 - Results summary of tests performed in the 30 LF benchmark graphs. The number of best costs solutions is shown for each of the hybrid heuristic variation and for BRKGA and SA metaheuristics.

		number of best solutions costs							
				HHM1		**HHM2**		**HHM3**	
n	#	**BRKGA**	**SA**	$z = 0$	$z = 1$	$z = 0$	$z = 1$	$z = 0$	$z = 1$
25	5	2	**5**	0	4	1	1	1	1
50	5	1	**4**	1	2	1	1	1	1
100	5	1	1	1	**5**	1	1	1	1
200	5	2	1	3	**4**	2	2	2	2
500	5	1	0	1	**3**	1	2	1	2
1000	5	1	2	1	**3**	1	1	1	2
total	30	8	12	7	**21**	7	8	7	9

Table 2.10 - Results summary of tests performed on the 30 LF benchmark graphs. The average computational is shown for each of the hybrid heuristic variation (HHM1, HHM2 and HHM3) and for BRKGA and SA metaheuristics. The hybrid heuristic variations' execution time is composed by the metaheuristics and the CPLEX times.

		average time (s)							
				HHM1		**HHM2**		**HHM3**	
n	#	**BRKGA**	**SA**	$z = 0$	$z = 1$	$z = 0$	$z = 1$	$z = 0$	$z = 1$
25	5	0.7	0.5	1.4	1.4	1.9	2.0	1.7	1.8
50	5	1.1	1.5	2.8	3.3	5.1	7.1	3.9	5.6
100	5	2.1	5.2	7.8	8.5	16.7	32.8	11.2	19.2
200	5	3.5	19.6	23.6	33.2	185.2	384.1	779.8	241.7
500	5	10.5	114.1	130.1	174.6	8166.1	8780.2	8769.7	8806.4
1000	5	29.8	441.8	720,2	548,4	9136,0	9158,1	9126,5	9163,5

The results regarding the overlapping cluster editing cost and execution time presented in the above tables, were similar to the results with random graph. However, we can observe, in Table 2.6, that the three hybrid heuristic versions were a slightly better in terms of solution cost. The HHM1, with $z = 1$ obtained best costs in 21 instances. We also observe better costs, in relation to the tests in random graphs, of the solutions generated by the hybrid heuristic variations with $z = 0$. This is because these 30 LF benchmark graphs originally have overlapping clusters. That is, even though, with $z = 0$, the minimum overlapping is required, the hybrid heuristic' solution may still has some clusters that overlap.

Table 2.11 presents the $FBCubed$ metric values obtained by the three hybrid heuristics versions. In particular, the number of best $FBCubed$ values and the avegare $FBCubed$ values achieved by each hybrid heuristic are shown.

In relation to the results of the *FBCubed* metric, it is observed that the best results, both in absolute number and the average metric values were obtained by the HHM1 with $z = 1$. In particular, the HHM1, with $z = 1$, obtained *FBCubed* values greater

Table 2.11 - Summary of the hybrid heuristic *FBCubed* results of tests performed on the 30 LF benckmark graphs. The number of best *FBCubed* values and the average *FBCubed* value obtained by each hybrid heuristic variation (HHM1, HHM2 and HHM3) are shown.

		number of best *FBCubed*						**average *FBCubed***					
		HHM1		**HHM2**		**HHM3**		**HHM1**		**HHM2**		**HHM3**	
n	#	$z=0$	$z=1$	$z=0$	$z=1$	$z=0$	$z=1$	$z=0$	$z=1$	$z=0$	$z=1$	$z=0$	$z=1$
25	5	0	**4**	0	0	0	1	0.46	**0.67**	0.51	0.54	0.51	0.60
50	5	0	**5**	0	0	0	0	0.38	**0.53**	0.34	0.38	0.34	0.38
100	5	0	**5**	0	1	0	1	0.31	**0.54**	0.33	0.35	0.33	0.35
200	5	0	**5**	0	0	0	0	0.28	**0.59**	0.29	0.28	0.29	0.28
500	5	1	**3**	0	1	0	0	0.22	**0.27**	0.18	0.23	0.22	0.22
1000	5	0	1	**2**	1	1	1	0.09	**0.23**	0.15	0.15	0.14	0.14
total	30	1	**23**	2	3	1	3	-	-	-	-	-	-

than 0.5 in 16 instances. With these values of the *FBCubed* metric, the generated solutions can be considered good-quality clusterings.

Analogously to results presented in random graphs instances, the performance of HHM2 and HHM3 were also negatively affect by the limited number of input clusters. These 30 instances generated by the Lancichinetti and Fortunato (2009b) algorithm have clusters that overlap. Then, to produce a good overlapping cluster editing solution, it is necessary a great variety of clusters. As M2 and M3 used a small number of clusters, because they are harder to solve, this implied in overlapping solutions with higher costs. This is corroborated by the values of the *FBCubed* metric obtained by HHM2 and HHM3, which were only good in small graphs.

2.6 Conclusions and future directions

In this chapter we proposed a new relaxation of the cluster editing problem, the overlapping cluster editing problem. In addition, three hybrid heuristics versions for this problem were introduced. These hybrid heuristics are based on coupling the BRKGA and SA metaheuristics, to generate solutions for the cluster editing problem, and the CPLEX (IBM Corporation, 2017) that uses the clusters from these solutions as input to solve one of three mixed integer linear program, also proposed in this work.

Taking into account the results in all 112 random graphs instances it can be considered that the hybrid heuristics variations produced good quality solutions in overall. The best hybrid heuristic variation was the HHM1, with $z = 1$, that obtained the minimum overlapping cluster editing costs in 51 instances. In addition, this variation was the faster hybrid heuristic. Futhermore, the SA metaheuristic obtained the best results in 28 instances, the HHM2, with $z = 1$, obtained the best results in

18 instances, the BRKGA metaheuristic obtained the best results in 15 instances and the HHM3, with $z = 1$, obtained the best results in 12 instances. Besides the Bastos et al. (BASTOS et al., 2016) instances being randomly generated without cluster formations, another reason that may have influenced the HHM2 and HHM3 results was the number of solutions in the cluster editing solutions set used as input. Because the models of these hybrid heuristics are harder to solve, only 10 cluster editing solutions were gathered from metaheuristics. Then, a small number of cluster editing solutions implies in a small number of good-quality clusters. However, it can be considered that, even with theses limitations, HHM2 and HHM3 achieved reasonable results. As HHM2 and HHM3 generate overlapping clustering through the generation of set-covers, they can be an alternative to HHM1 in others overlapping clustering problems where the computational time is not a main concern.

In the tests with the LF benchmark graphs, the HHM1 obtained better costs in 21 of the 30 instances. This hybrid heuristic version also achieved good values of the *FBCubed*, a supervised metric. Although improvements have yet to be made in HHM2 and HHM3, specially in overlapping solutions, these hybrid heuristic variations have proved to be promising.

In addition, with our new problem definition we presented an alternative to the original cluster editing problem. As shown in the tests realized in this work, in the overlapping cluster editing problem one can reduce the number of inter-cluster edges by overlapping two or more clusters. Then, it may be better insert a vertex in more than one cluster than to remove edges from that vertex.

For future work, some points of the hybrid heuristic should be improved, mainly the HHM2 and HHM3 versions. For example, the number of clusters to be used in an overlapping clustering solution and increase the variety of cluster editing solution set. To increase the variety of this set, other metaheuristics and other methods, such as the column generation method (OLIVEIRA et al., 2017), can be implemented. In addition, another direction is to apply our hybrid heuristic in the context of overlapping community detection, since it performed well retrieving ground-truth overlapping clustering. In order to achieve this, methods for community detection should be implemented instead of BRKGA and SA metaheuristics for clusters generation.

3 A HYBRID HEURISTIC FOR THE OVERLAPPING COMMUNITY DETECTION PROBLEM

This chapter is divided as follows. An introduction is presented in Section 3.1. Some related work are described in Section 3.2. In Section 3.3, we present the mathematical notation and some definitions necessary to understand this chapter. The details about our hybrid heuristic are given in Section 3.4. We show the experimental results and an analysis of them in Section 3.5. Our concluding remarks and considerations on future work are presented in Section 3.6.

3.1 Introduction

Community structures can be found in many real-world networks arising from several different areas, such as computer science, economics, biology, sociology and engineering (GIRVAN; NEWMAN, 2002; FORTUNATO, 2010). The task of identification of such structures is an interdisciplinary, widely known problem called *community detection* (FORTUNATO, 2010). Although there is no universally accepted formal definition of community, it is often assumed that a community is a set of elements with more links among them and less, or none, links to the elements that do not belong to the set (FORTUNATO, 2010; XIE et al., 2013). In graph theory context, these elements are vertices and the relationship between them can be represented by edges. So the objective in the community detection problem is to find disjoint sets, i.e., sets of highly connected vertices that share no vertices with other sets. These sets are also known as *clusters* and in this work this term is used interchangeably with communities.

In many real-world applications, however, vertices can belong to more than one cluster, that is, clusters may overlap (FORTUNATO; HRIC, 2016; CHAGAS et al., 2019). In social networks, for example, individuals frequently have several relationships with other individuals and they are usually associated to many groups (XIE et al., 2013). Overlapping communities are also frequently observed in data mining-related problems, since web pages, documents, users info and many other data can be categorized to more than one class (BONCHI et al., 2013). In a biological context, proteins often compose one or more protein complexes in protein interactions graphs (WANG et al., 2018). Therefore, identifying overlapping clusters in these networks is a relevant task and it is known as *overlapping community detection problem* (OCDP). In this problem, the objective is to partition vertices of a input graph into sets that are not necessarily disjoint.

There are some metrics in the literature used to evaluate the quality of a cluster of an overlapping clustering solution. Among these metrics one of most used and known is the *conductance measure* (ŠÍMA; SCHAEFFER, 2006). This metric evaluates the two main characteristics of a community: the number of edges inside of a cluster and the number of edges between this cluster and the remaining other vertices of the graph (FORTUNATO; HRIC, 2016). However, finding a cluster with minimal conductance is a NP-hard problem (ŠÍMA; SCHAEFFER, 2006). Then, exact methods, which can find clusters with optimal conductance, are only practical on small graphs whereas heuristics can find clusters at a reasonable time but without guarantee of optimality. An alternative is to combine heuristics with exact methods in order to produce high-quality solutions at a reasonable computational time. Such methods are known as *hybrid heuristics* or *matheuristics* (MANIEZZO et al., 2009), which lately have been successfully applied to optimization and clustering problems (OLIVEIRA et al., 2014; PEREIRA et al., 2015; OLIVEIRA et al., 2017; CHAGAS et al., 2019; MOUSSAVI et al., 2019).

The main contribution of this work is the proposal of a hybrid heuristic for detecting overlapping communities of a graph by the minimization of the conductance metric. Based on the work of Chagas et al. (2019), our hybrid heuristic consists of two algorithms that generate a set of clusters that is used to solve a MILP. We adapted the MILP for *overlapping cluster editing problem* proposed by Chagas et al. (2019) to the context of the overlapping community detection problem. An overlapping clustering is generated through the resolution of this MILP. In the sequence, some local search methods are used to improve the overall conductance of the clustering. As far as we known, there were no previous hybrid heuristics proposed for community detection problems. Furthermore, to the best of our knowledge, there are no MILP models in the literature for detecting overlapping clustering. The only work that we are aware of is the paper of Bennett et al. (2014). However, these authors proposed a mixed integer non linear programming for modularity maximization in the overlapping community detection.

3.2 Related work

Initial research in community detection was carried out considering only disjoint clusters. The overlapping community detection problem has been receiving increasingly attention recently mainly because of social networks analysis problems (ALGHAMDI; GREENE, 2019; GAO et al., 2019b). Indeed, several algorithms have been proposed for this problem lately (WHANG et al., 2016; CHANG et al., 2019;

GAO et al., 2019a; GAO et al., 2019b; SHENG et al., 2019). These methods can be divided into five categories: *clique percolation*, *link clustering*, *fuzzy detection*, *label propagation* and *local expansion* algorithms (XIE et al., 2013).

Algorithms that find a cluster by identifying overlapping fully *connected subgraphs*, which are known as *cliques*, are classified as clique percolation methods (XIE et al., 2013). An example of algorithm of this category is the *Clique Percolation Method* (PALLA et al., 2005b). This method was the first algorithm proposed for overlapping community detection and it is one of the most popular methods for this task. Link clustering methods find clusters through the partitioning of the graph by splitting edges rather than vertices (XIE et al., 2013; ALGHAMDI; GREENE, 2019). In fuzzy algorithms, vertices have a degree of membership related to each cluster ranging from 0 to 1, where the sum of all the memberships is equal to 1 (XIE et al., 2013; CHAGAS et al., 2019). Label propagation algorithms generate a cluster by inserting a vertex on it based on its adjacency affinities (ALGHAMDI; GREENE, 2019).

One of the most successful class of methods for overlapping community detection are the local expansion methods, which use the "seed-and-grow" strategy, i.e., growing a cluster from a given initial vertex (XIE et al., 2013; WHANG et al., 2016). Indeed, as shown by Xie et al. (2013), local expansion methods achieved the overall best results finding overlapping communities when comparing to other methods. There are several methods based on local expansion for overlapping community detection in the literature (LANCICHINETTI; FORTUNATO, 2009a; McDaid; HURLEY, 2010; LANCICHINETTI et al., 2011; WHANG et al., 2016; GAO et al., 2019a; GAO et al., 2019b).

The *Local fitness maximization* (LFM) (LANCICHINETTI; FORTUNATO, 2009a) is one of the most known methods that uses the "seed-and-grow" strategy. This algorithm starts finding a cluster from a random vertex and, at each iteration, a vertex is added if the fitness function value is increased. The algorithm stops when all vertices were assigned to at least one cluster. The *Model-based overlapping seed expansion* (MOSES) (McDaid; HURLEY, 2010) generates clusters using a statistical model and heuristics to greedily expanding a cluster through the maximization of its objective function (McDaid; HURLEY, 2010). Another algorithm that also uses a statistical model is the *Order statistics local optimization method* (OSLOM) (LANCICHINETTI et al., 2011). This method grows clusters optimizing a fitness function that measures the statistical significance of clusters in comparison to random variations (LANCICHINETTI et al., 2011).

The *Neighborhood-inflated seed expansion* (NISE) (WHANG et al., 2016) is a state-of-the-art algorithm for overlapping community detection. After identifying the most important region of the input graph, this algorithm finds a set of seeds on it and finds a cluster starting from each of these seeds by applying the *PageRank-Nibble* (ANDERSEN et al., 2006) algorithm. Another algorithm that uses a similar approach is the *Local expansion conductance minimization* (LECM) (GAO et al., 2019b). This method also applies the PageRank-Nibble algorithm in a set of seeds. However, in the LECM a series of local search methods for conductance improvement are used on every cluster found by the PageRank-Nibble algorithm. LECM is a state-of-the-art algorithm for detecting overlapping communities based on the conductance minimization (GAO et al., 2019b).

3.3 Mathematical notation and problem definition

Consider a *simple*, *undirected* and *unweighted* graph $G = (V, E)$, where V is a set of vertices and E is the set of edges in which $|V| = n$ and $|E| = m$. Two vertices $v \in V$ and $u \in V$ are *adjacent* if and only if $\{v, u\} \in E$. In this case, vertices v and u are *endpoints* of edge $\{v, u\}$. The *degree* of a vertex v, i.e., its number of adjacent vertices is defined by $deg(v) = |\{\{v, u\} \mid \{v, u\} \in E\}|$. A graph is *complete* if each vertex is adjacent to every other vertex. In a complete graph, $\forall v \in V$, $deg(v) = n-1$ and $m = \frac{n(n-1)}{2}$.

A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. A subgraph of G *induced* by a subset of vertices $V' \subseteq V$ is a graph $G_{V'} = (V', E_{V'})$ where $E_{V'} = \{\{v, u\} \mid v \in V' \wedge u \in V' \wedge \{v, u\} \in E\}$. In other words, $G_{V'}$ is induced by V' if it has all edges of G joining vertices of V'. G is *connected* if for every pair of vertices $v \in V$ and $u \in V$ there is a finite sequence of distinct edges connecting them. A *connected component* is a *maximal* connected subgraph, i.e., it is not a proper subset of any other connected component.

Although there is no standard formal definition of cluster in the literature (FORTUNATO; HRIC, 2016), in this work we consider that a cluster C is a set of vertices such that $C \subseteq V$. Note that, in our definition, C is a set of vertices, therefore it has no edges. Then, when we refer to "edges of C" we are considering edges of the original edge set E in which at least one of its vertices belongs to C.

There are two types of edges of a cluster C: *internal edges* and *external edges*. The set of internal edges of C, that is, edges in which both endpoints belong to C is given by $\textit{int-edges}(C) = \{(v, u) \in E \mid v, u \in C\}$. Note that, in our definition,

the internal edges set is formed by ordered pairs of vertices, then every edge is considered twice since $(v, u) \neq (u, v)$. The set of *external edges*, also known as *cut set*, is defined by $ext\text{-}edges(C) = \{\{v, u\} \in E \mid v \in C \wedge u \notin C\}$, i.e., edges where one of its vertices belongs to C and the other does not belongs to C. The number of external edges of C is called *cut* and it is represented by $cut(C)$. In addition, $adj(C) = \{v \mid \exists u \in C, \{v, u\} \in ext\text{-}edges(C)\}$ is the set of all vertices not contained in C that have at least one adjacent vertex in C. The number of edges between vertices belonging to C and a vertex v is $l(C, v) = |\{\{v, u\} \in E \mid u \in C\}|$. The set comprehending all edges of C is given by the union of the internal edges and the external edges and the number of edges of this set is the *degree* (or *volume*) $deg(C)$ of C. In other words, $deg(C) = |int\text{-}edges(C) \cup ext\text{-}edges(C)| = \sum_{v \in V} deg(v)$. Note that $deg(V) = 2m$.

With these definitions, we can define the *conductance* measure. The conductance of a cluster is the ratio between its cut and the minimum value between the cluster's degree and the degree of the set composed by all remaining graph vertices. The conductance is given by Equation (3.1).

$$\Phi(C) = \frac{cut(C)}{\min(deg(C), 2m - deg(C))}. \tag{3.1}$$

Two distinct clusters C_i and C_j are *disjoint* sets if $C_i \cap C_j = \emptyset$. If $C_i \cap C_j \neq \emptyset$, then C_i is an *overlapping cluster* of C_j and vice versa. The *Jaccard coefficient*, between two clusters C_i and C_j, is defined by $J(C_i, C_j) = |C_i \cap C_j|/|C_i \cup C_j|$. Then, two clusters C_i and C_j are equal when $J(C_i, C_j) = 1$. If $J(C_i, C_j) = 0$, then C_i and C_j have no elements in common, i.e., $C_i \cap C_j = \emptyset$ (CHAGAS et al., 2019).

A traditional *clustering* is a vertex-disjoint partitioning $\mathcal{C} = \{C_1, C_2, \ldots, C_l\}$ such that, for $1 \leq i, j \leq l$ and $i \neq j$, $C_i \subseteq V$, $C_i \neq \emptyset$, $\bigcup_{i=1}^{l} C_i = V$ and every pairwise intersection $C_i \cap C_j = \emptyset$. In an *overlapping clustering* $\exists C_i \in \mathcal{C}$ and $\exists C_j \in \mathcal{C}$, with $C_i \neq C_j$, such that $C_i \cap C_j \neq \emptyset$, i.e., the partition is not disjoint. In the community detection context, an overlapping clustering is often referred as *cover* (FORTUNATO, 2010).

3.4 Hybrid heuristic

Our hybrid heuristic is divided into three phases, namely *clusters set generation*, *model resolution* and *clusters improvement*. In the first phase, an heterogeneous set of good-quality clusters is generated through the execution of the LFM (LANCI-

CHINETTI et al., 2009) algorithm and the NISE (WHANG et al., 2016) algorithm. These two methods produce, each one, an overlapping clustering of the input graph and the clusters from both clustering solutions compose the clusters set. Thereafter, in the model resolution phase, this set is used as input to solve the MILP for overlapping clustering as proposed by (CHAGAS et al., 2018; CHAGAS et al., 2019). For simplicity, in this work, we call this MILP as *Overlapping Clustering Model* (OCM). The quality of the overlapping clustering generated by the OCM resolution is highly dependent on the quality and the variety of the clusters set (CHAGAS et al., 2019). In the third phase, we try to improve the conductance of each cluster of the overlapping clustering generated by the resolution of the OCM using the three methods for cluster refinement of the LECM (GAO et al., 2019b) algorithm.

Among a vast literature regarding local expansion methods (XIE et al., 2013; PADROL-SUREDA et al., 2010; McDaid; HURLEY, 2010; LANCICHINETTI et al., 2011; GAO et al., 2019b), we chose to implement the LFM (LANCICHINETTI et al., 2009) and NISE (WHANG et al., 2016) algorithms, two efficient and well known methods, to generate the OCM input clusters set. We utilized two local expansion methods because this class is one of the most successful class of methods for overlapping community detection (XIE et al., 2013). Even though both are algorithms of the same category, they have significant differences, which lead our hybrid heuristic to generate a diverse set of good-quality clusters in the first phase.

Since the LFM is a simple, widely known and efficient overlapping community detection algorithm, we utilized it to generate clusters due to two main reasons. First, LFM starts finding a cluster from a random vertex. Although it may not always generate good-quality clusters, expanding a cluster from a random vertex is important to maintain the heterogeneity of the clusters set. Second, the clusters' size of a LFM clustering can be controlled by an input parameter. This also contributes to the clusters set diversity, since we can run LFM using different values of the cluster control size parameter. In addition, several recent papers of proposing new overlapping community detection algorithms compares its results with LFM, e.g. (CHANG et al., 2019; DENG et al., 2019; YAN et al., 2018; ZHENG et al., 2019).

We implemented the NISE algorithm for generating clusters because it is one of the state-of-the-art overlapping community detection methods (WHANG et al., 2016; XU et al., 2016; GAO et al., 2019b). In tests carried out by (WHANG et al., 2016), it was shown that NISE achieved the bests results not only in the conductance measure, but also in the modularity measure, average association measure and retrieving

ground-truth clusters. Then, in our hybrid heuristic, while LFM contributes mainly to the diversity of the clusters set, NISE contributes adding high quality clusters to it. In addition, NISE has reasonable low computational cost since its time complexity is roughly $O(n + m)$ (WHANG et al., 2016).

The OCM (CHAGAS et al., 2019) was proposed for the *overlapping cluster editing* problem. In this problem, the goal is to partition the vertices of a graph into *maximal cliques*, that are not necessary disjoint, by adding and deleting edges (CHAGAS et al., 2019). A maximal clique is a complete subgraph not strictly contained in any other complete subgraph. Since the OCM presented good results, at a low computational cost, in the overlapping cluster editing problem and since it can be easily adapted to other overlapping clustering problems, we applied it in the overlapping community detection problem.

As Gao et al. (2016), Gao et al. (2019a), Gao et al. (2019b) state, the LECM is a state-of-the-art algorithm for conductance minimization in overlapping community detection. Indeed, the LECM achieved better results than NISE regarding the average conductance metric. Then, we utilized the three methods of the last phase of the LECM for conductance minimization seeking improve the conductance of the overlapping clustering obtained through the OCM resolution.

A briefly pseudocode of our hybrid heuristic is depicted in Algorithm (2) and figure 3.1 depicts the execution of the proposed method. First, the clusters set is generated by the execution of LFM and NISE. Repeated clusters are removed from this set in the filtering step at line 4. Then, CPLEX (IBM Corporation, 2019) solves the OCM using the clusters set as input. Thereafter, the conductance of each cluster of the overlapping clustering found by the OCM resolution is refined by LECM local search methods.

Each step of the Algorithm (2) is detailed in the remainder of this section and it is divided as follows. A briefly description of LFM and NISE algorithms are presented in Sections 3.4.1 and 3.4.2, respectively. The MILP utilized in this work is described in Section 3.4.3. The LECM methods for improve the conductance of the clustering generated by our hybrid heuristic are presented in Section 3.4.4.

3.4.1 Local Fitness Maximization algorithm

The LFM algorithm (LANCICHINETTI et al., 2009) is a simple but efficient greedy method for detecting both overlapping and hierarchical clusters. Its main idea is to

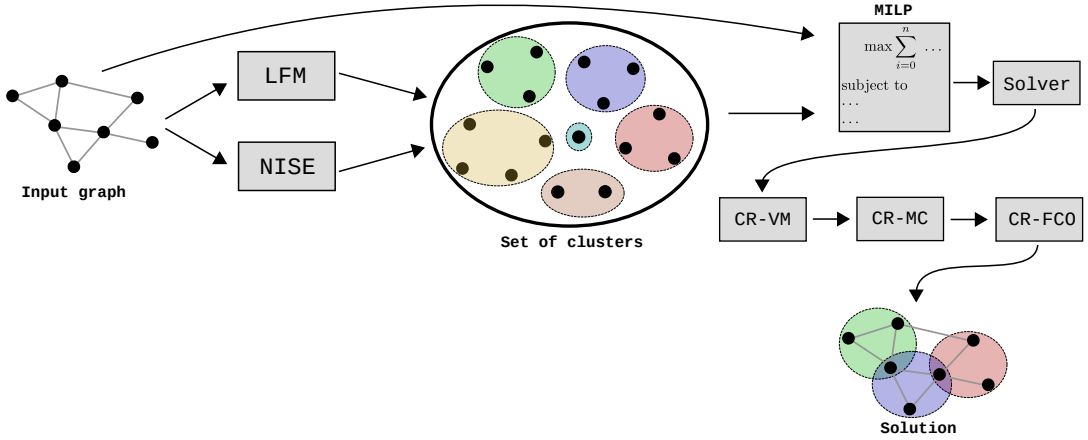
Algoritmo 2: Hybrid heuristic.

input : Graph $G = (V, E)$; OCM *model*; OCM number of clusters k; LFM cluster size control parameter b; NISE number of seeds ns; NISE teleport probability α; NISE acurracy ϵ; merging coefficient β; combining function balance parameter θ; conductance control parameter χ.

output: Clustering $\mathcal{C}$.

```
1 begin
    /* a set of clusters is generate by LFM and NISE                    */
2   clusters ← lfm(G, b);
3   clusters ← clusters ∪ nise(G, α, ϵ);
    /* Repeated clusters are removed from the set                       */
4   clusters′ ← filtering_step(clusters) ;
    /* The clusters′ set is used as input to solve OCM                  */
5   C ← cplex_solve(model, clusters′, k);
    /* LECM methods are used to improve the clustering                  */
6   improve_clustering(C, β, θ, χ);
7   return C;
8 end
```

Figure 3.1 - Overall execution of the proposed hybrid heuristic.



SOURCE: produced by the author.

create a cluster by, starting from a random vertex, iteratively inserting adjacent vertices seeking to maximize the cluster's LFM *fitness* value. The fitness value of a cluster is the ration between its number of internal edges and its degree to the power of b, where $b \in [0.5, 2.0]$ is a parameter to control the clusters size found by LFM (LANCICHINETTI et al., 2009). The function to compute this value is defined by Equation (3.2). In this equation, if b is close to 0.5, then LFM identifies large

clusters since the number of internal edges will be more relevant in the ratio. If b is close to 2, then LFM finds small clusters since the cluster's degree will be more relevant.

$$fit(C) = \frac{|int\text{-}edges(C)|}{deg(C)^b}. \tag{3.2}$$

After each vertex insertion, the algorithm checks if some vertex belonging to the cluster has a negative fitness value. The fitness of a vertex v in regard to a cluster C is the difference between the fitness of C with v and the fitness of C without v (LANCICHINETTI et al., 2009). This value is given by Equation (3.3).

$$fit(C, v) = fit(C \cup v) - fit(C \setminus v). \tag{3.3}$$

When there is no adjacent vertex to the cluster or all vertices adjacent to it have negative fitness value, then the insertion procedure stops and the cluster of the starting vertex is found. These steps are repeated until all vertices are assigned to, at least, one cluster.

A summarized pseudocode of LFM is shown in Algorithm 3. In the inner most while loop of Algorithm 3, a cluster is generate starting from a vertex randomly selected, at line 3, from vertices that were not assigned to any cluster. At each iteration, a vertex adjacent to the current cluster that has the largest positive fitness is inserted. If after this insertion the fitness of some of the cluster's vertices becomes negative, then it is removed at line 7. When there is no vertex adjacent to the cluster with positive fitness, this cluster is attached to the final clustering at line 9. The algorithm stops after all vertices were clusterized. For more details about the LFM algorithm, please see the work of Lancichinetti et al. (2009).

3.4.2 Neighborhood-Inflated Seed Expansion algorithm

The NISE algorithm is divided in four phases: filtering, seeding, seed expansion and propagation (WHANG et al., 2016). Firstly, the input graph is filtered in order to identify the its main region where NISE must focus on. Then, seeds are selected from this main region and, thereafter, they are expanded considering a personalized PageRank for compose, each one, a cluster. Remaining vertices that does not belong to the main region are inserted in one or more clusters in the propagation phase. Each of these phases are briefly described in the following subsections. Please see

Algoritmo 3: LFM (LANCICHINETTI et al., 2009).

input : Graph $G = (V, E)$; fitness cluster size control parameter b.
output: Clustering $\mathcal{C}$.

```
1  begin
2      while there is a vertex not yet assigned to any cluster do
3          C ← ∅; C ← C ∪ pick_vertex_at_random(V \ C) ;
4          while there is some vertex u ∈ adj(C) with fit(C, u) ≥ 0 do
5              forall u ∈ adj(C) with fit(C, u) ≥ 0 do
                   /* finds vertex u with the largest fitness computed by
                      Eq. (3.3). Inserts it into C                          */
6              end forall
7              while fit(C, v) < 0 for some v ∈ C do C ← C \ v ;
8          end while
9          C ← C ∪ C ;
10     end while
11     return C;
12 end
```

the work of Whang et al. (2016) for a full description of NISE.

3.4.2.1 Filtering

In the filtering step, the input graph is divided between a region where the method must be applied to and regions where it must not. For this purpose, NISE seek to identify the *biconnected core* and the *whiskers* of the given graph. A biconnected core of a graph is its largest connected component where all *biconnected components* with one edge were removed (WHANG et al., 2016). G is a biconnected graph if it continues connected after removing from it any single vertex and its adjacency (WHANG et al., 2016). Then, a biconnected component is a maximal induced subgraph that is biconnected, i.e., a biconnected induced subgraph that is not contained in any other biconnected induced subgraph. Whiskers are maximal subgraphs that are connected to the biconnected core only by a *bridge* edge (WHANG et al., 2016). Bridges are single-edge biconnected components that are attached to the biconnected core (WHANG et al., 2016).

The seeding and the seed expansion phases are applied only to the biconnected core. Then, clusters are detected only considering vertices belonging to this graph component. Since whiskers are maximal subgraphs that are attached to the biconnected core only by one edge, there is no overlap between any of the whiskers (WHANG et al., 2016). In this way, vertices belonging to whiskers are only clustered in the last

NISE step.

In order to detect the biconnected core, the recursive Hopcroft and Tarjan (1973) algorithm is utilized to find all biconnected components of the input graph. Then, all single-edge biconnected components are removed from it. The original graph is now partitioned into detached subgraphs. Each of these subgraphs, which are connected components, are identified by the breath-first-search algorithm and the largest among them is selected as the biconnected core. The remaing connected components are whiskers.

3.4.2.2 Seeding by Spread Hubs

Whang et al. (2016) proposed two methods for finding seeds for NISE: *Graclus centers* and *spread hubs*. The former uses the Graclus algorithm (DHILLON et al., 2007) to produce an initial clustering and then it selects a seed from each cluster. The latter finds a set of seeds by selecting vertices in decreasing degree order in which none is adjacent to a previous selected vertex. Since the spread hubs method is faster than Graclus centers (WHANG et al., 2016), we utilized the version of NISE with seeding by spread hubs.

The spread hubs method starts by sorting all vertices in decreasing degree order and marking them as unvisited. Then, the method picks one unvisited vertex of highest degree, sets it and its adjacency as a seed and marked them as visited. Note that a seed is composed by an unvisited vertex of highest degree and its adjacency. This procedure is repeated until ns seeds are selected. If there is a tie in the degree of unvisited vertices, then an *independent set* are selected among them to compose, each one, a seed. An independent set of vertices is a set where there is no adjacency between any of them.

3.4.2.3 Seed expansion

In this phase, clusters are generated, from each seed selected in the previous step, by the PageRank-Nibble algorithm (ANDERSEN et al., 2006). This algorithm computes an *approximate personalized PageRank* (ANDERSEN et al., 2006) vector for each seed and then performs a *sweep* over it for finding cluster with small conductance. A personalized PageRank vector is a stationary probability distribution of a random walk, over the vertices of a graph, starting from a non-uniform initial distribution (PAGE et al., 1999; ANDERSEN et al., 2006). In the PageRank-Nibble algorithm, Andersen et al. (2006) use an approximation, by a factor of ϵ, of person-

alized PageRank vectors which they called ϵ-*approximate PageRank*. The method for compute such vectors is depicted by Algorithm 4.

Algorithm 4 starts by initializing the ϵ-approximate PageRank vector p as 0 for all vertices of the graph and the residual vector r as $\frac{1}{|S|}$ for all vertices of the seed S and 0 for remaining vertices. A vertex with non-zero initial r value is a *restart vertex* (WHANG et al., 2016). Using the entire set S as restart vertices instead of single vertex is called *neighborhood inflation* and it generates clusters with better conductance (WHANG et al., 2016). Then, at each step, a vertex v such that $r[v] \geq \epsilon \cdot deg(v)$ is selected and a fraction of $(1 - \alpha)$ is transferred from its r value to its p value. The remaining α fraction of $r[v]$ is spread among vertex v adjacency. In order to maintain for which vertices $v \in V$ the inequality $r[v] \geq \epsilon \cdot deg(v)$ holds, the method uses a queue that stores vertices without repetition (ANDERSEN et al., 2006). In this way, whenever the r value of a vertex increases and becomes greater than ϵ times its degree, it is then pushed in the queue if it was not previous inserted. The algorithm stops when the queue is empty, that is, when there is no vertex $v \in V$ that satisfies the inequality $r[v] \geq \epsilon \cdot deg(v)$.

Algoritmo 4: ϵ-approximate PageRank (ANDERSEN et al., 2006).

input : biconnected core graph $G_{bc} = (V_{bc}, E_{bc})$; seed set S; teleport probability α; accuracy ϵ.
output: approximate PageRank scores p.

begin
  $p \leftarrow \vec{0}$;
  $r \leftarrow \frac{1}{|S|}$; `// for every` $v \in S$, `0 otherwise`
  unique_queue.push(S); `// queue without repeated vertices`
  while *unique_queue is not empty* **do**
    $v \leftarrow$ *unique_queue.pop*();
    foreach v *such that* $\{v, u\} \in E_{bc}$ **do**
      $r[u] \leftarrow r[u] + \frac{\alpha \cdot r[v]}{2 \cdot deg(v)}$;
      if $r[u] \geq \epsilon \cdot deg(u)$ **then** *unique_queue.push*(u) ;
    end foreach
    $p[v] \leftarrow p[v] + (1 - \alpha) \cdot r[v]$; $r[v] \leftarrow \frac{\alpha \cdot r[v]}{2}$;
    if $r[v] \geq \epsilon \cdot deg(v)$ **then** *unique_queue.push*(v) ;
  end while
  return p;
end

After computing the p vector, a *degree-normalized sweep* over it is realized to

generate a cluster (WHANG et al., 2016). This sweep is realized by sorting the vertices by the decreasing *probability-per-degree* order, i.e., an ordering such that $\frac{p[v_i]}{deg(v_i)} \geq \frac{p[v_{i+1}]}{deg(v_{i+1})}$ for $i = 1, \ldots, |V|$ (ANDERSEN et al., 2006). A cluster is find by selecting the first vertices from this sequence that achieves the smaller conductance.

3.4.2.4 Propagation

In the last NISE phase, clusters are found for vertices belonging to the whiskers identified in the filtering step. Each whisker is attached to the biconnected core by a bridge edge. One of the vertices of a bridge belongs to the biconnected core and the other one belongs to the whisker itself. In this way, all vertices of a whisker are inserted in all clusters that the bridge's vertex of the biconnected core is contained. As proved by Whang et al. (2016), this step always decrease the number of external edges of the clusters.

3.4.3 Mixed integer linear program for overlapping clustering

The MILP for overlapping community detection utilized in our hybrid heuristic is shown in Equations (3.4a) to (3.4e) (CHAGAS et al., 2019). From an input cluster set $CS = \{C_1, C_2, \ldots, C_N\}$, by solving OCM, an overlapping clustering $\mathcal{C} \subseteq CS$, where $|\mathcal{C}| = k$ and $\bigcup_{C \in \mathcal{C}} C = V$ is generated. In order to achieve this, OCM considers the Jaccard coefficient between each pair of clusters to regulate, by the overlapping control parameter, the overlap between them. This control is necessary because, depending on the graph context, it may be better to use clusters that share more vertices or less vertices (CHAGAS et al., 2019). In addition, the OCM's objective function seeks to maximize the quality coefficient associated to each selected cluster while minimizing the absolute difference of the Jaccard coefficient between the selected clusters and the overlapping parameter (CHAGAS et al., 2019). In this work we utilized the conductance metric as the quality coefficient associated to each cluster. Furthermore, the cover of every graph vertex is guaranteed in this model.

$$\max \sum_{i=1}^{N} (1 - \Phi(C_i)) \cdot y_i - u_i \tag{3.4a}$$

subject to

$$\sum_{j=1}^{N} \left| J(C_i, C_j) - z_i \right| \cdot (y_i + y_j - 1) \leq u_i, \ i = 1, 2, \ldots, N, \tag{3.4b}$$

$$\sum_{i=1}^{N} y_i = k, \tag{3.4c}$$

$$\sum_{i=1}^{N} a_{ji} \cdot y_i \geq 1, \ j = 1, 2, \ldots, n, \tag{3.4d}$$

$$y_i \in \{0, 1\}, u_i \in \mathbb{R}, \ i = 1, 2, \ldots, N. \tag{3.4e}$$

As defined by constraint (3.4e), there is two types of variables in OCM, the binary variables y_i and the real variables u_i, where $i = 1, 2, \ldots, N$. Binary variables y_i control which clusters C_i belong, or not, to the final clustering (CHAGAS et al., 2019). At the objective function (3.4a), a cluster quality coefficient is associated with each of these variables. Since the objective function is a maximization and the conductance value is better when it is close to 0, we utilized $1 - \Phi(C_i)$ as the quality coefficient associated to variable y_i.

The value of u_i variables are defined by the constraint (3.4b). Considering that a negative coefficient (-1) is attached to each of theses variables at the objective function, which is a maximization, the values of u_i variables must be as small as possible. Therefore, the objective function is also a *max-min* function (CHAGAS et al., 2019). In constraint (3.4b), if $y_i = 1$, i.e., C_i is selected to compose the clustering, then it must has the minimum absolute difference between its Jaccard coefficient, related to every other selected cluster, and the overlapping control parameter $z_i \in [0, 1]$. If $y_i = 0$, that is, C_i is not selected to compose the clustering, then it must has the maximum absolute difference between its Jaccard coefficient, related to every other not selected cluster, and z_i. Since the Jaccard coefficient between two clusters C_i and C_j is close to 1 when both share most of their vertices and near to 0 when they share almost none, it is necessary to adapt the z_i value to each these scenarios. In this work, we defined the values of each z_i by Equation (3.5). The aim is to use a large z_i value if cluster C_i has a large number of external edges in comparison to the total number of edges. This is because, if C_i has a big number of external edges, then it is expected that it overlaps with other clusters. Therefore, it is better to select clusters that shares vertices with C_i. On the other hand, if C_i has few external edges, then it is expected that it shares less vertices with other clusters.

$$z_i = \frac{cut(C_i)}{|E|}. \tag{3.5}$$

Constraint (3.4c) guarantees that the final clustering is composed by k clusters. At

constraint (3.4d), it is ensured that each vertex $j \in V$ is covered by at least one cluster C_i by checking whether the sum of the elements of each row of the *matrix of belonging* $A = (a_{ji})$ is greater than one. In the matrix A, if the vertex j belongs to C_i, then $a_{ji} = 1$ and $a_{ji} = 0$ otherwise (CHAGAS et al., 2019).

3.4.4 Cluster refinement methods

Likewise NISE, LECM algorithm is based on the "seed-and-grow" strategy which is divided into three steps. The first two steps are quite similar to the NISE seeding and seed expansion phases. Indeed, LECM also uses the PageRank-Nibble algorithm for generate a cluster from a seed. In the third step, LECM then tries to minimize the conductance of the clusters found by this algorithm by applying three cluster refinement methods. In the first cluster refinement method, it is verified if the conductance of each cluster found in the previous step can be decreased by inserting or removing vertices. Similar clusters are merged in the second method if the conductance is improved. The third method tries to find clusters for vertices not yet assigned to any cluster. Each of these methods is described in the following subsections. For more details about them, see the work of (GAO et al., 2016; GAO et al., 2019a; GAO et al., 2019b).

3.4.4.1 Vertices movement

The vertices movement method was proposed by Gao et al. (2016) seeking improve the conductance of all clusters of a clustering through the insertion and removal of vertices. For each vertex contained in a cluster C it is verified whether the remotion of it reduces the conductance of C. In addition, for each vertex adjacent to C it is also verified whether the insertion of it reduces the conductance of C. When an improve of the conductance is detected then the vertex is inserted/removed.

Instead of computing $\Phi(C)$ after each vertex v insertion/removal, which requires an algorithm of $O(deg(C))$ time cost per operation, Gao et al. (2016) proposed two functions that calculate the conductance decrease in $O(deg(v))$ time. These two functions are presented in Equations (3.6) and (3.7), where the Equation (3.6) computes the decrease of $\Phi(C)$ after insertion of v and Equation (3.7) computes the decrease of $\Phi(C)$ after removal of v.

$$\Phi_i(C, v) = \frac{\frac{cut(C)}{cut(C)+|int\text{-}edges(C)|} \cdot deg(v) - deg(v) + 2l(C, v)}{cut(C) + deg(v) + |int\text{-}edges(C)|}. \tag{3.6}$$

$$\Phi_r(C, v) = \frac{deg(v) - 2l(C, v) - \frac{cut(C)+2l(C,v)-deg(v)}{cut(C)+|int\text{-}edges(C)|-deg(v)} \cdot deg(v)}{cut(C) + |int\text{-}edges(C)|}. \tag{3.7}$$

Gao et al. (2016) proved that $\Phi_i(C, v) = \Phi(C) - \Phi(C \cup v)$ and $\Phi_r(C, v) = \Phi(C) - \Phi(C \setminus v)$. If the value of both Equations (3.6) and (3.7) are positive, then it denotes that the insertion of v, in Equation (3.6), and the removal of v, in Equation (3.7), decreased the conductance of the cluster.

A pseudocode of the vertex movement method is presented in Algorithm (5). The algorithm looks for a conductance decrease in every cluster $C \in \mathcal{C}$. In the first $foreach$ inner loop, it is verified whether the removal of each $v \in C$ reduces $\Phi(C)$. If a conductance decreased is found after removing v, then it is selected as a "move-out" vertex. In the second $foreach$ inner loop, it is verified whether the insertion of each $v \in adj(C)$, that is not a "move-out" vertex, reduces $\Phi(C)$. If a conductance decreased is found after inserting v, then it is selected as a "move-in" vertex. Thereafter, all "move-out" vertices are removed from C and all "move-in" vertices are inserted in C.

Algoritmo 5: Vertex movement method (GAO et al., 2016).

input : Graph $G = (V, E)$; Clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$.
output: Improved clustering $\mathcal{C}'$.

```
1  begin
2      foreach C_i ∈ C do
3          foreach v ∈ C_i do
4              if Φ_r(C_i, v) > 0 then move_out.insert(v) ;
5          end foreach
6          foreach v ∈ adj(C_i) do
7              if Φ_i(C_i, v) > 0 then move_in.insert(v) ;
8          end foreach
9          foreach v ∈ move_out do C_i ← C_i \ v;
10         foreach v ∈ move_in do C_i ← C_i ∪ v;
11     end foreach
12     return C';
13 end
```

3.4.4.2 Merging clusters

In this method, two similar clusters are combined if the conductance of the combined cluster is better than the original ones. In order to identify if two clusters C_i and C_j need to be merged, Gao et al. (2019b) utilized the *combining function* $cf(C_i, C_j)$, which is defined by Equation (3.8).

$$cf(C_i, C_j) = \theta \cdot J(C_i, C_j) + (1 - \theta) \cdot \frac{\Phi(C_i) + \Phi(C_j)}{2 \cdot \Phi(C_i \cup C_j) + \Phi(C_i) + \Phi(C_j)}. \tag{3.8}$$

The first part of the Equation (3.8) measures how similar the two clusters are by using the Jaccard coefficient whereas the second part of it measures the conductance improvement after the merging. Gao et al. (2019b) used a parameter θ to control which part of Equation (3.8) should be more relevant.

A pseudocode of the merging clusters method is presented in Algorithm 6. This algorithm iterates over all clusters $C_i \in \mathcal{C}$ looking for clusters to be combined. At each iteration, it is checked if the combining function between C_i and one of its overlapping clusters is greater than or equal to the *merging coefficient* β. Gao et al. (2019b) proved that the conductance of a combined cluster usually decreases if $\beta > 0.75$ and $\theta = 0.5$. When two clusters C_i and C_j are merged, all vertices of C_j are inserted into C_i and C_j is removed from $\mathcal{C}$. Then, the *foreach* inner loop restarts from the updated list of overlapping clusters of the new C_i.

Algoritmo 6: Merging clusters (GAO et al., 2019b).

input : Clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$; merging coefficient β.
output: Improved clustering $\mathcal{C}'$.

```
1  begin
2      foreach C_i ∈ C do
3          foreach C_j ∈ C that overlaps with C_i do
4              if cf(C_i, C_j) ≥ β then
5                  C_i ← C_i ∪ C_j; C ← C \ C_j;
6              end if
7          end foreach
8      end foreach
9      return C';
10 end
```

3.4.4.3 Finding clusters for outliers

Although the cover of each vertex is guaranteed in OCM, some vertices may be unclustered after the vertex movement method. Then, we also utilized the method of Gao et al. (2019b) for find clusters for vertices not contained in any cluster. This method, which is shown in Algorithm (7), uses a set of inequalities to assign an unclustered vertex to a cluster seeking not increasing its conductance. If inequalities of lines 4 or 8 holds, then the unclustered vertex v is inserted in a cluster of $adj\text{-}cluster(v)$, where $adj\text{-}cluster(v)$ is the set of all clusters that the adjacent vertices of v belong to. For more details about this method, please see Gao et al. (2019b).

Algoritmo 7: Finding clusters for outliers (GAO et al., 2019b).

input : Vertices not assigned to any cluster; clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$; conductance control paramenter χ.
output: Modified clustering $\mathcal{C}'$.

1 **begin**
2 **foreach** *unclustered vertex* v **do**
3 **foreach** $C_i \in adj\text{-}cluster(v)$ **do**
4 **if** $1 - \left(2 \cdot \frac{l(C_i,v)}{deg(v)}\right) < \frac{deg(C_i) - |int\text{-}edges(C_i)|}{deg(C_i)}$ **then** $C_i \leftarrow C_i \cup v$;
5 **end foreach**
6 **end foreach**
7 **foreach** *still unclustered vertex* v **do**
8 **if** $\max\limits_{C_i \subset adj\text{-}cluster(v)} \frac{l(C_i,v)}{deg(v)} > \chi$ **then**
9 $C_i \leftarrow \arg\max\limits_{C | C \subset adj\text{-}cluster(v)} \frac{l(C_i,v)}{deg(v)}$;
10 $C_i \leftarrow C_i \cup v$;
11 **end if**
12 **end foreach**
13 **return** $\mathcal{C}'$;
14 **end**

3.5 Experimental results and analysis

In this section results of the hybrid heuristic tests are presented and it is divided as follows. The setup used for tests and implementation details are described in Subsection 3.5.1. We detail the graph instances and metrics used to evaluate our approach in Subsections 3.5.2 and 3.5.3, respectively. Results of our approach in artificial graphs instances and real-world graph instances are present in Subsection 3.5.4

and in Subsection 3.5.5, respectively.

3.5.1 Setup and implementation details

All implementations were written in *C++* language and compiled with *g++* compiler version 7.4 using -O3 flag. For the resolution of models we used the IBM® ILOG® CPLEX® 12.9 (IBM Corporation, 2019). All the computational tests were executed on a computer with Intel® Xeon® E5-2687W v2 CPU 3.40GHz × 8 processor with 25MiB *cache* memory and 62GiB of RAM. The operating system installed on this machine is Ubuntu 18.04.3 64 bits with kernel 5.0.0-23-generic. In addition, in all CPLEX (IBM Corporation, 2019) executions were limited to 3 hours.

As suggested by Lancichinetti et al. (2009), several overlapping clustering can be generated by running the LFM with different b values in parallel. We then ran the LFM in parallel with 16 different b values ranging between 0.55 and 1.0 seeking to increase the diversity of clusters set. We utilized 16 different values of b because this was the number of cores in the computer that we used for the tests. The range of $[0.55, 1.0]$, for the b parameter values, was empirically defined following (LANCICHINETTI et al., 2009). Then, in each instance, the LFM result presented is related to the execution that achieved the best results. We utilized LFM_{best} to represent it.

Table 3.1 presents the parameters values of each method that we utilized for the tests. The values of the NISE parameters and of the LECM methods parameters were used following the values established by the authors' of each method (WHANG et al., 2016; GAO et al., 2019b). The number of seeds (ns) of the NISE algorithm and the number of clusters (k) of OCM model were defined for each instance.

Table 3.1 - Parameters values of each method utilized in this work. For LECM and NISE algorithms, we utilized the values established in their papers (WHANG et al., 2016; GAO et al., 2019b).

	LECM					**LFM**	**NISE**		
Parameter	α	β	ϵ	θ	χ	b	ns	α	ϵ
Value	0.99	0.8	10^{-4}	0.5	0.5	$[0.55, 1.0]$	-	0.99	10^{-4}

In addition, seeking evaluate each step of our hybrid heuristic, we utilized three versions of it in the tests performed in this work. The first hybrid heuristic version (HH-CR) is the complete version, that is, the method compose by LFM and NISE algorithms, the OCM and the clusters refinement local search (CR) methods. The second hybrid heuristic version (HH) is the basic version, i.e, the hybrid heuristic

without the clusters refinement local search methods (CR). We used the HH for analyse the influence of the CR methods on the results of our hybrid heuristic. The third version (HH-CR$_{LFM}$) is the complete version but without the NISE algorithm. The aim with this version was to analyse the difference on the hybrid heuristic results of using only LFM for generate the clusters set. A version of our method using only NISE was not possible, since the NISE algorithm do not always guarantees that every vertex is cover and we need this property for the feasibility of the OCM.

3.5.2 Instances

We utilized two types of graphs instances for evaluating our hybrid heuristic: *LF benchmark graphs* and *real-world graphs*. Following Gao et al. (2019b), we generated a set of 24 LF benchmark graphs of 10,000 vertices each and with ground truth overlapping clustering solutions. These instances were generated by the Lancichinetti and Fortunato (2009a) (LF) algorithm and were divided into four groups of six graphs according to the theirs generation parameters. The values of each LF parameter for generate each group of instances are shown in Table 3.2. These values are the same utilized by Gao et al. (2019a).

Table 3.2 - Parameters values of the LF algorithm (LANCICHINETTI; FORTUNATO, 2009a) used to generate each set of LF benckmark graphs. Each set LF_1, LF_2, LF_3 and LF_4 is composed by six graphs.

	LF algorithm parameter							
Set	n	k_{max}	k	$C_{\max}$	$C_{\min}$	O_n	O_m	μ
LF$_1$	10000	50	15	50	10	[0,5000]	2	0.1
LF$_2$								0.3
LF$_3$							4	0.1
LF$_4$								0.3

In Table 3.2, k_{max} is the maximum degree value of a vertex and k is the average degree of the vertices of the input graph. Parameters C_{max} and C_{min} are, respectively, the upper and lower bound of the number of vertices in a cluster. O_n is the number of vertices that belong to more than one cluster. O_m is the maximum number of clusters that a vertex can be contained in and μ is the LF mixing parameter. This parameter controls the fraction of edges connecting vertices that do not belong to the same cluster.

We utilized four real-world graphs in the tests carried out in this work, in which three of these graphs are widely known instances from the *Stanford Large Network Dataset Collection* (LESKOVEC; KREVL, 2014), namely *AstroPh*, *HepPh*

and *CondMat*. These instances are from collaboration networks, i.e., from graphs produced from networks of co-authored papers. In addition, we used an instance extracted from the Lattes Platform (<http://lattes.cnpq.br/>), a data base of Brazilian researchers' curricula, which can be found at <http://www.lac.inpe.br/~rafael.santos/Data/lattes_collab_graph.txt>. This graph is also a collaboration network, since it represents co-authored works between researchers. The number of vertices and edges of each of theses instances and presented in Table 3.4.

3.5.3 Metrics

Following Whang et al. (2016), we evaluated the maximum conductance of the clustering generated by each method by computing the *area under the curve* of the maximum *conductance-vs-coverage* plot. Given an overlapping clustering, this metric is computed by first sorting the clusters of the solution in increasing conductance value order. In the sequence, clusters are selected from this order until a minimum percentage of the vertices are covered (WHANG et al., 2016). A plot using x-axis as the vertices coverage and y-axis as the maximum conductance value of the selected clusters is generated. Then, the area under this plot is calculated. In this work, we utilized all vertices of the graph as the minimum covering. We refer to this score as *auc-cond*. In addition, we also computed the *average conductance* of each clustering, which is represented by *avg-cond*.

In order to compare the solution of each method to ground-truth overlapping clusterings, we utilized the *Generalized Normalized Mutual Information* (GNMI) measure (LANCICHINETTI; FORTUNATO, 2009a). In particular, we adopted the modified version of the GNMI proposed by Esquivel and Rosvall (2012), which an implementation can be found in <https://github.com/eXascaleInfolab/GenConvNMI>. In addition, we also utilized the *average F1* metric for evaluate clustering of instances with ground-truth solutions. Let $\mathcal{S}$ be a ground-truth overlapping clustering and $\mathcal{C}$ be an overlapping clustering generated by an algorithm. The average F1 score is defined by Equation (3.9) (GAO et al., 2019b; YANG; LESKOVEC, 2013).

$$F1_{avg}(\mathcal{S},\mathcal{C}) = \frac{1}{2}\cdot\left(\frac{1}{|\mathcal{C}|}\cdot\sum_{S_i\in\mathcal{S}} F1(S_i,C_{j^*}) + \frac{1}{|\mathcal{C}|}\sum_{C_j\in\mathcal{C}} F1(S_{i^*},C_j),\right) \tag{3.9}$$

where $i^* = \arg\max_i F1(S_i,C_j)$ and $j^* = \arg\max_j F1(S_i,C_j)$ (WHANG et al., 2016). The $F1$ measure given by Equation (3.10).

$$F1(S_i, C_j) = \frac{2 \cdot precision(S_i, C_j) \cdot recall(S_i, C_j)}{precision(S_i, C_j) + recall(S_i, C_j)}. \tag{3.10}$$

Equation (3.10) is the *harmonic mean* between *precision* and *recall* scores. These metrics are defined by Equations (3.11) and (3.12), respectively.

$$precision(S_i, C_j) = \frac{|S_i \cap C_j|}{|S_i|}. \tag{3.11}$$

$$recall(S_i, C_j) = \frac{|S_i \cap C_j|}{|C_j|}. \tag{3.12}$$

3.5.4 Tests in artificial graphs

In this subsection, we present results of tests carried out using the 24 LF benchmark graphs. As described in Subsection 3.5.2, these instances were divided into four sets LF_1, LF_2, LF_3 and LF_4 of six instances each. Since there is a ground-truth overlapping clustering for each of these graphs, we presented the GNMI and the F1_{avg} results of each method in addition to the conductance metrics. Table 3.3 shows the number of best metrics results in each set of LF benckmark graphs obtained by each method. For the LFM algorithm, it is presented the LFM version that achieved the best value of the GNMI (LFM_{best}) in each instance. The detailed results of each method in each instance are presented in Appendix B. In this table, the average size of the OCM's input clusters set (N_{avg}) utilized in each set of instances is also presented. In addition, for all tests realized with LF benchmark graphs utilized OCM's number of clusters $k = 600$ and NISE's number of seeds $ns = 600$.

As presented in Table 3.3, the HH-CR achieved the overall absolute number of best results considering all metrics. Considering all versions of our hybrid heuristic, they obtained 13 of 24 bests results of *auc-cond* metric; 23 of 24 bests results of *avg-cond* metric; 9 of 24 bests results of *GNMI* metric; and 15 of 24 bests results of $F1_{avg}$ metric. From these results, only on GNMI metric the largest number of best values were not obtained by one of our methods. The LFM algorithm, which results are the best among its 16 executions, obtained the best value of the GNMI in 17 of 24 instances. Considering the *auc-cond* measure, the greatest number of best values was achieved by LECM. However, when considering just the results of HH-CR among the hybrid heuristic versions, as one can see in Appendix B, it obtained best *auc-cond* results in 13 instances against 12 bests results of LECM. Furthermore, the HH-CR obtained the best average conductance in 22 of the 24 graphs. The greatest number

Table 3.3 - Summary results of each algorithm in each set of LF benchmark graphs. The number of best metrics' values is presented for each set.

Set	N_{avg}	Metric	Number of best metric values					
			LECM	**LFM$_{best}$**	**NISE**	**HH**	**HH-CR**	**HH-CR$_{LFM}$**
LF$_1$	2500	*auc-cond*	**5**	0	0	0	1	0
		avg-cond	0	0	0	0	**6**	0
		GNMI	0	**5**	0	0	0	1
		$F1_{avg}$	0	0	0	0	0	**6**
LF$_2$	4629	*auc-cond*	2	0	0	0	2	**3**
		avg-cond	0	0	0	0	**6**	0
		GNMI	0	**4**	0	0	1	2
		$F1_{avg}$	0	3	0	0	1	**4**
LF$_3$	5532	*auc-cond*	1	0	0	1	**3**	**3**
		avg-cond	1	0	0	0	**4**	1
		GNMI	0	**4**	0	0	0	2
		$F1_{avg}$	0	**5**	0	0	0	1
LF$_4$	7468	*auc-cond*	**4**	1	1	1	3	2
		avg-cond	0	0	0	0	**6**	0
		GNMI	0	**4**	0	0	1	2
		$F1_{avg}$	0	**3**	0	1	1	2
All	5032	*auc-cond*	**12**	1	1	2	9	8
		avg-cond	1	0	0	0	**22**	1
		GNMI	0	**17**	0	0	2	7
		$F1_{avg}$	0	11	0	1	2	**13**
Total	-	-	13	29	1	3	**35**	29

of best $F1_{avg}$ score was achieved by HH-CR$_{LFM}$.

From results shown in Table 3.3, it can be noted that is better to use the cluster refinement local search methods, since the HH-CR achieved better results than HH. In addition, considering conductance metrics, the HH-CR also obtained a greater number of bests results in comparison with HH-CR$_{LFM}$ results, even tough HH-CR$_{LFM}$ achieved better results in the *GNMI* and $F1_{avg}$ measures. This hybrid heuristic version achieved better results in supervised metrics because it uses only LFM algorithm to produce the input clusters set for the OCM and LFM obtained better results retrieving ground-truth solutions. Then, a clusters set with clusters more similar to the ground-truth is produced. However, as the focus of this work is the conductance minimization, the hybrid heuristic version with NISE and LFM is more suitable since it achieved the bests results related to conductance and reasonable *GNMI* and $F1_{avg}$ results.

3.5.5 Tests in real-world graphs

In this subsection, results of the tests performed in four real-world instances are presented. Since these instances do not have ground-truth overlapping clustering, only the results regarding the conductance measure is shown. We used the following values for the OCM's number of clusters and NISE's number of seeds: $k = 200$ and $ns = 200$ (*HepPh*), $k = 300$ and $ns = 250$ (*LattesCollab*), $k = 300$ and $ns = 250$

(*AstroPh*) and $k = 300$ and $ns = 250$ (*CondMat*). For instances of the Staford Collection (LESKOVEC; KREVL, 2014), the number of seeds of NISE that we used was the same utilized by its authors. All the OCM's number of clusters and the NISE's number of seeds in the *LattesCollab* instance were empirically defined.

Table 3.4 presents the values of *auc-cond* and *avg-cond* obtained by each method in each instance. The value of both metrics are better when is close to one. For the LFM algorithm, it is presented the LFM version that achieved the best value of the *auc-cond* (LFM_{best}) in each instance. In addition, the number of vertices (n) and edges (m) of each graph and the size of the OCM's clusters set (N) are also shown in Table 3.4.

Table 3.4 - Metrics results of each method in real-world graphs.

					Metrics values					
Instance	***n***	***m***	***N***	**Metric**	**LECM**	**LFM_{best}**	**NISE**	**HH**	**HH-CR**	**HH-CR_{LFM}**
HepPh	11,204	117,619	5392	*auc-cond*	0.180	0.063	0.105	0.065	0.060	**0.059**
				avg-cond	0.319	0.136	0.329	0.109	**0.107**	0.121
LattesCollab	13,121	23866	13069	*auc-cond*	0.088	0.069	0.066	0.054	**0.050**	0.056
				avg-cond	0.153	0.885	0.143	**0.072**	0.074	0.074
AstroPh	17,903	196,972	6505	*auc-cond*	0.349	0.102	0.168	0.100	**0.080**	0.081
				avg-cond	0.439	0.218	0.379	0.150	**0.143**	0.154
CondMat	21,363	91,286	12417	*auc-cond*	0.232	0.120	0.112	0.118	**0.093**	0.094
				avg-cond	0.307	0.177	0.296	0.119	**0.118**	0.120
Best values	-	-	-	-	0	0	0	1	**6**	1

It can be observed, in Table 3.4, that the greatest number of best results, in terms of low *auc-cond* and low *avg-cond*, were obtained by HH-CR. Indeed, all best results were achieved by our hybrid heuristic versions. Only the best value of *auc-cond*, in the instance HepPh, and the best value of *avg-cond*, in the instance LattesCollab, were not obtained by HH-CR. These results corroborate the results presented in Subsection 3.5.4.

The computational time, in seconds, of each method related to the HH-CR in the four real world instances are shown in Table 3.5. Since we executed 16 versions of the LFM in parallel, the LFM time shown in Table 3.5 is related to the LFM execution that taken longer to finish (LFM_{max}). This is because the execution of the NISE only starts after all LFM executions finish. In addition, in this table it is presented the computational cost of each HH-CR step and the total execution time, which is the sum of each HH-CR phase.

As one can see, the high computational time of the HH-CR, in three of the four instances, is due to the LFM execution, since the others HH-CR phases presented

Table 3.5 - Computational time, in seconds, of our hybrid heuristic. It is shown the execution time for each step of the HH-CR and its total computational time it is presented at column “total".

Computational time (s)						
			HH_CR			
Instance	**LFM_max**	**NISE**	**LFM_max + NISE**	**OCM**	**CR**	**Total**
HepPh	283.57	7.12	290.69	5.44	0.91	297.04
LattesCollab	12.41	1.53	13.94	40.17	0.09	54.20
AstroPh	931.97	11.56	943.53	9.57	3.16	956.26
CondMat	494.63	8.55	503.18	35.5	1.06	539.74

low execution time. In addition, it is worthily to highlight the low computational cost for solve the OCM, which was already pointed in Chagas et al. (2019).

3.6 Conclusions and future directions

In this work, we proposed the HH-CR, a hybrid heuristic for the overlapping community detection by the conductance minimization. This method is composed by coupling two well-known community detection algorithms namely, NISE and LFM to a mixed-integer linear program (OCM). Furthermore, local search methods for conductance minimization were utilized to improve the solution generated by our hybrid heuristic. We also evaluate two other versions of our proposed method: a version where the local search methods were not applied, and another version where only the LFM algorithm was utilized for generating the input clusters' set.

With the experimental tests carried out in this work in synthetic and real-world graphs, we showed that the complete version of the hybrid heuristic obtained the best results among the other versions. In addition, these tests show that our method can produce overlapping clustering with better overall conductance than NISE and LECM, two state-of-the-art overlapping community detection algorithms.

For future work, we could implement a faster algorithm instead of using the LFM, which was the main reason for the high computational cost of the HH-CR. However, it is necessary to use a algorithm that has some key features of the LFM that were essential for the results of our method: starting a cluster from random vertex, ensure that each vertex is inserted into a cluster and controlling the size of the clusters by an input parameter. In addition, we will propose a modification of the OCM to handle larger instances.

4 A BRANCH AND PRICE METHOD FOR THE p-MEDIAN PROBLEM WITH OVERLAP CONTROL

This chapter is organized as follows. An introduction is presented in Section 4.1. Some relevant related works are presented in Section 4.2. In Section 4.3, the mathematical formulations of the p-median problem with overlap control are shown. The algorithms we propose to solve one of these formulations are detailed in Section 4.4. Section 4.5 presents the results of our computational experiments. Our concluding remarks and considerations on future work are presented in Section 4.6.

4.1 Introduction

Facility location problems aim to locate a number of facilities from m potential locations known *a priori* in order to fulfill the demands of n clients at a minimum cost. In the graph theory context, both potential locations and clients are vertices of an input graph. In this sense, clients vertices assigned to a facility vertex consist of a subset of vertices allocated to a facility. In this work, we refer to such subsets as *clusters* centered around the facility. Then, location problems can be described as the problem of partitioning the vertices of a graph into a given number of clusters, where the partition cost is minimized. In addition, if vertices can be allocated to more than one facility, we can also consider these problems as covering problems with a cardinality constraint.

Among these problems, the p-median problem (PMP), first introduced by Hakimi (1964), is the problem of selecting p distinct vertices, also known as medians, minimizing the sum of the distances from each vertex to its closest median. This minimization version of the PMP is a well known NP-hard problem (KARIV; HAKIMI, 1979).

While in the p-median problem each node is assigned to exactly one facility on the basis of a weight function between the demand vertex and the facility vertex, in some applications it is desirable that vertices be assigned to more than one facility at a time, effectively covering the vertex more than once. This overlap is useful, for example, to backup coverage, where the service provided by facilities are critical and may become unavailable due to unpredictable reasons such as weather and electricity problems (PANTELI et al., 2019). In addition, the number of demand vertices can also increase in certain regions and instead of opening new facility locations, one can use a close by facility, not necessarily the closest one, to satisfy the demand of the new clients (ARAÚJO et al., 2020). For instance, in the COVID-19 pandemic

context, where hospitals located in highly dense urban areas can handle the demand of a regular day, but are now facing overwhelming demand (MILLER et al., 2020). An alternative would be assign the excess demand to a temporary healthcare structure or even to a backup hospital (ARAÚJO et al., 2020). On the other hand, a hospital located in a less populated region might not be dealing with a burden on its system (MILLER et al., 2020). Therefore, in order the reduce operational costs, not every area should be served by an additional temporary facility or an extra hospital.

Other examples arise in computer networks, where some critical systems must have higher redundancy that others, or more generally to any context in which some entities being served are more important than others (WANG et al., 2009). From the provider's perspective, such as in the hospital example, facilities more prone to failure may be elected to require extra coverage for its users. It then becomes useful to assign a client to more than one facility at a time and also be able to adapt covering strategies.

To this end, we introduce the *p-median problem with overlap control* (PMPOC) which is similar to the PMP but imposes that (some) vertices may be assigned to more than one median. Furthermore, the number of vertices shared between medians can be controlled by simple parameters determining for example how many vertices can be shared by a facility and the degree of multiple coverage of a client. Different overlapping statistics exist, and a widely used one is the *Jaccard similarity coefficient* (JACCARD, 1912), which measures the similarity of two sets: if they are identical, the coefficient equals one and if they do not overlap at all, the coefficient equals zero.

To solve the PMPOC, we first propose a non-linear mixed-integer programming model for it. The non-linearity stems from the Jaccard coefficient, that requires computing intersections and unions of the clusters, modeled explicitly. An implicit cluster formulation requires enumerating the clusters and their similarities *a priori*. To solve such a model, we derive a column generation (CG) algorithm that iteratively generates new clusters and computes the similarity coefficients with all other known clusters. The CG is applied at each node of a search tree, which is explored in parallel by a branch-and-price (B&P) algorithm.

4.2 Related work

Although the literature of the PMP is vast (BARBAROS et al., 1983; DASKIN; MAASS, 2015; MARÍN; PELEGRÍN, 2019; MLADENOVIĆ et al., 2007; REESE,

2006), to the best of our knowledge, there are few works concerning the PMP where vertices can be assigned to more than one median. One of these studies is the work of Wang et al. (2009), who introduced the backup 2-center problem and the backup 2-median problem. These authors were motivated by a problem where they had to locate two servers, which may fail, in a tree network. Then, every vertex is assigned to two servers simultaneously. When one of them fails, the other server fulfills all vertices' demands. Wang et al. (2009) proposed a linear algorithm for the backup 2-center problem and a log-linear algorithm for the backup 2-median problem.

Another example is the study of Karatas et al. (2016), where the authors compared the coverage location problem (CLP) and the PMP with the requirement that each vertex must be served by q medians. Note that, when $q = 1$, the PMP is defined. In the CLP, one seeks to locate facilities covering all demand vertices minimizing some criteria, such as the number of facilities. These authors evaluated the mathematical formulations of both problems considering five different decision criteria. Panteli et al. (2019) also studied the PMP with multiple coverage, which they called as multiple assignment p-median problem (MPMP). As Karatas et al. (2016), they also used a parameter to define the number of medians that each vertex must be assigned to. Panteli et al. (2019) proposed a biclustering heuristic to solve the MPMP and compared it with the solutions from a commercial solver. Although the heuristic did not obtained optimal solutions in the tested instances, it was able to generated MPMP solutions at a low computational cost.

A recent constraint that allows managing the overlap between clusters was presented by Chagas et al. (2019), where the amount of overlap is controlled by a user-defined parameter. This constraint was proposed in the context of the overlapping cluster editing problem, where the objective is to minimize the number of edges' addition and deletion in order to partition the vertices of the input graph into maximal cliques that may overlap. The *overlap control constraint* was also successfully applied to the CLP by Araújo et al. (2020). The authors used this constraint to handle the overlapping between the facilities' coverage. These authors called this problem as the coverage location problem with overlap control.

CG algorithms were already successfully applied to p-median problems. For example, Lorena and Senne (2004) proposed a CG method for the capacitated p-median problem (CPMP) and applied it to some real-world instances. The CPMP, is a PMP where each vertex has a demand and each median has a capacity, which must not be exceeded. The authors used a pricing sub-problem with Lagrangean relaxation that

increased the CG convergence. Another example is the work of Garía et al. (2011), who introduced a column-and-row generation algorithm using a branch-and-bound approach for the PMP.

The first two B&P algorithms proposed in the p-median context, that we are aware of, are the works of Ceselli and Righini (2005) and Senne et al. (2005). In the first paper, the authors developed a B&P method for the CPMP. Ceselli and Righini (2005) utilized two different branching rules and tested the algorithm against other methods from the literature and a general purpose solver. The proposed algorithm obtained the best results on medium and large size instances. Senne et al. (2005) introduced a B&P algorithm for the PMP using Lagrangean relaxation in the pricing sub-problem. They utilized the *partitioning with identical subsets* method (RYAN; FOSTER, 1981) as the branching rule. This method is used in this work and is detailed in Section 4.4.2.1. Senne et al. (2005) showed that the proposed B&P can solve small and medium sizes instances of the PMP at a small computational cost.

Another relevant algorithm is the branch-and-cut-and-price method for the PMP developed by Avella et al. (2007). The authors also obtained good quality PMP solutions at a low execution time. In addition, an example of the B&P applied to other PMP variant is the study of Güden and Haldun (2019). These authors proposed a B&P algorithm for the dynamic PMP, where the facilities locations can change over time.

4.3 Formal problem description and mathematical formulation

Let $G = (V, E)$ be an *undirected*, *weighted* and *connected* graph, where V is a set of n vertices, E is the set of m edges, and to each edge $(i, j) \in E$ is associated a weight $d_{ij} \in \mathbb{R}$. In the facility location context, d_{ij} is the distance between vertices i and j, which is often the euclidean distance. Then, $D = (d_{ij})$ is an $n \times n$ *distance matrix* of non-negative real values. In addition, a subset $C \subseteq V$ is called a cluster. The PMP requires that exactly p facilities be selected among the vertices of V, and that all other vertices are assigned to the closest facility. It is implicitly assumed that a facility vertex is assigned to itself.

The PMP can be formulated as the integer linear program (PMP-ILP) (REVELLE; SWAIN, 1970), presented by (4.1a)–(4.1e).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{4.1a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \ i = 1, \ldots, n, \tag{4.1b}$$

$$\sum_{j=1}^{n} x_{jj} = p, \tag{4.1c}$$

$$x_{ij} - x_{ii} \leq 0, \ i, j = 1, \ldots, n, \tag{4.1d}$$

$$x_{ij} \in \{0, 1\}, \ i, j = 1, \ldots, n, \tag{4.1e}$$

where the decision variables x_{ij}, known as *allocation variables*, are defined by

$$x_{ij} = \begin{cases} 1, & \text{if vertex } i \in V \text{ is assigned to facility vertex } j \in V, \\ 0, & \text{otherwise.} \end{cases}$$

In the PMP-ILP, the objective function (4.1a) minimizes the assignment distances. Equations (4.1b) are the single assignment constraints and define that each vertex i is allocated to only one facility. Note that even if an inequality (greater than or equal to) is used, the objective function cannot improve by allowing double assignments. Constraint (4.1c) ensures that p vertices are selected as facilities. Constraints (4.1d) impose that a vertex i can only be assigned to an open facility j, i.e., only if $x_{jj} = 1$. Decision variables x_{ij} are defined as binaries by constraints (4.1e).

The PMP can also be considered as the problem of partitioning set V into p disjoint clusters of minimum cost, where the cost c_i of a cluster C_i, $i = 1, \ldots, p$, is computed by Equation (4.2). In addition, the median is the vertex $j \in C_i$ which achieved the minimum cost c_i.

$$c_i = \min_{j \in C_i} \left(\sum_{k \in C_i} d_{kj} \right). \tag{4.2}$$

Given N clusters, the amount of overlap between each cluster and the remaining $N-1$ clusters can be determined by an overlap control constraint (OCC) (CHAGAS et al., 2019) using a similarity measure and a user-defined overlap control parameter $z \in [0, 1]$. The overlap control aims to select clusters whose similarity measure

between them is as close as possible to z. Then, the overlap decreases when z is closer to zero and it increases when z is closer to one. In order to achieve this, Chagas et al. (2019) added N variables $u_i \in \mathbb{R}$, $-N-1 \leq u_i \leq N+1$, to the objective function whose values are defined by inequality 4.3.

$$\sum_{\substack{j=1 \\ j \neq i}}^{N} \Big| J(C_i, C_j) - z \Big| \cdot (y_i + y_j - 1) \leq u_i, \;\; i = 1, \ldots, n, \tag{4.3}$$

where variables y_i, $i = 1, \ldots, N$, are binary variables with values defined as

$$y_i = \begin{cases} 1, & \text{if cluster } i \text{ is selected to compose the final clustering,} \\ 0, & \text{otherwise,} \end{cases} \tag{4.4}$$

and $J(C_i, C_j)$ is the *Jaccard coefficient* (JACCARD, 1912) between clusters C_i and C_j. This similarity measure is defined in Equation (4.5).

$$J(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}. \tag{4.5}$$

As the OCEP is a minimization problem, variables u_i penalize the solution with the value computed by its related constraint, which is the absolute difference between the similarity measure and z. However, in the context of the PMP it is necessary to ensure uniformity in the service level provided to each client, even if they can still have a different number of assignments. Then, instead of using N OCCs, one for each cluster, we use only one OCC which is the same for all clusters. In this way, the amount of overlap between every pair of clusters is controlled uniformly. In addition, since two variables, y_i and u_i, exist for every cluster i, two new columns are added to the problem whenever a new cluster is generated. In addition, the number of rows of the problem is also increased through the column generation process, as an overlapping control constraint is defined for every cluster. In this way, the problem becomes larger and, therefore, harder to solve. In order to overcome this issue, we modified the original OCC and defined just one overlapping control constraint instead of N. The modified OCC is defined by Equation (4.6).

$$\sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \Big| J(C_i, C_j) - z \Big| \cdot (y_i + y_j - 1) \leq u, \tag{4.6}$$

If both clusters C_i and C_j are selected ($y_i = 1$ and $y_j = 1$), then $J(C_i, C_j)$ should be

as close as possible to z in order to minimize the penalty in the objective function. In case only one of them is selected, the penalty is zero. If neither C_i nor C_j are selected ($y_i = 0$ and $y_j = 0$), the value of $J(C_i, C_j)$ should be as different as possible from z in order to yield the minimum penalty in the objective function.

In order to adapt the OCC to the PMP context to model the PMPOC, it is also necessary to define the Jaccard coefficient in terms of the decision variables x_{ij}. For this, consider the cardinality of the intersection and the union sets given by Equations (4.7) and (4.8), respectively. The Jaccard coefficient is thus defined as the ratio between the values of these both equations.

$$|\cap_{ij}| = \sum_{k=1}^{n} x_{ki} x_{kj} \tag{4.7}$$

$$|\cup_{ij}| = \left(\sum_{k=1}^{n} x_{ki} + x_{kj} \right) - |\cap_{ij}|. \tag{4.8}$$

Then, the OCC can be added to the model presented in Equations (4.1a)–(4.1e), yielding the PMPOC. The PMPOC can be cast as the following *mixed-integer non-linear program.*

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} + u \tag{4.9a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} \geq 1, \ i = 1, \ldots, n, \tag{4.9b}$$

$$\sum_{i=1}^{n} x_{ii} = p, \tag{4.9c}$$

$$x_{ij} - x_{ii} \leq 0, \ i, j = 1, \ldots, n, \tag{4.9d}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} \left| \frac{|\cap_{ij}|}{|\cup_{ij}|} - z \right| \cdot (x_{ii} + x_{jj} - 1) \leq u, \tag{4.9e}$$

$$x_{ij} \in \{0, 1\}, u \in \mathbb{R}, \ i, j = 1, \ldots, n. \tag{4.9f}$$

This formulation differs from that of the PMP-ILP as variable u was added to the objective function (4.1a). In addition, constraints (4.1b) were updated to (4.9b) to

allow multiple assignments (cover constraint).

Note, however, that constraints (4.9e) are non-linear, therefore this formulation cannot be solved by the Dantzig's simplex algorithm. An alternative is to first to formulate the PMP-ILP as a *set covering problem with cardinality constraint* (GARFINKEL et al., 1974; SWAIN, 1974) and then add the OCC as follows. Let $\mathcal{C} = \{C_1, \ldots, C_{2^n}\}$ be the *power set* of V, i.e., the set of all possible subsets of V where $|\mathcal{C}| = 2^n$. Furthermore, consider the decision variables y_i, $i = 1, \ldots, 2^n$, as defined in Equation (4.4). The new PMPOC model is defined by (4.10a)–(4.10f) (PMPOC-MP). This *mixed-integer linear program* (MILP) can also be obtained by applying the Dantzig-Wolfe decomposition (DANTZIG; P., 1960) to the PMPOC and, in this context, it is known as *master problem* (MP) (LORENA; SENNE, 2004).

$$\min \sum_{i=1}^{2^n} c_i y_i + u \tag{4.10a}$$

subject to

$$\sum_{j=1}^{2^n} a_{ij} y_j \geq 1, \ i = 1, \ldots, n, \tag{4.10b}$$

$$\sum_{i=1}^{2^n} y_i = p, \tag{4.10c}$$

$$\sum_{i=1}^{2^n} \sum_{\substack{j=1 \\ j \neq i}}^{2^n} \Big| J(C_i, C_j) - z \Big| \cdot (y_i + y_j - 1) \leq u, \tag{4.10d}$$

$$\sum_{j=1}^{N} b_{ij} y_j \leq 1, \ i = 1, \ldots, n, \tag{4.10e}$$

$$y_i \in \{0, 1\}, u \in \mathbb{R}, \ i = 1, \ldots, 2^n. \tag{4.10f}$$

The objective function (4.10a) minimizes the clusters costs c_i, which are computed by Equation (4.2), plus the u penalty value. It is guaranteed by cover constraints (4.10b) that each vertex is contained in at least one cluster, where matrix $A = (a_{ij})$ is a $n \times 2^n$ matrix such that

$$a_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ belongs to cluster } C_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

Constraint (4.10c) is similar to constraint (4.1c). Inequalities (4.10d) define the

overlap control constraint. In addition, it is necessary to ensure that all p medians are distinct. Then, constraints (4.10e) impose that if cluster j is selected, its median i cannot be the median of any other selected cluster, where matrix $B = (b_{ij})$ is given by

$$b_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ is the median of cluster } C_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.12}$$

The PMPOC-MP, however, is not practical since it requires all possible clusters to be known *a priori*, which is an exponential number. In order to efficiently solve this formulation, we present in the next section a solution algorithm based on CG and on B&P.

Although the objective function (4.10a) is composed by the clusters costs and the value of variable u, only the c_i coefficients have practical meaning. Then, in this work, whenever we describe the cost of a solution, we are considering only the sum of the clusters costs, i.e, the PMP cost.

4.4 Solution algorithms

As we indicated, formulation (4.10a)–(4.10f) is only tractable if all clusters in $\mathcal{C}$ are known. For any instance of practical interest, it is prohibitive to even try to enumerate them. In order to solve this formulation and obtain optimal solutions for the problem, our algorithm works with only a subset $C \subset \mathcal{C}$, where $|C| = N$ and $N \ll 2^n$. This formulation is known as the *restricted master problem* (RMP) and the remaining clusters can be generated through a CG process.

In brief, the CG algorithm solves a relaxed version of the RMP with an initial set of variables (columns). Then, by solving n sub-problems, a pricing algorithm adds up to n new columns to the RMP, restarting the process. Each column has a reduced cost and only columns with negative reduced costs are inserted in the RMP. This procedure stops when the pricing algorithm cannot find new columns for the RMP. The CG algorithm is detailed in Section 4.4.1.

If the solution found by the CG is still fractional, then its RMP becomes the root node of a search tree, which is explored by a B&P algorithm in a parallel breadth-first search fashion. Two child nodes are then created from the root node following a *branching rule*, which imposes different constraints to the sub-problem of each child's RMP. Then, this process, known as branching, is repeated whenever a fractional solution is found. When an integer solution is obtained at a node, the branching ends

and the node becomes a leaf. The algorithm stops when all nodes were explored, i.e., a branch cannot be generated in any node or when an stopping criterion, e.g., the maximum execution time, is met. The B&P algorithm is detailed in Sections 4.4.2.

4.4.1 Column generation

The CG algorithm works by iteratively solving the RMP and identifying variables that can potentially improve its objective function value. These steps are described next.

4.4.1.1 Initial pool

The RMP is initially defined with only a subset of all possible clusters. In our algorithm, we considered an initial pool of clusters generated by Algorithm 8 (SENNE et al., 2005). At each iteration, p clusters of random medians are generated. Then, each graph vertex is assigned to the closest one. These steps are repeated until the size of the pool is greater than a threshold.

Algoritmo 8: Generation of an initial pool of columns.

input : vertex set V; distance matrix $D = (d_{ij})$; number of medians p; minimum number of clusters in the pool $minSize$.
output: pool of initial clusters $\mathcal{C}$; initial incumbent solution.

begin
  $\mathcal{C} \leftarrow \emptyset$;
  while $|\mathcal{C}| < minSize$ **do**
    $U \leftarrow \{u_1, \ldots, u_p\}$; `// generate a set of` p `random vertices`
    $C \leftarrow \{C_i \mid C_i = \{u_i\} \wedge u_i \in U, i = 1, \ldots, p\}$;
    foreach $v \in V \setminus U$ **do**
      $i \leftarrow \arg\min_{u \in U}(d_{uv})$; `// find the index` i `of the closest median`
      $C_i \leftarrow C_i \cup v$; `// insert` v `in the` i`th cluster`
    end foreach
    foreach $C_i \in C$ **do**
      $cost(C_i)$; `// compute Equation (4.2)`
    end foreach
    if $v(C) \leq v(incumbent)$ **then** $incumbent \leftarrow C$;
    $\mathcal{C} \leftarrow \mathcal{C} \cup C$;
  end while
  return $[\mathcal{C}, incumbent]$;
end

Note that a feasible solution is generated at each iteration of lines 3-15. Then, we keep the best solution generated through these lines as the initial incumbent. In addition, to ensure the feasibility of the PMPOC-RMP, we also added p dummy clusters to the initial set. All vertices of V are inserted in each of these clusters.

4.4.1.2 Restricted Master Problem

The PMPOC-RMP is defined as follows.

$$\min(1-\alpha)\sum_{i=1}^{N} c_i y_i + \alpha u \tag{4.13a}$$

$$\text{subject to } (4.10b), (4.10c), (4.10d), (4.10e),$$

$$y_i \in [0,1], u \in \mathbb{R},\ i = 1,\ldots,N. \tag{4.13b}$$

Note that constraints (4.10b), (4.10c), (4.10d), (4.10e) are defined only for the N clusters of the initial set instead of all 2^n possible clusters. In addition, the integrality requirement of variables y of the PMPOC-MP is relaxed in order to make the PMPOC-RMP easier to solve. Furthermore, we added a new parameter $\alpha \in [0,1]$ to the objective function to balance the relevance between the y variables and the u variable. Note that, when $\alpha = 0$, the PMP is defined as the OCC would take no effect. We also normalized the distances matrix $D = [d_{ij}]$ to the interval $[0,1]$ in order to compute the coefficients of the y variables in the objective function, since the u variable is composed by a sum of values in this same interval. The normalization was carried out using the *min-max scaling* method, which computed by Equation (4.14).

$$d'_{ij} = \frac{d_{ij} - \min_{ij}(d)}{\max_{ij}(d) - \min_{ij}(d)}. \tag{4.14}$$

4.4.1.3 Pricing sub-problem

Given a solution for the PMPOC-RMP, we obtain the dual values π_i, ρ, σ, and μ_i associated with constraints (4.10b), (4.10c), (4.10d) and (4.10e). In the sequence, new columns are added to the problem by solving the *pricing sub-problem*. Our pricing sub-problems, one for each $j = 1,\ldots,N$, are presented below (LORENA; SENNE, 2004).

$$v(pricing)_j = \min \sum_{i=1}^{n} (d'_{ij} - \pi_i) x_{ij}, \tag{4.15}$$

where distances d'_{ij} are given by Equation (4.14).

This equation is easily solved by considering $x_{ij} = 0$ if $(d'_{ij} - \pi_i) > 0$ and $x_{ij} = 1$ otherwise. Then all columns j where $v(pricing)_j$ values satisfy the inequality

$$v(pricing)_j - \sigma - \rho - \sum_{i}^{n} \mu_i < 0, \tag{4.16}$$

i.e., columns with negative reduced costs, can be added to the PMPOC-RMP.

The pricing sub-problem then optimally identifies all possible new columns j with a negative reduced price, and returns them to the PMPOC-RMP for another iteration. At each iteration, up to n new columns can be identified. Note also that the sub-problems are independent, hence all n sub-problems can be solved in parallel.

Note that a cluster C_j is not a column itself, but a column on the matrix A of coefficients related to the cover constraints (4.10b) and on the matrix B of coefficients corresponding to constraints (4.10e) of the y_j variable. A PMPOC-RMP column $\mathcal{K}$ is composed by these coefficients and the coefficients of constraints (4.10c) and (4.10d) as shown below.

$$\mathcal{K} = \begin{bmatrix} A_j \\ 1 \\ \mathcal{J} \\ B_j \end{bmatrix}, \tag{4.17}$$

where $\mathcal{J}$ is the coefficient of variable y_j in the OCC.

When a new column is added to the PMPOC-RMP, it is necessary to update the OCC coefficients of all y variables. In addition, the value of the right-hand side of the OCC is also updated.

4.4.1.4 Column substitution

In order to limit the size of the PMPOC-RMP column pool and, therefore, prevent it to become too hard to solve, we employed a column substitution procedure. Whenever the number of columns of the PMPOC-RMP exceeds a threshold, new

columns are not appended to it anymore, but they replace the old columns with the highest reduced costs. To achieved this, we sort the PMPOC-RMP column pool in decreasing reduced costs order and the column substitution takes place following this order.

Note that removing the columns from the PMPOC-RMP would be a costly operation, then the coefficients (4.17) of the old column are updated to the coefficients of the new column. In addition, we always maintain the p dummy columns in the column pool to ensure the model feasibility.

4.4.1.5 CG pseudocode

A CG pseudocode is depicted by Algorithm 9. The algorithm takes as input an initial column pool $\mathcal{C}$. At each iteration, at most n columns are generated and added to the RMP if their reduced costs are negative. When the maximum number of columns in the RMP is reached, the column substitution procedure is utilized.

Algoritmo 9: Column generation algorithm.

input : Initial column pool $\mathcal{C}$; Graph $G = (V, E)$; distance matrix $D = (d_{ij})$; number of medians p; overlap control parameter z; minimum number of clusters in the pool $minSize$; Maximum number of columns in the RMP $maxRmpSize$.

```
1  begin
2      pmpoc_rmp.init(G, p, z, C);
3      do
4          solver.solve(pmpoc_rmp);
5          [π, ρ, σ, μ] ← solver.getDuals(pmpoc_rmp);
6          columnFound ← false;
7          for j = 1, ..., n do
8              C_j ← pricing.solve(D, j, π);
9              if pricing.getObjVal() − ρ − σ − (Σ_{i=1}^{n} μ_i) < 0 then
10                 columnFound ← true;
11                 if pmpoc_rmp.nbColumns() > maxRmpSize then
                       // replace the column with the highest reduced cost
12                     pmpoc_rmp.replaceHighestRC(C_j);
13                 else
14                     pmpoc_rmp.append(C_j);
15         end for
16     while columnFound;
17 end
```

4.4.2 Branch-and-price

The CG algorithm provides an optimal solution for the linear relaxation of the PMPOC-RMP. This is not necessarily a solution for the PMPOC as some variables might take fractional values. If all variables are integer, the solution is optimal for the PMPOC. However, if some variables are fractional, branching needs to take place. Then, each branch of the tree is solved using the algorithm just described in Section 4.4.1.

This section is organized as follows. In Section 4.4.2.1, we present the branching rule utilized. A pseudocode of our B&P, compiling all methods described in this work, is presented in Section 4.4.2.3.

4.4.2.1 Branching rule

Our branching rule is based on the partitioning with identical subsets (PIS) rule described by Ryan and Foster (1981). This method was already used to solve a PMP (SENNE et al., 2005) and works as follows. Given two vertices q and r and a cluster C_i, three cases may arise: either both q and r belong to C_i, only one of them belong to C_i, or none of them belong to this cluster. The PIS branching rule creates two child nodes with the following constraints: on the left child, either both q and r appear in cluster C_i or none of them are present at this cluster. On the right child node, at most one vertex q or r must belong to C_i but not both, i.e., they do not appear together. This branching rule creates two child nodes and contemplates all three scenarios described above. Then, new columns are generated, at each branch side, respecting the related PIS constraint. Moreover, all columns from the parent node which violate the child node's PIS rule are removed from the child node's RMP.

In order to determine the vertex q, we proceed as in Senne et al. (2005). Let $\bar{y_i}$ be the value of the variable y_i in the current PMPOC-RMP solution and $C = \{C_i \mid 0 < \bar{y_i} < 1, i = 1, \ldots, N\}$ be the set of clusters related to the decision variables y_i with fractional values. In addition, consider the set of clusters $Q(i) = \{C_j \mid i \in C_j, j = 1, \ldots, |C|\}$, i.e, the set of clusters that vertex i belongs to. Then, q is selected as the vertex contained in the largest number of clusters as defined by Equation (4.18).

$$q = \underset{1 \leq i \leq n}{\arg\max} \left(|Q(i)|\right). \tag{4.18}$$

The choice of vertex r is performed based on the set $R(i) = \{C_j \mid C_j \in Q(q) \wedge i \in$

$C_j, j = 1, \ldots, |C|\}$, i.e., the subset of clusters of $Q(q)$ where vertex i is contained. In addition, consider that $R(i) \neq \emptyset$. Then, r is defined as the vertex belonging to the minimum number of clusters which q is also contained, that is,

$$r = \arg\min_{1 \leq i \leq n} (|R(i)|). \tag{4.19}$$

The new columns are generated using the pricing sub-problem presented in Section 4.4.1.3 with the PIS constraints. Analogously to the pricing sub-problem (4.15), all columns satisfying inequality (4.16) are inserted in the RMP related to the current node. On a left branch node, vertices q and r must appear together in a cluster C_j or not appear at all in this cluster. In order to achieve this, the sub-problem shown in Equations (4.20a) and (4.20b) is solved.

$$v(pricing)_j = \min \sum_{i=1}^{n} (d'_{ij} - \pi_i) x_{ij} \tag{4.20a}$$

subject to

$$x_{qj} = x_{rj}. \tag{4.20b}$$

On a right branch node, vertices q and r must not belong to a same cluster C_j. These columns are generated by solving the sub-problem presented in Equations (4.21a) and (4.21b).

$$v(pricing)_j = \min \sum_{i=1}^{n} (d'_{ij} - \pi_i) x_{ij} \tag{4.21a}$$

subject to

$$x_{qj} + x_{rj} \leq 1. \tag{4.21b}$$

Note that constraints (4.20) and (4.21) are accumulated from the root node to the current child node through the related path in the B&P search tree. Thus, the pricing sub-problem of a node is composed by its own constraint (4.20), if it is a left node, or (4.21), if it is a right node, and its ancestor's constraints. These sub-problems cannot be easily solved as the pricing sub-problem of the root node, and are solved using a commercial solver.

4.4.2.2 Pruning

The tree exploration is improved by using a pruning procedure. Whenever the cost of a solution, either fractional or integer, found by the CG algorithm in a node is greater than the best integer solution found so far, the node is pruned.

4.4.2.3 B&P pseudocode

A simplified pseudocode of the B&P is shown in Algorithm (10). As mentioned, we utilized a parallel breadth-first search algorithm to explore the B&P tree. Then, a queue is used to store the nodes created whenever branching takes place. A node is dequeued and assigned to an idle thread, which solves the related PMPOC-RMP by the CG algorithm and creates a new branch if the solution is fractional and its cost is less then the incumbent cost. If the cost of a fractional solution is not less then the cost of the incumbent solution, then the tree is pruned and a branch is not performed. Moreover, in the event that all threads are busy solving models, the algorithm waits for one of them to finish to dequeue the next queue node.

In order to save memory we utilized two kinds of column pools in the B&P algorithm: a global and a local one. The local pool keeps only columns generated by the CG on the related node. Then, all CG columns insertions and substitutions are performed on this pool. When the CG algorithm finishes, its local pool is copied to the global pool, which is used as the initial pool to build the PMPOC-RMP of the subtree nodes respecting the corresponding PIS constraints. Since nodes of the subtree use different columns from the global pool, the column substitution is not applied to it. The particular case is the root node, where its local pool is the global pool. Then, the column substitution is also carried out on the global pool.

4.5 Computational experiments and analysis

We now present the computational tests designed to evaluate our proposed B&P algorithm. All implementations were written in *C++* language and compiled with *g++* compiler version 10.1 using -O3 flag. For the resolution of models we used the Gurobi™ 9.0.2 (OPTIMIZATION, 2020). All the computational tests were executed on a computer with Intel® Core™i9-9900K CPU 3.60GHz × 16 processor with 16MiB *cache* memory and 128GiB of RAM. The operating system installed on this machine is Ubuntu 18.04.4 64 bits. The tests were carried out using the well-known PMP instances from the OR-library (BEASLEY, 1985; BEASLEY, 1990). This set consists of 40 instances with $n \in [100, 900]$ and $p \in [5, 200]$. The bigger the $\frac{n}{p}$ ratio the harder

Algoritmo 10: Branch-and-price algorithm.

input : Graph $G = (V, E)$; distance matrix $D = (d_{ij})$ number of medians p; overlap control parameter z; minimum number of clusters in the pool $minSize$.

```
1  begin
2      C ← generateInitialPool(); // Algorithm (8)
3      pmpoc_rmp.init(G, p, z, C); // build the initial RMP
4      root ← createNode(pmpoc_rmp);
5      unexploredNodes.enqueue(root);
6      while !unexploredNodes.empty() do
7          if threads.idle() then
               /* give the next node in the queue to a worker thread   */
8              node ← unexploredNodes.dequeue();
9              solution ← cg.solve(node); // solve using CG algorithm (9)
10             if isFractional(solution) then
11                 [leftChild, rightChild] ← node.branch();
12                 unexploredNodes.enqueue(leftChild);
13                 unexploredNodes.enqueue(rightChild);
14             end if
15         else
16             wait(); // wait for an idle thread
17         end if
18     end while
19 end
```

is to find a PMP solution (CHRISTOFIDES; BEASLEY, 1982; SENNE et al., 2005).

In all experiments realized in this work, the following B&P parameters values were used: 1000 initial columns; p dummy columns; 4000 as the maximum number of columns at each node; 2000 as the maximum number of CG iterations and $3h$ as the execution time limit. Furthermore, up to 16 threads were utilized through the B&P execution since the CPU of the machine where the tests were carried out has 16 threads.

We evaluated our method using three values of the overlap control parameter $z = \{0, 0.5, 1\}$, i.e., with minimum amount of overlap, with an intermediary value and with maximum amount of overlap, respectively. For each of these values, we tested with $\alpha = \{0.05, 0.5, 0.95\}$, that is, with more relevance to the clusters costs, with equal relevance between the clusters costs and the OCC value and with more relevance to the OCC value, respectively. Moreover, in order to evaluate the algorithm in the PMP context, we also tested with $\alpha = 0$ when $z = 0$. Thus, the B&P

was tested in ten scenarios for each of the 40 instances.

The results of tests performed with the proposed B&P algorithm are presented in tables 4.1 and 4.2. The acronyms presented is these tables are:

- *opt*: the optimal PMP cost, obtained by the commercial solver;
- ts: the tree size (considering only explored nodes);
- th: the tree height;
- ncg: the number of columns generated;
- np: the number of tree prune;
- *cost*: the PMP cost of the solution;
- $|sc|$: the normalized sum of the number of coverage of each vertex, i.e., the sum of the number of clusters that each vertex belongs to divided by n;
- mc(%): percentage of vertices contained in more than one cluster.

The remaining of this section is organized as follows. Subsection 4.5.1 presents the results of the tests of our algorithm performed in the PMP context, i.e., using $z = 0$ and $\alpha = 0$. The results of the experiments with different values of the overlap control parameter are shown in Subsection 4.5.2.

4.5.1 PMP tests results

In order to evaluate the effectiveness of our method and ensure that it is working properly, we applied it to solve the OR-library instances in the PMP context, that is, with parameters $z = 0$ and $\alpha = 0$. The results of these experiments are presented in Table 4.1. In addition to the table information described in the previous section, in Table 4.1 the gap (%) between the optimal PMP solution and the solution found by the B&P algorithm is shown. The gap is computed by Equation (4.22).

$$gap = \frac{cost - opt}{opt} \cdot 100. \tag{4.22}$$

From Table 4.1 it can be noticed that the B&P algorithm obtained the optimal PMP solution in 12 of the 40 instances (bolded costs). Considering the remaining

Table 4.1 - Results of the experiments in the OR-library instances (BEASLEY, 1985; BEASLEY, 1990) using $z = 0$ and $\alpha = 0$.

				B&P						
Instance	***n***	***p***	***opt***	**ts**	**th**	**np**	**ncg**	**cost**	**gap(%)**	**t(*s*)**
pmed1		5	5819	1	0	0	4397	**5819**	0.00	3.84
pmed2		10	4093	15	4	7	46578	**4093**	0.00	769.59
pmed3	100	10	4250	29	5	12	55372	**4250**	0.00	831.59
pmed4		20	3034	1	0	0	5545	**3034**	0.00	3.47
pmed5		33	1355	1	0	0	3305	**1355**	0.00	1.74
pmed6		5	7824	1222	14	375	1689472	7827	0.04	11396.20
pmed7		10	5631	1	0	0	14210	**5631**	0.00	37.02
pmed8	200	20	4445	1	0	0	47052	**4445**	0.00	45.32
pmed9		40	2734	3	1	0	69483	**2734**	0.00	561.25
pmed10		67	1255	3	1	0	10697	**1255**	0.00	578.28
pmed11		5	7696	52	8	0	665897	8498	10.42	13821.60
pmed12		10	6634	141	8	16	3367049	6639	0.08	12031.70
pmed13	300	30	4374	1	0	0	17476	**4374**	0.00	27.24
pmed14		60	2968	19	5	9	4437794	2979	0.37	4909.11
pmed15		100	1729	1	0	0	12682	**1729**	0.00	8.57
pmed16		5	8162	1	0	0	399087	9179	12.46	949.43
pmed17		10	6999	58	7	0	1043513	8715	24.52	12874.70
pmed18	400	40	4809	15	3	7	5688576	4866	1.19	5040.01
pmed19		80	2845	3	1	1	865816	2859	0.49	1432.68
pmed20		133	1789	1	0	0	11562	**1789**	0.00	8.40
pmed21		5	9138	1	0	1	500505	9996	9.39	2178.24
pmed22		10	8579	25	7	0	1471261	10309	20.17	18021.40
pmed23	500	50	4619	5	2	2	4357584	4625	0.13	8264.04
pmed24		100	2961	13	4	2	11765777	3034	2.47	11879.20
pmed25		167	1828	1	0	0	112931	1830	0.11	65.54
pmed26		5	9917	1	0	1	1200404	11562	16.59	6293.14
pmed27		10	8307	1	0	0	1200117	9764	17.54	5263.18
pmed28	600	60	4498	15	4	0	15268367	4524	0.58	16707.00
pmed29		120	3033	15	4	3	17050266	3043	0.33	15538.80
pmed30		200	1989	32	6	0	2660719	2000	0.55	15105.70
pmed31		5	10086	1	0	0	1400305	11187	10.92	10042.50
pmed32	700	10	9297	1	0	0	699544	10759	15.73	4753.99
pmed33		70	4700	31	5	0	18769506	4715	0.32	16574.10
pmed34		140	3013	27	5	1	17194437	3071	1.92	16124.70
pmed35		5	10400	1	0	1	800205	11775	13.22	5452.39
pmed36	800	10	9934	1	0	0	800210	11922	20.01	4775.64
pmed37		80	5057	15	4	0	453706	5245	3.72	14714.30
pmed38		5	11060	1	0	1	900105	12265	10.90	8270.78
pmed39	900	10	9423	1	0	0	899847	11650	23.63	6377.96
pmed40		90	5128	13	4	5	10844189	5144	0.31	15619.20

instances where our method did not found the optimal solutions, the average gap was 7.79% with a maximum value of 24.52%. The B&P stopped at the root node without generating columns with negative reduced costs in 19 instances. In the others 21 instances, the branching procedure took place, since an integer solution was not found. From these tests, we can conclude that the B&P can generate good quality PMP solutions.

The largest tree explored was in instance *pmed6*, where 1222 nodes were solved within 11396.2 seconds. Since we implemented a prune procedure, 375 sub-trees were pruned in this instance, avoiding processing unnecessary nodes. Even though we set $3h$ as the B&P time limit, in some instances a node was still being solved

when the time limit was reached. This is the reason for the execution in some of them took more than $3h$.

4.5.2 Overlap control results

In this subsection we present the summary of the results with each pair of values $z = \{0, 0.5, 1\}$ and $\alpha = \{0.05, 0.5, 0.95\}$ in each of the 40 OR-library instances. Since the difficult an instance in the PMP context is related to its n/p ratio, we organized the instances into sets of same n/p values. For example, instances *pmed3* and *pmed8* are in the same set as both have equal n/p ratio ($\frac{100}{10} = \frac{200}{20}$). Then, we computed the average results for each of the n/p sets related to each pair of z and α. These results are shown in the Table C.1. The first column of this table shows the rounded n/p ratio; the second column is the number of instances with the same n/p value; the third and the fourth columns are the z and α parameters values, respectively; and the remaining columns are the average results for each information. The complete results are presented in Table 4.2 of Appendix C.

Table 4.2 - Summary of the overlap control results in the OR-library instances (BEASLEY, 1985; BEASLEY, 1990).

				B&P						
n/p	**#**	**z**	**α**	**ts**$_a$	**th**$_a$	**np**$_a$	**ncg**$_a$	$\|sc\|_a$	**mc**$_a$**(%)**	$t_a(s)$
3	6	0	0.05	1.00	0.00	0.00	700782.33	1.33	0.33	666.96
			0.50	1.00	0.00	0.00	700782.33	1.34	0.33	554.54
			0.95	1.00	0.00	0.00	668182.33	1.33	0.33	528.48
		0.5	0.05	44.00	5.83	13.50	4701391.50	4.57	0.69	2083.99
			0.50	3.00	0.83	0.67	203149.00	3.33	0.58	120.54
			0.95	1.00	0.00	0.00	362632.33	3.62	0.56	223.35
		1	0.05	17.17	4.50	3.17	4467165.33	7.68	0.70	3403.00
			0.50	1.00	0.00	0.00	108982.33	58.53	0.76	59.12
			0.95	1.00	0.00	0.00	201032.33	85.30	0.93	85.71
5	7	0	0.05	1.00	0.00	0.00	800717.14	1.20	0.20	856.67
			0.50	1.00	0.00	0.00	800717.14	1.20	0.20	568.08
			0.95	1.00	0.00	0.00	800717.14	1.20	0.20	534.88
		0.5	0.05	36.71	5.29	7.86	7705701.57	3.82	0.64	5414.77
			0.50	1.00	0.00	0.00	324817.14	3.03	0.47	203.26
			0.95	1.00	0.00	0.00	440074.29	3.00	0.37	279.51
		1	0.05	39.86	3.29	5.86	6958425.00	7.41	0.61	3493.47
			0.50	1.00	0.00	0.00	178474.29	6.75	0.54	105.98
			0.95	1.00	0.00	0.00	600817.14	47.71	0.79	297.17
10	10	0	0.05	1.00	0.00	0.00	920607.00	1.10	0.10	1334.55
			0.50	1.00	0.00	0.00	920607.00	1.10	0.10	696.51
			0.95	1.00	0.00	0.00	920607.00	1.10	0.10	595.56
		0.5	0.05	291.60	10.60	62.70	11550302.80	2.49	0.61	12682.83
			0.50	1.00	0.00	0.00	523507.00	2.10	0.29	349.39
			0.95	1.00	0.00	0.00	455427.00	2.01	0.30	281.61
		1	0.05	215.60	19.80	97.40	5443384.50	3.40	0.53	3209.45
			0.50	1.00	0.00	0.00	224607.00	3.58	0.45	149.82

Continued on next page

Table 4.2: Conclusion.

				B&P						
n/p	**#**	**z**	$\boldsymbol{\alpha}$	$\mathbf{ts}_a$	$\mathbf{th}_a$	$\mathbf{np}_a$	$\mathbf{ncg}_a$	$\lvert sc\rvert_a$	$\mathbf{mc}_a(\%)$	$t_a(s)$
			0.95	1.00	0.00	0.00	673307.00	13.34	0.59	378.97
			0.05	1.00	0.00	0.00	480768.00	1.03	0.03	458.43
		0	0.50	1.00	0.00	0.00	480768.00	1.03	0.03	280.29
			0.95	1.00	0.00	0.00	480768.00	1.03	0.03	271.02
			0.05	58.40	6.60	2.60	21070468.00	1.46	0.34	13254.70
[20, 40]	5	0.5	0.50	1.00	0.00	0.00	480768.00	1.37	0.19	300.64
			0.95	1.00	0.00	0.00	480768.00	1.36	0.19	284.39
			0.05	49.60	5.40	7.20	15873068.00	1.40	0.26	10398.69
		1	0.50	1.00	0.00	0.00	480768.00	1.48	0.26	322.74
			0.95	1.00	0.00	0.00	480768.00	1.48	0.26	306.95
			0.05	1.00	0.00	0.00	1050483.75	1.02	0.02	1321.99
		0	0.50	1.00	0.00	0.00	1050483.75	1.02	0.02	755.63
			0.95	1.00	0.00	0.00	1050483.75	1.02	0.02	690.15
			0.05	11.00	3.00	0.00	8346833.75	1.81	0.36	16002.93
[50, 70]	4	0.5	0.50	1.00	0.00	0.00	1050483.75	1.26	0.12	749.90
			0.95	1.00	0.00	0.00	1050483.75	1.17	0.09	664.63
			0.05	11.25	2.75	1.25	9096458.75	1.98	0.33	13651.08
		1	0.50	1.00	0.00	0.00	1050483.75	1.93	0.26	1291.50
			0.95	1.00	0.00	0.00	1050483.75	1.81	0.36	994.81
			0.05	1.00	0.00	0.00	1200406.25	1.01	0.01	1980.40
		0	0.50	1.00	0.00	0.00	1200406.25	1.01	0.01	1026.62
			0.95	1.00	0.00	0.00	1200406.25	1.01	0.01	908.56
			0.05	7.00	2.75	0.00	6997506.25	1.33	0.18	15747.08
[80, 120]	4	0.5	0.50	1.00	0.00	0.00	1200406.25	1.31	0.16	1201.54
			0.95	1.00	0.00	0.00	1200406.25	1.35	0.30	795.91
			0.05	8.00	2.75	0.00	8196903.00	1.82	0.34	16581.48
		1	0.50	1.00	0.00	0.00	1200406.25	1.50	0.11	1811.20
			0.95	1.00	0.00	0.00	1200406.25	1.87	0.37	1254.24
			0.05	1.00	0.00	0.00	1600205.00	1.01	0.01	3567.40
		0	0.50	1.00	0.00	0.00	1600205.00	1.01	0.01	2254.35
			0.95	1.00	0.00	0.00	1600205.00	1.01	0.01	1612.41
			0.05	1.67	1.33	0.00	2533071.67	1.53	0.47	16892.87
[140, 180]	3	0.5	0.50	1.00	0.00	0.00	1600205.00	1.10	0.06	1328.89
			0.95	1.00	0.00	0.00	1600205.00	1.10	0.05	1234.88
			0.05	3.00	1.67	0.33	4798605.00	1.37	0.17	17449.23
		1	0.50	1.00	0.00	0.00	1600205.00	1.41	0.19	3358.60
			0.95	1.00	0.00	0.00	1600205.00	1.38	0.18	2866.55

From Table 4.2, we can assume that the overlap control worked properly and the B&P was able to find PMPOC solutions. In all instances sets, as shown by $|sc|_a$ and $mc_a(\%)$ results, the minimum overlapping between the clusters were found using $z = 0$, as expected. However, even in the $z = 0$ and $\alpha = 0.05$ scenario, some overlap is observed. The reason why this occurs is the fact that constraints (4.10b) allow it and the selection of the p-medians relies on the generated column pool.

Note that the percentage of vertices belonging in more than one cluster increase as the z value increase, although this increase is note proportional neither linear. This can observed mainly in the easier instances (small n/p ratio), where there are more

medians and the vertices can be divided fairly. Furthermore, similar behavior can be seen in the average normalized total number of covers ($|sc|_a$). In the easier set of instances ($n/p = 3$) and for $z = 1$ and $\alpha = 0.95$ each vertex was covered, on average, 85 times. In such scenarios one could decrease the overlap control parameters values to avoid such extreme solutions.

Some outliers results were observed in the harder instances sets. For example, in instance set $n/p = [140, 180]$ with $z = 0.5$, the highest overlap degree was found with $\alpha = 0.05$. As the number of vertices per median is high, the overlap become fuzzy and harder to control. Also, recollect that our PMPOC restricted master problem is a modification of the original one, where there is a OCC for each vertex. Then, the overlap control is not precise as would be in the original model.

The branching took place only when using $\alpha = 0.05$, i.e., when the PMP part of the objective function (4.13a) is more relevant than the u variable. For the others α the tree size was equal to one, meaning that the execution stopped at the rode node. This is due the fact that the branching is performed over the y variables rather than the u variable. Then, if the overlap control has equal or more relevance than the PMP part, an integer solution is find more easily since PMP part of the objective function ins not as relevant as the $\alpha = 0.05$.

In addition, as one can see from results presented in Appendix C, the number of columns generated is nearly the same in most of the instances of the same set of n/p values, e.g., instances *pmed3* and *pmed8*. This corroborates with the fact that instances with similar n/p ratio have the same level of difficult in the PMP context.

Note that the average execution time, in some scenarios, was greater than the time limit of $3h$. As explained in Subsection 4.5.1, we did not stop a node execution when the time limit is reached.

4.6 Conclusions and future directions

In this chapter, we proposed a new variant of the p-median problem namely, p-median problem with overlap control. In addition, a parallel branch-and-price algorithm was developed to solve the PMPOC.

In the PMP context, our method was able to find good-quality solutions at a reasonable computational time. Indeed, the B&P found optimal solutions in 12 of 40 instances. In the instances where our algorithm did not found the optimal solutions, the average gap was of 7.79%.

Considering the overlap control tests, the results of the application of the PMPOC model and the B&P algorithm proved to be satisfactory. The overlap control worked properly in instances of small to medium n/p ratio. Regarding the harder instances, our method found good solutions in terms of overlapping between the medians, but a careful parameter adjustment should be carried out as the overlap control is not precise as in the easier instances.

Further investigation needs to be done to test other branching rules. Moreover, tests comparing a depth-first search exploration against the implemented breadth-first search should be performed. In addition, we should apply the proposed B&P algorithm in larger instances and adapt it and the model to other overlapping clustering problems.

5 A PARALLEL ADAPTIVE LARGE NEIGHBORHOOD SEARCH FOR MULTIPLE ASSIGNMENT p-MEDIAN PROBLEMS

This chapter is organized as follows. An introduction is presented in Section 5.1. The mathematical formulations are presented in Section 5.2. The proposed parallel adaptive large neighborhood search is detailed in Section 5.3. The results of the computational tests are shown in Section 5.4. Our concluding remarks and considerations on future work are presented in Section 5.5.

5.1 Introduction

Facility location problems are extensively studied and an essential topic in operations research. The aim is to determine the location of a given number of facilities, whereas the cost of serving the clients is minimized or the profit maximized. These problems have several applications in logistics and data mining (NG; HAN, 1994; HANSEN et al., 2009; GRANGIER et al., 2016; CONTARDO et al., 2019).

A classical facility location problem is the *p-median problem* (PMP) (HAKIMI, 1964). Given a graph, the PMP's objective is to choose p vertices, also known as medians, minimizing the sum of the distances from each vertex to its closest median. However, there are some applications where a client should be assigned to more than one facility, which is not allowed in the PMP. These cases are common in some critical services provided by emergency facilities, such as hospitals and fire stations, and computer networks, where a backup coverage is needed (WANG et al., 2009; CHAGAS et al., 2019; PANTELI et al., 2019; ARAÚJO et al., 2020). Then, Panteli et al. (2019) proposed the *multiple p-median problem* (MPMP), which generalizes the PMP requesting to each vertex be assigned to $mc \geq 1$ medians. Note that when $mc = 1$ the PMP is defined. These authors also proposed the *Biclustering Multiple Median algorithm* (BIMM) to solve the MPMP and compared it with a commercial solver.

Even though the literature related to p-medians problems is vast (BARBAROS et al., 1983; REESE, 2006; MLADENOVIĆ et al., 2007; DASKIN; MAASS, 2015; MARÍN; PELEGRÍN, 2019), to the best of our knowledge, there are few works concerning variations of these problems where vertices can be assigned to more than one median. One of these studies is the work of Wang et al. (2009), who introduced the backup 2-center problem and the backup 2-median problem. In these problems, every vertex is served by two medians. Another study that we are aware of is the work of Karatas et al. (2016). The authors introduced the requirement of each vertex

to be assigned to a number of facilities and compared it under five different criteria.

Then, following Panteli et al. (2019), we also relaxed the assignment constraints of the *capacitated p-median problem* (CPMP) and the *p-center problem* (PCP), two variations of the PMP and classical facility location problems. The CPMP is a modification of the PMP where each median has a capacity and each client has a demand. All clients' demands must be fulfilled without exceeding the medians' capacities. Introduced by Hakimi (1965), the PCP aims to minimize the maximum distance between a vertex and its closest median. Then, we introduced, in this work, the *capacitated multiple p-median problem* (CMPMP) and the *multiple p-center problem* (MPCP). These problems are similar to the original ones, but they required each vertex to be served by mc medians.

The PMP, CPMP and PCP were proven to be NP-hard (GAREY; JOHNSON, 1979; KARIV; HAKIMI, 1979; MASUYAMA et al., 1981). Since the MPMP, CMPMP and the MPCP are generalizations of these problems, there are also NP-hard. Hence, exact methods are only practical in small-sized instances and metaheuristics are an alternative to obtaining solutions at a low computational time. Therefore, we developed an *adaptive large neighborhood search* (ALNS) (PISINGER; ROPKE, 2007; ROPKE; PISINGER, 2006) and applied it to MPCP, CMPMP and MPCP. In order to take advantage of modern multi-core CPU, we implemented a parallelized version of the ALNS metaheuristic (PALNS) (ROPKE, 2009). The ALNS was proposed in the context of vehicle routing problems and it has been successfully applied to several other problems (AVCI; AVCI, 2019; LAHYANI et al., 2019; HAMMAMI et al., 2020). Also, the ALNS was previously applied to a facility location problem (PEREIRA et al., 2015).

5.2 Mathematical notation and problems definitions

Let $G = (V, E)$ be an *undirected*, *weighted* and *connected* graph, where V is the set of vertices and E is the set of edges, where $|V| = n$, $|E| = m$ and to each edge $(i, j) \in E$ is associated a weight $d_{ij} \in \mathbb{R}$. In facility location problems, d_{ij} is often the euclidean distance or the length of the shortest path between vertices i and j, but dissimilarity values are also common. In all these cases, the triangular inequality is not violated.

It is assumed that there is a distance d_{ij} between every pair of vertices $i, j \in V$. Note that even though an edge joining vertices i and j may not exists in the original graph, (i, j) can be added to E with d_{ij} equal to the length of the shortest path

between these vertices since G is connected and the triangular inequality holds. In this way, $D = (d_{ij})$ is an $n \times n$ *distance matrix* of non-negative real values. The *radius* of a median j is determined by the farthest distance of a vertex i assigned to it.

Let P be the set of the p open medians. Since it is required that all vertices must be assigned to mc facilities in the MPMP, CMPMP and the MPCP, it is implicitly assumed that each vertex is always assigned to the mc closest medians from the p open ones. Let $\phi_b(i)$ be the $(mc+1)$-th nearest open median from vertex i and let $\phi_f(i)$ be the farthest of the mc medians which vertex i is currently assigned to. Whenever a facility j is closed, each vertex i that was assigned to j is automatically assigned to $\phi_b(i)$. Whenever a facility j is open, every vertex i whose $\phi_f(i)$ is farthest than j, is removed from $\phi_f(i)$ and assigned to j.

In this work, a subset $C \subseteq V$ is also called a cluster. In addition, facility and center are used interchangeably to denote a median vertex.

The remainder of this section is divided as follows. In Subsection 5.2.1, the MPMP formulation is presented. The integer linear program of the CMPMP is described in Subsection 5.2.2. The MPCP formulation is shown in Subsection 5.2.3.

5.2.1 MPMP formulation

The MPMP requires that exactly p medians be selected from V and that all other vertices are assigned to the closest mc medians. It is implicitly assumed that a facility vertex is assigned to itself. The MPMP can be formulated as the integer linear program (MPMP-ILP) (PANTELI et al., 2019) as shown by Equations (5.1a)-(5.1e).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{5.1a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} \geq mc, \qquad i = 1, \ldots, n, \tag{5.1b}$$

$$\sum_{j=1}^{n} x_{jj} = p, \tag{5.1c}$$

$$x_{ij} \leq x_{jj}, \qquad i = 1, \ldots, n, \ j = 1, \ldots, n, \tag{5.1d}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, n, \ j = 1, \ldots, n, \tag{5.1e}$$

where decision variables x_{ij} control whether client i is allocated at facility j or not, i.e.,

$$x_{ij} = \begin{cases} 1, & \text{if vertex } i \in V \text{ is assigned to median vertex } j \in V, \\ 0, & \text{otherwise.} \end{cases}$$

In the MPMP-ILP, the objective function (5.1a) minimizes the sum of distances between every vertex i assigned to each median j. Constraints (5.1b) are the multiple assignment constraints and impose that every vertex must be covered by at least mc clusters. Equation (5.1c) guarantee that p vertices are open medians. A vertex i can only be assigned to a vertex j if j is an open facility, i.e, only if $x_{jj} = 1$ and this is ensured by Inequalities (5.1d). Constraints 5.1e define variables x_{ij} as binary. Note that the difference between the MPMP-ILP and the the PMP model (REVELLE; SWAIN, 1970) is only the constraint (5.1b), which is relaxed in the former to allow multiple assignments.

5.2.2 CMPMP formulation

Analogously to the MPMP, the CMPMP seeks to select p vertices as medians whereas the sum of distances of every graph vertex to the mc nearest medians is minimized. However, in the CMPMP the total demand of all vertices assigned to a facility cannot exceed its capacity. The CMPMP can be formulated as the integer linear program (CMPMP-ILP) as shown by (5.2a)-(5.2e).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \tag{5.2a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} \geq mc, \qquad i = 1, \ldots, n, \tag{5.2b}$$

$$\sum_{j=1}^{n} x_{jj} = p, \tag{5.2c}$$

$$\sum_{i=1}^{n} q_i x_{ij} \leq Q x_{jj}, \qquad j = 1, \ldots, n, \tag{5.2d}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, n, \; j = 1, \ldots, n. \tag{5.2e}$$

The MPMP-ILP and CMPMP-ILP are similar formulations, the only difference between them are the capacity constraints (5.2d). These constraints are a modified

version of constraints (5.1d) which ensure that the every median capacity Q is not exceeded by the sum of its clients demands, where q_i is the demand of client i. The constraint (5.2b) is a generalization of the assignment constraint of the original CPMP formulation (MULVEY; BECK, 1984).

In addition, note that if the p medians are fixed, the CMPMP reduces to the *generalized multi-assignment problem* (GMAP) (PARK et al., 1998), which is much easier to solve than the CMPMP (FLESZAR; HINDI, 2008). The GMAP integer linear formulation (GMAP-ILP) can be defined as shown in Equations (5.3a) to (5.3d).

$$\min \sum_{i=1}^{n} \sum_{j \in P} d_{ij} x_{ij} \tag{5.3a}$$

subject to

$$\sum_{j \in P} x_{ij} \geq mc, \qquad i = 1, \ldots, n, \tag{5.3b}$$

$$\sum_{i=1}^{n} q_i x_{ij} \leq Q, \qquad \forall j \in P, \tag{5.3c}$$

$$x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, n, \ j \in P. \tag{5.3d}$$

Note that if we relax the integrality requirement of variables of x variables of the GMAP-ILP and if we substitute them by $\frac{x'_{ij}}{q_i}$, it becomes a *minimum-cost network flow problem* (MCNFP) (FLESZAR; HINDI, 2008). Also, it is necessary to create a dummy sink node to deal with the possibly remaining flow (capacity). A MCNFP can be solved in polynomial time and it yields a fair lower bound to the GMAP (FLESZAR; HINDI, 2008).

5.2.3 MPCP formulation

In the MPCP, p vertices, from V, are selected as medians and each vertex is assigned to the nearest mc of them. We relaxed the mathematical formulation of the PCP (DASKIN, 1995) in order allow each vertex to allow multiple assignments. Let $\sigma_{mc}(i)$ be the sum of the distances between a vertex i and the mc medians which it is assigned to. The objective is to minimize $v(MPCP) = \max_{i \in V}(\sigma_{mc}(i))$, i.e, the maximum sum of distances between a vertex and the mc medians that serve it. The MPCP mixed-integer linear program (MPCP-MILP) is presented by (5.4a)-(5.4f).

$$\min\ z \tag{5.4a}$$

subject to

$$\sum_{j=1}^{n} x_{ij} \geq mc, \qquad i = 1, \ldots, n, \tag{5.4b}$$

$$\sum_{j=1}^{n} x_{jj} = p, \tag{5.4c}$$

$$x_{ij} \leq x_{jj}, \qquad i = 1, \ldots, n,\ j = 1, \ldots, n, \tag{5.4d}$$

$$\sum_{j=1}^{n} d_{ij} x_{ij} \leq z, \qquad i = 1, \ldots, n, \tag{5.4e}$$

$$z \in \mathbb{R},\ x_{ij} \in \{0, 1\}, \qquad i = 1, \ldots, n,\ j = 1, \ldots, n. \tag{5.4f}$$

The MPCP-MILP shares similarities with the models presented in the previous subsections. However, there are some differences between them, such as the extra continuous variable $z \in \mathbb{R}$. The value of z is minimized by the objective function (5.4a) which lower bound is given by constraint (5.4e). In other words, the maximum sum of distances between a vertex and its mc closest medians is minimized.

5.3 Parallel adaptive large neighborhood search

Proposed by Pisinger and Ropke (PISINGER; ROPKE, 2007; ROPKE; PISINGER, 2006), the ALNS is a neighborhood-based metaheuristic where the neighborhood functions are continuously evaluated and their usage selection is adapted based on their performance. These functions are divided into two sets namely, destroy and repair operators. At each iteration, a destroy and a repair method are randomly selected and applied to the current solution. The probability to pick out a method is computed considering its score, which is obtained through the previous metaheuristic' iterations. If a pair of destroy and repair operators improves the current solution or if it generates a new solution, their score are increased. Methods with a higher score have a higher probability to be chosen.

In order to take advantage of modern multi-core CPU, we implemented a parallelized version of the ALNS metaheuristic. We developed our PALNS following the guidelines of Ropke (2009). However, instead of sharing the current solution among all worker threads, we let each thread has its own local copy and only the best solution is shared between them. Then, each worker thread performs the ALNS main

loop independently and updates the global incumbent solution whenever a better solution is found. In this way, we can avoid the overhead of sharing the current solution among all threads, since there are destroy/repair operators with different time complexity and the main thread must wait to the worker threads to finish applying their destroy/repair operators.

A pseudocode of our PALNS is shown in Algorithm 11. All these steps, with the exception of lines 10 and 21, are performed locally by each worker thread. In addition, all threads start with the same set of parameters and destroy/repair methods. At line 2, a random initial solution is generated and it becomes the current solution s and the incumbent solution s_{best}. A destroy method f_d and a repair method f_r are selected, at line 5, from Γ_d and Γ_r, respectively, using a roulette wheel mechanism based on the their current weights. Then, at line 6, the selected operators f_d and f_r are applied to the current solution s and a neighbor solution s' is generated. If the cost of s' is less than the cost of s, then it becomes s. In addition, if s' cost is less than s_{best} cost, it becomes the new incumbent, which is updated to all threads at line 10. The temperature t is locally updated through the PALNS execution, where it is cooled at line 17. In addition, we also utilized a re-heating process, shown at lines 19 to 21, that updates t with t_r whenever it becomes less than the temperature threshold t_t. Moreover, we update the worker thread current solution with the global incumbent whenever the re-heating temperature is reached.

At each iteration of Algorithm 11, the selected operators f_d and f_r are reward as follows: at line 11, with σ_1, if f_d and f_r generated a new best solution; at line 13, with σ_2, if f_d and f_r not improved the best solution but found a better current solution; at line 16, with σ_3, if the *simulated annealing* (SA) acceptance criteria (KIRKPATRICK et al., 1983), presented in Algorithm 12, is satisfied and if s' is a new solution. A hash-table data structure is used to keep track of the visited solutions. At line 18, the weight of the neighborhood functions are updated, at every θ iterations, using their accumulated scores.

The same PALNS implementation, presented in the above pseudocodes, is shared between the MPMP, CMPMP and MPCP contexts. Only solutions data structure, the destroy and repair methods and some minor features are adapted to each problem. These details are presented in the following subsections.

Algoritmo 11: PALNS.

input : set of destroy operators Γ_d; set of repair operators Γ_r; reward values σ_1, σ_2 and σ_3; maximum number of iterations max_iter; segment size θ; initial temperature t_{init}; threshold for re-heating t_t; re-heating temperature t_r; cooling rate α.

output: incumbent solution s_{best}.

```
1  begin
2      s_best ← s ← gen_initial_solution();
3      t ← t_init; i ← 0;
4      while (i < max_iter) do
5          [f_d, f_r] ← select_operators(Γ_d, Γ_r);
6          s' ← f_r(f_d(s));
7          if (cost(s') < cost(s)) then
8              s ← s';
9              if (cost(s') < cost(s_best)) then
10                 s_best ← s';
11                 reward(f_d, f_r, σ_1);
12             else
13                 reward(f_d, f_r, σ_2);
14         else if (accept(s', s, t) and is_new(s')) then
15             s ← s';
16             reward(f_d, f_r, σ_3);
17         t ← t · α; i ← i + 1;
18         if (i mod θ = 0) then update_weights(Γ) ;
19         if (t < t_t) then
20             t ← t_r;
21             s ← s_best;
22     end while
23     return s_best;
24 end
```

Algoritmo 12: SA acceptance criteria.

input : current solution s; new solution s'; current temperature t.

output: $true$ if inequality holds, $false$ otherwise.

```
1  begin
2      if (rand(0, 1) < exp(((−(s'.cost() − s.cost()))/t))) then return true ;
3      else return false ;
4  end
```

5.3.1 Initial solution

Since it is well known that the initial solution makes no difference on the ALNS final solution quality (BARRENA et al., 2014; C. CORDEAU J-F, 2012b; C. CORDEAU J-F, 2012a; LAHYANI et al., 2019), our implementation starts with a random solution. For each problem, p medians are selected at random and then each vertex is assigned to the closest mc of them.

In the CMPMP context, however, using this random method can lead to a scenario where some few vertices may not fit into any cluster, even though the total remaining capacity is greater than the total remaining demand. Then every time a set of p vertices are selected, the GMAP-ILP is solved. If the problem is feasible, then the p medians are used as initial solution. Otherwise, p new medians are randomly selected and this procedure is repeated until feasibility is detected.

5.3.2 Neighborhood functions

In this section, the PALNS neighborhood functions are described. All destroy and repair operators only close and open medians, respectively.

5.3.2.1 Destroy operators

The following destroy operators were implemented:

- *randomly close k medians* (d_1): in this method, k medians, chosen at random, are closed. Following Pereira et al. (2015), the value of k is computed by Equation (5.5), where $\omega_1 \in [0, 1]$ and $\omega_2 \in [0, 1]$ are two random variables given by an uniform distribution. Equation (5.5) gives a random value following a non-uniform decreasing distribution in the interval $[1, \frac{p}{2}]$, i.e., values close to 1 are more likely to occur than values close to $\frac{p}{2}$. This same operator is applied to all problems;

$$k = \lfloor \frac{p}{2}(|\omega_1 - \omega_2|) + 1 \rfloor. \tag{5.5}$$

- *close highest cost median* (d_2): close the median with the highest cost. For the MPMP and CMPMP, the median with the maximum sum of the d_{ij} distances is closed. For the MPCP, the median with the maximum radius is closed;

- *close minimum loss median* (d_3): close the median which yields the mini-

mum increase in the objective function. This operator was inspired by the heuristics of Resende and Werneck (2007) and of Mladenović et al. (2003). Let C_j be the set of vertices assigned to median j. The increasing in the objective function by closing median j is computed by Equation (5.6), for the MPMP and CMPMP, and by Equation (5.7), for the MPCP. Then, the median $j \in P$ with the minimum δ_j^l is closed.

$$\delta_j^l = \sum_{i \in C_j} (d_{i\phi_b(i)} - d_{ij}). \tag{5.6}$$

$$\delta_j^l = \max_{i \in C_j} (v(MPCP), \sigma_{mc}(i) - d_{ij} + d_{i\phi_b(i)}). \tag{5.7}$$

- *close farthest median from the farthest vertex* (d_4): close the farthest median from the farthest vertex, i.e., the median $\phi_f(i)$ of the vertex i with the maximum $\sigma_{mc}(i)$. This operator is only applied to the MPCP.

5.3.2.2 Repair operators

The repair operators are repeated until the number of open medians is equal to p. The following repair operators were implemented:

- *randomly open a median* (r_1): randomly open a median. The non-median vertex to be open is selected using an uniform distribution. This same operator is applied to all problems;
- *open smallest sum of distances* (r_2): greedily selects the non-median vertex with the smallest sum of distances to all other vertices. The sum of distances from a vertex to all other vertices is computed *a priori*. This same operator is applied to all problems;
- *open maximum gain median* (r_3): open a median which yields the maximum decrease in the objective function. This operator was inspired by the heuristics of Resende and Werneck (2007) and of Mladenović et al. (2003). The decrease in the objective function by opening a non-median vertex j is given by Equation (5.8), for the MPMP and CMPMP, and by Equation (5.9), for the MPCP. Then, the vertex $j \notin P$ with the maximum δ_j^g becomes a facility.

$$\delta_j^g = \sum_{i=1}^{n} \max(0, d_{i\phi_f(i)} - d_{ij}). \tag{5.8}$$

$$\delta_j^g = v(MPCP) - \max_{i \in V}(\sigma_{mc}(i) - \max(0, d_{i\phi_f(i)} - d_{ij})). \quad (5.9)$$

- *open closest median from the farthest vertex* (r_4): open the closest non-median vertex from the vertex whose the sum of distances is maximum, i.e, open the closest non-median vertex from vertex $k = \arg\max_{i \in V}(\sigma_{mc}(i))$. This operator is only applied to the MPCP.

5.3.3 CMPMP details

In order to evaluate a CMPMP solution, we implemented the Fleszar and Hindi (2008) approach. The idea proposed by the authors is to avoid solve the related GMAP-ILP every time a new solution is generated as much as possible. Given a solution s, two lower bounds are used:

- LB1(s): the MPMP cost, i.e., the capacities are ignored and just the assignment costs are considered;

- LB2(s): the MCNFP cost. As mentioned in Subsection 5.2.2, the linear relaxation of the GMAP-ILP is a MCNFP. Then, a complete bipartite network flow is built using the p medians as source vertices and the remaining client vertices as sink ones. An additional dummy sink vertex is needed to handle the remaining flow. The MCNFP is solved by a network simplex algorithm from a commercial solver.

With these lower bounds, two CMPMP solutions s and s' can be compared by Algorithm 13. If $LB1(s') \geq cost(s)$ or if $LB2(s') \geq cost(s)$, then the CMPMP cost of s' cannot be better than the cost of s. Otherwise, then the GMAP-ILP related to s' is solved. Additional hash tables are also used in Algorithm 13 to store MCNFP and GMAP-ILP costs, avoiding to recompute them. Moreover, the implementation of the Algorithm 13 can still be improved. For further details, please refer to the work of Fleszar and Hindi (2008).

Indeed, all CMPMP destroy and repair operators work only at the MPMP level, i.e., the medians' capacities and the vertices' demands are ignored. They are only considered when the related MCNFP or the related GMAP-ILP need to be solved, as presented in Algorithm 13.

Algoritmo 13: is better procedure (FLESZAR; HINDI, 2008).

input : new solution s'; solution s.
output: *true* if s' is better than s, $false$ otherwise.

```
1 begin
2   if (LB1(s') ≥ cost(s)) then return false ;
3   if (LB2(s') ≥ cost(s)) then return false ;
    /* compute cost(s') by solving the related GMAP-ILP          */
4   if (GMAP-ILP is feasible ∧ cost(s') < cost(s)) then
5     return true
6   else return false ;
7 end
```

5.3.4 PALNS setup

In this subsection, PALNS implementation details and the parameters used are presented. Each PALNS thread has its own operators score and its current solution. The input parameters and the operators are the same for all threads and only the incumbent solution is shared. Whenever the reheating procedure takes place, the incumbent solution is updated to the current solution of each thread. In addition, as presented in Subsection 5.3.2, the destroy operators d_1, d_2 and d_3 and repair operators r_1, r_2 and r_3 are applied to all problems. The exception are the d_4 and the r_4 methods, which are applied only to the MPCP.

The PALNS parameters' values, common to all problems, are presented in Table 5.1. In this table, θ is the PALNS segment size, α is the cooling rate, t_t is the threshold temperature for re-heating and σ_1, σ_2 and σ_3 are the reward values. Specifically, CALIBRA software (ADENSO-DÍAZ; LAGUNA, 2006) was used to obtain these values. The CALIBRA tests were performed in 10 instances of Beasley (1985), Beasley (1990) for the MPMP and MPCP and in 10 instances of Baldacci et al. (2002) for the CMPMP.

Table 5.1 - PALNS parameters' values utilized in the MPMP, CMPMP and MPCP.

Parameter	$\boldsymbol{\theta}$	$\boldsymbol{\alpha}$	$\boldsymbol{t_t}$	$\boldsymbol{\sigma_1}$	$\boldsymbol{\sigma_2}$	$\boldsymbol{\sigma_3}$
Value	200	0.99	10^{-3}	10	5	2

The others PALNS parameters namely, maximum number of iterations (max_iter), initial temperature (t_{init}) and re-heating temperature (t_r) were adjusted to each

scenario. In all MPMP and MPCP instances, we utilized $max\_iter = 2500$, $t_{init} = 1500$ and $t_r = 750$. For the CMPMP instances of up to 100 vertices, it was used $max\_iter = 3500$, $t_{init} = 2000$ and $t_r = 1000$. In CMPMP instances with 150 and 200 vertices, we utilized $max\_iter = 20000$, $t_{init} = 12500$ and $t_r = 7500$. For CMPMP instances with more than 200 vertices, it was used $max\_iter = 35000$, $t_{init} = 15000$ and $t_r = 10000$.

5.4 Computational experiments and analysis

All implementations were written in *C++* language and compiled with *g++* compiler version 10.1.0. For the resolution of all models we used the IBM® ILOG® CPLEX® 12.10 (IBM Corporation, 2020). The MCNFP was solved using the simplex network optmizer of this commercial solver. All the computational tests were executed on a computer with Intel® i7-8086K® CPU 4.00GHz × 12 processor with 12MiB *cache* memory and 62GiB of RAM. The operating system installed on this machine is Ubuntu 18.04.4 64 bits with kernel 5.3.0-40-generic.

All CPLEX executions were limited to 3 hours. In addition, we ran the PALNS 20 times in all instances and selected the best solution. Then, the average execution time, over the 20 runs, is presented.

To evaluate our heuristic in the MPMP and the MPCP context, we used the OR-library instances (BEASLEY, 1985; BEASLEY, 1990). This well-known set contains 40 instances with sizes ranging between 100 vertices and 900 vertices. For the CMPMP, the sets of instances that we utilized are described below.

- Instances of Osman and Christofides (1994): this set contains 20 CPMP instances divided into 10 graphs with 50 vertices and 10 graphs with 100 vertices;
- Instances of Baldacci et al. (2002): set of 20 instances with sizes $n = 150$ and $n = 200$, which are larger versions of the Osman and Christofides (1994) instances;
- Instances of Lorena and Senne (2003), Lorena and Senne (2004): this set consists of six real-world problem instances, with sizes ranging between 100 and 402 vertices, from the city of São José dos Campos, Brazil.

Following Panteli et al. (2019), we performed tests with our method using two pairs of p and mc values in each instance: $p = 10$ and $mc = 5$ and $p = 20$ and $mc = 10$.

Then, there is two scenarios for each instance of each problem. In the CMPMP instances, we also increased their capacities by the factor of mc to make them feasible. In addition, in some CMPMP instances we use the original p to avoid infeasibility.

The tests results are shown in the next subsections. In Subsection 5.4.1, we present experiments comparing the sequential version of our method with the parallel one. In the sequence, the results of the tests in the MPMP, CMPMP and the MPCP are shown in Subsections 5.4.2, 5.4.3 and 5.4.4, respectively. In these subsections, the CPLEX and the PALNS solutions costs ($cost$) and their executions times ($t(s)$), in seconds, are presented for each instance of each problem. Specifically, as we ran our heuristic 20 times in each instance, the average computational time ($t_{avg}(s)$) is shown. Also, the gap (%) between the solution cost of PALNS and the solution cost of CPLEX is presented. Let $cost_{best}$ be the cost of the best-known feasible solution and $cost'$ be the cost of a given solution. The gap between $cost'$ and $cost_{best}$ is determined by Equation (5.10).

$$gap = \frac{cost' - cost_{best}}{cost_{best}} \cdot 100. \tag{5.10}$$

5.4.1 Single thread and multithread analysis

In order to compare the impact of the parallelization of the PALNS against the sequential version of it, we performed tests in the MPMP context. For these tests, we selected nine OR-library instances, one of each size $n = \{100, 200, 300, 400, 500, 600, 700, 800, 900\}$ and used $p = 20$ and $mc = 10$ in all of them. The ALNS and the PALNS were executed 10 times in each instance. The results of these experiments are presented in Table 5.2. In this table, column σ is the standard deviation related to the solutions costs and $t_{avg}^{best}(s)$ is the average time, in seconds, that the method took to find the best solution.

In results presented by Table 5.2, both heuristics versions found the optimal solutions in all instances. Besides, it can be noticed that the PALNS took less computational time than the ALNS to find the best solution in every instance tested. The PALNS found the best solution in almost all of each of its 10 runs since the standard deviation observed was zero in all but one instance. We can then conclude that the parallel version is a better choice than the sequential one, as it finds the best solution faster and is more consistent.

Table 5.2 - Tests with $p = 20$ and $mc = 10$ in some OR-library instances to evaluate the benefits of the parallelization of the ALNS.

		ALNS			PALNS		
Instance	n	**cost**	σ	$t_{avg}^{best}(s)$	**cost**	σ	$t_{avg}^{best}(s)$
pmed1	100	84027	79.31	0.05	84027	0.00	0.05
pmed6	200	102347	42.35	0.21	102347	0.00	0.08
pmed11	300	93903	2.70	0.18	93903	0.00	0.13
pmed16	400	101027	4.22	0.77	101027	0.00	0.46
pmed21	500	114895	0.00	0.66	114895	0.00	0.25
pmed26	600	119392	0.00	0.11	119392	0.00	0.11
pmed31	700	123848	3.38	1.26	123848	2.21	0.24
pmed35	800	125727	0.00	0.58	125727	0.00	0.31
pmed38	900	133369	6.10	0.32	133369	0.00	0.31

5.4.2 MPMP results

Table 5.3 shows the MPMP results of our method in the OR-library instances (BEASLEY, 1985; BEASLEY, 1990). This table also presents the costs of the BIMM heuristic solutions, as shown by Panteli et al. (2019). Also, the gap between the solutions generated by this algorithm and the CPLEX solutions are presented. Since the BIMM was executed, by its authors, in a different machine than ours, its running time is not shown because a fair comparison cannot be made. Besides, the best solutions costs are bolded.

Table 5.3 - MPMP results in the OR-library instances (BEASLEY, 1985; BEASLEY, 1990).

				CPLEX		BIMM		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	**cost**	**gap (%)**	$t_{avg}(s)$
pmed1	100	10	5	**40592**	0.41	41462	2.14	**40592**	0.00	0.35
		20	10	**84027**	0.39	88745	5.61	**84027**	0.00	0.46
pmed2		10	5	**39421**	0.40	40134	1.81	**39421**	0.00	0.34
		20	10	**80660**	0.54	83021	2.93	**80660**	0.00	0.44
pmed3		10	5	**43345**	0.56	44000	1.51	**43345**	0.00	0.31
		20	10	**88180**	0.36	91166	3.39	**88180**	0.00	0.46
pmed4		10	5	**46854**	0.58	51351	9.60	**46854**	0.00	0.31
		20	10	**95441**	0.67	104680	9.68	95454	0.01	0.50
pmed5		10	5	**34167**	0.42	35054	2.60	**34167**	0.00	0.33
		20	10	**70836**	0.29	72192	1.91	**70836**	0.00	0.49
pmed6	200	10	5	**50759**	3.59	52734	3.89	**50759**	0.00	1.09
		20	10	**102341**	3.24	105089	2.69	**102341**	0.00	1.40
pmed7		10	5	**44978**	2.86	46621	3.65	**44978**	0.00	1.12
		20	10	**91465**	2.68	95486	4.40	**91465**	0.00	1.58
pmed8		10	5	**49837**	2.88	51064	2.46	**49837**	0.00	1.15
		20	10	**101003**	2.61	103998	2.97	**101003**	0.00	1.59
pmed9		10	5	**47636**	3.14	48638	2.10	**47636**	0.00	1.29
		20	10	**96365**	3.10	99371	3.12	**96365**	0.00	1.43
pmed10		10	5	**36864**	3.38	37968	2.99	**36864**	0.00	1.11
		20	10	**74770**	4.18	77136	3.16	**74770**	0.00	1.55

Continued on next page

Table 5.3: Continuation.

				CPLEX		BIMM		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	**cost**	**gap (%)**	$t_{avg}(s)$
pmed11	300	10	5	**46297**	19.16	47657	2.94	**46297**	0.00	2.41
		20	10	**93903**	13.30	94851	1.01	**93903**	0.00	2.92
pmed12		10	5	**53082**	18.10	54997	3.61	**53082**	0.00	2.48
		20	10	**106863**	19.81	111812	4.63	**106863**	0.00	3.43
pmed13		10	5	**48257**	18.78	49012	1.56	**48257**	0.00	2.43
		20	10	**97837**	14.21	99802	2.01	**97837**	0.00	3.50
pmed14		10	5	**55342**	20.43	56304	1.74	**55342**	0.00	2.36
		20	10	**111488**	19.85	113774	2.05	**111488**	0.00	3.27
pmed15		10	5	**47426**	17.12	47581	0.33	**47426**	0.00	2.41
		20	10	**96190**	19.53	98231	2.12	**96190**	0.00	3.41
pmed16	400	10	5	**49941**	47.65	51171	2.46	**49941**	0.00	4.16
		20	10	**101027**	47.73	103530	2.48	**101027**	0.00	5.08
pmed17		10	5	**53403**	49.11	55475	3.88	**53403**	0.00	4.19
		20	10	**107608**	70.44	111679	3.78	**107608**	0.00	5.99
pmed18		10	5	**59089**	50.53	59734	1.09	**59089**	0.00	3.69
		20	10	**119282**	51.68	121202	1.61	**119282**	0.00	5.42
pmed19		10	5	**56234**	49.40	57270	1.84	**56234**	0.00	4.14
		20	10	**113107**	50.92	115688	2.28	**113107**	0.00	5.09
pmed20		10	5	**58389**	49.58	59239	1.46	**58389**	0.00	4.18
		20	10	**118523**	44.04	121468	2.48	**118523**	0.00	6.01
pmed21	500	10	5	**56961**	93.45	57735	1.36	**56961**	0.00	5.51
		20	10	**114895**	87.38	116754	1.62	**114895**	0.00	8.30
pmed22		10	5	**62650**	135.57	64217	2.50	**62650**	0.00	6.26
		20	10	**125994**	149.91	132925	5.50	**125994**	0.00	8.90
pmed23		10	5	**60660**	107.13	62488	3.01	**60660**	0.00	6.44
		20	10	**122437**	100.05	127093	3.80	**122437**	0.00	8.96
pmed24		10	5	**60210**	105.11	61725	2.52	**60210**	0.00	6.46
		20	10	**121462**	127.16	124517	2.52	**121462**	0.00	8.64
pmed25		10	5	**54793**	90.52	56284	2.72	**54793**	0.00	6.41
		20	10	**111435**	83.16	114231	2.51	**111435**	0.00	9.23
pmed26	600	10	5	**59347**	154.40	59955	1.02	**59347**	0.00	10.92
		20	10	**119392**	172.47	121537	1.80	**119392**	0.00	12.92
pmed27		10	5	**57705**	143.48	58046	0.59	**57705**	0.00	8.80
		20	10	**116498**	135.63	117508	0.87	**116498**	0.00	12.88
pmed28		10	5	**58252**	195.00	59076	1.41	**58252**	0.00	9.10
		20	10	**117933**	136.07	120718	2.36	**117933**	0.00	12.90
pmed29		10	5	**60745**	160.02	61661	1.51	**60745**	0.00	7.84
		20	10	**122339**	150.88	125649	2.71	**122339**	0.00	10.94
pmed30		10	5	**65738**	177.32	66300	0.85	**65738**	0.00	9.05
		20	10	**133069**	139.75	133935	0.65	**133069**	0.00	12.58
pmed31	700	10	5	**61463**	244.27	62571	1.80	**61463**	0.00	10.53
		20	10	**123848**	240.92	129303	4.40	**123848**	0.00	17.44
pmed32		10	5	**67073**	290.61	68186	1.66	**67073**	0.00	12.17
		20	10	**134470**	569.17	137108	1.96	**134470**	0.00	17.93
pmed33		10	5	**66024**	239.31	67924	2.88	**66024**	0.00	12.66
		20	10	**132822**	228.99	136182	2.53	**132822**	0.00	17.88
pmed34		10	5	**63475**	218.37	64656	1.86	**63475**	0.00	13.40
		20	10	**127779**	240.73	130290	1.97	**127779**	0.00	17.66

Continued on next page

Table 5.3: Conclusion.

				CPLEX		BIMM		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	**cost**	**gap (%)**	$t_{avg}(s)$
pmed35	800	10	5	**62408**	432.30	62937	0.85	**62408**	0.00	13.95
		20	10	**125727**	427.53	127188	1.16	**125727**	0.00	23.02
pmed36		10	5	**70805**	409.19	72878	2.93	**70805**	0.00	17.07
		20	10	**142084**	693.30	149330	5.10	**142084**	0.00	22.61
pmed37		10	5	**74125**	381.64	74661	0.72	**74125**	0.00	16.46
		20	10	**149976**	265.97	152607	1.75	**149976**	0.00	21.73
pmed38	900	10	5	**66456**	704.86	68235	2.68	**66456**	0.00	21.31
		20	10	**133369**	1091.66	135485	1.59	**133369**	0.00	28.50
pmed39		10	5	**66129**	456.37	66604	0.72	**66129**	0.00	20.80
		20	10	**133246**	831.62	136345	2.33	**133246**	0.00	29.47
pmed40		10	5	**75386**	460.13	78237	3.78	**75386**	0.00	20.48
		20	10	**151713**	654.96	153743	1.34	**151713**	0.00	29.43
# best		-	-	80	-	0	-	79	-	-

All CPLEX solutions shown in Table 5.3 are optimal and were achieved within the time limit of three hours. Our heuristic obtained better MPMP costs than the BIMM in all the 80 instances, decreasing the total average gap from 2.55% to 0.0002%. Indeed, 79 of the 80 solutions costs are optimal. Only in the *pmed4* instance, with $p = 20$ and $mc = 10$, the PALNS did not achieve optimality. However, the solution found in this instance scenario has a gap of just 0.01% from the optimal one. Moreover, our method's MPMP results can be considered consistent since the maximum coefficient of variation, related to the 20 runs in all instances, was 0.01% for the solutions' costs.

As one can note from Table 5.3, the proposed method was significantly better than the CPLEX in terms of computational time in almost all instances. Apart from the smallest instances (with $n = 100$), both methods showed similar execution times, and the PALNS outperforms the CPLEX. Indeed, our method was up to 38 times faster than the commercial solver in the largest ones. Considering the average of the executions times in all 80 scenarios, the PALNS was nearly 19 times faster than the CPLEX.

5.4.3 CMPMP results

Table 5.4 shows the CMPMP results of our method in the instances of Osman and Christofides (1994) (*CCPX01* to *CCPX20*), Baldacci et al. (2002) (*CCPX21* to *CCPX40*) and Lorena and Senne (2003), Lorena and Senne (2004) (*SJC1* to *SJC4b*). These table has the same structure of the previous one.

Table 5.4 - CMPMP results in Osman and Christofides (1994), Baldacci et al. (2002) and Lorena and Senne (2003), Lorena and Senne (2004) instances.

				CPLEX		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	$t_{avg}(s)$
CCPX01	50	10	5	**7260**	1.02	**7260**	0.00	0.21
		20	10	**14773**	0.64	**14773**	0.00	0.32
CCPX02		10	5	**6556**	0.85	**6556**	0.00	0.21
		20	10	**13330**	0.56	**13330**	0.00	0.32
CCPX03		10	5	**7092**	0.89	**7092**	0.00	0.21
		20	10	**14388**	0.57	**14388**	0.00	0.31
CCPX04		10	5	**6237**	0.61	**6237**	0.00	0.21
		20	10	**12768**	0.47	**12768**	0.00	0.32
CCPX05		10	5	**6538**	0.76	**6538**	0.00	0.21
		20	10	**13414**	0.53	**13414**	0.00	0.32
CCPX06		10	5	**7108**	1.05	**7108**	0.00	0.21
		20	10	**14308**	0.72	**14308**	0.00	0.32
CCPX07		10	5	**6453**	0.94	**6453**	0.00	0.21
		20	10	**13148**	0.59	**13148**	0.00	0.33
CCPX08		10	5	**6701**	0.52	**6701**	0.00	0.20
		20	10	**14054**	0.55	**14054**	0.00	0.32
CCPX09		10	5	**6166**	0.68	**6166**	0.00	0.20
		20	10	**12627**	0.47	**12627**	0.00	0.32
CCPX10		10	5	**7081**	0.77	**7081**	0.00	0.21
		20	10	**14407**	0.55	**14407**	0.00	0.32
CCPX11	100	10	5	**13312**	6.30	13313	0.01	1.13
		20	10	**27044**	7.95	**27044**	0.00	0.71
CCPX12		10	5	**13703**	12.97	**13703**	0.00	1.45
		20	10	**27619**	13.32	**27619**	0.00	0.73
CCPX13		10	5	**14334**	13.60	**14334**	0.00	1.13
		20	10	**28875**	51.75	**28875**	0.00	0.71
CCPX14		10	5	**13466**	7.77	**13466**	0.00	0.99
		20	10	**27582**	13.96	**27582**	0.00	0.72
CCPX15		10	5	**14343**	10.67	**14343**	0.00	1.18
		20	10	**28917**	10.05	**28917**	0.00	0.71
CCPX16		10	5	**13165**	17.23	**13165**	0.00	2.26
		20	10	**26315**	70.42	26318	0.01	0.72
CCPX17		10	5	**14574**	13.09	**14574**	0.00	2.78
		20	10	**29237**	60.49	29248	0.04	0.72
CCPX18		10	5	**14618**	11.45	**14618**	0.00	1.42
		20	10	**29392**	11.75	**29392**	0.00	0.72
CCPX19		10	5	**13945**	12.51	13946	0.01	2.27
		20	10	**28111**	17.66	**28111**	0.00	0.74
CCPX20		10	5	**14609**	13.37	**14609**	0.00	1.42
		20	10	**29420**	29.74	**29420**	0.00	0.72

Continued on next page

Table 5.4: Continuation.

				CPLEX		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	$t_{avg}(s)$
CCPX21	150	15	5	**16940**	87.80	**16940**	0.00	25.42
		20	10	**43455**	898.78	43458	0.01	7.79
CCPX22		15	5	**17153**	78.45	**17153**	0.00	5.92
		20	10	**44048**	648.11	**44048**	0.00	7.25
CCPX23		15	5	**16089**	77.81	**16089**	0.00	12.43
		20	10	**40457**	741.33	**40457**	0.00	6.98
CCPX24		15	5	**17196**	99.16	**17196**	0.00	5.87
		20	10	**43741**	749.66	**43741**	0.00	7.10
CCPX25		15	5	**16823**	98.90	**16823**	0.00	6.62
		20	10	**42721**	703.69	**42721**	0.00	7.17
CCPX26		15	5	**17398**	228.28	17405	0.04	5.99
		20	10	**44057**	803.07	**44057**	0.00	7.15
CCPX27		15	5	**16608**	63.74	**16608**	0.00	6.40
		20	10	43543	657.55	**43541**	0.00	6.98
CCPX28		15	5	**16628**	77.31	**16628**	0.00	5.39
		20	10	**42071**	874.13	**42071**	0.00	6.85
CCPX29		15	5	**16842**	74.53	**16842**	0.00	19.61
		20	10	**43471**	609.52	**43471**	0.00	7.34
CCPX30		15	5	**16342**	104.93	16343	0.01	8.85
		20	10	**41844**	888.72	**41844**	0.00	6.88
CCPX31	200	20	5	**18844**	1574.28	18845	0.01	81.06
			10	57557	10801.00	**57518**	-0.07	28.71
CCPX32		20	5	**18471**	10800.00	**18471**	0.00	20.86
			10	**56390**	3153.30	56392	0.00	16.48
CCPX33		20	5	**18926**	545.05	18927	0.01	11.44
			10	**58227**	10800.80	**58227**	0.00	13.88
CCPX34		20	5	**18266**	478.33	**18266**	0.00	14.25
			10	56996	2462.55	**56995**	0.00	13.57
CCPX35		20	5	**18949**	142.71	**18949**	0.00	23.45
			10	**59257**	10800.00	**59257**	0.00	20.80
CCPX36		20	5	**17840**	134.00	**17840**	0.00	12.44
			10	**56663**	1693.63	**56663**	0.00	11.80
CCPX37		20	5	**18608**	497.20	**18608**	0.00	38.57
			10	**57759**	2522.29	57760	0.00	17.10
CCPX38		20	5	**18793**	439.09	18794	0.01	49.24
			10	55381	2790.44	**55380**	0.00	36.43
CCPX39		20	5	**18241**	196.84	**18241**	0.00	26.51
			10	**54097**	2204.17	**54097**	0.00	14.18
CCPX40		20	5	**18338**	574.23	**18338**	0.00	13.35
			10	58469	10801.10	**58442**	-0.05	11.52
SJC1	100	10	5	**211622**	4.34	**211622**	0.00	0.93
		20	10	**432101**	6.13	**432101**	0.00	0.67
SJC2	200	15	5	**426525**	781.34	**426525**	0.00	23.37
		20	10	**1042490**	876.84	1344710	28.99	50.62
SJC3a	300	25	5	**576200**	988.68	**576200**	0.00	42.76
			10	1735320	10801.20	**1713450**	-1.26	42.04
SJC3b	300	30	5	**520050**	803.34	520164	0.02	46.68
			10	1537630	10802.40	**1537610**	0.00	45.58
SJC4a	402	30	5	**772352**	2681.67	772681	0.04	122.39
			10	2281640	10800.30	**2274990**	-0.29	72.54
SJC4b	402	40	5	**647064**	2123.86	647342	0.04	136.40
			10	2001770	10800.10	**1969080**	-1.63	112.91

Continued on next page

Table 5.4: Conclusion.

				CPLEX		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	$t_{avg}(s)$
# best		-	-	83	-	76	-	-

From the 92 CMPMP instances scenarios, the PALNS achieved the best solutions costs in 76 of them. In these 76 instances, our method obtained 65 solutions equal to the CPLEX's and improved the best-known solutions in nine instances (negative gap). Considering the others 16 instances where our method did not found the best solutions, the gap in 15 of them was not greater than 0.04%. The outlier was instance *SJC2*, with $p = 20$ and $mc = 10$, with a gap of 28.99%. Besides this case, it can be considered that the PALNS obtained consistent results, as results presented in the previous subsection. The maximum coefficient of variation, related to the 20 runs of our heuristic in all instances, was 0.2% for the solutions costs.

Regarding the computational times, the proposed heuristic was significantly better than the CPLEX in all instances. The PALNS was at least 10 times faster than the commercial solver in 59% of the instances and at least 100 times faster in 21% of them. For example, in instance *CCPX40*, with $p = 20$ and $mc = 10$, our method was 938 times faster than CPLEX and still obtained a gap of -0.05%.

5.4.4 MPCP results

In Table 5.5, the MPCP tests results in the OR-library instances are presented. This table has the same structure than the previous ones and we also tested each instance with two pairs of p and mc values.

Table 5.5 - MPCP results in the OR-library instances (BEASLEY, 1985; BEASLEY, 1990).

				CPLEX		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	$t_{avg}(s)$
pmed1	100	10	5	**651**	148.35	663	1.84	0.35
		20	10	**1286**	181.79	1311	1.94	0.53
pmed2		10	5	**672**	135.68	674	0.30	0.35
		20	10	**1320**	487.84	1325	0.38	0.51
pmed3		10	5	**713**	218.70	719	0.84	0.35
		20	10	**1506**	1.40	**1506**	0.00	0.50
pmed4		10	5	**744**	673.95	750	0.81	0.36
		20	10	**1458**	2508.49	**1474**	1.10	0.52
pmed5		10	5	**607**	16.19	**607**	0.00	0.35
		20	10	**1215**	187.08	1224	0.74	0.49

Continued on next page

Table 5.5: Continuation.

				CPLEX		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	$t_{avg}(s)$
pmed6	200	10	5	454	10800.20	**452**	-0.44	1.06
		20	10	**875**	7729.60	891	1.83	1.47
pmed7		10	5	**418**	10800.20	421	0.72	1.07
		20	10	**819**	10800.00	826	0.85	1.53
pmed8		10	5	**472**	1167.64	474	0.42	1.07
		20	10	**942**	5589.05	943	0.11	1.47
pmed9		10	5	**441**	5929.48	**441**	0.00	1.06
		20	10	**879**	547.02	886	0.80	1.37
pmed10		10	5	**374**	1285.30	377	0.80	1.05
		20	10	**754**	4.85	757	0.40	1.40
pmed11	300	10	5	**311**	10800.30	314	0.96	2.14
		20	10	**608**	8175.55	614	0.99	3.01
pmed12		10	5	**344**	9642.45	345	0.29	2.18
		20	10	**740**	9.71	**740**	0.00	2.79
pmed13		10	5	**323**	10800.20	**323**	0.00	2.17
		20	10	**633**	2769.54	640	1.11	2.86
pmed14		10	5	**356**	5013.15	357	0.28	2.07
		20	10	**736**	78.81	739	0.41	2.33
pmed15		10	5	**311**	10800.30	**311**	0.00	2.19
		20	10	**611**	8254.75	615	0.65	2.96
pmed16	400	10	5	**246**	10800.40	**246**	0.00	3.74
		20	10	**483**	10800.10	485	0.41	5.12
pmed17		10	5	**237**	10800.30	**237**	0.00	3.82
		20	10	**468**	10800.10	471	0.64	4.98
pmed18		10	5	**278**	10800.50	**278**	0.00	3.78
		20	10	**556**	10786.30	560	0.72	4.68
pmed19		10	5	**248**	10801.20	249	0.40	3.90
		20	10	**483**	10800.10	486	0.62	5.45
pmed20		10	5	**269**	10800.40	271	0.74	3.82
		20	10	**529**	10800.10	533	0.76	5.24
pmed21	500	10	5	**213**	10800.50	**213**	0.00	5.81
		20	10	**416**	10800.10	421	1.20	7.72
pmed22		10	5	247	10800.50	**240**	-2.83	5.38
		20	10	**496**	10800.30	499	0.60	6.02
pmed23		10	5	232	10800.30	**230**	-0.86	6.01
		20	10	**449**	10800.10	454	1.11	8.15
pmed24		10	5	**218**	10800.60	**218**	0.00	5.74
		20	10	**431**	10802.40	438	1.62	7.00
pmed25		10	5	**234**	10800.30	235	0.43	5.76
		20	10	**463**	4719.56	466	0.65	7.30
pmed26	600	10	5	201	10801.30	**196**	-2.49	8.28
		20	10	**387**	10800.10	390	0.78	10.50
pmed27		10	5	**194**	10801.30	**194**	0.00	8.05
		20	10	**388**	10800.10	391	0.77	10.60
pmed28		10	5	**238**	571.51	**238**	0.00	7.07
		20	10	**556**	10.92	**556**	0.00	10.13
pmed29		10	5	205	10800.30	**199**	-2.93	8.14
		20	10	**391**	10812.50	**391**	0.00	11.07
pmed30		10	5	215	10800.40	**211**	-1.86	8.01
		20	10	**428**	10837.20	430	0.47	9.30

Continued on next page

Table 5.5: Conclusion.

				CPLEX		PALNS		
Instance	n	p	mc	**cost**	$t(s)$	**cost**	**gap (%)**	$t_{avg}(s)$
pmed31	700	10	5	166	10800.50	**160**	-3.61	11.29
		20	10	**315**	10800.10	**315**	0.00	15.79
pmed32		10	5	**299**	98.35	**299**	0.00	9.14
		20	10	**697**	14.46	**697**	0.00	13.61
pmed33		10	5	179	10800.50	**169**	-5.59	11.14
		20	10	**336**	10800.40	337	0.30	14.01
pmed34		10	5	204	10800.50	**195**	-4.41	10.52
		20	10	**438**	247.54	**438**	0.00	12.12
pmed35	800	10	5	165	10800.80	**159**	-3.64	14.67
		20	10	**317**	10800.20	319	0.63	19.99
pmed36		10	5	**179**	6857.59	181	1.12	13.75
		20	10	**426**	35.04	**426**	0.00	18.36
pmed37		10	5	171	10800.80	**168**	-1.75	14.14
		20	10	**344**	10800.20	345	0.29	16.16
pmed38	900	10	5	177	10801.60	**167**	-5.65	17.57
		20	10	**384**	538.66	**384**	0.00	23.31
pmed39		10	5	**313**	820.39	**313**	0.00	14.99
		20	10	**722**	32.40	**722**	0.00	20.91
pmed40		10	5	175	10801.00	**154**	-12.00	19.06
		20	10	**304**	10800.3	306	0.66	24.04
# best		-	-	66	-	36	-	-

As can be observed from Table 5.5, the PALNS achieved the best results in 36 of 80 the MPCP instances scenarios. From these 36 best results, our method obtained 23 solutions equal to the CPLEX's and improved the best-known solutions in 13 instances (negative gap). Considering the other 44 instances scenarios in which our heuristic did not found the best solutions, the maximum gap observed was 1.94%. However, when taking into account all instances, the PALNS presented an overall average gap of -0.18%. Moreover, the generated solutions are consistent, as the maximum coefficient of variation observed, related to the 20 runs in all instances, was of 0.65%.

The PALNS did not performed in the MPCP, in terms of solution quality, as it performed in the MPMP and CMPMP, because in the MPCP there are several solutions with the same cost. Then, in some instances, our heuristic got stuck in some plateau and could not found an optimal solution, only local ones.

Regarding the execution time, as results presented in the previous subsections, the PALNS performance was significantly better than CPLEX's. In 83% of the instances scenarios, our method was at least 100 times faster than CPLEX and at least 1000 times faster in 55% of them. For example, in instance *pmed40*, with $p = 10$ and $mc = 5$, the PALNS improved the CPLEX solution in 12%, taking 568 times less

computational time.

5.5 Conclusions and future directions

An effective parallel adaptive large neighborhood search heuristic was proposed for facility location problems with multiple assignments in this work. We applied the PALNS in the multiple p-median problem, in the capacitated multiple p-median problem and in the multiple p-center problem.

The computational tests results show that our heuristic can generate optimal solutions at a low computational cost in all problems tested. Specifically, the PALNS found 79 optimal solutions of the 80 instances scenarios spending, on average, nearly 19% times less execution time than the commercial solver. In addition, our heuristic outperformed the BIMM algorithm in all instances.

In the CMPMP context, the PALNS found the best solution in 76 of the 92 instances. From these 76 best solutions costs, our method could outperform the CPLEX in nine of them. The gap between the PALNS solutions and the CPLEX solutions in 15 of the 16 remaining instances was at most 0.04%. In CMPMP experiments, the proposed heuristic was, on average, 97 times faster than the CPLEX.

Regarding the MPCP tests, the PALNS generated the best solutions in 36 of the 80 instances. Indeed, 13 of these 36 solutions were better than the CPLEX'S. The maximum gap observed in the remaining 44 instances was 1.94%. On average, the PALNS was nearly 1007 times faster than the commercial solver in the MPCP tests.

However, further investigation needs to be done to improve the results in all the problems. In the MPMP, parameters should be adjusted to reduce the execution time on instances with 100 vertices. For the CMPMP and MPCP, more operators should be implemented in order to improve the solutions. Furthermore, an alternative MPCP objective function should be used to add more information to the solution cost.

6 CONCLUSIONS AND FUTURE DIRECTIONS

Four new overlapping clustering problems variations, namely overlapping cluster editing problem, p-median problem with overlap control, capacitated multiple p-median problem and the multiple p-center problem were proposed in this work. These modifications were introduced to deal with non-disjoint clusters that arise from real-world applications since the original problems do not handle it. Efficient methods were proposed to solve all these new problems. Furthermore, a hybrid heuristic was introduced for the overlapping community detection problem. In all contexts, the solution algorithms could generate good quality solutions.

In Chapter 2, three versions of a hybrid heuristic were proposed to solve the OCEP. These hybrid heuristics are composed by coupling two metaheuristics with a MILP. The HHM1 presented the best results in the cluster editing cost, execution time and controlling the overlap between the clusters.

With the good results obtained by the first overlapping clustering MILP from the Chapter 2, we decided to apply it to the overlapping community detection problem. Then, a hybrid heuristic was introduced to solve the OCDP by conductance minimization. Two state-of-art OCDP methods from the literature were implemented to generate a set of input clusters to the MILP. Also, local search procedures were used to improve the solutions found by the hybrid heuristic. Experimental tests indicate that the proposed method can detect overlapping clusters with better overall conductance than some state-of-art algorithms.

Driven by the good results achieved by the overlapping clustering model in both Chapters 2 and 3, the OCM was applied to the context of the p-median problem. Then, the PMPOC was presented in Chapter 4 and a parallel branch-and-price algorithm to solve it was developed. Through a series of computational tests, it was shown that the model can control the overlap between the facilities and the proposed B&P can find good-quality solutions.

Following the work developed in the Chapter 4, another study in the facility location context was done in Chapter 5. In this chapter, a parallel adaptive large neighborhood search was introduced to solve the MPMP, the CMPMP and the MPCP, which the two later ones were proposed in this work. The PALNS obtained the best known solutions in 76% of the instances at a low execution time.

These contributions can be considered useful to overlapping clustering problems

since new problems variations were introduced to allow clusters to overlap. In addition, it was shown that the OCM is suitable to different overlapping clustering contexts as it was successfully applied to the OCEP, to the OCDP and to the PM-POC.

For future work, the OCM could be applied to another facility location problems, such as the CPMP and the PCP. In this sense, the proposed B&P could be adapted to these problems contexts. Other overlapping clustering problems should be explored as well. In addition, the PALNS could also be applied to another facility location problems with multiple assignment, e.g., the capacitated multiple p-center problem.

REFERENCES

ADENSO-DÍAZ, B.; LAGUNA, M. Fine-tuning of algorithms using fractional experimental designs and local search. **Operations Research**, v. 54, n. 1, p. 99–114, 2006. xvii, 17, 19, 98

AGGARWAL, C. C. An introduction to cluster analysis. In: AGGARWAL, C. C.; REDDY, C. K. (Ed.). **Data clustering algorithms and applications**. Boca Raton, FL, USA: CRC Press, 2013. chapter 1, p. 1–27. 1

AHN, Y.-Y.; BAGROW, J. P.; LEHMANN, S. Link communities reveal multiscale complexity in networks. **Nature**, v. 466, n. 1, p. 761–764, 2010. 8

ALGHAMDI, E.; GREENE, D. Active semi-supervised overlapping community finding with pairwise constraints. **Applied Network Science**, v. 4, n. 63, p. 27, 2019. 38, 39

ALJARAH, I.; MAFARJA, M.; HEIDARI, A. A.; FARIS, H.; ZHANG, Y.; MIRJALILI, S. Asynchronous accelerating multi-leader salp chains for feature selection. **Applied Soft Computing**, v. 71, p. 964–979, 2018. 10

AMIGÓ, E.; GONZALO, J.; ARTILES, J.; VERDEJO, F. A comparison of extrinsic clustering evaluation metrics based on formal constraints. **Information Retrieval**, v. 12, n. 4, p. 461–486, 2009. 25

ANDERSEN, R.; CHUNG, F.; LANG, K. Local graph partitioning using pagerank vectors reid. In: ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 47., 2006. **Proceedings...** Berkeley, CA, USA: IEEE, 2006. p. 475–486. 40, 47, 48, 49

ANDERSEN, R.; GLEICH, D.; MIRROKNI, V. Overlapping clusters for distributed computation. In: ACM INTERNATIONAL CONFERENCE ON WEB SEARCH AND DATA MINING, 5., 2012. **Proceedings...** Seattle, USA: ACM, 2012. p. 273–282. 1

ANDRADE, C. E.; RESENDE, M. G. C.; KARLOFF, H. J.; MIYAKAWA, F. K. Evolutionary algorithms for overlapping correlation clustering. In: CONFERENCE ON GENETIC AND EVOLUTIONARY COMPUTATION, 2014. **Proceedings...** Vancouver, Canada: ACM, 2014. p. 405–412. 10, 15

ARAÚJO, E. J.; CHAVES, A. A.; LORENA, L. A. N. A mathematical model for the coverage location problem with overlap control. **Computers & Industrial Engineering**, v. 146, p. 106548, 2020. 63, 64, 65, 87

ATAY, Y.; KOC, I.; BABAOGLU, I.; KODAZ, H. Community detection from biological and social networks: a comparative analysis of metaheuristic algorithms. **Applied Soft Computing**, v. 50, n. 1, p. 194–211, 2017. 10

AVCI, M. G.; AVCI, M. An adaptive large neighborhood search approach for multiple traveling repairman problem with profits. **Computers & Operations Research**, v. 111, p. 367–385, 2019. 88

AVELLA, P.; SASSANO, A.; VASIL'EV, I. Computational study of large-scale p-median problems. **Mathematical programming (Ser. A)**, v. 109, p. 89–114, 2007. 66

BAADEL, S.; THABTAH, F.; LU, J. Mcoke: Multi-cluster overlapping k-means extension algorithm. **International Journal of Computer and Information Engineering**, v. 9, p. 427–430, 2015. 7

_____. Overlapping clustering: a review. In: SAI COMPUTING CONFERENCE. **Proceedings...** London, UK: IEEE, 2016. p. 233–237. 7

BALDACCI, R.; HADJICONSTANTINOU, E.; MANIEZZO, V.; MINGOZZI, A. A new method for solving capacitated location problems based on a set partitioning approach. **Computers and Operations Research**, v. 54, n. 4, p. 365–386, 2002. xviii, 98, 99, 103, 104

BANSAL, N.; BLUM, A.; CHAWLA, S. Correlation clustering. **Machine Learning**, v. 56, n. 1, p. 89–113, 2004. 1, 5, 9

BARBAROS, C. T.; RICHARD, L. F.; TIMOTHY, J. L. State of the art—location on networks: a survey. part i: the p-center and p-median problems. **Management Science**, v. 29, n. 4, p. 482–497, 1983. 64, 65, 87

BARRENA, E.; ORTIZ, D. C.; COELHO, L. C.; LAPORTE, G. Single-line rail rapid transit timetabling under dynamic passenger demand. **Transportation Research Part B**, v. 70, p. 134–50, 2014. 95

BARTHÉLEMY, J.-P.; BRUCKER, F. Np-hard approximation problems in overlapping clustering. **Journal of Classification**, v. 18, n. 2, p. 159–183, 2001. 8, 9

BASTOS, L.; OCHI, L. S.; PROTTI, F.; SUBRAMANIAN, A.; MARTINS, I. C.; PINHEIRO, R. G. S. Efficient algorithms for cluster editing. **Journal of Combinatorial Optimization**, v. 31, n. 1, p. 347–371, 2016. 5, 10, 17, 25, 36, 129

BEASLEY, J. E. A note on solving large p-median problems. **European Journal of Operational Research**, v. 21, p. 270–273, 1985. xviii, xix, 78, 81, 82, 98, 99, 101, 106, 139

_____. Or-library: distributing test problems by electronic mail. **Journal Operational Research Society**, v. 41, n. 11, p. 1069–1072, 1990. xviii, xix, 78, 81, 82, 98, 99, 101, 106, 139

BEN-DOR, A.; SHAMIR, R.; YAKHIMI, Z. Clustering gene expression patterns. **Journal of Computer Biology**, v. 6, n. 3-4, p. 281–297, 1999. 1, 5

BEN N'CIR, C.-E.; CLEUZIOU, G.; ESSOUSSI, N. Overview of overlapping partitional clustering methods. In: CELEBI, M. E. (Ed.). **Proceedings...** Cham, Switzerland: Springer International Publishing, 2015. chapter 8, p. 245–275. 7, 8

BENELALLAM, A.; CUADRADO, J. S.; CABOT, J. Efficient model partitioning for distributed model transformations. In: INTERNATIONAL CONFERENCE ON SOFTWARE LANGUAGE ENGINEERING, 2016. **Proceedings...** Amsterdam, Netherlands: ACM SIGPLAN, 2016. p. 226–238. 1

BENNETT, L.; KITTAS, A.; LIU, S.; PAPAGEORGIOU, L. G.; TSOKA, S. Community structure detection for overlapping modules through mathematical programming in protein interaction networks. **PLoS ONE**, v. 9, n. 11, p. e112821, 2014. 38

BERTRAND, P.; JANOWITZ, M. F. The k-weak hierarchical representations: an extension of the indexed closed weak hierarchies. **Discrete Applied Mathematics**, v. 127, p. 199–220, 2003. 7

BÖCKER, S.; BAUMBACH, J. Cluster editing. In: BONIZZONI, P.; BRATTKA, V.; LÖWE, B. (Ed.). **CiE 2013 LNCS**. Heidelberg: Springer, 2013. v. 7921, p. 33–44. 5

BÖCKER, S.; BRIESEMEISTER, S.; BUI, Q. B. A.; TRUSS, A. Going weighted: parameterized algorithms for cluster editing. **Theoretical Computer Science**, v. 410, n. 52, p. 5467–5480, 2009. 1

BÖCKER, S.; BRIESEMEISTER, S.; KLAU, G. W. Exact algorithms for cluster editing: evaluation and experiments. **Algorithmica**, v. 60, n. 2, p. 316–334, 2011. 5

BONCHI, F.; GIONI, A.; UKKONEN, A. Overlapping correlation clutering. In: IEEE INTERNATIONAL CONFERENCE ON DATA MINING, 11., 2011. **Proceedings...** Vancouver,BC, Canada: IEEE, 2011. p. 51–60. 1, 8, 9

_____. _____. **Knowledge and Information Systems**, v. 35, n. 1, p. 1–32, 2013. 1, 8, 9, 11, 37

C. CORDEAU J-F, L. G. C. L. Consistency in multi-vehicle inventory-routing. **Transportation Research Part C: Emerging Technologies**, v. 24, p. 270–287, 2012. 95

_____. The inventory-routing problem with transshipment. **Computers & Operations Research**, v. 39, p. 2537–2548, 2012. 95

CALVO, B.; SANTAFÉ, G. scmamp: statistical comparison of multiple algorithms in multiple problems. **The R Journal**, v. 8, n. 1, p. 248–256, 2016. 28

CANISIUS, S.; MARTENS, J.; WESSELS, L. A novel independence test for somatic alterations in cancer shows that biology drives mutual exclusivity but chance explains most co-occurrence. **Genome Biology**, v. 17, n. 1, p. 261:1–17, 2016. 9

CESELLI, A.; RIGHINI, G. A branch-and-price algorithm for the capacitated p-median problem. **Networks**, v. 45, n. 3, p. 125–142, 2005. 66

CHAGAS, G. O.; LORENA, L. A. N.; SANTOS, R. D. C. dos. A hybrid heuristic for the overlapping cluster editing problem. In: BRAZILIAN SYMPOSIUM ON OPERATIONS RESEARCH. **Proceedings...** Rio de Janeiro, Brazil: SOBRAPO, 2018. p. 83867. 42

_____. _____. **Applied Soft Computing**, v. 81, p. 105482, 2019. 1, 2, 5, 37, 38, 39, 41, 42, 43, 49, 50, 51, 61, 65, 67, 68, 87

CHANG, F.; ZHANG, B.; LI, H.; HUANG, M.; LI, B.; ZHAO, Y. Discovering overlapping communities in ego-nets using friend intimacy. **Journal of Intelligent & Fuzzy Systems**, v. 36, n. 6, p. 5167–5175, 2019. 38, 39, 42

CHARIKAR, M.; GURUSWAMI, V.; WIRTH, A. Clustering with qualitative information. **Journal of Computer and System Sciences**, v. 71, n. 3, p. 360–383, 2005. xv, xix, 11, 31, 32, 129, 130, 131

CHESLER, E. J.; LU, L.; SHOU, S.; QU, Y.; GU, J.; WANG, J.; HSU, H. C.; MOUNTZ, J. D.; BALDWIN, N. E.; LANGSTON, M. A.; THREADGILL, D. W.; MANLY, K. F.; WILLIAMS, R. W. Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. **Nature Genetics**, v. 37, n. 3, p. 233–242, 2005. 5

CHIERICHETTI, F.; DALVI, N.; KUMAR, R. Correlation clustering in mapreduce. In: INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, 2014. **Proceedings...** New York, USA: ACM, 2014. p. 641–650. 5

CHRISTOFIDES, N.; BEASLEY, J. A tree search algorithm for the p-median problem. **European Journal of Operational Research**, v. 10, p. 196–204, 1982. 79

CONTARDO, C.; IORI, M.; KRAMER, R. A scalable exact algorithm for the vertex p-center problem. **Computers and Operations Research**, v. 103, p. 211–220, 2019. 87

DAMASCHKE, P. Fixed-parameter enumerability of cluster editing and related problems. **Theory of Computing Systems**, v. 46, n. 2, p. 261–283, 2010. 8, 9

DAMASCHKE, P.; MOGREN, O. Editing simple graphs. **Journal of Graphs Algorithms and Applications**, v. 18, n. 4, p. 557–576, 2014. 5

DANTZIG, G. B.; P., W. Decomposition principle for linear programs. **Operations Research**, v. 8, n. 1, p. 101–111, 1960. 70

DAS, S.; ABRAHAM, A.; KONAR, A. **Metaheuristic clustering**. Chennai, India.: Springer-Verlag Berlin Heidelberg, 2009. 340 p. 10

DASKIN, M. **Network and discrete location: models, algorithms, and applications**. New York: Wiley, 1995. 91

DASKIN, M. S.; MAASS, K. L. The p-median problem. In: LAPORTE, G.; NICKEL, S.; GAMA, F. S. da (Ed.). **Location science**. Cham: Springer International, 2015. p. 21–45. 64, 65, 87

DEHNE, F.; LANGSTON, M. A.; LUO, X.; PITRE, S.; SHAW, P.; ZHANG, Y. The cluster editing problem: Implementations and experiments. In: BODLAENDER, H. L.; LANGSTON, M. A. (Ed.). **Parameterized and exact computation: second international workshop**. Zürich, Switzerland: Springer Berlin Heidelberg, 2006. p. 13–24. 6

DELVAUX, S.; HORSTEN, L. On best transitive approximations to simple graphs. **Acta Informatica**, v. 40, n. 9, p. 637–655, 2004. 5

DEMAINE, E. D.; EMANUEL, D.; FIAT, A.; IMMORLICA, N. Correlation clustering in general weighted graphs. **Theoretical Computer Science**, v. 361, n. 2-3, p. 172–187, 2006. 1

DENG, C.; DENG, H.; LIU, Y. Detection of microblog overlapping community based on multidimensional information and edge distance matrix. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE AND SECURITY, 2019. **Proceedings...** Hohhot, China: Springer, 2019. p. 121–136. 42

DHILLON, I. S.; GUAN, Y.; KULIS, B. Weighted graph cuts without eigenvectors: a multilevel approach. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 29, n. 11, p. 1944–1957, 2007. 47

ESQUIVEL, A. V.; ROSVALL, M. Comparing network covers using mutual information. **ArXiv**, abs/1202.0425, 2012. 57

FARRIS, H.; AL-ZOUBI, A. M.; HEIDARI, A. A.; ALJARAH, I.; MAFARJA, M.; HASSONAH, M. A.; FUJITA, H. An intelligent system for spam detection and identification of the most relevant features based on evolutionary random weight networks. **Information Fusion**, v. 48, p. 67–83, 2019. 10

FARRIS, H.; MAFARJA, M. M.; HEIDARI, A. A.; ALJARAH, I.; AL-ZOUBI, A. M.; MIRJALILI, S.; FUJITA, H. An efficient binary salp swarm algorithm with crossover scheme for feature selection problems. **Knowledge-Based Systems**, v. 154, p. 43–67, 2018. 10

FELLOWS, M. R.; GUO, J.; KOMUSIEWICZ, C.; NIEDERMEIER, R.; UHLMANN, J. Graph-based data clustering with overlaps. In: INTERNATIONAL CONFERENCE ON COMPUTING AND COMBINATORICS, 2009. **Proceedings...** Niagara Falls, USA: Springer, 2009. v. 5609, p. 516–526. 5, 8, 9

_____. _____. **Discrete Optimization**, v. 8, n. 1, p. 2–17, 2011. 5, 6, 8, 9

FILHO, G. F. d. S.; CABRAL, L. d. A. F.; OCHI, L. S.; PROTTI, F. Hybrid metaheuristic for bicluster editing problem. **Electronic Notes in Discrete Mathematics**, v. 39, p. 35–42, 2012. 10

FLESZAR, K.; HINDI, K. S. An effective vns for the capacitated p-median problem. **European Journal of Operational Research**, v. 191, p. 612–622, 2008. 91, 97, 98

FORTUNATO, S. Community detection in graphs. **Physics Reports**, v. 486, n. 3-5, p. 75–174, 2010. 37, 41

FORTUNATO, S.; HRIC, D. Community detection in networks: a user guide. **Physics Reports**, v. 659, p. 1–44, 2016. 37, 38, 40

FU, Q.; BANERJEE, A. Multiplicative mixture models for overlapping clustering. In: EIGHTH IEEE INTERNATIONAL CONFERENCE ON DATA MINING. **Proceedings...** Pisa, Italy: IEEE, 2008. p. 791–796. 7

GAO, Y.; ZHANG, H.; ZHANG, Y. A fast and high quality approach for overlapping community detection through minimizing conductance. In: INTERNATIONAL CONFERENCE ON DATA SCIENCE IN CYBERSPACE (DSC), 1., 2016. **Proceedings...** Changsha, China: IEEE, 2016. p. 688–693. 43, 51, 52

_____. Overlapping communities from lines and triangles in complex networks. **Physica A**, v. 521, p. 455–466, 2019. 38, 39, 43, 51, 56

_____. Overlapping community detection based on conductance optimization in large-scale networks. **Physica A**, v. 522, p. 69–79, 2019. xviii, 38, 39, 40, 42, 43, 51, 53, 54, 55, 56, 57

GARCÍA, S.; FERNZÁNDEZ, A.; LUENGO, J.; HERRERA, F. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power. **Information Sciences**, v. 180, n. 10, p. 2044–2064, 2010. 28

GAREY, M. R.; JOHNSON, D. S. **Computers and intractability: a guide to the theory of NP-completeness**. San Francisco, USA: W. H. Freeman, 1979. 340 p. 88

GARFINKEL, R. S.; NEEBE, W.; RAO, M. R. An algorithm for the m-median location problem. **Transportation Science**, v. 8, n. 3, p. 217–236, 1974. 70

GARÍA, S.; LABBÉ, M.; MARÍN, A. Solving large p-median problems with a radius formulation. **INFORMS Journal on Computing**, v. 23, n. 4, p. 546–556, 2011. 66

GIRVAN, M.; NEWMAN, M. E. J. Community structure in social and biological networks. **Proceedings of National Academy of Sciences**, v. 99, n. 12, p. 7821–7826, 2002. 37

GLOVER, F.; KOCHENBERGER, G. A. **Handbook of metaheuristics**. Dordrecht, Netherlands: Kluwer Academic Publishers, 2003. 557 p. 15

GONÇALVES, J. F.; RESENDE, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, v. 17, n. 5, p. 487–525, 2011. 10, 12, 14, 15

GRANGIER, P.; GENDREAU, M.; LEHUÉDÉA, F.; ROUSSEAU, L.-M. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. **European Journal of Operations Research**, v. 254, n. 1, p. 80–91, 2016. 87

GÜDEN, H.; HALDUN, S. The dynamic p-median problem with mobile facilities. **Computers & Industrial Engineering**, v. 135, p. 615–627, 2019. 66

GUO, J.; KANJ, I. A.; KOMUSIEWICZ, C.; UHLMANN, J. Editing graphs into disjoint unions of dense clusters. **Algorithmica**, v. 61, n. 4, p. 949–970, 2011. 8

GUO, J.; KOMUSIEWICZ, C.; NIEDERMEIER, R.; UHLMANN, J. A more relaxed model for graph-based data clustering: s-plex editing. In: GOLDBERG, A. V.; ZHOU, Y. (Ed.). **Algorithmic aspects in information and management: 5th International Conference**. San Francisco, CA, USA: Springer, 2009. p. 226–239. 1

_____. A more relaxed model for graph-based data clustering: s-plex cluster editing. **SIAM Journal on Discrete Mathematics**, v. 24, n. 4, p. 1662–1683, 2010. 8

HAKIMI, S. L. Optimum locations of switching centers and the absolute centers and medians of a graph. **Operations Research**, v. 12, n. 3, p. 450–459, 1964. 63, 87

_____. Optimal distribution of switching centers in a communications network and some related graph-theoretic problems. **Operations Research**, v. 13, p. 462–475, 1965. 88

HAMMAMI, F.; REKIK, M.; COELHO, L. C. A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. **Computers & Operations Research**, v. 123, 2020. 88

HANSEN, P.; BRIMBERG, J.; UROŠEVIĆ, D.; MLADENOVIĆ, N. Solving large p-median clustering problems by primal–dual variable neighborhood search. **Data Mining and Knowledge Discovery**, v. 19, p. 351–375, 2009. 87

HEGGERNES, P.; LOKSHTANOV, D.; NEDERLOF, J.; PAUL, C.; TELLE, J. A. Generalized graph clustering: recognizing (p,q)-cluster graphs. In: GRAPH THEORETIC CONCEPTS IN COMPUTER SCIENCE, 36., 2010. **Proceedings...** Crete, Greece: Springer, 2010. p. 171–183. 8

HEIDARI, A. A.; ABBASPOUR, R. A.; JORDEHI, A. R. An efficient chaotic water cycle algorithm for optimization tasks. **Neural Computing and Applications**, v. 28, p. 57–85, 2017. 10

HEIDARI, A. A.; PAHLAVANI, P. An efficient modified grey wolf optimizer with Lévy flight for optimization tasks. **Applied Soft Computing**, v. 60, p. 115–134, 2017. 10

HOLLAND, J. H. **Adaptation in natural and artificial systems**. Cambridge, England: MIT Press, 1975. 205 p. 15

HOPCROFT, J.; TARJAN, R. Algorithm 447: efficient algorithms for graph manipulation. **Communications of the ACM**, v. 16, n. 6, p. 372–378, 1973. 47

HRUSCHKA, E. R.; CAMPELLO, R. J. G. B.; FREITAS, A. A.; CARVALHO, A. C. P. L. F. d. A survey of evolutionary algorithms for clustering. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 39, p. 133–155, 2009. 10, 15, 17

IBM Corporation. **IBM ILOG CPLEX optimization studio V12.7.0 documentation**. 2017. 12, 29, 31, 33, 35

_____. **IBM ILOG CPLEX optimization studio V12.8.0 documentation**. 2018. 24, 129

_____. **IBM ILOG CPLEX Optimization Studio V12.9.0 documentation**. 2019. 43, 55

_____. **IBM ILOG CPLEX optimization studio V12.10.0 documentation**. 2020. 99

IMANE, M.; NADJET, K. Hybrid bat algorithm for overlapping community detection. In: IFAC CONFERENCE ON MANUFACTURING MODELLING, 8., 2016. **Proceedings...** Troyes, France: Elsevier, 2016. p. 1454–1459. 10

JACCARD, P. The distribution of the flora in the alpine zone. **The New Phytologist**, v. 11, n. 2, p. 37–50, 1912. 64, 68

JIANG, D.; PEI, J. Mining frequent cross-graph quasi-cliques. **ACM Transactions on Knowledge Discovery from Data**, v. 2, n. 4, p. 16:1–42, 2009. 5

JOSÉ-GARCÍA, A.; GÓMEZ-FLORES, W. Automatic clustering using nature-inspired metaheuristics: a survey. **Applied Soft Computing**, v. 41, n. 4, p. 192–213, 2016. 10

KARATAS, M.; RAZI, N.; TOZAN, H. A comparison of p-median and maximal coverage location models with q-coverage requirement. **Procedia Engineering**, v. 149, p. 169–176, 2016. 65, 87

KARIV, O.; HAKIMI, S. L. An algorithmic approach to network location problems. ii: the p-medians. **SIAM Journal on Applied Mathematics**, v. 37, n. 3, p. 539–560, 1979. 63, 88

KHANMOHAMMADI, S.; ADIBEIG, N.; SHANEHBANDY, S. An improved overlapping k-means clustering method for medical applications. **Expert Systems with Applications**, v. 67, p. 12–18, 2017. 2

KIRKPATRICK, S.; GELATT, C. D.; VECCH, M. P. Optimization by simulated annealing. **Science**, v. 220, n. 4598, p. 671–680, 1983. 12, 14, 17, 93

KOMUSIEWICZ, C.; UHLMANN, J. Cluster editing with locally bounded modifications. **Discrete Applied Mathematics**, v. 160, n. 15, p. 2259–2270, 2012. 5

KUMPULA, J. M.; KIVELÄ, M.; KASKI, k.; SARAMÄKI, J. Sequential algorithm for fast clique percolation. **Physical Review E**, v. 74, p. 026109 (7pp.), 2008. 8

LAHYANI, R.; GOUGUENHEIM, A.-L.; COELHO, L. C. A hybrid adaptive large neighbourhood search for multi-depot open vehicle routing problems. **International Journal of Production Research**, v. 57, p. 6963–6976, 2019. 88, 95

LANCICHINETTI, A.; FORTUNATO, S. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. **Physical Review E**, v. 80, n. 1, p. 016118, 2009. xviii, 39, 56, 57

_____. _____. **Physical Review E**, v. 80, n. 1, p. 016118, 2009. 25, 35

LANCICHINETTI, A.; FORTUNATO, S.; KERTÉSZ. Detecting the overlapping and hierarchical community structure in complex networks. **New Journal of Physics**, v. 11, p. 033015 (18pp), 2009. 8, 42, 43, 44, 45, 46, 55

LANCICHINETTI, A.; RADICCHI, F.; RAMASCO, J. J.; FORTUNATO, S. Finding statistically significant communities in networks. **PLoS ONE**, v. 6, n. 4, p. e18961, 2011. 39, 42

LESKOVEC, J.; KREVL, A. **SNAP Datasets: Stanford large network dataset collection**. 2014. Available from: <http://snap.stanford.edu/data>. 56, 60

LEVIN, M. S. Combinatorial clustering: literature review, methods, examples. **Journal of Communications Technology and Eletronics**, v. 60, n. 12, p. 1403–1428, 2015. 7, 8

LI, T.; PINTADO, F. De la P.; CORCHADO, J. M.; BAJO, J. Multi-source homogeneous data clustering for multi-target detection from cluttered background with misdetection. **Applied Soft Computing Journal**, v. 60, p. 436–446, 2017. 1

LIU, H.; ZHANG, P.; ZHU, D. On editing graphs into 2-club clusters. In: FRONTIERS IN ALGORITHMICS AND ALGORITHMIC ASPECTS IN INFORMATION AND MANAGEMENT: JOINT INTERNATIONAL CONFERENCE, 2012. **Proceedings...** Beijing, China: Springer, 2012. p. 235–246. 8

LORENA, L. A. N.; SENNE, E. L. F. Local search heuristics for capacitated p-median problems. **Networks and Spatial Economics**, v. 3, p. 407–419, 2003. xviii, 99, 103, 104

_____. A column generation approach to capacitated p-median problems. **Computers & Operations Research**, v. 31, n. 6, p. 863–876, 2004. xviii, 65, 70, 73, 99, 103, 104

LORENA, L. H. N.; QUILES, M. G.; CARVALHO, A. C. P. d. L. F. de; LORENA, L. A. N. Preprocessing technique for cluster editing via integer linear programming. In: HUANG, D.-S.; BEVILACQUA, V.; PREMARATNE, P.; GUPTA, P. (Ed.). **Intelligent computing theories and application**. Bengaluru, India: Springer, 2018. p. 287–297. 5

MAFARJA, M.; ALJARAH, I.; FARIS, H.; HAMMOURI, A. I.; AL-ZOUBI, A. M.; MIRJALILI, S. Binary grasshopper optimisation algorithm approaches for feature selection problems. **Expert Systems with Applications**, v. 117, n. 1, p. 267–286, 2019. 10

MAFARJA, M.; ALJARAH, I.; HEIDARI, A. A.; FARIS, H.; FOURNIER-VIGER, P.; LI, X.; MIRJALILI, S. Binary dragonfly optimization for feature selection using time-varying transfer functions. **Knowledge-Based Systems**, v. 161, n. 1, p. 185–204, 2018. 10

MAIZA, M. I.; N'CIR, C.-E. B.; ESSOUSSI, N. Overlap regulation for additive overlapping clustering methods. In: INTERNATIONAL CONFERENCE ON RESEARCH CHALLENGES IN INFORMATION SCIENCE, 10., 2016. **Proceedings...** Grenoble, France: IEEE, 2016. p. 1–6. 1, 6

MANIEZZO, V.; STÜTZLE, T.; VOSS, S. **Matheuristics: hybridizing metaheuristics and mathematical programming**. New York, USA: Springer, 2009. 269 p. 7, 38

MARÍN, A.; PELEGRÍN, M. p-median problems. In: LAPORTE, G.; NICKEL, S.; GAMA, F. S. da (Ed.). **Location science**. Cham: Springer International Publishing, 2019. p. 25–50. 64, 65, 87

MASUYAMA, S.; IBARAKI, T.; HASEGAWA, T. The computational complexity of the m-center problems on the plane. **Transactions of the Institute of Electronics and Communication Engineers of Japan. Section E**, E64, n. 2, p. 57–64, 1981. 88

McDaid, A.; HURLEY, N. Detecting highly overlapping communities with model-based overlapping seed expansion. In: INTERNATIONAL CONFERENCE ON ADVANCES IN SOCIAL NETWORKS ANALYSIS AND MINING, 2010. **Proceedings...** Odense, Denmark: IEEE, 2010. p. 112–119. 39, 42

METROPOLIS, N.; ROSENBLUTH, A. W.; ROSENBLUTH, M. N.; TELLER, A. H.; TELLER, E. Equation of state calculations by fast computing machines. **Journal of Chemical Physics**, v. 21, p. 1087–1092, 1953. 17, 19

MILLER, I. F.; BECKER, A. D.; GRENFELL, B. T.; METCALF, C. J. E. Disease and healthcare burden of covid-19 in the united states. **Nature Medicine**, 2020. 64

MLADENOVIĆ, N.; BRIMBERG, J.; HANSEN, P.; Moreno-Pérez, J. A. The p-median problem: a survey of metaheuristic approaches. **European Journal of Operational Research**, v. 179, n. 3, p. 927–939, 2007. 64, 65, 87

MLADENOVIĆ, N.; LABBĆ, M.; HANSEN, P. Solving the p-center problem with tabu search and variable neighborhood search. **Networks**, v. 42, n. 1, p. 48–64, 2003. 96

MOUSSAVI, S. E.; MAHDJOUB, M.; GRUNDER, O. A matheuristic approach to the integration of worker assignment and vehicle routing problems: application to home healthcare scheduling. **Expert Systems With Applications**, v. 125, p. 317–332, 2019. 38

MULVEY, J. M.; BECK, M. P. Solving capacitated clustering problems. **European Journal of Operational Research**, v. 18, p. 339–348, 1984. 91

NANDA, S. J.; PANDA, G. A survey on nature inspired metaheuristic algorithms for partitional clustering. **Swarm and Evolutionary Computation**, v. 16, p. 1–18, june 2014. 10

NATANZON, A.; SHAMIR, R.; SHARAN, R. Complexity classification of some edge modification problems. **Discrete Applied Mathematics**, v. 113, n. 1, p. 109–128, 2001. 5

NEPUSZ, T.; PETRÓCZI, A.; NÉGYESSY, L.; BAZSÓ, F. Fuzzy communities and the concept of bridgeness in complex networks. **Physical Review E**, v. 77, p. 016107 (12pp.), 2008. 8

NG, R.; HAN, J. Efficient and effective clustering methods for spatial data mining. In: CONFERENCE VERY LARGE DATABASES (VLDB), 20., 1994. **Proceedings...** San Francisco, USA: ACM, 1994. p. 144–155. 87

OLIVEIRA, R. M.; CHAVES, A. A.; LORENA, L. A. N. A comparison of two hybrid methods for constrained clustering problems. **Applied Soft Computing Journal**, v. 54, p. 256–266, 2017. 36, 38

OLIVEIRA, R. M.; LORENA, L. A. N.; CHAVES, A. A.; MAURI, G. R. Hybrid heuristics based on column generation with path-relinking for clustering problems. **Expert Systems with Applications**, v. 41, p. 5277–5284, 2014. 38

OPTIMIZATION, L. G. **Gurobi optimizer reference manual**. 2020. Available from: <<http://www.gurobi.com>>. 78

OSMAN, I. H.; CHRISTOFIDES, N. Capacitated clustering problems by hybrid simulated annealing and tabu search. **International Transactions in Operational Research**, v. 1, p. 317–336, 1994. xviii, 99, 103, 104

PADROL-SUREDA, A.; PERARNAU-LLOBET, G.; PFEIFLE, J.; MUNTÉS-MULERO, V. Overlapping community search for social networks. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 26. **Proceedings...** Long Beach, CA, USA: IEEE, 2010. p. 992–995. 42

PAGE, L.; BRIN, S.; MOTWANI, R.; WINOGRAD, T. **The PageRank citation ranking: bringing order to the web**. 1999. 1999-66. 47

PALLA, G.; DERÉNYI, I.; FARKAS, I.; VICSEK, T. Uncovering the overlapping community structure of complex networks in nature and society. **Nature**, v. 435, p. 814–818, 2005. 1

PALLA, G.; DERENYI, I.; FARKAS, I.; VICSEK, T. Uncovering the overlapping community structures of complex networks in nature and society. **Nature**, v. 435, n. 7043, p. 814–818, 2005. 39

PANTELI, A.; BOUTSINAS, B.; GIANNIKOS, L. On solving the multiple p-median problem based on biclustering. **Operational Research**, 2019. 63, 65, 87, 88, 89, 99, 101

PARK, J. S.; LIM, B. H.; LEE, Y. A lagrangian dual-based branch-and-bound algorithm for the generalized multi-assignment problem june. **Management Science**, v. 44, n. 12, p. S271–S282, 1998. 91

PEREIRA, M. A.; COELHO, L. C.; LORENA, L. A. N.; SOUZA, L. C. A hybrid method for the probabilistic maximal covering location-allocation problem. **Computers & Operations Research**, v. 57, n. Suppl. C, p. 51–59, 2015. 7, 38, 88, 95

PÉREZ-SUÁREZ, A.; MARTÍNEZ-TRINIDAD, J. F.; CARRASCO-OCHOA, J. A.; MEDINA-PAGOLA, J. E. Oclustr: a new graph-based algorithm for overlapping clustering. **Neurocomputing**, v. 121, n. 9, p. 234–247, 2013. 1, 7

PISINGER, D.; ROPKE, S. A general heuristic for vehicle routing problems. **Computers & Operations Research**, v. 34, p. 2403–2435, 2007. 88, 92

QU, J. A hybrid algorithm for community detection using PSO and EO. **Advances in Information Sciences and Service Sciences**, v. 5, n. 7, p. 187–195, 2013. 10

QUADE, D. Using weighted rankings in the analysis of complete blocks with additive block effects. **Journal of the American Statistical Association**, v. 74, n. 367, p. 680–683, 1979. 28

RANA, S.; JASOLA, S.; KUMAR, R. A review on particle swarm optimization algorithms and their applications to data clustering. **Artificial Intelligence Review**, v. 35, n. 3, p. 211–222, 2011. 10

REESE, J. Solution methods for the p-median problem: an annotated bibliography. **Networks**, v. 48, p. 125–142, 2006. 64, 65, 87

RESENDE, M. G. C.; WERNECK, R. F. A fast swap-based local search procedure for location problems. **Annals of Opeartions Research**, v. 150, p. 205–230, 2007. 96

REVELLE, C.; SWAIN, R. Central facilities location. **Geographical Analysis**, v. 2, p. 20–42, 1970. 66, 90

ROPKE, S. Parallel large neighborhood search - a software framework. In: METAHEURISTIC INTERNATIONAL CONFERENCE CDROM, 8., 2009. **Proceedings...** Hamburg, Germany: Springer New York, 2009. p. id1–id10. 88, 92

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Transportation Science**, v. 40, n. 4, p. 455–472, 2006. 88, 92

RYAN, D. M.; FOSTER, B. A. An integer programming approach to scheduling. In: WREN, A. (Ed.). **Computer scheduling of public transport urban passenger vehicle and crew scheduling**. North-Holland, Amsterdam: North-Holland Publishing Company, 1981. p. 269–280. 66, 76

SCHAEFFER, S. E. Graph clustering. **Computer Science Review**, v. 1, n. 1, p. 27–64, 2007. 1

SENNE, E. L. F.; LORENA, L. A. N.; PEREIRA, M. A. A branch-and-price approach to p-median location problems. **Computers & Operations Research**, v. 32, n. 6, p. 1655–1664, 2005. 66, 72, 76, 79

SHAMIR, R.; SHARAN, R.; TSUR, D. Cluster graph modification problems. **Discrete Applied Mathematics**, v. 144, n. 1-2, p. 173–182, 2004. 1, 5, 11

SHENG, J.; WANG, K.; SUN, Z.; WANG, B.; KHAWAJA, F. Overlapping community detection via preferential learning model. **Physica A**, v. 527, p. 121265, 2019. 38, 39

ŠÍMA, J.; SCHAEFFER, S. E. On the np-completeness of some graph cluster measures. In: INTERNATIONAL CONFERENCE ON CURRENT TRENDS IN THEORY AND PRACTICE OF COMPUTER SCIENCE, 2006. **Proceedings...** Merin, Czech Republic: Springer, 2006. p. 530–537. 38

SWAIN, R. W. A parametric decomposition approach for the solution of uncapacitated location problems. **Management Science**, v. 21, n. 2, p. 955–961, 1974. 70

WANG, H.-L.; WU, B. Y.; CHAO, K.-M. The backup 2-center and backup 2-median problems on trees. **Networks**, v. 53, p. 39–49, 2009. 64, 65, 87

WANG, K.; SAHO, Y.; ZHOU, W. Matheuristic for a two-echelon capacitated vehicle routing problem with environmental considerations in city logistics service. **Transportation Research Part D**, v. 57, p. 262–276, 2017. 7

WANG, R.; LIU, G.; WANG, C.; SU, L.; SUN, L. Predicting overlapping protein complexes based on core-attachment and a local modularity structure. **BMC Bioinformatics**, v. 19, p. 15, 2018. 37

WHANG, J.; GLEICH, D.; DHILLON, I. Overlapping community detection using neighborhood-inflated seed expansion. **IEEE Transactions on Knowledge and Data Engineering**, v. 28, n. 5, p. 1272–1284, 2016. xviii, 38, 39, 40, 42, 43, 45, 46, 47, 48, 49, 55, 57

WITTKOP, T.; BAUMBACH, J.; LOBO, F. P.; RAHMANN, S. Large scale clustering of protein sequences with force - a layout based heuristic for weighted cluster editing. **BMC Bioinformatics**, v. 8, n. 1, p. 396:1–12, 2007. 5

XIE, J.; KELLEY, S.; SZYMANSKI, B. Overlapping community detection in networks: the state-of-the-art and comparative study. **ACM Computing Surveys**, v. 45, n. 4, p. 43:1–43:35, 2013. 1, 2, 8, 37, 39, 42

XIE, J.; SZYMANSKI, B. K. Towards linear time overlapping community detection in social networks. In: PACIFIC-ASIA CONFERENCE ON ADVANCES IN KNOWLEDGE DISCOVERY AND DATA MINING, 16., 2012. **Proceedings...** Kuala Lumpur, Malaysia: Springer, 2012. p. 25–36. 8

XU, R.; WUNSCH II, D. Survey of clustering algorithms. **IEEE Transactions on Neural Networks**, v. 16, n. 3, p. 645–678, 2005. 1

XU, Y.; XU, H.; ZHANG, D.; ZHANG, Y. Finding overlapping community from social networks based on community forest model. **Knowledge-Based Systems journal**, v. 109, p. 238–255, 2016. 42

YAN, Y.; YU, G.; YAN, X.; XIE, H. Community cores expansion for overlapping community detection in complex networks. **Modern Physics Letters B**, v. 32, n. 33, p. 11, 2018. 42

YANG, J.; LESKOVEC, J. Overlapping community detection at scale: a nonnegative matrix factorization approach. In: ACM INTERNATIONAL CONFERENCE ON WEB SEARCH AND DATA MINING, 6., 2013. **Proceedings...** Rome, Italy: ACM, 2013. p. 587–596. 57

ZAHN, C. T. J. Symmetric relations by equivalence relations. **Journal of the Society for Industrial and Applied Mathematics**, v. 12, n. 4, p. 840–847, 1964. 8

ZHENG, J.; WANG, S.; LI, D.; ZHANG, B. Personalized recommendation based on hierarchical interest overlapping community. **Information Sciences**, v. 479, p. 55–75, 2019. 42

APPENDIX A - OCEP DETAILED RESULTS

Table A.1 shows the results, in terms of overlapping cluster editing costs, of the tests performed in order to evaluate the metaheuristics' influence on each of the three models. In these tests, 12 random graphs, three of each value $n = \{25, 50, 100, 200\}$, were used.

Table A.1 - Overlapping cluster editing costs of all hybrid heuristic versions in 12 random graphs instances. For each MILP it is shown the results of tests using just BRKGA, just SA and both metaheuristics to generate input clusters.

		M1			M2			M3		
Instance	z	**BRKGA**	**SA**	**both**	**BRKGA**	**SA**	**both**	**BRKGA**	**SA**	**both**
cmpr_101_5_25	0	68	**58**	68	52	61	**48**	52	61	**48**
	1	51	43	**42**	47	57	**46**	47	57	**46**
cmpr_102_1_25	0	**35**	62	39	31	36	**23**	31	36	**23**
	1	21	44	**20**	26	36	**21**	26	36	**21**
cmpr_102_4_25	0	68	74	**66**	**51**	60	60	**51**	60	60
	1	46	**43**	45	**48**	70	56	**48**	70	56
cmpr_105_2_50	0	132	194	**81**	114	123	**104**	114	123	**101**
	1	152	187	**90**	102	105	**96**	102	105	**96**
cmpr_105_3_50	0	**177**	202	204	162	188	**152**	**152**	188	**152**
	1	144	157	**134**	153	159	**142**	153	159	**140**
cmpr_105_6_50	0	360	446	**355**	333	319	**303**	333	319	**303**
	1	315	299	**283**	322	321	**312**	322	321	**312**
cmpr_109_1_100	0	249	485	**219**	281	**256**	299	281	**265**	299
	1	202	186	**184**	291	317	**263**	293	317	**263**
cmpr_109_2_100	0	505	805	**487**	512	488	**445**	512	488	**445**
	1	449	705	**417**	486	613	**439**	486	613	**452**
cmpr_110_8_100	0	1971	2364	**1931**	1917	1829	**1825**	1917	1829	**1825**
	1	1836	1804	**1715**	2107	1827	**1793**	2107	1827	**1824**
cmpr_113_1_200	0	939	1123	**871**	1083	1432	**1050**	1083	1432	**1054**
	1	856	1254	**808**	1088	1076	**1034**	1083	868	**851**
cmpr_113_2_200	0	**1920**	2544	2107	2090	**1941**	2037	2090	**1941**	2037
	1	1854	1800	**1787**	2020	2099	**1994**	2020	**1876**	1994
cmpr_113_10_200	0	9795	9709	**9477**	9595	9106	**8967**	9595	9029	**8979**
	1	8658	8703	**8438**	9769	**8808**	8934	9769	**8915**	8934
Best values		3	2	**19**	2	3	**19**	3	4	**18**

Table A.2 shows results of the tests in random graphs (BASTOS et al., 2016) with up to 100 vertices in which the optimal cluster editing cost is known. These optimal costs were obtained by CPLEX (IBM Corporation, 2018) solving the Charikar et al. (2005) linear integer programming model. The CPLEX execution time for solving this model are also presented. The cluster editing costs of the solutions generated by the BRKGA and SA metaheuristics and their execution times, in seconds, are shown. With regard to hybrid heuristic variations (HHM1, HHM2 and HHM3) results, in Table A.2, the solutions costs, the execution time, in seconds, and the number of vertices belonging to more than one cluster (*ovlp*) are presented. In particular, the execution time of each hybrid heuristic versions presented in all tables is only the execution time spent by CPLEX in the resolution of the models. Then,

the total execution time of the hybrid heuristic versions is composed by the sum of metaheuristics' time and the CPLEX time.

Table A.3 presents results of the hybrid heuristic tests in random graphs with sizes ranging from 50 vertices to 100 vertices. This table has the same structure as Table A.2, except for the results of the Charikar et al. (2005) model resolution. Since $3h$ was used as CPLEX maximum execution time, it was not possible to obtain cluster editing optimal solutions for these instances.

Table A.4 presents results of the hybrid heuristic tests in random graphs with sizes ranging from 200 vertices to 1000 vertices. This table has the same structure as Table A.3. It were also not possible to obtain cluster editing optimal solutions for instances presented in Table A.4.

In Table A.5 results of the all hybrid heuristic versions tests on the 30 LF benchmark graphs are shown. These instances have sizes ranging from 25 to 1000 vertices. In order to differentiate each graph, the number of edges (m) is presented. For each size, there are five instances that have graph density ranging from sparse to dense. In addition, the results of the supervised metric *FBCubed* are shown for the three variations of the hybrid heuristic. All the 30 instances have ground truth overlapping clustering.

Table A.2 - Tests results of hybrid heuristic, with $z = 0$ and $z = 1$, on random graphs with sizes ranging from 21 vertices to 100 vertices. In addition, metaheuristics' results are presented. Costs of Charikar et al. (2005) model solved by CPLEX are also shown.

Instance	n	CPLEX		BRKGA		SA		HHM1						HHM2						HHM3					
								$(z=0)$			$(z=1)$			$(z=0)$			$(z=1)$			$(z=0)$			$(z=1)$		
		cost	t (s)	cost	t (s)	cost	t (s)	cost	*ovlp*	t (s)	cost	*ovlp*	t (s)	cost	*ovlp*	t (s)	cost	*ovlp*	t (s)	cost	*ovlp*	t (s)	cost	*ovlp*	t (s)
cmpr_101_1_22	22	**8**	0.04	9	0.61	**8**	0.37	23	1	0.04	**8**	1	0.59	16	1	1.95	12	0	3.35	16	1	0.34	12	0	2.14
cmpr_101_1_25	25	**8**	0.05	9	1.95	**8**	0.41	24	1	0.09	10	1	0.63	15	0	0.98	18	0	4.31	15	0	0.33	18	0	2.98
cmpr_101_2_21	21	**15**	0.06	**15**	1.85	**15**	0.34	24	2	0.04	**15**	2	0.40	25	0	2.23	24	2	2.83	25	0	0.35	24	2	2.27
cmpr_101_2_25	25	**19**	0.15	**19**	1.08	**19**	0.42	34	0	0.08	**19**	2	0.52	31	0	2.02	46	1	3.39	31	0	0.67	46	1	1.97
cmpr_101_3_23	23	**21**	0.08	**21**	2.13	**21**	0.40	46	2	0.08	57	1	0.39	28	1	1.57	26	2	3.70	28	1	0.39	26	2	1.89
cmpr_101_3_25	25	**26**	0.09	**26**	0.48	**26**	0.45	42	1	0.08	101	4	0.49	40	0	1.37	38	2	4.98	40	0	0.65	38	2	2.55
cmpr_101_4_25	25	37	0.38	38	0.66	37	0.47	64	0	0.09	**34**	5	0.37	46	0	0.74	47	2	3.33	46	0	0.71	47	2	1.62
cmpr_101_5_25	25	44	1.19	46	0.61	44	0.48	68	0	0.05	**42**	2	0.35	56	0	1.36	58	3	3.55	56	0	0.71	58	3	1.89
cmpr_101_6_25	25	**65**	3.85	**65**	0.86	**65**	0.50	95	1	0.42	121	11	0.36	101	3	0.41	76	2	1.06	101	3	0.56	76	2	0.62
cmpr_101_7_25	25	**72**	4.61	75	0.66	**72**	0.46	133	0	0.09	73	7	0.32	80	0	0.72	78	5	1.25	80	0	0.85	78	5	0.59
cmpr_101_8_25	25	**80**	6.87	81	0.72	**80**	0.48	95	1	0.41	81	4	0.31	84	0	0.97	**80**	2	0.83	84	0	0.69	**80**	2	0.77
cmpr_101_9_25	25	99	10.39	105	0.68	100	0.52	137	0	0.38	121	13	0.08	128	0	0.33	96	20	1.01	128	25	0.22	**95**	2	0.31
cmpr_101_10_25	25	91	6.04	92	0.66	92	0.50	141	0	0.38	92	11	0.12	99	0	0.72	**90**	25	1.49	99	0	0.42	102	2	0.45
cmpr_102_1_24	24	20	0.08	20	0.66	20	0.40	29	1	0.05	**18**	1	0.50	28	1	0.73	28	0	3.56	28	1	0.33	28	0	2.85
cmpr_102_1_25	25	21	0.07	21	2.52	21	0.46	39	1	0.05	**20**	3	0.49	23	0	2.26	21	0	4.02	23	0	0.77	21	0	3.85
cmpr_102_2_21	21	**19**	0.08	**19**	0.64	**19**	0.35	28	2	0.04	**19**	6	0.45	27	0	0.81	35	1	1.30	27	0	0.58	35	1	1.48
cmpr_102_2_25	25	**24**	0.19	**24**	0.80	**24**	0.43	37	0	0.08	**24**	2	0.43	**24**	0	1.38	33	2	3.37	**24**	0	0.34	33	2	2.18
cmpr_102_3_25	25	**34**	0.47	**34**	0.58	**34**	0.45	74	2	0.07	39	3	0.49	38	1	1.75	37	2	3.43	38	1	0.38	37	2	2.55
cmpr_102_4_25	25	**45**	0.31	**45**	0.47	**45**	0.47	66	0	0.38	**45**	3	0.29	60	0	1.03	56	1	2.81	60	0	0.77	56	1	1.37
cmpr_102_5_25	25	**55**	1.60	**55**	0.50	**55**	0.49	71	0	0.08	56	4	0.29	74	0	0.83	60	3	3.03	74	0	0.42	60	3	0.97
cmpr_102_6_25	25	**77**	7.23	78	1.58	78	0.46	111	1	0.28	81	7	0.21	87	0	1.18	83	3	1.32	87	0	0.25	83	3	1.05
cmpr_102_7_25	25	**83**	8.23	85	0.60	83	0.46	105	0	0.28	87	4	0.49	89	3	1.21	94	4	2.26	89	3	0.39	94	4	0.71
cmpr_102_8_25	25	**80**	4.60	81	0.53	**80**	0.50	128	1	0.26	84	8	0.16	88	0	0.90	99	5	0.70	88	0	0.87	99	5	0.38
cmpr_102_9_25	25	89	18.24	91	0.50	89	0.55	135	0	0.36	114	13	0.13	90	0	0.73	**86**	25	0.76	90	0	0.37	114	8	0.35
cmpr_102_10_25	25	**102**	12.87	103	0.56	**102**	0.50	134	0	0.26	115	10	0.11	112	0	0.58	119	11	1.08	112	0	0.24	130	0	0.25
cmpr_105_1_45	45	33	1.18	34	2.61	33	1.19	162	1	0.16	**30**	6	2.04	84	1	2.60	47	0	16.40	84	1	1.41	47	0	10.17
cmpr_105_1_50	50	42	1.78	43	1.10	42	1.33	165	1	0.16	**38**	7	2.50	65	0	4.48	58	3	15.72	65	0	1.87	58	3	11.29
cmpr_105_2_50	50	88	5.14	94	0.91	90	1.33	138	1	0.16	**81**	6	1.52	104	0	3.70	96	2	13.09	101	0	2.24	96	2	10.12
cmpr_105_3_50	50	137	77.52	140	2.59	137	1.34	204	0	0.12	**134**	8	1.79	152	1	4.07	142	2	17.50	152	1	1.53	140	3	9.95
cmpr_105_4_50	50	209	7349.68	221	1.07	212	1.33	267	0	0.48	**206**	10	1.52	220	1	11.74	224	7	14.85	220	1	3.05	224	7	13.67
cmpr_105_6_50	50	298	10696.43	382	0.95	303	1.40	355	0	0.60	**283**	10	1.25	303	0	9.46	312	6	14.63	303	0	2.97	312	6	6.89
cmpr_106_1_50	50	**48**	0.53	55	1.03	52	1.39	314	1	0.09	49	5	1.19	72	0	4.16	50	0	4.78	72	0	0.99	50	0	3.26
cmpr_106_2_50	50	111	1.34	116	0.84	112	1.37	189	0	0.44	**108**	6	1.31	134	0	6.36	134	1	6.61	132	0	3.67	134	1	3.37
cmpr_106_3_50	50	177	117.23	179	0.86	179	1.36	230	0	0.12	**174**	7	1.24	195	4	4.10	193	3	6.89	195	4	2.63	193	3	5.42
cmpr_106_4_50	50	230	2780.21	234	0.92	231	1.36	335	1	0.17	**223**	15	1.34	242	2	7.23	227	4	7.50	242	2	1.55	227	4	5.73
cmpr_109_1_98	98	186	47.45	198	2.73	193	4.72	579	2	0.37	**183**	7	7.75	292	1	9.85	365	9	44.95	292	1	9.51	365	8	85.43
cmpr_109_1_100	100	187	69.98	205	1.96	194	5.06	219	1	0.59	**184**	6	6.94	299	1	13.13	263	7	347.54	299	1	13.36	263	4	423.01
cmpr_110_1_100	100	**256**	4.05	380	1.83	345	4.99	782	3	0.46	276	54	4.24	303	2	10.44	318	19	27.61	303	2	5.60	318	19	14.74
cmpr_110_2_100	100	**468**	69.39	565	2.64	558	4.90	567	3	0.22	486	38	3.44	538	1	22.47	546	17	33.16	538	1	22.34	546	17	21.05
cmpr_110_3_100	100	**739**	2347.71	887	1.81	788	5.06	886	9	0.68	757	37	4.24	806	5	18.07	763	15	69.89	806	5	9.79	761	16	29.04
Best values	-	**22**	-	**10**	-	**16**	-	**0**	-	-	**21**	-	-	**1**	-	-	**3**	-	-	**1**	-	-	**2**	-	-

Table A.3 - Tests results of hybrid heuristic versions, with $z = 0$ and $z = 1$, on random graphs with sizes ranging from 50 vertices to 100 vertices. In addition, metaheuristics' results are presented.

Instance	n	BRKGA		SA		HHM1						HHM2						HHM3					
						($z = 0$)			($z = 1$)			($z = 0$)			($z = 1$)			($z = 0$)			($z = 1$)		
		cost	*t (s)*	**cost**	*t (s)*	**cost**	*ovlp*	*t (s)*	**cost**	*ovlp*	*t (s)*	**cost**	*ovlp*	*t (s)*	**cost**	*ovlp*	*t (s)*	**cost**	*ovlp*	*t (s)*	**cost**	*ovlp*	*t (s)*
cmpr_105_5_50	50	245	0.9	**243**	1.4	291	2	0.4	475	13	1.6	257	2	8.5	251	0	10.1	257	2	3.1	251	0	7.6
cmpr_105_7_50	50	314	1.0	311	1.4	367	1	0.4	**304**	14	1.2	323	1	3.7	320	5	7.8	323	1	2.3	320	5	6.6
cmpr_105_8_50	50	416	1.3	408	1.4	459	2	0.5	**386**	13	1.0	554	0	2.2	411	6	3.8	554	0	2.0	411	6	2.8
cmpr_105_9_50	50	420	0.9	420	1.4	504	1	0.3	**402**	12	0.7	414	2	4.1	439	7	6.1	414	2	1.3	439	7	5.1
cmpr_105_10_50	50	**448**	1.0	449	1.5	604	2	0.7	464	34	0.4	467	50	4.3	495	2	1.1	467	49	2.0	495	2	0.7
cmpr_106_5_50	50	304	1.3	292	1.4	322	1	0.3	**288**	26	1.1	310	0	13.2	590	6	9.6	310	0	4.6	590	2	5.5
cmpr_106_6_50	50	308	2.1	307	1.4	453	1	0.1	693	20	0.9	320	3	2.8	**292**	13	8.5	320	3	2.6	297	13	6.7
cmpr_106_7_50	50	392	0.8	386	1.5	439	0	0.6	**369**	20	0.9	394	4	3.4	379	1	5.5	394	4	1.4	379	1	2.5
cmpr_106_8_50	50	394	1.0	394	1.4	451	0	0.3	381	20	0.6	406	1	3.9	381	0	3.9	406	1	1.1	381	0	2.2
cmpr_106_9_50	50	431	1.3	430	1.4	536	1	0.4	**414**	26	0.5	415	5	6.4	416	11	1.6	454	5	1.8	416	11	2.1
cmpr_106_10_50	50	468	0.9	459	1.5	570	1	0.0	478	27	0.3	564	0	0.7	**446**	0	1.0	564	0	0.4	**446**	0	0.4
cmpr_109_2_100	100	453	1.8	442	5.0	487	1	0.4	**417**	16	6.3	445	1	70.2	439	3	206.9	445	1	24.5	452	2	323.4
cmpr_109_3_100	100	641	2.0	633	4.9	1354	2	0.3	**608**	12	5.4	686	1	77.5	655	6	517.3	686	1	21.6	655	6	389.0
cmpr_109_4_100	100	901	2.0	883	4.9	983	1	0.9	**862**	30	5.7	905	0	34.7	870	6	121.1	905	0	25.7	870	6	82.8
cmpr_109_5_100	100	1112	1.9	1083	5.1	1199	1	0.7	**1037**	24	4.7	1091	5	45.3	1076	10	128.0	1091	5	16.9	1076	10	125.1
cmpr_109_6_100	100	1313	1.9	1272	5.0	1333	1	0.8	**1234**	29	4.3	1303	5	63.6	1247	22	97.0	1303	5	20.8	1250	21	207.7
cmpr_109_7_100	100	1602	1.9	1551	5.0	2070	1	0.3	2371	30	3.6	**1459**	6	39.9	1539	13	44.8	1567	6	10.9	1539	13	29.7
cmpr_109_8_100	100	1759	1.8	1677	5.1	1708	2	0.6	1664	48	2.8	**1543**	99	35.2	1669	10	38.7	1681	4	11.9	1669	10	27.3
cmpr_109_9_100	100	1954	1.9	1923	5.2	1940	1	0.9	2071	47	1.7	1953	89	19.2	**1897**	14	30.9	1953	0	6.0	**1897**	14	18.9
cmpr_109_10_100	100	2121	1.8	**2088**	5.3	2410	4	0.2	2121	55	0.9	2330	0	4.3	2320	0	3.3	2393	0	1.2	2320	0	1.4
cmpr_110_4_100	100	1125	1.9	1026	5.0	1069	1	0.6	**987**	44	4.0	1065	97	16.7	1408	20	31.0	1065	6	8.6	1408	20	24.8
cmpr_110_5_100	100	1367	1.7	1265	5.0	1273	4	0.6	**1240**	20	4.1	1288	7	15.9	1252	10	83.6	1288	7	9.1	1252	10	33.9
cmpr_110_6_100	100	1517	2.5	1476	5.1	1628	3	0.9	**1444**	36	3.4	1478	1	23.6	1468	19	53.8	1478	1	11.4	1468	19	41.1
cmpr_110_7_100	100	1699	1.9	1666	5.0	1860	3	1.5	**1613**	21	3.2	1580	5	32.6	1647	16	29.5	1917	5	8.4	1647	16	19.2
cmpr_110_8_100	100	1850	2.0	1829	5.3	1931	6	2.7	**1715**	59	2.4	1825	100	38.4	1793	26	44.7	1825	9	11.1	1793	26	18.1
cmpr_110_9_100	100	1963	2.2	1946	5.2	2211	4	0.3	1847	67	1.4	1785	0	6.8	1868	35	16.5	**1778**	0	7.2	1868	35	21.9
cmpr_110_10_100	100	2041	1.8	**2039**	5.7	2311	5	0.2	2083	56	1.0	2091	94	2.4	2092	2	3.9	2289	0	0.6	2092	2	1.4
Best values	-	**1**	-	**3**	-	**0**	-	-	**17**	-	-	**2**	-	-	**3**	-	-	**1**	-	-	**2**	-	-

Table A.4 - Tests results of hybrid heuristic versions, with $z = 0$ and $z = 1$, on random graphs with sizes ranging from 200 vertices to 1000 vertices. In addition, metaheuristics results are presented.

Instance	n	BRKGA		SA		HHM1						HHM2						HHM3					
						($z = 0$)			($z = 1$)			($z = 0$)			($z = 1$)			($z = 0$)			($z = 1$)		
		cost	*t (s)*	cost	*t (s)*	cost	*ovlp*	*t (s)*	cost	*ovlp*	*t (s)*	cost	*ovlp*	*t (s)*	cost	*ovlp*	*t (s)*	cost	*ovlp*	*t (s)*	cost	*ovlp*	*t (s)*
cmpr_113_1_200	200	857	3.3	844	17.8	871	0	0.8	**808**	18	26.1	1050	2	819.3	1034	8	1782.3	1054	2	1321.3	851	5	10898.4
cmpr_113_2_200	200	1873	3.2	1850	17.9	2107	0	0.8	**1787**	25	23.3	2037	11	653.7	1994	17	4005.5	2037	10	2281.2	1994	17	3110.4
cmpr_113_3_200	200	2841	3.2	**2813**	18.5	3167	3	0.7	3011	52	17.6	2996	13	2936.9	3156	19	2483.6	2996	13	3096.6	3163	21	2111.6
cmpr_113_4_200	200	3847	3.7	**3795**	18.1	4722	5	0.7	3969	41	17.0	3851	13	1813.8	3826	26	10802.1	4009	13	1683.0	3826	26	10804.3
cmpr_113_5_200	200	4822	3.4	4736	18.3	5308	2	0.9	4935	60	16.1	4565	8	1345.4	**4722**	29	10804.1	4761	8	2607.6	4760	24	10803.4
cmpr_113_6_200	200	5628	3.2	**5549**	18.3	6257	3	0.5	5970	35	13.3	6150	15	133.0	5531	25	8570.2	6150	15	89.0	5531	25	2092.4
cmpr_113_7_200	200	6699	3.4	6618	18.7	6674	3	4.4	6969	72	11.5	6661	12	790.0	**6563**	32	10800.9	6661	12	692.9	**6563**	32	10800.9
cmpr_113_8_200	200	7573	3.3	7428	18.8	7436	9	2.8	**7300**	80	11.0	7403	10	696.2	7374	41	10801.8	7403	10	381.2	7374	41	6586.7
cmpr_113_9_200	200	8381	3.2	8239	19.2	8230	6	3.2	8111	92	10.0	8260	22	352.0	**8104**	38	694.9	8260	22	242.3	**8104**	38	921.5
cmpr_113_10_200	200	8712	3.5	9071	19.2	9477	11	0.7	**8438**	180	3.3	8967	167	12.8	8934	0	10.3	8979	0	4.4	8934	0	10.7
cmpr_114_1_200	200	2044	3.4	1988	18.6	1724	42	1.1	**1420**	150	15.1	1766	44	157.3	1797	78	947.6	1766	44	114.4	1800	77	245.0
cmpr_114_2_200	200	2981	3.3	2785	18.5	2694	13	2.1	**2334**	117	14.9	2914	29	483.9	2572	48	228.1	2914	29	328.3	2563	52	991.6
cmpr_114_3_200	200	3746	3.4	3687	18.6	3633	10	2.2	**3343**	91	14.5	3850	29	676.7	3570	40	4008.5	3850	29	10060.8	3570	40	6981.4
cmpr_114_4_200	200	4930	3.3	4714	18.5	4714	7	1.0	**4433**	72	14.2	4719	23	681.4	4539	52	1754.1	4719	23	678.2	4539	52	1726.5
cmpr_114_5_200	200	5635	3.3	5463	18.5	5486	4	3.4	5714	56	11.5	5522	16	888.9	**5440**	47	1597.4	5522	16	1652.3	**5440**	47	562.9
cmpr_114_6_200	200	6627	3.4	6457	18.8	6485	3	2.9	6622	81	13.5	**6388**	27	385.5	6791	37	400.0	**6388**	27	742.3	6791	37	2174.7
cmpr_114_7_200	200	7310	3.4	7227	19.0	7309	10	1.1	**7035**	94	11.0	7664	2	1865.5	7325	79	1820.1	7664	2	783.2	7336	77	346.4
cmpr_114_8_200	200	8065	4.2	7967	19.0	8013	21	1.7	8116	91	12.0	7967	16	4179.6	**7922**	53	7996.9	7959	16	696.4	**7922**	53	3413.7
cmpr_114_9_200	200	8560	3.4	8549	19.7	9353	9	0.7	**8359**	105	5.0	8628	4	312.4	8620	11	415.3	8628	4	35.8	8620	11	1072.1
cmpr_114_10_200	200	**8907**	4.5	8991	19.6	9866	13	0.3	9704	200	2.5	9531	0	9.5	9491	0	5.8	9753	0	2.4	9491	0	2.1
cmpr_117_1_500	500	5987	10.0	5769	107.6	9802	15	1.8	6373	26	81.5	6683	51	15.9	7731	500	11026.6	5779	46	4.3	**5778**	0	10956.0
cmpr_117_2_500	500	11976	10.6	11787	108.5	16240	5	1.9	**11685**	52	71.6	12451	76	10802.7	13303	500	10808.1	12451	0	10803.3	13123	99	10805.2
cmpr_117_3_500	500	18279	9.8	18120	108.2	22383	32	1.6	**18016**	215	71.6	19016	121	10802.2	19205	500	10943.7	19057	59	10802.9	18126	0	10804.2
cmpr_117_4_500	500	24951	10.0	**24435**	108.2	26368	0	2.7	30050	279	56.4	24580	0	10801.9	25829	85	10812.8	24472	16	10803.0	25713	61	10824.0
cmpr_117_5_500	500	30597	10.7	30481	108.3	31479	9	6.7	32097	83	55.4	30568	154	10806.4	**30230**	500	10805.3	30463	0	10804.6	30550	45	10825.8
cmpr_117_6_500	500	36741	10.7	36584	109.0	38340	9	3.4	38202	352	51.6	**35501**	84	10802.4	35593	500	10801.6	36935	58	10803.8	36782	103	10825.0
cmpr_117_7_500	500	42846	10.6	42645	109.2	45164	9	1.8	**42373**	405	45.3	43786	500	10805.9	43409	186	10810.0	43574	55	10802.5	43654	112	10818.8
cmpr_117_8_500	500	49076	10.2	48697	110.5	48821	5	22.4	49879	205	42.9	48639	320	10802.7	**48578**	166	10804.9	48701	176	10803.5	48749	103	10802.4
cmpr_117_9_500	500	55014	10.5	54634	111.8	56385	32	5.6	55502	339	39.6	55825	82	10805.4	54541	138	10803.9	55825	91	10804.2	**54464**	141	10843.5
cmpr_117_10_500	500	**58618**	11.4	59909	114.3	62108	39	0.7	62108	500	8.2	59791	0	7547.1	61212	0	10.7	59791	0	580.3	61212	0	8.7
cmpr_118_1_500	500	16066	10.1	16135	109.1	15710	41	10842.1	**13677**	330	63.1	15364	110	10802.6	16366	254	10838.8	15613	119	10803.5	16310	66	10820.3
cmpr_118_2_500	500	21238	10.4	20534	109.3	20443	23	402.3	21345	286	53.9	21210	123	10813.1	**20010**	499	10822.1	21126	112	10805.6	20308	145	10808.8
cmpr_118_3_500	500	27442	10.5	26958	109.6	34720	38	1.5	26600	321	59.0	26983	105	10801.6	**26347**	231	10812.5	27056	87	10806.5	26605	153	10816.3
cmpr_118_4_500	500	32937	10.4	**32587**	108.6	37309	26	1.7	33933	271	52.0	32675	115	10803.3	32618	208	10805.3	32693	98	10801.6	32824	117	10819.1
cmpr_118_5_500	500	37558	10.5	37432	109.4	37638	22	34.7	38139	324	54.6	38501	88	10813.7	**36360**	500	10806.8	38501	71	1992.7	37554	141	10816.9
cmpr_118_6_500	500	43208	10.2	**42897**	109.0	43116	17	113.9	43617	187	47.0	43223	363	10801.4	43159	135	10806.5	42997	13	10803.5	43004	58	10810.7
cmpr_118_7_500	500	47451	9.8	47131	110.1	50697	18	1.7	51888	190	42.7	47323	307	10802.2	**46789**	475	10817.3	47354	118	10807.9	48219	3	3509.1
cmpr_118_8_500	500	52275	10.5	51897	111.1	53692	20	3.0	52067	220	41.6	51887	403	10802.2	51879	126	10815.7	51896	90	10803.3	**51846**	143	10813.7
cmpr_118_9_500	500	56652	10.5	56291	110.8	60537	35	1.7	56441	142	36.4	56274	147	10801.8	**55595**	167	10808.9	56644	7	10802.2	55720	163	10804.2
cmpr_118_10_500	500	**58064**	11.4	59147	114.3	61817	37	0.5	61128	487	8.2	60222	300	23.7	59953	0	13.4	60222	0	9.5	59953	0	20.1
cmpr_121_1_1000	1000	25737	29.0	**24669**	423.8	31678	172	10805.5	38112	679	94.3	36479	1000	10824.4	38052	1000	10885.7	24730	0	10818.1	28351	242	10909.3
cmpr_121_2_1000	1000	50162	29.0	**49182**	428.0	55110	163	1132.6	78464	643	103.5	56336	1000	10825.8	57872	1000	10855.9	49483	0	10829.0	49312	0	10812.6
cmpr_121_3_1000	1000	74694	30.2	**73708**	416.9	79203	114	1582.1	77423	668	102.7	80042	1000	10818.1	81586	1000	10885.0	73747	0	10811.2	73737	0	10803.5
cmpr_121_4_1000	1000	98716	29.2	97838	428.9	103842	110	15.7	111923	704	104.9	102207	1000	10819.3	104118	1000	10812.4	98318	0	10811.8	**97824**	0	10826.6
cmpr_121_10_1000	1000	**238131**	35.2	245233	437.2	248313	68	1.0	248989	1000	14.8	246021	0	292.6	246763	0	297.0	246021	0	33.4	246763	0	33.1
Best values	-	**4**	-	**9**	-	**0**	-	-	**14**	-	-	**2**	-	-	**12**	-	-	**1**	-	-	**8**	-	-

Table A.5 - Hybrid heuristic results, with $z = 0$ and $z = 1$, on LF benchmark graphs. Results of metaheuristics are also presented.

n	m	BRKGA		SA		HHM1						HHM2						HHM3					
						$z = 0$			$z = 1$			$z = 0$			$z = 1$			$z = 0$			$z = 1$		
		cost	t (s)	cost	t (s)	cost	FBC	t (s)	cost	FBC	t (s)	cost	FBC	t (s)	cost	FBC	t (s)	cost	FBC	t (s)	cost	FBC	t (s)
25	139	85	0.7	**84**	0.5	140	0.43	0.1	84	**0.62**	0.1	103	0.50	0.7	137	0.60	0.3	103	0.50	0.4	137	0.60	1.8
25	28	14	0.7	**13**	0.5	29	0.51	0.1	**13**	**0.70**	0.5	23	0.60	0.9	25	0.59	3.0	26	0.59	1.0	25	0.59	0.3
25	156	**90**	0.8	**90**	0.5	155	0.48	0.4	**90**	**0.67**	0.2	133	0.44	0.5	131	0.43	0.3	133	0.44	0.4	104	0.66	0.2
25	158	80	0.5	**75**	0.5	139	0.39	0.4	88	0.62	0.1	118	0.52	0.5	98	0.58	0.2	118	0.52	0.3	87	**0.64**	0.2
25	265	**35**	0.6	**35**	0.5	55	0.48	0.2	**35**	**0.74**	0.1	**35**	0.50	0.7	**35**	0.50	0.2	**35**	0.50	0.4	**35**	0.50	0.3
50	113	73	1.3	71	1.4	92	0.39	0.1	**65**	**0.48**	1.7	90	0.43	7.2	105	0.39	15.1	90	0.43	3.6	105	0.39	9.4
50	471	339	1.0	**331**	1.5	350	0.19	0.4	334	**0.24**	0.7	352	0.18	3.5	375	0.23	6.0	352	0.18	1.9	375	0.23	4.7
50	591	421	1.0	**381**	1.5	509	0.49	0.1	421	**0.61**	0.4	536	0.27	0.9	515	0.39	0.4	536	0.27	0.3	515	0.39	0.4
50	614	404	1.0	**367**	1.5	481	0.29	0.1	404	**0.64**	0.5	464	0.29	0.4	534	0.38	0.7	464	0.29	0.4	534	0.38	0.6
50	1039	**186**	1.4	**186**	1.6	**186**	0.52	0.1	**186**	**0.69**	0.1	**186**	0.52	0.5	**186**	0.52	0.2	**186**	0.52	0.2	**186**	0.52	0.1
100	264	187	1.8	174	4.9	787	0.27	0.3	**149**	**0.50**	7.0	282	0.35	24.5	192	0.49	100.7	282	0.35	14.6	192	0.49	37.0
100	1286	719	2.9	787	5.0	957	0.36	0.6	**711**	**0.41**	4.3	756	0.40	19.2	733	**0.41**	11.8	756	0.40	7.2	733	**0.41**	10.6
100	1703	1095	1.9	1123	5.2	1141	0.30	0.8	**1082**	**0.36**	3.2	1185	0.30	4.4	1179	0.35	18.4	1185	0.30	4.4	1179	0.35	17.1
100	2250	1559	1.8	1582	5.5	1542	0.39	0.7	**1493**	**0.51**	1.4	1559	0.38	8.6	2148	0.24	6.3	1559	0.38	3.0	2148	0.24	4.5
100	4121	**829**	1.9	**829**	5.5	**829**	0.25	0.1	**829**	**0.94**	0.1	**829**	0.25	0.5	**829**	0.25	0.2	**829**	0.25	0.3	**829**	0.25	0.1
200	450	344	3.4	316	17.8	1490	0.27	0.5	**239**	**0.57**	29.7	645	0.33	585.0	757	0.28	1106.4	645	0.33	3659.1	757	0.28	627.4
200	6330	4693	3.3	4390	18.9	**4264**	0.27	0.5	6852	**0.30**	11.7	4329	0.26	156.3	4324	0.24	559.7	4329	0.26	104.6	4324	0.24	353.1
200	7467	4564	3.5	4624	19.6	4897	0.38	1.3	**4049**	**0.42**	8.0	4625	0.37	66.0	4336	0.40	140.4	4625	0.37	18.7	4320	0.40	227.1
200	14051	**5849**	3.9	**5849**	21.0	**5849**	0.29	0.1	**5849**	**0.78**	0.2	**5849**	0.29	1.2	**5849**	0.29	1.1	**5849**	0.29	0.3	**5849**	0.29	0.4
200	16543	**3357**	3.4	4476	20.6	**3357**	0.20	0.1	**3357**	**0.86**	0.7	**3357**	0.20	1.9	**3357**	0.20	0.7	**3357**	0.20	0.7	**3357**	0.20	0.4
500	2440	2362	10.6	1977	107.7	3079	0.26	3.3	**1841**	**0.35**	71.1	3485	0.12	10801.6	4347	0.28	10844.3	2028	0.31	10801.2	2033	0.31	10924.0
500	21285	19692	10.6	18348	113.2	23303	0.08	2.0	18348	0.10	65.1	20231	0.06	10800.7	18637	**0.16**	10803.0	20192	0.06	10817.0	**17211**	0.09	10809.3
500	24177	22811	9.8	21044	112.5	21781	**0.13**	6.5	19730	0.08	64.3	20197	0.10	10803.3	**19572**	0.10	10818.8	20010	0.10	10804.2	20168	0.07	10856.3
500	33550	30793	10.5	27685	112.3	25772	0.16	15.4	**24256**	**0.19**	48.9	26752	0.15	10801.2	24678	0.16	10811.0	26608	0.15	10802.8	24838	0.16	10819.0
500	93255	**31495**	11.1	48172	124.9	**31495**	0.46	0.2	**31495**	**0.63**	0.4	**31495**	0.46	0.8	**31495**	0.46	1.0	**31495**	0.46	0.5	**31495**	0.46	0.4
1000	4996	6614	29.0	**4139**	421.1	13712	0.08	4.2	12202	0.25	102.7	18625	0.20	10809.5	18830	0.20	10820.1	4227	**0.39**	10819.5	4213	**0.39**	10909.3
1000	70857	69189	29.0	68377	433.1	80846	0.06	4.6	67075	0.08	101.3	69336	0.15	10856.3	66387	**0.17**	10897.7	68731	0.04	10827.7	**65492**	0.07	10812.6
1000	94399	91048	29.0	83608	427.3	88805	0.08	1229.5	**81594**	0.10	92.9	87110	**0.11**	10817.1	91554	0.10	10856.3	89709	0.03	10820.0	91722	0.02	10873.6
1000	129055	125019	29.0	107619	436.3	136973	0.05	4.6	**102190**	0.07	86.3	115246	**0.11**	10832.5	118144	0.10	10857.8	121666	0.03	10806.3	124462	0.02	10863.7
1000	306286	**193214**	33.4	193214	491.4	**193214**	0.20	0.2	**193214**	**0.63**	0.7	**193214**	0.20	6.6	**193214**	0.20	0.8	**193214**	0.20	1.1	**193214**	0.20	0.5
Best values	-	**8**	-	**12**	-	**7**	**1**	-	**20**	**23**	-	**7**	**2**	-	**8**	**3**	-	**7**	**1**	-	**9**	**3**	-

APPENDIX B - LF BENCHMARK DETAILED RESULTS

In this appendix, the deitaled results of tests performed on the 24 LF benchmark graphs are shown. These results are divided into four tables that presents the results of each method in each set of instances.

Table B.1 - Results of each algorithm in the set LF_1 of LF benchmark graphs.

				Metrics values					
Instance	***m***	***N***	**Metric**	**LECM**	**LFM**$_{best}$	**NISE**	**HH**	**HH-CR**	**HH-CR**$_{LFM}$
LFB1	76116	1949	*auc-cond*	0.083	0.103	0.080	0.100	**0.078**	0.079
			avg-cond	0.100	0.175	0.101	0.104	**0.099**	0.100
			GNMI	0.959	0.995	0.988	0.992	0.999	**1.000**
			$F1_{avg}$	0.983	0.961	0.971	0.997	0.999	**1.000**
LFB2	77467	2301	*auc-cond*	**0.161**	0.199	0.176	0.183	0.174	0.187
			avg-cond	0.185	0.251	0.192	0.183	**0.175**	0.204
			GNMI	0.912	**1.000**	0.924	0.950	0.954	0.998
			$F1_{avg}$	0.857	0.977	0.833	0.887	0.890	0.987
LFB3	75469	2384	*auc-cond*	**0.227**	0.267	0.249	0.258	0.247	0.252
			avg-cond	0.257	0.333	0.270	0.254	**0.245**	0.273
			GNMI	0.874	**0.999**	0.897	0.943	0.952	0.996
			$F1_{avg}$	0.854	0.966	0.833	0.889	0.893	**0.984**
LFB4	75928	2517	*auc-cond*	**0.280**	0.324	0.306	0.316	0.302	0.311
			avg-cond	0.315	0.384	0.332	0.316	**0.308**	0.339
			GNMI	0.867	**0.999**	0.891	0.942	0.954	0.995
			$F1_{avg}$	0.833	0.972	0.793	0.889	0.896	**0.984**
LFB5	75461	2781	*auc-cond*	**0.328**	0.366	0.355	0.359	0.344	0.352
			avg-cond	0.362	0.449	0.383	0.362	**0.352**	0.376
			GNMI	0.847	**0.997**	0.897	0.948	0.958	0.990
			$F1_{avg}$	0.829	0.950	0.792	0.895	0.904	**0.976**
LFB6	76040	3065	*auc-cond*	**0.369**	0.402	0.398	0.397	0.378	0.387
			avg-cond	0.402	0.473	0.430	0.406	**0.397**	0.419
			GNMI	0.851	**0.997**	0.866	0.953	0.961	0.990
			$F1_{avg}$	0.822	0.957	0.780	0.906	0.914	**0.980**
Best values	-	-	-	5	5	0	0	**7**	**7**

Table B.2 - Results of each algorithm in the set LF_2 of LF benchmark graphs.

Instance	m	N	Metric	Metrics value					
				LECM	**LFM**$_{best}$	**NISE**	**HH**	**HH-CR**	**HH-CR**$_{LFM}$
LFB7	76266	3877	*auc-cond*	0.267	0.304	0.298	0.300	0.235	**0.234**
			avg-cond	0.299	0.357	0.328	0.308	**0.297**	0.298
			GNMI	0.901	0.984	0.940	0.995	**1.000**	**1.000**
			$F1_{avg}$	0.978	0.962	0.947	0.994	**1.000**	**1.000**
LFB8	75996	4309	*auc-cond*	0.350	0.372	0.379	0.369	**0.332**	0.336
			avg-cond	0.371	0.399	0.405	0.367	**0.358**	0.360
			GNMI	0.918	0.995	0.923	0.990	0.994	**0.996**
			$F1_{avg}$	0.933	0.980	0.871	0.970	0.974	**0.980**
LFB9	75282	4731	*auc-cond*	**0.394**	0.422	0.444	0.419	0.395	0.396
			avg-cond	0.424	0.466	0.476	0.412	**0.402**	0.413
			GNMI	0.888	**0.994**	0.880	0.979	0.981	0.988
			$F1_{avg}$	0.898	**0.967**	0.811	0.943	0.944	0.961
LFB10	76919	4553	*auc-cond*	0.440	0.463	0.497	0.461	0.439	**0.437**
			avg-cond	0.469	0.506	0.527	0.456	**0.447**	0.455
			GNMI	0.809	**0.992**	0.865	0.975	0.978	0.984
			$F1_{avg}$	0.882	0.959	0.758	0.939	0.942	**0.961**
LFB11	75818	4793	*auc-cond*	0.478	0.500	0.546	0.498	**0.467**	**0.467**
			avg-cond	0.508	0.553	0.577	0.502	**0.493**	0.497
			GNMI	0.758	**0.990**	0.834	0.968	0.972	0.977
			$F1_{avg}$	0.856	0.947	0.729	0.932	0.940	**0.951**
LFB12	76471	5508	*auc-cond*	**0.508**	0.529	0.578	0.713	0.736	0.746
			avg-cond	0.539	0.578	0.613	0.521	**0.500**	0.505
			GNMI	0.712	**0.985**	0.705	0.035	0.051	0.053
			$F1_{avg}$	0.811	**0.930**	0.674	0.617	0.630	0.621
Best values	-	-	-	2	7	0	0	**10**	9

Table B.3 - Results of each algorithm in the set LF_3 of LF benchmark graphs.

Instance	m	N	Metric	Metrics values					
				LECM	**LFM**$_{best}$	**NISE**	**HH**	**HH-CR**	**HH-CR**$_{LFM}$
LFB13	76111	1935	*auc-cond*	0.437	0.277	0.208	0.262	0.188	**0.187**
			avg-cond	**0.099**	0.161	0.100	0.105	0.100	0.100
			GNMI	0.952	0.994	0.986	0.995	0.999	**1.000**
			$F1_{avg}$	0.978	0.968	0.974	0.997	0.999	**1.000**
LFB14	76573	5210	*auc-cond*	0.482	0.398	0.348	0.180	**0.173**	**0.173**
			avg-cond	0.469	0.337	0.241	0.255	**0.222**	0.240
			GNMI	0.907	0.992	0.904	0.958	0.969	**0.996**
			$F1_{avg}$	0.780	**0.875**	0.738	0.826	0.790	0.822
LFB15	76223	6403	*auc-cond*	0.479	0.493	0.461	**0.076**	0.096	**0.076**
			avg-cond	0.509	0.459	0.379	0.311	**0.290**	0.304
			GNMI	0.850	**0.983**	0.840	0.065	0.190	0.170
			$F1_{avg}$	0.659	**0.772**	0.619	0.562	0.529	0.525
LFB16	76464	6531	*auc-cond*	0.507	0.559	0.553	0.552	**0.492**	0.701
			avg-cond	0.507	0.549	0.491	0.400	**0.381**	0.386
			GNMI	0.753	**0.979**	0.688	0.036	0.125	0.158
			$F1_{avg}$	0.566	**0.731**	0.564	0.487	0.442	0.448
LFB17	76288	6678	*auc-cond*	**0.570**	0.610	0.615	0.699	0.735	0.737
			avg-cond	0.546	0.612	0.590	0.513	0.499	**0.482**
			GNMI	0.606	**0.968**	0.574	0.900	0.916	0.122
			$F1_{avg}$	0.537	**0.677**	0.532	0.612	0.579	0.448
LFB18	76244	6436	*auc-cond*	0.269	0.304	0.291	0.299	**0.237**	0.239
			avg-cond	0.606	0.659	0.653	0.569	**0.552**	0.557
			GNMI	0.559	**0.958**	0.498	0.044	0.057	0.089
			$F1_{avg}$	0.526	**0.612**	0.520	0.472	0.445	0.451
Best values	-	-	-	2	**9**	0	1	7	7

Table B.4 - Results of each algorithm in the set LF_4 of LF benchmark graphs.

				Metrics values					
Instance	m	N	**Metric**	**LECM**	**LFM**$_{best}$	**NISE**	**HH**	**HH-CR**	**HH-CR**$_{LFM}$
LFB19	76951	3991	*auc-cond*	0.454	0.415	0.396	0.410	**0.307**	**0.307**
			avg-cond	0.299	0.369	0.328	0.308	**0.296**	0.297
			GNMI	0.902	0.985	0.935	0.994	**1.000**	**1.000**
			$F1_{avg}$	0.979	0.955	0.961	0.994	**1.000**	**1.000**
LFB20	75960	7319	*auc-cond*	**0.479**	0.500	0.511	0.637	0.671	0.685
			avg-cond	0.512	0.456	0.442	0.400	**0.374**	0.391
			GNMI	0.948	0.985	0.916	0.981	0.988	**0.996**
			$F1_{avg}$	0.807	0.832	0.776	**0.840**	0.816	0.827
LFB21	75518	8159	*auc-cond*	**0.538**	0.572	0.595	0.690	0.723	0.728
			avg-cond	0.526	0.551	0.550	0.448	**0.422**	0.425
			GNMI	0.862	**0.973**	0.764	0.051	0.134	0.140
			$F1_{avg}$	0.654	**0.722**	0.642	0.556	0.538	0.533
LFB22	76449	8330	*auc-cond*	**0.599**	0.628	0.660	0.711	0.753	0.753
			avg-cond	0.572	0.613	0.634	0.524	**0.497**	0.506
			GNMI	0.717	**0.963**	0.599	0.050	0.119	0.137
			$F1_{avg}$	0.567	**0.632**	0.547	0.500	0.478	0.493
LFB23	75908	8475	*auc-cond*	0.646	0.670	0.704	0.722	**0.557**	0.764
			avg-cond	0.630	0.666	0.696	0.591	**0.570**	0.571
			GNMI	0.577	**0.941**	0.480	0.054	0.104	0.115
			$F1_{avg}$	0.503	**0.554**	0.480	0.474	0.450	0.449
LFB24	76670	8533	*auc-cond*	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
			avg-cond	0.680	0.706	0.741	0.650	**0.631**	0.641
			GNMI	0.475	**0.907**	0.446	0.059	0.077	0.874
			$F1_{avg}$	0.431	0.468	0.418	0.456	0.438	**0.490**
Best values	-	-	-	4	8	1	2	**11**	6

APPENDIX C - DETAILED B&P PMPOC RESULTS

Table C.1 shows the B&P results in all the 40 OR-library instances with each pair of z and α values. This table presents the detailed results of Table 4.2.

Table C.1 - B&P results with overlap control in OR-library instances (BEASLEY, 1985; BEASLEY, 1990).

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
pmed1	100	5	5819	0	0.05	1	0	0	200905	11264	104	0.04	138.32
					0.50	1	0	0	200905	13165	104	0.04	110.96
					0.95	1	0	0	200905	13837	104	0.04	111.30
				0.5	0.05	118	10	13	23589205	10771	108	0.05	11928.80
					0.50	1	0	0	200905	13949	130	0.17	107.64
					0.95	1	0	0	200905	14776	130	0.20	108.23
				1	0.05	130	10	29	25988005	10150	106	0.03	11579.30
					0.50	1	0	0	200905	15712	141	0.32	100.66
					0.95	1	0	0	200905	16006	139	0.27	101.75
pmed2	100	10	4093	0	0.05	1	0	0	200910	9514	109	0.09	119.88
					0.50	1	0	0	200910	11896	109	0.09	102.13
					0.95	1	0	0	200910	9860	109	0.09	100.72
				0.5	0.05	5	2	2	1000510	10269	117	0.09	1445.54
					0.50	1	0	0	200910	15694	164	0.31	101.71
					0.95	1	0	0	200910	16379	165	0.35	102.97
				1	0.05	3	1	1	600710	11881	124	0.13	504.41
					0.50	1	0	0	200910	17957	172	0.39	97.35
					0.95	1	0	0	200910	16142	174	0.36	94.77
pmed3	100	10	4250	0	0.05	1	0	0	200910	12066	109	0.09	134.36
					0.50	1	0	0	200910	11318	109	0.09	98.57
					0.95	1	0	0	200910	11739	109	0.09	102.19
				0.5	0.05	128	8	0	25588210	14560	202	0.70	11643.80
					0.50	1	0	0	200910	20991	206	0.65	101.27
					0.95	1	0	0	200910	17080	161	0.34	102.57
				1	0.05	132	9	10	26387810	12662	122	0.13	11655.40
					0.50	1	0	0	200910	19085	174	0.32	102.34
					0.95	1	0	0	200910	19177	174	0.38	96.42
pmed4	100	20	3034	0	0.05	1	0	0	200920	11905	119	0.19	131.22
					0.50	1	0	0	200920	12314	120	0.20	100.09
					0.95	1	0	0	200920	12302	119	0.19	93.39
				0.5	0.05	135	9	17	25188420	14294	137	0.19	11966.40
					0.50	1	0	0	200920	26051	255	0.64	99.67
					0.95	1	0	0	200920	22183	213	0.47	95.00
				1	0.05	136	8	1	27187419	15879	149	0.22	11554.90
					0.50	1	0	0	200920	27497	271	0.64	95.74
					0.95	1	0	0	200920	25895	231	0.49	93.51

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	200956	9132	132	0.32	138.94
				0	0.50	1	0	0	200956	8628	132	0.32	95.52
					0.95	1	0	0	5356	8628	132	0.32	1.79
					0.05	173	10	41	27587256	14478	281	0.61	11770.60
pmed5	100	33	1355	0.5	0.50	1	0	0	200956	22182	251	0.44	97.93
					0.95	1	0	0	200956	24095	290	0.57	95.03
					0.05	37	18	14	3799156	13282	160	0.32	7018.03
				1	0.50	1	0	0	200956	18133	222	0.57	105.43
					0.95	1	0	0	200956	20208	233	0.59	104.68
					0.05	1	0	0	400805	13823	205	0.03	332.25
				0	0.50	1	0	0	400805	13255	205	0.03	247.71
					0.95	1	0	0	400805	14057	207	0.04	233.62
					0.05	64	7	0	25588205	10005	225	0.13	14526.70
pmed6	200	5	7824	0.5	0.50	1	0	0	400805	15694	238	0.12	238.28
					0.95	1	0	0	400805	15836	260	0.14	219.86
					0.05	52	6	0	20790605	10200	200	0.00	12188.10
				1	0.50	1	0	0	400805	18678	267	0.22	244.64
					0.95	1	0	0	400805	17700	270	0.27	244.14
					0.05	1	0	0	400810	12241	209	0.05	349.38
				0	0.50	1	0	0	400810	10754	209	0.05	221.74
					0.95	1	0	0	400810	10937	209	0.05	209.76
					0.05	64	7	0	25588210	15154	301	0.36	12850.10
pmed7	200	10	5631	0.5	0.50	1	0	0	400810	17904	314	0.28	238.86
					0.95	1	0	0	400810	20065	306	0.30	234.16
					0.05	50	7	7	19991010	11138	223	0.06	12416.50
				1	0.50	1	0	0	400810	18694	327	0.30	248.47
					0.95	1	0	0	400810	20652	319	0.35	237.10
					0.05	1	0	0	400820	12073	219	0.10	273.42
				0	0.50	1	0	0	400820	12199	219	0.10	202.74
					0.95	1	0	0	400820	11615	219	0.10	200.39
					0.05	64	6	0	25588220	22469	460	0.62	13337.50
pmed8	200	20	4445	0.5	0.50	1	0	0	400820	22887	390	0.34	228.54
					0.95	1	0	0	400820	23858	389	0.35	221.48
					0.05	54	6	1	20790620	64590	1153	1.00	12789.00
				1	0.50	1	0	0	400820	27940	455	0.50	235.84
					0.95	1	0	0	400820	26610	448	0.44	228.09
					0.05	1	0	0	400840	11727	239	0.20	310.20
				0	0.50	1	0	0	400840	12311	241	0.21	197.55
					0.95	1	0	0	400840	11518	240	0.20	193.22
					0.05	25	9	7	9996040	34889	742	0.77	11618.10
pmed9	200	40	2734	0.5	0.50	1	0	0	400840	32755	570	0.60	227.75
					0.95	1	0	0	400840	27124	465	0.46	220.36
					0.05	56	7	1	20786599	85192	1690	0.91	12379.20
				1	0.50	1	0	0	400840	41840	692	0.63	222.79
					0.95	1	0	0	400840	37032	640	0.59	216.77

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	400872	9129	266	0.33	334.34
				0	0.50	1	0	0	400872	9267	268	0.34	211.15
					0.95	1	0	0	400872	9033	266	0.33	210.30
					0.05	1	0	0	400872	49318	1177	0.87	214.89
pmed10	200	67	1255	0.5	0.50	1	0	0	400872	27776	579	0.60	246.78
					0.95	1	0	0	400872	29290	670	0.76	227.90
					0.05	62	9	5	22389872	70019	1759	0.93	13029.50
				1	0.50	1	0	0	400872	60305	1226	0.79	221.40
					0.95	1	0	0	400872	210567	4128	1.00	157.67
					0.05	1	0	0	600705	11469	304	0.01	597.52
				0	0.50	1	0	0	600705	13293	304	0.01	513.19
					0.95	1	0	0	600705	13907	304	0.01	461.08
					0.05	31	5	0	18591705	11537	377	0.24	14649.80
pmed11	300	5	7696	0.5	0.50	1	0	0	600705	17484	416	0.20	498.73
					0.95	1	0	0	600705	13190	360	0.16	357.98
					0.05	28	5	0	16792605	18500	525	0.62	13652.80
				1	0.50	1	0	0	600705	16483	367	0.17	410.12
					0.95	1	0	0	600705	14419	366	0.18	376.88
					0.05	1	0	0	600710	13789	309	0.03	567.32
				0	0.50	1	0	0	600710	13334	309	0.03	329.61
					0.95	1	0	0	600710	13452	309	0.03	331.69
					0.05	31	5	0	18591710	16895	551	0.60	14217.40
pmed12	300	10	6634	0.5	0.50	1	0	0	600710	18197	431	0.22	378.47
					0.95	1	0	0	600710	17404	412	0.19	356.51
					0.05	1	0	0	600710	24455	583	0.53	485.07
				1	0.50	1	0	0	600710	19027	444	0.24	387.54
					0.95	1	0	0	600710	22138	469	0.27	373.80
					0.05	1	0	0	600750	11616	329	0.10	467.67
				0	0.50	1	0	0	600750	11278	329	0.10	318.96
					0.95	1	0	0	600750	11540	330	0.10	299.88
					0.05	31	5	0	18591750	26149	792	0.72	13310.10
pmed13	300	30	4374	0.5	0.50	1	0	0	600750	25806	616	0.32	385.79
					0.95	1	0	0	600750	22168	578	0.33	357.54
					0.05	7	2	3	4198950	38275	998	0.72	3743.43
				1	0.50	1	0	0	600750	33578	803	0.52	402.01
					0.95	1	0	0	600750	35346	856	0.43	385.27
					0.05	1	0	0	600780	13247	359	0.20	493.01
				0	0.50	1	0	0	600780	14051	359	0.20	332.73
					0.95	1	0	0	600780	13538	360	0.20	328.38
					0.05	31	5	0	18591780	57187	1404	0.77	13938.70
pmed14	300	60	2968	0.5	0.50	1	0	0	600780	42660	923	0.66	367.50
					0.95	1	0	0	600780	43168	895	0.42	358.31
					0.05	1	0	0	600780	86477	1903	0.84	331.69
				1	0.50	1	0	0	517980	63814	1410	0.65	332.61
					0.95	1	0	0	600780	60712	1332	0.48	353.05

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	600800	11136	399	0.33	507.22
				0	0.50	1	0	0	600800	10994	399	0.33	371.96
					0.95	1	0	0	600800	11610	401	0.34	352.78
					0.05	49	13	23	160442	42230	1218	0.61	308.64
pmed15	300	100	1729	0.5	0.50	1	0	0	600800	47686	1160	0.59	354.41
					0.95	1	0	0	600800	40010	973	0.57	362.70
					0.05	1	0	0	600800	85200	2200	0.85	360.11
				1	0.50	1	0	0	20900	144343	3196	0.48	9.84
					0.95	1	0	0	600800	884197	17831	1.00	248.68
					0.05	1	0	0	800605	13308	404	0.01	992.98
				0	0.50	1	0	0	800605	14413	404	0.01	642.51
					0.95	1	0	0	800605	12618	404	0.01	555.59
					0.05	15	4	0	11995005	10160	425	0.06	14187.00
pmed16	400	5	8162	0.5	0.50	1	0	0	800605	16025	503	0.18	749.30
					0.95	1	0	0	800605	15834	455	0.10	509.47
					0.05	15	4	0	11995005	19336	637	0.41	14047.50
				1	0.50	1	0	0	800605	16742	504	0.07	606.03
					0.95	1	0	0	800605	15311	486	0.14	577.67
					0.05	1	0	0	800610	12071	409	0.02	904.88
				0	0.50	1	0	0	800610	12775	409	0.02	491.43
					0.95	1	0	0	800610	13515	409	0.02	468.73
					0.05	15	4	0	11995010	16965	708	0.59	12750.50
pmed17	400	10	6999	0.5	0.50	1	0	0	800610	18303	533	0.18	539.95
					0.95	1	0	0	800610	16283	515	0.13	503.19
					0.05	15	4	0	11995010	22590	743	0.69	15324.50
				1	0.50	1	0	0	800610	20490	617	0.25	632.36
					0.95	1	0	0	800610	20134	600	0.14	577.95
					0.05	1	0	0	800640	14050	439	0.10	690.93
				0	0.50	1	0	0	800640	14008	442	0.10	471.02
					0.95	1	0	0	800640	13962	441	0.10	432.67
					0.05	15	4	0	11995040	47222	1424	0.74	15563.10
pmed18	400	40	4809	0.5	0.50	1	0	0	800640	29230	803	0.22	541.18
					0.95	1	0	0	800640	28539	778	0.30	499.96
					0.05	1	0	0	800640	47999	1390	0.77	531.80
				1	0.50	1	0	0	800640	42504	1194	0.42	620.25
					0.95	1	0	0	800640	44963	1195	0.31	562.69
					0.05	1	0	0	800720	14070	480	0.20	1102.12
				0	0.50	1	0	0	800720	13248	481	0.20	513.87
					0.95	1	0	0	800720	13127	481	0.20	470.99
					0.05	25	4	12	80965	50524	1699	0.62	155.37
pmed19	400	80	2845	0.5	0.50	1	0	0	800720	31414	878	0.27	535.86
					0.95	1	0	0	800720	29833	897	0.40	560.86
					0.05	81	7	38	124144	33535	1186	0.54	174.47
				1	0.50	1	0	0	80720	109321	3144	0.51	61.80
					0.95	1	0	0	800720	613532	15645	1.00	358.05

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	800797	13659	532	0.33	756.38
				0	0.50	1	0	0	800797	13651	532	0.33	632.78
					0.95	1	0	0	800797	13716	533	0.33	563.57
					0.05	21	6	9	19597	48498	1644	0.62	109.64
pmed20	400	133	1789	0.5	0.50	1	0	0	6797	44490	1327	0.43	3.99
					0.95	1	0	0	800797	62323	1652	0.54	531.38
					0.05	1	0	0	5197	64640	2003	0.48	3.65
				1	0.50	1	0	0	28797	214027	5308	0.70	15.81
					0.95	1	0	0	1197	2137120	46966	1.00	0.95
					0.05	1	0	0	1000505	15396	504	0.01	1356.81
				0	0.50	1	0	0	1000505	14735	506	0.01	874.79
					0.95	1	0	0	1000505	15481	504	0.01	799.06
					0.05	7	3	0	6997505	20179	612	0.20	12247.30
pmed21	500	5	9138	0.5	0.50	1	0	0	1000505	19770	688	0.20	1148.88
					0.95	1	0	0	1000505	15496	543	0.05	573.72
					0.05	7	3	0	6997505	25345	852	0.36	13464.80
				1	0.50	1	0	0	1000505	19782	684	0.10	909.94
					0.95	1	0	0	1000505	21548	676	0.20	808.93
					0.05	1	0	0	1000510	16188	512	0.02	1168.32
				0	0.50	1	0	0	1000510	15998	513	0.03	633.15
					0.95	1	0	0	1000510	15123	515	0.03	607.45
					0.05	7	3	0	6997510	26983	1144	0.79	16015.20
pmed22	500	10	8579	0.5	0.50	1	0	0	1000510	19893	632	0.13	646.47
					0.95	1	0	0	1000510	19555	581	0.03	606.95
					0.05	7	2	3	6997510	38688	1258	0.19	12583.90
				1	0.50	1	0	0	1000510	25062	731	0.13	949.19
					0.95	1	0	0	1000510	26480	824	0.09	830.68
					0.05	1	0	0	1000550	14808	552	0.10	1334.40
				0	0.50	1	0	0	1000550	14184	554	0.11	607.99
					0.95	1	0	0	1000550	13975	549	0.10	560.46
					0.05	139	14	0	14321703	38343	1495	0.67	16037.50
pmed23	500	50	4619	0.5	0.50	1	0	0	1000550	28443	1040	0.32	637.29
					0.95	1	0	0	1000550	31472	1057	0.25	610.52
					0.05	1	0	0	4818	50988	1960	0.68	4.16
				1	0.50	1	0	0	1550	28661	1087	0.32	0.98
					0.95	1	0	0	1000550	377307	11086	1.00	551.80
					0.05	1	0	0	1000600	14652	601	0.20	1138.40
				0	0.50	1	0	0	1000600	13986	601	0.20	698.14
					0.95	1	0	0	1000600	14427	600	0.20	647.69
					0.05	11	4	5	23451	54226	2132	0.74	60.50
pmed24	500	100	2961	0.5	0.50	1	0	0	184600	46908	1595	0.44	125.37
					0.95	1	0	0	1000600	54024	1731	0.29	663.46
					0.05	1	0	0	2973	149804	5004	0.71	2.02
				1	0.50	1	0	0	16600	125461	4029	0.33	9.99
					0.95	1	0	0	1000600	1202410	33218	1.00	446.81

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	1000669	12065	668	0.34	873.99
				0	0.50	1	0	0	1000669	12113	668	0.33	816.60
					0.95	1	0	0	1000669	12111	668	0.34	861.48
					0.05	13	4	5	21054	32414	1656	0.56	48.64
pmed25	500	167	1828	0.5	0.50	13	5	4	7669	8555	1000	0.63	18.70
					0.95	1	0	0	156169	66005	2081	0.48	111.21
					0.05	1	0	0	3167	231018	8174	0.73	2.20
				1	0.50	1	0	0	1169	2677880	73207	1.00	1.07
					0.95	1	0	0	1169	2733070	72724	1.00	1.06
					0.05	1	0	0	1200405	15429	605	0.01	1987.37
				0	0.50	1	0	0	1200405	16914	605	0.01	1072.89
					0.95	1	0	0	1200405	15236	606	0.01	960.49
					0.05	3	2	0	3599205	21493	902	0.23	13448.80
pmed26	600	5	9917	0.5	0.50	1	0	0	1200405	20938	811	0.13	1472.34
					0.95	1	0	0	1200405	32062	1220	1.00	709.35
					0.05	7	3	0	8396805	25971	1014	0.46	19249.70
				1	0.50	1	0	0	1200405	18935	756	0.07	1527.35
					0.95	1	0	0	1200405	25139	765	0.15	1361.90
					0.05	1	0	0	1200410	13665	609	0.02	1528.70
				0	0.50	1	0	0	1200410	13441	609	0.02	803.63
					0.95	1	0	0	1200410	15735	609	0.02	736.49
					0.05	3	2	0	3599210	25125	1066	0.21	13530.20
pmed27	600	10	8307	0.5	0.50	1	0	0	1200410	18101	749	0.11	804.52
					0.95	1	0	0	1200410	18497	713	0.09	741.62
					0.05	7	2	2	8396810	20547	931	0.38	16284.10
				1	0.50	1	0	0	1200410	32227	1395	0.34	1532.60
					0.95	1	0	0	1200410	39800	1582	1.00	747.55
					0.05	1	0	0	1200480	13134	662	0.10	1361.96
				0	0.50	1	0	0	1200480	14214	660	0.10	785.59
					0.95	1	0	0	1200480	13741	663	0.10	718.95
					0.05	1252	14	0	8628002	36675	1719	0.69	11295.10
pmed28	600	60	4498	0.5	0.50	1	0	0	1200480	34000	1401	0.24	817.02
					0.95	1	0	0	1200480	32662	1379	0.21	796.44
					0.05	1	0	0	2880	82399	3494	0.41	1.88
				1	0.50	1	0	0	8280	77640	3404	0.87	5.94
					0.95	1	0	0	53280	67717	2892	0.48	26.98
					0.05	1	0	0	1200600	14214	724	0.21	1338.97
				0	0.50	1	0	0	1200600	14241	720	0.20	927.49
					0.95	1	0	0	1200600	14337	729	0.21	866.80
					0.05	5	2	2	15296	59123	2757	0.71	39.55
pmed29	600	120	3033	0.5	0.50	1	0	0	30000	40958	1739	0.34	20.60
					0.95	1	0	0	34800	64977	2587	0.30	27.79
					0.05	3	1	1	3000	319866	11212	0.46	8.69
				1	0.50	1	0	0	7200	186687	7282	0.59	4.14
					0.95	1	0	0	1200600	1702050	55829	1.00	610.65

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	1200600	15434	808	0.34	1390.90
				0	0.50	1	0	0	1200600	15506	813	0.35	1199.22
					0.95	1	0	0	1200600	15325	808	0.35	1180.95
					0.05	7	2	3	19128	106102	4357	0.87	51.52
pmed30	600	200	1989	0.5	0.50	1	0	0	1800	77370	3242	0.77	1.46
					0.95	1	0	0	16200	64351	2375	0.42	11.86
					0.05	1	0	0	4800	92920	4181	0.86	4.51
				1	0.50	1	0	0	1200	3613230	103486	1.00	1.18
					0.95	1	0	0	1200	3470510	99918	1.00	1.20
					0.05	1	0	0	1400305	15665	706	0.01	2094.80
				0	0.50	1	0	0	1400305	17260	704	0.01	2060.67
					0.95	1	0	0	1400305	15006	705	0.01	1417.42
					0.05	3	2	0	4198905	17277	944	0.31	24081.20
pmed31	700	5	10086	0.5	0.50	1	0	0	1400305	16045	755	0.05	1074.44
					0.95	1	0	0	1400305	17967	764	0.03	1002.74
					0.05	3	2	0	4198905	24663	1147	0.29	16782.00
				1	0.50	1	0	0	1400305	20962	986	0.25	2436.77
					0.95	1	0	0	1400305	22093	929	0.23	2052.10
					0.05	1	0	0	1400310	15593	713	0.02	1993.42
				0	0.50	1	0	0	1400310	17205	711	0.02	1072.56
					0.95	1	0	0	1400310	16196	710	0.01	955.56
					0.05	3	2	0	4198910	30702	1343	0.21	19816.50
pmed32	700	10	9297	0.5	0.50	1	0	0	1400310	18481	800	0.03	1049.90
					0.95	1	0	0	1400310	18902	795	0.07	951.97
					0.05	3	2	0	4198910	33791	1485	0.14	12083.50
				1	0.50	1	0	0	1400310	43518	1908	0.39	2274.10
					0.95	1	0	0	1400310	27440	1220	0.18	2024.11
					0.05	1	0	0	1400420	15481	778	0.11	2384.47
				0	0.50	1	0	0	1400420	15872	779	0.11	1123.64
					0.95	1	0	0	1400420	16388	784	0.12	956.10
					0.05	795	26	391	5934602	33403	1886	0.73	21711.00
pmed33	700	70	4700	0.5	0.50	1	0	0	244720	38613	1702	0.16	183.84
					0.95	1	0	0	136220	41254	1755	0.14	115.27
					0.05	1574	102	777	1292867	18237	1319	0.51	2012.32
				1	0.50	1	0	0	9520	90780	3870	0.42	9.03
					0.95	1	0	0	74620	89352	3973	0.48	47.25
					0.05	1	0	0	1400560	14945	843	0.20	1482.76
				0	0.50	1	0	0	1400560	14729	845	0.21	1206.69
					0.95	1	0	0	1400560	14794	843	0.20	1143.69
					0.05	25	4	12	43959	45652	2726	0.72	124.79
pmed34	700	140	3013	0.5	0.50	1	0	0	55860	69917	3111	0.38	46.07
					0.95	1	0	0	41860	58065	2478	0.25	30.80
					0.05	1	0	0	4060	53927	2730	0.56	3.34
				1	0.50	1	0	0	25060	136567	5852	0.45	14.76
					0.95	1	0	0	1260	2571480	87806	1.00	1.33

Continued on next page

Table C.1: Continuation.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
					0.05	1	0	0	1600205	15706	804	0.01	3360.53
				0	0.50	1	0	0	1600205	16254	804	0.01	1936.67
					0.95	1	0	0	1600205	16896	804	0.01	1677.36
					0.05	1	1	0	1600205	19411	1216	0.45	11314.20
pmed35	800	5	10400	0.5	0.50	1	0	0	1600205	21772	920	0.12	1358.12
					0.95	1	0	0	1600205	19806	877	0.06	1129.30
					0.05	3	2	0	4798605	18853	1027	0.17	15835.00
				1	0.50	1	0	0	1600205	22173	1170	0.23	3191.26
					0.95	1	0	0	1600205	22815	1141	0.18	2713.23
					0.05	1	0	0	1600210	18251	809	0.01	2382.10
				0	0.50	1	0	0	1600210	19129	809	0.01	1245.54
					0.95	1	0	0	1600210	18265	812	0.02	1142.79
					0.05	1	1	0	1600210	23313	1310	0.53	12228.50
pmed36	800	10	9934	0.5	0.50	1	0	0	1600210	21756	959	0.06	1244.30
					0.95	1	0	0	1600210	21483	958	0.07	1098.26
					0.05	3	2	0	4798610	32842	1595	0.73	15048.70
				1	0.50	1	0	0	1600210	33163	1562	0.18	3177.58
					0.95	1	0	0	1600210	29622	1392	0.17	2812.74
					0.05	1	0	0	1600320	18430	892	0.12	2438.53
				0	0.50	1	0	0	1600320	19058	893	0.12	1392.39
					0.95	1	0	0	1600320	16924	888	0.11	1152.32
					0.05	26	6	12	150960	32479	1947	0.50	386.68
pmed37	800	80	5057	0.5	0.50	1	0	0	17120	36272	1628	0.17	12.26
					0.95	1	0	0	1920	30202	1919	0.63	1.44
					0.05	1	0	0	5812	95092	4502	0.73	8.54
				1	0.50	1	0	0	10720	79333	3520	0.30	9.04
					0.95	1	0	0	1600320	816832	32874	1.00	780.91
					0.05	1	0	0	1800105	16996	905	0.01	5246.87
				0	0.50	1	0	0	1800105	17862	904	0.00	2765.71
					0.95	1	0	0	1800105	16243	904	0.00	1742.44
					0.05	1	1	0	1800105	23531	1558	0.65	15283.20
pmed38	900	5	11060	0.5	0.50	1	0	0	1800105	18495	964	0.02	1554.10
					0.95	1	0	0	1800105	18946	993	0.07	1572.61
					0.05	3	1	1	5398305	18921	1070	0.05	19730.70
				1	0.50	1	0	0	1800105	23638	1227	0.09	4447.78
					0.95	1	0	0	1800105	22995	1245	0.13	3834.33
					0.05	1	0	0	1800110	18171	911	0.01	3584.43
				0	0.50	1	0	0	1800110	15310	910	0.01	1516.29
					0.95	1	0	0	1800110	16918	909	0.01	1319.09
					0.05	3	2	0	5398310	24813	1390	0.22	23105.20
pmed39	900	10	9423	0.5	0.50	1	0	0	1800110	22678	1122	0.12	1435.65
					0.95	1	0	0	1800110	18184	1034	0.04	1391.12
					0.05	3	1	0	5398297	36049	2062	0.15	19563.90
				1	0.50	1	0	0	1800110	33868	1918	0.20	4201.49
					0.95	1	0	0	1800110	64984	3258	1.00	2268.46

Continued on next page

Table C.1: Conclusion.

						B&P							
Instance	***n***	***p***	***opt***	***z***	***α***	**ts**	**th**	**np**	**ngc**	**cost**	**sc**	**mc(%)**	**t(*s*)**
pmed40	900	90	5128	0	0.05	1	0	0	1800270	18307	1014	0.13	4139.89
					0.50	1	0	0	1800270	18299	1018	0.13	1862.09
					0.95	1	0	0	1800270	18427	1015	0.12	1431.95
				0.5	0.05	461	21	222	3704031	26970	2037	0.61	22098.00
					0.50	1	0	0	568170	41498	2150	0.15	485.02
					0.95	1	0	0	11070	29909	1570	0.13	7.88
				1	0.05	382	78	182	348738	24976	1586	0.26	843.56
					0.50	1	0	0	11970	120165	5931	0.45	15.41
					0.95	1	0	0	1800270	1017980	43245	1.00	1015.49