UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

RAFAEL DE SANTIAGO

Efficient Modularity Density Heuristics in Graph Clustering and Their Applications

Thesis presented in partial fulfillment
of the requirements for the degree of
Doctor of Computer Science

Advisor: Prof. Dr. Luís da Cunha Lamb

Porto Alegre
June 2017

CIP – CATALOGING-IN-PUBLICATION

Santiago, Rafael de

Efficient Modularity Density Heuristics in Graph Clustering and Their Applications / Rafael de Santiago. – Porto Alegre: PPGC da UFRGS, 2017.

158 f.: il.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2017. Advisor: Luís da Cunha Lamb.

1. Clustering. 2. Modularity density maximization. 3. Heuristic search. 4. Multilevel heuristics. 5. Local search. 6. Column generation. I. Lamb, Luís da Cunha. II. Efficient Modularity Density Heuristics in Graph Clustering and Their Applications.

In memoriam of my loving brothers

Elisangela M. Miranda and Luís Fernando Máximo

"Menor que meu sonho não posso ser."

— Lindolf Bell

ACKNOWLEDGEMENTS

First, I would like to thank my family. My mother (Vania), father (Mario), brother (Diego), and my wife (Lucia) for all support during my life. Even in the darkest moments, even when I was wrong, they helped me to understand my mistakes and get up to try again. Especially my dear wife who believe in my dreams as if they were hers.

I would like to thank John Steele Weickert who supported me with English classes. Without him, my papers would never be accepted.

I would like to thank my advisors, colleagues, friends, and teachers André Luís Alice Raabe, Cesar Albenes Zeferino, Luís Carlos Martins, and Rudimar Luís Scaranto Dazzi. They turned this dream possible, helping me to receive the support of the university where I work (UNIVALI). Especially Prof. Cesar who presented me to the most important person in this process: my advisor.

When I was looking for a PhD program, my advisor was the only professor who believed in my intention to do a good job. For all psychological and technical support, thank you Prof. Luís C. Lamb! You made this possible!

Prof. Luciana Salete Buriol and Prof. Marcus Rolf Petter Ritt are special people, inspiring teachers who always helped me when I looked for them. I thank them for the patience and lessons.

I am grateful to LIA team. They followed every step of this journey and showed strong confidence in my command. In this team, I would like to thank especially, Alex Rese, Eduardo Santos, Fernando Concatto, Rafael Marini, Rafael Schmitt, Rodrigo Lyra, Rudimar Dazzi, and Thiago Pereira.

I would like to thank my colleagues in the PhD. They helped me to avoid the madness and focus on the important things. They are André Pereira, Clávison Zapelini, Diego Noble, Felipe Grando, Gabriel Ramos, Marcelo Souza, and Ricardo Grunitzki.

The examination board helped to improve the presentation of our results. Without this review, the publication of the thesis results could be harder. I would like to thank Prof. Felipe Meneguzzi, Prof. Leandro Wives, Prof. Ricardo Araujo, and Prof. Roberto da Silva.

Finally, I would like to thank UFRGS and UNIVALI. UFRGS is a great university that helped me to pursue the objective of being a Doctor in Computer Science; I am grateful especially to the PPGC program. UNIVALI is the university that made my academic life possible since my undergraduate days, supporting my PhD for most of the time; I am grateful especially to Prof. Valdir Cechinel Filho. In the name of these universities, I would like to thank their staff for direct and indirect support.

ABSTRACT

Modularity Density Maximization is a graph clustering problem which avoids the resolution limit degeneracy of the Modularity Maximization problem. This thesis aims at solving larger instances than current Modularity Density heuristics do, and show how close the obtained solutions are to the expected clustering. Three main contributions arise from this objective. The first one is about the theoretical contributions about properties of Modularity Density based prioritizers. The second one is the development of eight Modularity Density Maximization heuristics. Our heuristics are compared with optimal results from the literature, and with GAOD, iMeme-Net, HAIN, BMD-λ heuristics. Our results are also compared with CNM and Louvain which are heuristics for Modularity Maximization that solve instances with thousands of nodes. The tests were carried out by using graphs from the "Stanford Large Network Dataset Collection". The experiments have shown that our eight heuristics found solutions for graphs with hundreds of thousands of nodes. Our results have also shown that five of our heuristics surpassed the current state-of-the-art Modularity Density Maximization heuristic solvers for large graphs. A third contribution is the proposal of six column generation methods. These methods use exact and heuristic auxiliary solvers and an initial variable generator. Comparisons among our proposed column generations and state-of-the-art algorithms were also carried out. The results showed that: (i) two of our methods surpassed the state-of-the-art algorithms in terms of time, and (ii) our methods proved the optimal value for larger instances than current approaches can tackle. Our results suggest clear improvements to the state-of-the-art results for the Modularity Density Maximization problem.

Keywords: Clustering. modularity density maximization. heuristic search. multilevel heuristics. local search. column generation.

LIST OF ABBREVIATIONS AND ACRONYMS

AP	Auxiliar Problem
AP-ILS	Auxiliar Problem Iterated Local Search
AP-LS	Auxiliar Problem Local Search
CNM	Clauset, Newman, and Moore heuristic
CM	Coarsening Merger
GAOD	Genetic Algorithm to Optimize D
HAIN	Hybrid Artificial Immune Network
HLSMD	Hybrid Local Search for Modularity Density Maximization
LIA	Laboratory of Applied Intelligence (*Laboratório de Inteligência Aplicada*)
LNM	Local Node Moving
MCN	Move and Coarse Nodes
MD	Multilevel Density
MDM	Multilevel Density Modularity
MILP	Mixed Integer Linear Programming
MP	Master Problem
PPGC	*Programa de Pós-Graduação em Computação*
RMP	Reduced Master Problem
UFRGS	Federal University of *Rio Grande do Sul*
UNIVALI	University of *Vale do Itajaí*

LIST OF FIGURES

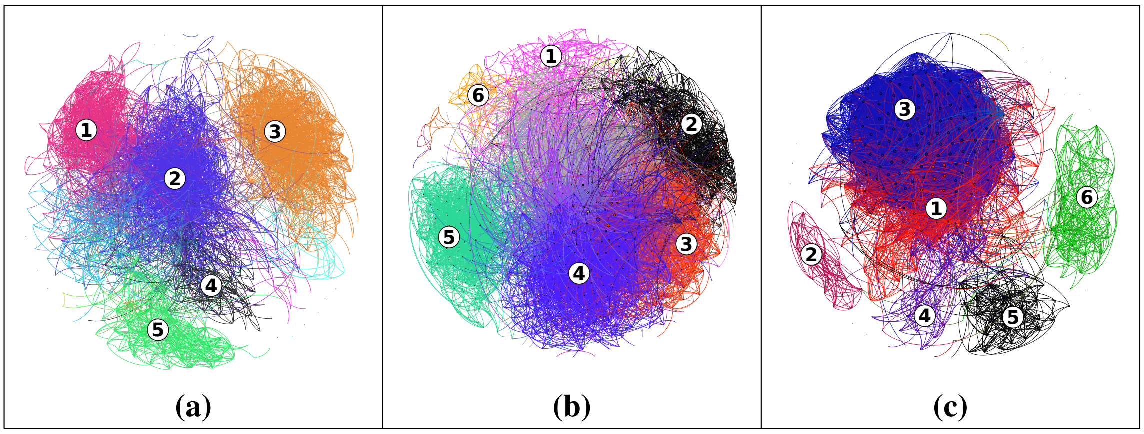
CONTENTS

1 INTRODUCTION

Graph clustering identification has recently attracted interest from several areas of Computer Science (CHAN; YEUNG, 2011; EATON; MANSBACH, 2012; FORTUNATO; CASTELLANO, 2012; XIE; KELLEY; SZYMANSKI, 2013; TIAN et al., 2014; BARCZ et al., 2015; PEI; CHAKRABORTY; SYCARA, 2015; SANKAR; RAVINDRAN; SHIVASHANKAR, 2015). Identifying clusters in networks is a relevant application in different fields, e.g.: (*i*) astronomy, for automatic stellar cluster recognition (SCHMEJA, 2011); (*ii*) biology, for finding protein complexes (NEPUSZ; YU; PACCANARO, 2012) and mapping metabolic reactions (GUIMERA; AMARAL, 2005); (*iii*) health sciences, to identify functional memory involved in olfactory recognition (MEUNIER et al., 2014); (*iv*) social sciences, to recognize criminal organizations (FERRARA et al., 2014; CALDERONI; BRUNETTO; PICCARDI, 2017).

An example of a recent motivation of such a problem can be seen in the clustering solutions for three social graphs of Figure 1.1. The clusterings of the figure are found from the Facebook profile of three different people called (a), (b), and (c). Each connection is a friendship between two friends of the respective “owner” of such graph. The clustering method did not request any other information than the graph, although it discovered meaningful groups. For the graph in (a), cluster 1 has people who worked with him in the past; cluster 2 is composed of people of (a)’s hometown; cluster 3 is composed of people who currently works in with (a); clusters 4 and 5 have the relatives of (a) and (a)’s fianceé. For the graph in (b), cluster 1 is composed of (b)’s family; cluster 6 has relatives of (b)’s wife; clusters 2, 3, 4, and 5 are composed of teachers and students of the Information Systems, Internet Systems, Computer Science, and Mechanical Engineering courses respectively, that are placed where (b) is a teacher. For the graph in (c), clusters 1 and 4 are composed of the family of (c)’s mother and father respectively; co-workers are identified in cluster 2; people who lived in the (c)’s hometown are in cluster 3; clusters 5 and 6 are composed of his colleagues when he studied in Biology and Computer Science courses respectively. These results could be used for shopping recommendation and criminal investigations.

Figure 1.1: Three examples of clustering in social networks.



Usually, clusters can be detected from graphs where nodes are entities in the real world, and edges denote the relationship between two nodes. The methods used for detection are typically applied to overlapping and non-overlapping clusters. In the first group, a node can (possibly) belong to several clusters, whereas in the second, each node belongs to a single cluster.

To detect non-overlapping clusters, some methods only use the graph topology. The strength of relationships can be expressed as edge or arc weights. In these methods, the modular property of a set of nodes is used to detect clusters. The modular property of a set of nodes is characterized by a higher density of edges inside than outside clusters (RADICCHI et al., 2004; FORTUNATO; CASTELLANO, 2012; XIE; KELLEY; SZYMANSKI, 2013).

Modularity optimization has been first introduced by Newman and Girvan (2004) and uses the modular feature to determine which nodes belong to each cluster. A modularity solution is a partition of disjoint nodes that maximizes an objective function which represents the difference between the internal and the expected cluster connectivity.

However, there are some degeneracies of Modularity Maximization reported in the literature. The "resolution limit" has been proved by Fortunato and Barthélemy (2007) in which cliques with a different number of nodes are merged into the optimal solution, even if they are connected by a single edge. This behavior is a degeneracy for Modularity Maximization because the modular property is violated for optimal partitions. Two other degeneracies are reported by Good, Montjoye and Clauset (2010): there is an exponential number of suboptimal solutions, and the number of nodes has a positive correlation with the optimal modularity value. Efforts to avoid these degeneracies are made by reformulating the objective function of Modularity Maximization in, e.g. (MUFF; RAO; CAFLISCH, 2005; ARENAS; FERNANDEZ; GOMEZ, 2008; LI et al., 2008; CAFIERI; HANSEN; LIBERTI, 2010; TRAAG; DOOREN; NESTEROV, 2011; GRANELL; GÓMEZ; ARENAS, 2012; CHAKRABORTY et al., 2013; CHEN; NGUYEN; SZYMANSKI, 2013; CHEN; KUZMIN; SZYMANSKI, 2014).

One of these Modularity Maximization reformulations is known as Modularity Density Maximization. The Modularity Density Maximization problem has been introduced by Li et al. (2008). It has a new objective function which uses the number of nodes of each cluster to normalize the objective value instead of the total number of edges. The objective function maximizes the sum of all differences between the internal and external connectivity of each cluster.

The exact solving of the Modularity Density Maximization is a binary nonlinear problem (0–1 NLP) which makes use of conventional solvers difficulty. Li et al. (2008) have presented exact binary non-linear (0-1 NLP) mathematical programming which makes it difficult to use with common solvers. Karimi-Majd, Fathian and Amiri (2014) have shown an improvement to the Li et al. (2008) model, where the parameter of the number of clusters is not required. Costa (2015) has created four different mixed integer linear models converted from the 0-1 NLP. Instances with at most 40 nodes have been tested, and the results point to the difficulty in the exact solving.

Heuristics for Modularity Density Maximization have been reported in Liu and

Zeng (2010), Gong et al. (2012), and Karimi-Majd, Fathian and Amiri (2014). Liu and Zeng (2010) have presented the genetic algorithm GAOD, Gong et al. (2012) have presented the memetic algorithm iMeme-Net, and Karimi-Majd, Fathian and Amiri (2014) have created a hybrid artificial immune network heuristic HAIN. Karimi-Majd, Fathian and Amiri (2014) have reported solutions for instances with at most 6,594 edges and compared the HAIN with GAOD and iMeme-Net. The HAIN method has surpassed them by finding higher Modularity Density Maximization values in less time (KARIMI-MAJD; FATHIAN; AMIRI, 2014). Costa et al. (2016) have presented eight divisive heuristics for Modularity Density Maximization that have provided solutions close to the optimal one. These divisive heuristics are based on mathematical programming formulations. Izunaga, Matsui and Yamamoto (2016) have introduced methods to obtain lower and upper bounds on the Modularity Density Maximization objective value. As regards lower bounds, they have created a hybrid heuristic that combines a spectral method and dynamic programming; for upper bounds, they have used a variant of semidefinite programming called 0-1 SDP.

The current Modularity Density Maximization heuristics reported in the literature are limited to solve instances of less than $5,000$ nodes in over 500 seconds. This time has been obtained by HAIN. In contrast, there are heuristics for Modularity Maximization that solve instances with more than $300,000$ nodes in less than 5 seconds (see Louvain method in Section 2.1.1.2). Thus, we still lack a Modularity Density Maximization heuristic that solves similar larger instances to enable the comparison among heuristics that use Modularity Density Maximization and Modularity Maximization-based functions for graphs with more than $5,000$ nodes.

Large graphs are the usual instances for clustering problems. For example, Clauset, Newman and Moore (2004) have shown a clustering detection method in a graph with hundreds of thousands of nodes obtained from items listed on the Amazon web site in August 2003. Palla et al. (2005), have been described instances of protein-protein interactions of the *Saccharomyces cerevisiae* fungus with more than $30,000$ nodes. In the "Stanford Large Network Dataset Collection" (LESKOVEC; KREVL, 2014), there are several graphs obtained from social, web communities, communication, citation and signed networks with over tens of thousands of nodes. These are some examples that show the importance of clustering detection in large networks.

1.1 Objective

The main goal of this thesis is to solve larger instances than the current Modularity Density Maximization heuristics are able to tackle, and analyze how close the obtained solutions are to the expected clustering.

1.2 Method

The research of the thesis is mostly quantitative because it looks for heuristics for Modularity Density Maximization. These heuristics were evaluated by considering the runtime, the reached objective value, and ground truth analyses.

The research for new improved heuristics for Modularity Density Maximization was divided into the following two main lines:

1. Constructive and multilevel heuristics: development of novel Modularity Density Maximization heuristics inspired by CNM (CLAUSET; NEWMAN; MOORE, 2004), Louvain (BLONDEL et al., 2008), and multilevel methods (ROTTA; NOACK, 2011) which are heuristics that find solutions for the Modularity Maximization problem;
2. Column generation algorithms: development of heuristics based on relaxations over exact mathematical models of Li et al. (2008), Karimi-Majd, Fathian and Amiri (2014), and Costa (2015) by using column generation methods.

The results were compared with the existing literature data for exact and heuristic methods for Modularity Density Maximization. The amortized complexity was calculated for each novel heuristic.

The resulting partitions of the experiments were used in ground truth experiments with graph instances in which expected partitions are known. These experiments are also used in Lancichinetti et al. (2011), Lancichinetti and Fortunato (2012), Pizzuti (2012), Gong et al. (2012), Jiang and McQuay (2012), Chakraborty et al. (2013), Meo et al. (2013), Xie, Kelley and Szymanski (2013), Darst, Nussinov and Fortunato (2014), Meo et al. (2014), Hric, Darst and Fortunato (2014), Jarukasemratana and Murata (2014), Jia et al. (2014), Karimi-Majd, Fathian and Amiri (2014), Meunier et al. (2014), Park and Lee (2014), Sun (2014), Zhao et al. (2015). To measure the closeness of the heuristic results to the expected partition, we used the "Matthews Correlation Coefficient" (ϕ) that takes into account true and false positives, and true and false negatives (MATTHEWS, 1975). These experiments enabled the verification of the distance between heuristic results and expected partitions.

1.3 Contributions of this Research

The results of this research can be divided into three main contributions: (*i*) the theoretical results over failures of the Modularity Density Maximization based function, (*ii*) the eight new constructive and multilevel heuristics, and (*iii*) the six new column generation algorithms.

Our theoretical results showed that the Modularity Density Maximization objective function fails when it is used in some constructive heuristics. So, we present an alternative prioritizer function that detects clusters with densely connected nodes. A feature of this alternative function is also discussed for star-shaped clusters, suggesting future researches in this direction.

In the context of heuristic contributions, some of the eight new constructive and multilevel heuristics surpassed the objective function value reported by iMeme-Net, HAIN, and BMD-λ for some real graphs. Ground truth experiments in artificial random graphs were performed and suggest that some of our heuristics lead to better cluster detection than CNM and Louvain which are known clustering heuristics used in large graphs.

The algorithmic contributions are about six new column generation methods. Comparisons of our proposed methods with state-of-the-art algorithms showed that: (*i*) two of our methods surpass the exact state-of-the-art algorithms in terms of time, and (*ii*) our methods provide optimal values for larger instances than current approaches can tackle.

1.4 Contributions to the Literature

The following publications result from our research:

- SANTIAGO, R.; LAMB, L. C. Identifying Relationship Patterns Inside Communities. **International Joint Conference on Artificial Intelligence School - Doctoral Consortium**. 2014. **(poster paper)**
- SANTIAGO, R.; LAMB, L. C. On the Role of Degree Influence in Suboptimal Modularity Maximization. In: **Proceedings of IEEE Congress on Evolutionary Computation**. Vancouver: IEEE, 2016. p. 4618–4625. ISBN 978-1-5090-0622-9. **(full paper)**
- SANTIAGO, R.; LAMB, L. C. Efficient Stochastic Local Search for Modularity Maximization. In: **Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion**. Denver: ACM, 2016. p. 51–52. ISBN 9781450343237. **(poster paper)**
- SANTIAGO, R.; ZUNINO, W.; CONCATTO, F.; LAMB, L. C. A New Model and Heuristic for Infection Minimization by Cutting Relationships. In: HIROSE, A. et al. (Ed.). **Lecture Notes in Computer Science**. Kyoto: Springer International Publishing, 2016. v. 9948, cap. Neural Inf, p. 500–508. ISBN 978-3-319-46671-2. **(full paper)**
- SANTIAGO, R.; LAMB, L. C. Efficient modularity density heuristics for large graphs. **European Journal of Operational Research**, v. 258, n. 3, p. 844–865,

May 2017. **(full paper)**

- SANTIAGO, R.; LAMB, L. C. Exact computational solution of Modularity Density Maximization by effective column generation. **Computers & Operations Research**, v. 86, n. 3, p. 18–29, October 2017. **(full paper)**
- SCHMITT R., RAMOS P.; SANTIAGO, R.; LAMB, L. C. Novel Clique Enumeration Heuristic for Detecting Overlapping Clusters. In: **IEEE Congress on Evolutionary Computation**. Donostia: IEEE, 2017. **(full paper)**
- CONCATTO, F.; ZUNINO, W.; GIANCOLI, L.; SANTIAGO, R.; LAMB, L. C. Genetic Algorithm for Epidemic Mitigation by Removing Relationships. In: **Genetic and Evolutionary Computation Conference**. Berlin: ACM, 2017. **(full paper)**
- SANTIAGO, R.; LAMB, L. C. Efficient Quantitative Heuristics for Graph Clustering. In: **Genetic and Evolutionary Computation Conference Companion**. Berlin: ACM, 2017. **(poster paper)**

1.5 Contributions to Scientific Projects

The author of this thesis is the lead researcher at Laboratory of Applied Intelligence (LIA, *Laboratório de Inteligência Aplicada*) of University of *Vale do Itajaí*. In the following, we list the scientific projects developed at LIA that are motivated by our preliminary results and were/are advised by the author of this thesis.

- MARINI, R. Variable Neighborhood Search for the Modularity Clustering Problem, 2014 - 2015. Sponsored by the government of the state of Santa Catarina through law 170;
- SANTOS, E. S. Ant Colony Optimization for the Modularity Clustering Problem, 2014 - 2015. Sponsored by the government of the state of Santa Catarina through law 170;
- RESE, A. L. R. Particle Swarm Optimization for the Modularity Clustering

Problem, 2014 - 2015. Computer Science undergraduate dissertation;

- MARINI, R. Iterated Local Search for the Modularity Clustering Problem, 2014 - 2015. Computer Science undergraduate dissertation;
- CÂNDIDO, G. Modularity for Overlapping Community Detection, 2015. Computer Science undergraduate dissertation;
- RAMOS, P. Overlapping Community Detection by using the "Clique Percolation Method", 2015. Computer Science undergraduate dissertation;
- ZUNINO, W. Heuristic for Mitigation of the Contagious Spreading in Networks, 2015 - 2016. Sponsored by the government of the state of Santa Catarina through law 170;
- SCHMITT, R. O. Clique Heuristic for the Overlapping Community Detection, 2015 - 2016. Sponsored by the government of the state of Santa Catarina through law 170;
- BARAGATTI, M. V. Heuristic for the Maximization of the Contagious Spreading, 2015 - 2016. Computer Science undergraduate dissertation;
- MORAES, R. R. E. Column Generation Heuristic for the Community Detection, 2015 - 2016. Computer Science undergraduate dissertation;
- BRUZACA, M. M. A* Algorithm for the Modularity Density Problem, 2016. Computer Science undergraduate dissertation;
- LOPES, M. E. P. Hybrid Local Search for the Signed Modularity Density Problem, 2016. Computer Science undergraduate dissertation;
- IZIDORO, M. F. Parallel Heuristic for the Modularity Density Problem, 2016. Computer Science undergraduate dissertation;
- SOUZA, M. L. Constructive Searches Analyses for the Community Detection, 2015 - 2017. Sponsored by the government of the state of Santa Catarina through law 171;
- LUCHTENBERG, G. Column Generation Heuristic for the Community

Detection, 2015 - 2017. Sponsored by the government of the state of Santa Catarina through law 171;

- SCHMITT, R. O. Neural Network for the Overlapping Community Detection, 2016 - 2017. Sponsored by the government of the state of Santa Catarina through law 170;
- GALVAGNO, I. Heuristic to Manage Resources in the Control of Aedes Aegypt, 2016 - 2017. Sponsored by the government of the state of Santa Catarina through law 170;
- ARALDI, J. E. Bat Heuristic for the Modularity Density Problem, 2016 - 2017. Computer Science undergraduate dissertation;
- PERUSSOLO, L. C. Competitive Contagious Spreading Analyses, 2016 - 2017. Computer Science undergraduate dissertation;
- GIANCOLLI, L. A. Minimization of Contagious Spreading in Weighted and Directed Networks, 2016 - 2017. Computer Science undergraduate dissertation;
- SCHMITT, R. O. Shiokawa-Fujiwara-Onizuka Heuristic Adaptation for Signed Networks, 2016 - 2017. Sponsored by the government of the state of Santa Catarina through law 170.

1.6 Organization

The remainder of this document is divided into seven coming chapters:

- Chapter 2 presents the background of Modularity Maximization and Modularity Density Maximization problems;
- Chapter 3 shows the theoretical contributions of this work;
- Chapter 4 and 5 are about our heuristic and algorithm contributions respectively;
- Chapter 6 presents the ground truth analyses, comparing our heuristics and the optimal solutions of Modularity Maximization and Modularity Density Maximization;

- Chapter 7 concludes the thesis and suggests further investigations.

2 BACKGROUND

In this chapter, we review the literature related to the thesis research. The first section is on the Modularity Maximization problem, its main properties and methods. Although this problem is not listed in the thesis objectives, the Modularity Maximization is important because it has degeneracies that inspired the development of the Modularity Density Maximization problem which is related to our objectives. The second section is about the Modularity Density Maximization problem, presenting basic concepts, properties, and known methods to solve it.

2.1 Modularity Maximization

In 2004, Newman and Girvan proposed a function to evaluate partitions composed of modular clusters (NEWMAN; GIRVAN, 2004). The work formalized the Modularity Maximization problem and introduced two heuristics for this problem, called “shortest-path betweenness” and “random walks”, based on earlier studies by Newman (NEWMAN, 2003).

The objective function of this search problem has two main components. The first one is a gain factor which is the total number of edges inside the cluster. The second one is the penalty factor and is the sum of the probability of each pair of nodes from the same cluster being connected. The two components are normalized by the total number of edges of the graph. Thus, a Modularity Maximization method has to find the partition which has the maximal difference between the number of edges inside of the clusters and the expected number of edges.

The evaluation function was initially designed for undirected graph instances. The graph $G = (V, E)$ is the input of the problem, where V is the set of nodes, and E the set of edges. The evaluation for undirected graphs can be seen in Equation (2.1), where d_i is the degree of node $i \in V$. Equation (2.1) is the modularity function for a given partition C. Partition C is a set of disjoint clusters. Versions of the Modularity Maximization objective function for weighted and directed graphs are reported in Newman (2004) and Leicht and Newman (2008) respectively.

$$Q(C) = \frac{1}{2|E|} \sum_{c \in C} \sum_{i,j \in c} \left(a_{ij} - \frac{d_i d_j}{2|E|} \right) \tag{2.1}$$

Modularity Maximization is a hard computational problem (BRANDES et al., 2008). This feature has led the research for heuristic methods, as seen in Agarwal and Kempe (2008), Aloise et al. (2013), Clauset, Newman and Moore (2004), Djidjev and Onus (2013), Krzakala et al. (2013), Nascimento and Pitsoulis (2013), Newman (2013), Pizzuti (2012), Rotta and Noack (2011). There are also efforts in the field of mathematical programming to propose exact methods (ALOISE et al., 2010; BRANDES et al., 2008; XU; TSOKA; PAPAGEORGIOU, 2007).

Some issues have been reported for Modularity Maximization. The "resolution limit" in Modularity Maximization has been proved by Fortunato and Barthélemy (2007). They found that Modularity Maximization may fail to identify clusters smaller than a specific scale. For optimal solutions of Modularity Maximization, different cliques may be merged into a single cluster, even if they are connected by a single edge. So, important substructures can be missed into the optimal solution. This issue can most likely occur with clusters which have less than $\sqrt{2m}$ internal edges, where m is the number of edges of the graph. This behavior is an issue in modularity-based problems because the modular property is broken in optimal solutions. Two other issues are reported by Good, Montjoye and Clauset (2010): there is an exponential number of suboptimal solutions, and the number of nodes have a positive correlation with optimal modularity value. The degeneration about the number of exponential solutions is addressed by our results reported in Appendix A. Efforts to avoid the resolution limit are made by reformulating the objective function of Modularity Maximization (ARENAS; FERNANDEZ; GOMEZ, 2008; CAFIERI; HANSEN; LIBERTI, 2010; CHAKRABORTY et al., 2013; CHEN; NGUYEN; SZYMANSKI, 2013; CHEN; KUZMIN; SZYMANSKI, 2014; GRANELL; GÓMEZ; ARENAS, 2012; LI et al., 2008; MUFF; RAO; CAFLISCH, 2005; TRAAG; DOOREN; NESTEROV, 2011).

The following subsection presents heuristics for Modularity Maximization. They inspired some of the contributions reported in this thesis which are described in Chapter 4.

2.1.1 Heuristics for Modularity Maximization

As an NP-Hard problem, Modularity Maximization cannot solve large instances in polynomial time. In this section, heuristics for large graphs are described for the Modularity Maximization problem, which inspired the creation of our eight novel heuristics presented in Chapter 4.

2.1.1.1 CNM

Clauset, Newman and Moore (2004) reported that a heuristic for Modularity Maximization called CNM (or Fast Greedy) resulted in high modularity partitions for graphs with over 400,000 nodes and 2,000,000 edges. CNM is a constructive heuristic, where the set of data structures lead to fast results. Binary heaps are used to store the highest option when the search is constructing the solution, applying the ΔQ function (2.2), to identify the difference between the incumbent solution and the next movement. The method has a time complexity of $O(|E| d \log |V|)$, where d is the depth of the dendrogram. In recent years, a new version of heuristic replaced the binary with Fibonacci heaps.

In this heuristic, the starting partition is composed of $|V|$ singleton clusters, where each node belongs to a cluster of size 1. The matrix ΔQ_{ij} is filled with the gain value from merging each pair of i and j clusters, calculated by Function (2.2). Before the first iteration could be performed, a total of $|V|$ binary max heaps are created and assigned to each line of the matrix ΔQ_{ij}, plus a heap, called H that stores the highest values of each line heap. For each starting cluster, a data structure a_i is defined with the value

${}^{d_i}/_{2m}$. For each iteration, the pair of clusters with the highest ΔQ_{ij} value is merged.

$$\Delta Q(i,j) = \begin{cases} \frac{1}{2|E|} - \frac{d_i S d_j}{4|E|^4}, & \text{if } i \text{ and } j \text{ are connected} \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

The following steps are performed during each iteration after the preparation of data structures:

1. select the highest gain stored in the heap H, to decide the two clusters to be merged;
2. update ΔQ_{ij} and a_i;
3. go to step 1, until only one cluster remains.

The updates in the matrix are performed using rules for different merge configurations. Clusters i and j are selected to be merged: (*i*) if a cluster k is connected to both i and j, then

$$\Delta Q'_{jk} = \Delta Q_{ik} + \Delta Q_{jk}; \tag{2.3}$$

(*ii*) if a cluster k is connected to i and is not connected to j, then

$$\Delta Q'_{jk} = \Delta Q_{ik} - 2a_j a_k; \tag{2.4}$$

and (*iii*) if a cluster k is connected to j and is not connected to i, then

$$\Delta Q'_{jk} = \Delta Q_{jk} - 2a_i a_k. \tag{2.5}$$

2.1.1.2 Louvain Method

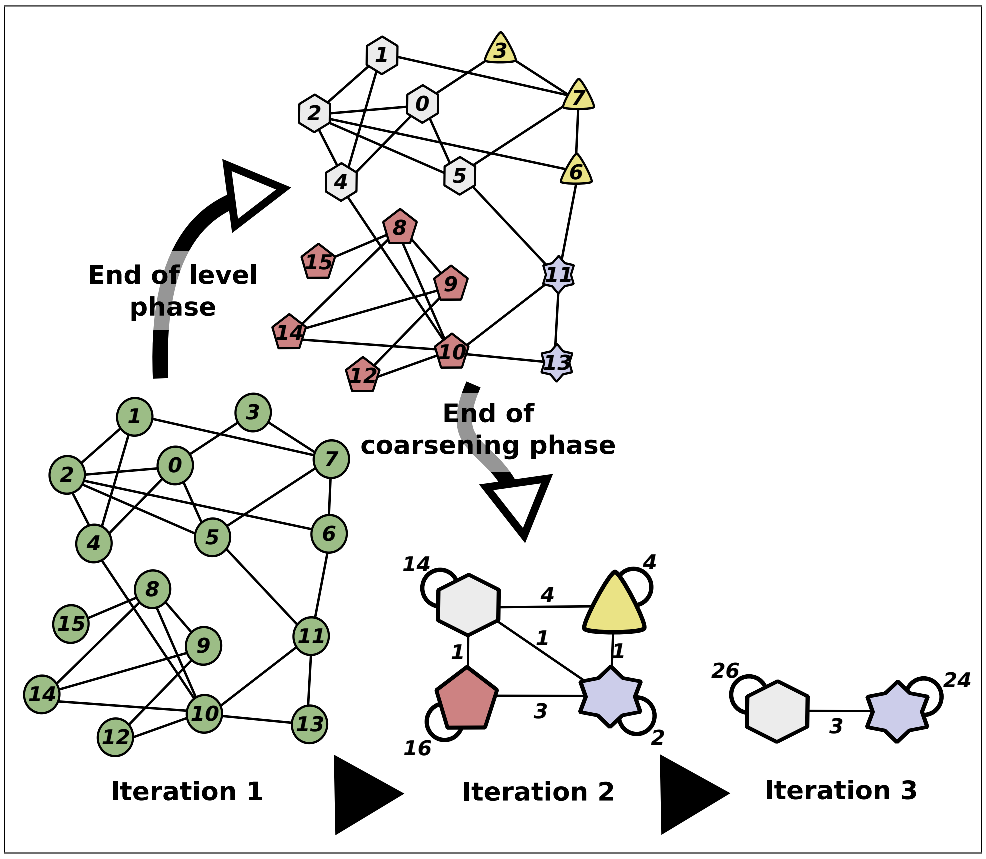
Blondel et al. (2008) created a local search heuristic to find modular clusters in large graphs. The method is seen in Algorithm 1. Given a graph $G = (V, E)$, the

Algorithm 1: Louvain method (BLONDEL et al., 2008).

```
   Input  : $G(V, E)$
1  $partition \leftarrow \{\{v\} : v \in V\}$
2  $best \leftarrow partition$
3  $improvement \leftarrow true$
4  while $improvement$ do
5      $improvement \leftarrow false$
       // level phase
6      repeat
7          $moves \leftarrow 0$
8          foreach $v \in V$ do
9              $\Delta Q_v \leftarrow 0$
10             $c_v \leftarrow \emptyset$
11             foreach $c \in C_{N(v)}$ do
12                 if $\Delta Q_v < \Delta Q_{coarse}(v, c)$ then
13                     $\Delta Q_v \leftarrow \Delta Q_{coarse}(v, c)$
14                     $c_v \leftarrow c$
15             if $c_v \neq \emptyset$ then
16                 $moves \leftarrow moves + 1$
17                 in $partition$, move node $v$ to cluster $c_v$
18     until $moves = 0$
       // coarsening phase
19     if $Q(best) < Q(partition)$ then
20         $best \leftarrow partition$
21         $improvement \leftarrow true$
22         clusters are coarsened as nodes in $partition$ and graph $G(V, E)$
23 return $best$
```

heuristic starts with a partition composed of $|V|$ singleton nodes. After that, iterations are performed until no improvement in the current partition is found. Each iteration is composed of the level and coarsening phases. The level phase executes a local search that removes each node from its clusters and moves it to the cluster that maximizes the Modularity Maximization objective function. This procedure is repeated until no improvement is found. Then the coarsening phase changes the graph instance by transforming the cluster in coarsened nodes. This procedure increases the internal number of edges of each node, and the edges that connect each pair of coarsened nodes

Figure 2.1: Louvain method executed on a graph with 16 nodes, which demanded three iterations.



Source: Adapted from Blondel et al. (2008).

are merged and their weights are summed. Then, a new iteration is made by applying the level phase again. The Louvain method iterates until no improvement is found for either of the two phases. Figure 2.1 shows three iterations of the execution of the Louvain method over a graph with 16 nodes.

Blondel et al. (2008) show that the Louvain method surpasses the Q values (values obtained through the objective function of the Modularity Maximization problem) and is faster than CNM (CLAUSET; NEWMAN; MOORE, 2004), CNM adaptation (WAKITA; TSURUMI, 2007), and the LP method (PONS; LATAPY, 2005). The tested instances are “Karate” with 34 nodes and 77 edges, “Arxiv” with 9,000 nodes and 24,000 edges, “Internet” with 70,000 nodes and 351,000 edges, “Web nd.edu” with 325,000 nodes and 1 million edges, “Phone” with 2.04 million nodes and 5.4 million edges, “Web uk-2015” with 39 million nodes and 783 million edges, and “Web WebBase 2001” with 118 million nodes and 1 billion edges.

2.1.1.3 Multilevel Heuristics

Rotta and Noack (2011) reported experimental results over existing and new coarsening and refinement methods. Their results support a new multilevel heuristic which led to competitive results when compared to already known heuristics for Modularity Maximization.

Coarsening are methods that merge a pair of clusters at each iteration by using a priority criterion. They are divided into single or multi-step coarseners. A single-step coarsener joins a pair of clusters at each iteration until the partitions become a single cluster, or before it if a stopping criterion is met. An example of this kind of method is CNM (CLAUSET; NEWMAN; MOORE, 2004). A multi-step coarsener method iteratively merges l disjoint clusters with the highest priority criteria. Algorithm 2 shows this method.

The refinement methods are heuristics which perform a local search by iteratively moving the nodes from their current clusters to increase the Q value of the partition. Rotta and Noack (2011), divided this kind of heuristic into three subtypes. The first subtype is called the complete greedy refinement and performs the best node movement at each iteration until there is no movement that improves the Q value. This method is seen in Algorithm 3.

The second refinement method is called fast greedy, and it iterates to each node $v \in V$ and moves v to a cluster that increases the modularity value. Algorithm 4 shows the fast greedy heuristic.

The third refinement heuristic is an adaptation of the Kerningan-Lin method

Algorithm 2: Multi-step coarsener (ROTTA; NOACK, 2011).

```
   Input  : $G(V, E), l$
1  create a partition composed of the cluster set $\{\{v\} : v \in V\}$
2  while $\exists$ cluster pair $(C, D)$ where $\Delta Q_{(C,D)}$ do
3      $l \leftarrow$ merge fraction $\times |\{(C, D) : \Delta Q_{(C,D)} > 0\}|$
4      mark all clusters as unmerged
5      for $l$ most prioritized pairs $(C, D)$ do
6          if $C$ and D are marked as unmerged then
7              merge clusters $C$ and D
8              mark clusters $C$ and D as merged
```

Algorithm 3: Complete greedy (ROTTA; NOACK, 2011).

```
   Input  : $G(V, E), partition$
1  repeat
2      $(v, D) \leftarrow$ best node move
3      if $\Delta Q_{v \to D} > 0$ then
4          move node v to cluster D
5  until $\Delta Q_{v \to D} \leq 0$
```

Algorithm 4: Fast greedy (ROTTA; NOACK, 2011).

```
   Input  : $G(V, E), partition$
1  repeat
2      foreach $v \in V$ do
3          $D \leftarrow$ best cluster for v
4          if $\Delta Q_{v \to D} > 0$ then
5              move node $v$ to cluster D
6  until no improvement is found
```

(KERNIGHAN; LIN, 1970). It is similar to the fast greedy method, but it tries to escape from a local optimum. At each iteration, the method performs the best node movements. There is no requirement for modularity improvement. This heuristic method is seen in Algorithm 5.

Algorithm 5: Adaptation of Kerninghan-Lin method (ROTTA; NOACK, 2011).

```
   Input : $G(V, E), partition$
1  repeat
2      $peak \leftarrow partition$ mark all nodes as moved
3      while unmoved nodes exist do
4          $(v, D) \leftarrow$ best move with $v$ unmoved
5          move $v$ to cluster D
6          mark $v$ as moved
7          if $Q(partition) > Q(peak)$ then
8              $peak \leftarrow partition$
9          if $k$ has moved since the last peak then
10             break
11         $partition \leftarrow peak$
12 until no improved partition found
13 return $partition$
```

Algorithm 6: Multilevel clustering (ROTTA; NOACK, 2011).

```
   Input : $G(V, E), coarsener, refiner$
   // coarsening phase
1  $level[1] \leftarrow G$
2  for $l \leftarrow 1$ to ... do
3      $level[l + 1] \leftarrow coarsener(level[l])$
4      if no clusters merged then
5          break
6  $l_{max} \leftarrow l$
   // refinement phase
7  $partition \leftarrow$ nodes of the $l_{max}$
8  for $l \leftarrow l_{max} - 1$ to 1 do
9      project $partition$ from $level[l + 1]$ to $level[l]$
10     $partition \leftarrow refiner(level[l], partition)$
```

A multilevel heuristic that combines a single-step greedy coarsener with a fast greedy refiner led to the best results when efficiency is required. The best result without scalable requirement is obtained by replacing the fast greedy with the Kernighan-Lin adaptation presented in Algorithm 5 (ROTTA; NOACK, 2011).

2.2 Modularity Density Maximization

Motivated mainly by the resolution limit degeneracy of Modularity Maximization, Li et al. (2008) created a search problem known as Modularity Density Maximization. Its objective function uses the number of nodes of each cluster to normalize the objective value instead of the total number of edges. The function maximizes the difference between the number of internal and external edges of each cluster. The Modularity Density Maximization function is given by Equation (2.6), where E_c is the set of internal edges of cluster c, and $E_{\overline{c}}$ is the set of edges that connect an internal with an external node.

$$D(C) = \sum_{c \in C} \left(\frac{2E_c - E_{\overline{c}}}{|c|} \right) \tag{2.6}$$

Costa (2015) simplified Equation (2.6) by replacing $E_{\overline{c}}$ with twice the number of internal edges minus the total degree of nodes inside of the cluster c, resulting in Equation (2.7).

$$D(C) = \sum_{c \in C} \left(\frac{2|E_c| + 2|E_c| - \sum_{v \in c} d_v}{|c|} \right) = \sum_{c \in C} \left(\frac{4|E_c| - \sum_{v \in c} d_v}{|c|} \right) \tag{2.7}$$

Function D_λ for quantified Modularity Density Maximization is shown in Equation (2.8). This function is used to obtain the "ratio association" to find small clusters when $\lambda > 0.5$, and the "ratio cut" to find large clusters when $\lambda < 0.5$. With $\lambda = 0.5$, the function is equal to Equation (2.7). Li et al. (2008) suggested that this function can be used to find the appropriate level of topological structure of graphs to find proper partitions.

$$D_\lambda(C) = \sum_{c \in C} \left(\frac{4\lambda|E_c| - (2-2\lambda)\big(\sum_{v \in c} d_v - 2|E_c|\big)}{|c|} \right) \tag{2.8}$$

Li et al. (2008) demonstrated that Modularity Density Maximization avoids resolution limit and preserves modular property. They showed that Modularity Density Maximization does not divide a clique into two parts. This proof is detailed in subsection 2.2.1.1. The optimal partitions can identify modular clusters correctly with different sizes. Experiments confirmed these properties by testing a linear integer exact model to optimize the D value. Artificial graphs were created with already known clusters, and they were submitted to the exact D solver, Girvan and Newman (2002) algorithm and Newman (2006) spectral method. The D solver found better partitions than the other two methods. The real graphs "Karate", "Football" and "Journal Index" were also submitted and showed that D maximization found precise partitions.

Other Objective Functions for Modularity Density Maximization

For weighted graph instances, an equation similar to Function (2.7) is also suggested by Li et al. (2008). Let $W(c,c)$ be the sum of all weights of edges from E_c and $W(c,\overline{c})$ be the sum of all edges from $E_{\overline{c}}$. The Modularity Density Maximization objective function for weighted graphs is seen in Equation (2.9).

$$D_w(C) = \sum_{c \in C} \left(\frac{2W(c,c) - W(c,\overline{c})}{|c|} \right) \tag{2.9}$$

For signed graphs that represent positive and negative connections, Li, Liu and Liu (2014) developed two evolutionary and two memetic algorithms. They extended the original D value functions (Equations (2.8) and (2.9)). This function is in Equation (2.10), where $W^+(c,c)$ is the sum of all positive weights of the edges which are inside of the cluster c, $W^-(c,c)$ is the absolute value of the sum of all negative weights of edges which are inside of the cluster c, $W^+(c,\overline{c})$ is the sum of all positive weights of edges which connect the cluster c to the other cluster, $W^-(c,\overline{c})$ is the absolute value of the sum of all negative weights of edges which connect cluster c to another cluster.

$$\begin{aligned} D_\lambda^{\pm}(C) &= \sum_{c\in C}\left(\frac{2\lambda W^{+}(c,c)+2(1-\lambda)W^{+}(c,\overline{c})}{|c|}\right) \\ &- \sum_{c\in C}\left(\frac{2(1-\lambda)W^{-}(c,c)+2\lambda W^{-}(c,\overline{c})}{|c|}\right) \end{aligned} \quad (2.10)$$

The D optimization was adapted to bipartite graphs, as can be seen in Li et al. (2015). There are two objective functions for bipartite graphs. The first is for an unweighted bipartite graph $G = (U, V, E)$, where U and V are disjoint set of nodes, $u \in U$, and $v \in V$ for all $\{u, v\} \in E$, $E_{U,V}$ is the set of edges between nodes U and V, U_c and V_c are the set of nodes if the cluster c which belong to cluster U and V respectively, and E_{U_c,V_c} is the set of edges between the nodes from U_c and V_c. This function can be seen in Equation (2.11).

$$DB(C) = \frac{1}{|E_{U,V}|}\sum_{c\in C}\left(\frac{|E_{U_c,V_c}|^2}{|U_c|\times|V_c|}\right) \quad (2.11)$$

The second function of Modularity Density Maximization for weighted bipartite graph $G = (U, V, E, W)$, where U and V are the two disjoint sets of nodes, E is the set of edges between U and V, and $W : E \to [0, 1]$ is the function which defines the weight of each edge (LI et al., 2015). The Modularity Density Maximization function for the weighted bipartite graph is in Equation (2.12). The value $W(U, V)$ is the sum of all weights of the edges between U and V, U_c and V_c are the set of nodes which belong to cluster c from U and V respectively, $W(U_c, V_c)$ is the sum of all weights of the edges between U_c and V_c.

$$DB_w(C) = \frac{1}{W(U,V)}\sum_{c\in C}\left(\frac{W(U_c,V_c)^2}{|U_c|\times|V_c|}\right) \quad (2.12)$$

2.2.1 Theoretical Background

This section presents the theoretical properties of Modularity Density Maximization. The resolution limit avoidance for cliques and indivisible clusters are described.

2.2.1.1 Resolution Limit Avoidance

This section shows that Modularity Density Maximization avoids the resolution limit. Li et al. (2008) described three theorems based on the arguments of Fortunato and Barthélemy (2007) to demonstrate that resolution limit degeneracy is avoided. The first theorem explains that Modularity Density Maximization does not divide a clique into two parts. The second one is about the solution of most modular networks, and the third one is about how Modularity Density Maximization detects clusters of different sizes. These three are explained below in the following theorems.

Theorem 1. *Modularity Density Maximization does not divide a clique into two parts (LI et al., 2008).*

Proof. This proof is by contradiction. Suppose that P is a partition which divides a clique into two parts. The first part is C_1 and the second part is C_2. They have n_1 and n_2 nodes, respectively. The number of edges between them is $n_1 \cdot n_2$. Where D_P is the Modularity Density Maximization of partition P, and D_C is the Modularity Density Maximization of a partition which does not divide the clique $C_1 \cup C_2$. D_P and D_C values can be seen in Equation (2.13).

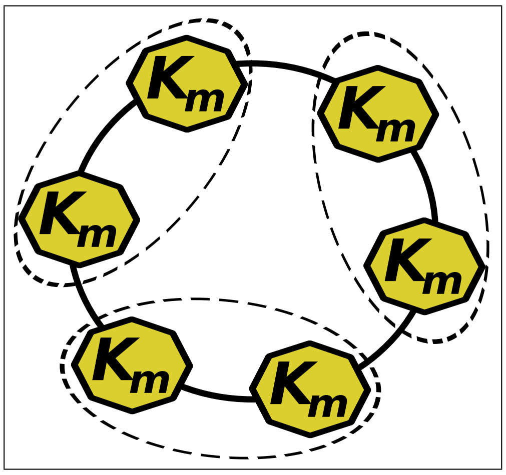
$$\begin{aligned} D_C &= n_1 + n_2 + 1 \\ D_P &= \frac{n_1(n_1) - n_1 n_2}{n_1} + \frac{n_2(n_2 - 1) - n_1 n_2}{n_2} = -2 \end{aligned} \tag{2.13}$$

Since $D_C > D_P$, Li et al. (2008) show that Modularity Density Maximization does not divide a clique. □

Theorem 2. *Modularity Density Maximization resolves most modular networks (LI et al., 2008).*

Proof. To prove this result, Li et al. (2008) used the graph $G = (V, E)$ of the ring of m cliques from Fortunato and Barthélemy (2007) which was used to show the resolution limit for the Modularity Maximization problem. A similar graph can be seen in Figure 2.2. These cliques are called K_m. Each clique K_m has $n > 3$ nodes and ${(n^2-n)}/{2}$ edges. Suppose that there are $m > 2$ cliques, and the number of nodes of this graph is $|V| = nm$ and its number of edges is $|E| = mn(n-1)/2_m$.

Figure 2.2: A graph with six identical clusters connected by single links, composing a ring network.



Source: Adapted from Fortunato and Barthélemy (2007).

Suppose that each cluster of the partition P is composed of one clique K_m. The modular clusters of this graph are the cliques. Each cluster of this partition P has D_{single} value for the objective function of Modularity Density Maximization. Another partition P' is composed of clusters where each one of them has k cliques. The value of the Modularity Density Maximization for this partition is D_k. Equation (2.14) shows the difference $D_{single} - D_k$. This difference is higher than zero, so partition P has a higher D value than P', proving that Modularity Density Maximization clusters do not merge two or more cliques of a graph made up of a ring of them.

$$\begin{aligned} D_{single} - D_k &= m\left(n - 1 - \frac{2}{n}\right) - \frac{m}{k}\frac{kn(n-1) + 2(k-3)}{kn} \\ &> m\left[(n-1) - \frac{3}{n} - \frac{n-1}{2}\right] > 0 \end{aligned} \tag{2.14}$$

□

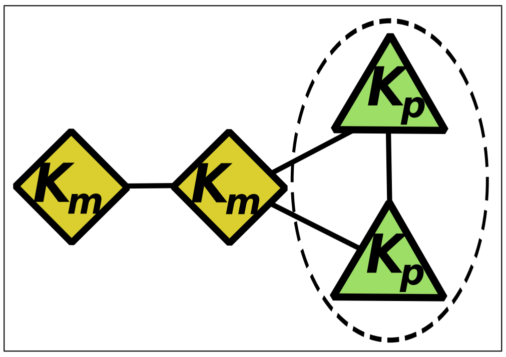
Theorem 3. *Modularity Density Maximization can detect clusters of different sizes (LI et al., 2008).*

Proof. This proof uses the graph in Figure 2.3, which is based on the schematic example of Fortunato and Barthélemy (2007). Suppose a graph $G = (V, E)$ composed of two cliques K_p of size p and two cliques K_m of size m, where $3 \leq p \leq m$. Equation (2.15) shows the difference between the D values of different partitions, where $D_{separate}$ denotes the D value of the partition in which each cluster is a clique, D_{merge} denotes the D value of the partition that has two smaller cliques K_p merged into a single cluster. The difference is larger than zero for $p > 3$. Hence Modularity Density Maximization does not merge the cliques for a graph with the same structure of the graph G.

$$\begin{aligned} D_{separate} - D_{merge} &= \\ \left[\frac{m(m-1)-1}{m} + \frac{m(m-1)-3}{m} + 2(p-1) - \frac{4}{p}\right] \\ - \left[\frac{m(m-1)-1}{m} + \frac{m(m-1)-3}{m} + (p-1)\right] \\ = 2(p-1) - \frac{4}{p} - (p-1) > 0 \end{aligned} \tag{2.15}$$

□

Figure 2.3: A graph with four cliques, where K_m has m nodes, K_p has p nodes, and $p < m$.



Source: Adapted from Fortunato and Barthélemy (2007).

2.2.1.2 Indivisible Clusters

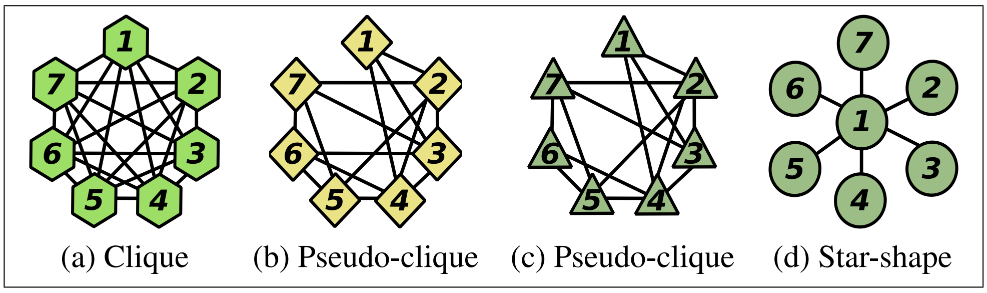
Wang et al. (2008) showed that a ring graph and an ad hoc graph composed of indivisible clusters are not divided into optimal D partitions. The indivisible cluster is a subgraph, where the number of inside edges is more than a half of the outside edges.

Consider a cluster $N = (V_N, E_N)$ where V_N is the set of nodes in N, and E_N is the set of edges in N. The cluster N is indivisible if $|E_N| \geq F(|V_N|)$ (Equation (2.16)), where $|V_N| \geq 4$, and $n = |V_N|$. Cluster N is also indivisible if $|E_N| \geq {}^{(|V_N|^2 - 2|V_N|)}/_{2}$. Figure 2.4 shows five different indivisible clusters.

$$F(n) = \begin{cases} C_n^2 - \frac{n}{2}, & n \geq 4, n \text{ is even} \\ C_n^2 - \frac{n}{2} + 1, & n \geq 4, n \text{ is odd} \end{cases} \tag{2.16}$$

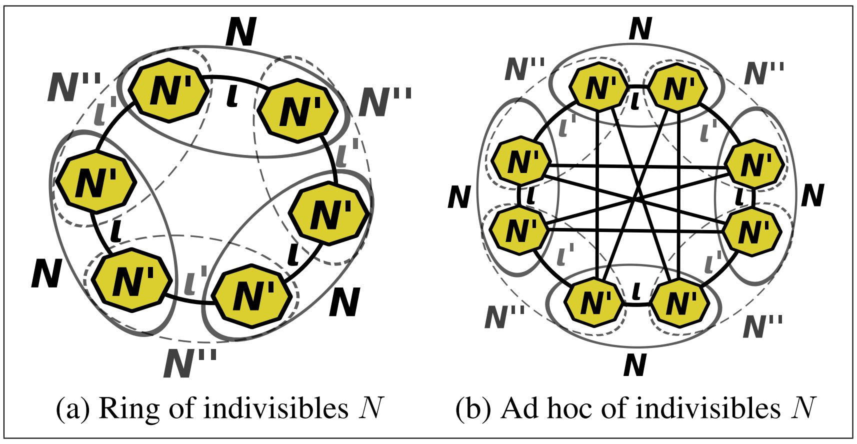
Suppose the graphs of Figure 2.5. For these graphs suppose that Ns are indivisible clusters where their elements are distributed uniformly, the N's are bipartitions of N, and N''s are composed of two adjacent N's from different N clusters. The number of nodes of N' is equal to ${}^{|N|}/_{2}$. The edge set ι connects the two bipartite clusters N' which are from the same N cluster, and the edge set ι' connects the two adjacent

Figure 2.4: Examples of modules that are indivisible clusters for optimal D partitions.



Source: Adapted from Wang et al. (2008).

Figure 2.5: Two graphs to support the explanation about indivisible clusters and optimal D partitions.



Source: Adapted from Wang et al. (2008).

N' clusters which are inside N''. The N clusters make up partition R which follows the definitions of clusters from Radicchi et al. (2004), and N'' clusters compose the recombined partition, R'.

For the ring graph in Figure 2.5 (a), Wang et al. (2008) showed the difference between D values of R and R' partitions. These values are called D_R and $D_{R'}$ respectively. Equation (2.17) shows that Modularity Density Maximization generates a higher value for partition R than partition R' because $\iota < \iota'$.

$$\begin{aligned} D_R - D_{R'} &= \frac{|E|}{|N|}\big(2|E_N| - 2\iota\big) - \frac{|E|}{|N''|}\big(4|E_{N''}| + 2\iota - 2\iota'\big) \\ &= \frac{|E|}{|N|}\big(4\iota' - 4\iota\big) > 0 \end{aligned} \tag{2.17}$$

For the ad hoc graph in Figure 2.5 (b), Wang et al. (2008) showed that Modularity Density Maximization evaluates with a higher D value for R than R'. Equation (2.18) shows that D_R is higher than $D_{R'}$.

$$\begin{aligned} D_R - D_{R'} = |E|\left[\frac{2|E_N| - (|E| - 1)}{|N|}\iota\right] - \frac{|E|}{|N''|}\left[2|E_{N''}| - \big(|E| - 5\big)\iota - 4\iota'\right] \\ = \frac{|E|}{|N|}(4\iota' - 4\iota) > 0 \end{aligned} \tag{2.18}$$

2.2.2 Degeneracies

This section presents two known degeneracies of the Modularity Density Maximization problem. They are negative D for weighted networks and, the possibility that some optimal partitions present nodes in non-adjacent clusters.

2.2.2.1 Negative D_w

Yang and Luo (2009) reported that Function (2.9) presents a degeneration in which some clusters can obtain negative D_w. A negative cluster has internal weights smaller than external weights. To overcome this degeneracy, they present the Normalized Modularity Density Maximization (NMD_w) function, presented here in Equation (2.19), where $W(c, C)$ is the total sum of edge weights which have any endpoint node in cluster c.

$$NMD_w(C) = \sum_{c \in C} \left(\frac{2W(c,c) - W(c,\overline{c})}{W(c,C)} \right) \tag{2.19}$$

Yang and Luo (2009) also show that NMD_w can resolve the resolution limit by using similar examples from Fortunato and Barthélemy (2007).

2.2.2.2 Cluster of Strangers

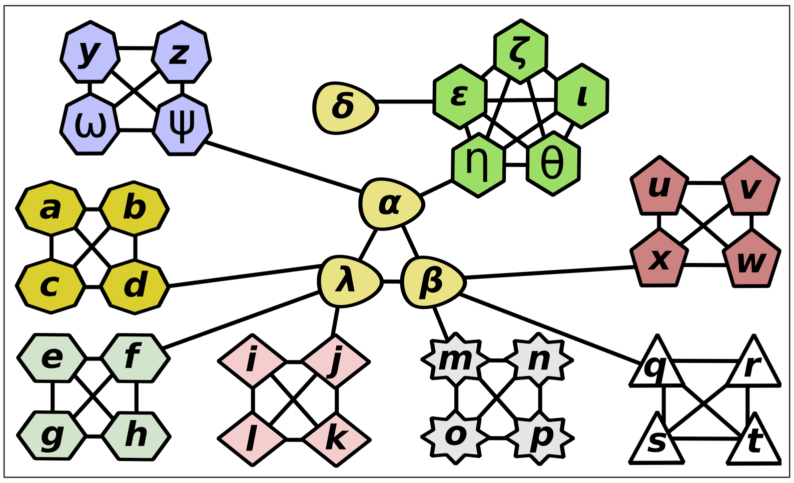
Costa (2014) and Costa (2015) found a degeneration in the Modularity Density Maximization problem. The nodes of degree 1 can belong to a non-neighbor cluster in the optimal partition.

The graph in Figure 2.6 shows this degeneracy. In this figure, the optimal solution consists of nine clusters represented with nodes with nine different shapes. There is a clique with three nodes connected to seven cliques with four nodes and one clique with five nodes. The clique with five nodes is connected to node δ which has degree 1. The D value obtained by considering that it is in the cluster of its neighbor's clique is 22.083. The optimal partition has D=22.1, where each clique belongs to its cluster, and node δ is inside the cluster of the clique with three nodes. This happens on clusters where the optimal value is negative, so by adding a node of degree 1 the denominator of the function D is increased by one, resulting in a better D value.

2.2.3 Heuristics

This section presents three heuristics for the Modularity Density Maximization problem. They are the genetic algorithm GAOD, the memetic algorithm iMeme-Net, and the artificial immune network HAIN.

Figure 2.6: A graph to support the explanation about one-degree node degeneration.



Source: Adapted from Costa (2015).

2.2.3.1 Genetic Algorithm GAOD

Liu and Zeng (2010) created the GAOD genetic algorithm for Modularity Density Maximization. The GAOD is showed in Algorithm 7. The inputs for GAOD are a graph $G = (V, E)$, the number of generations, the probability p_c of the crossover operator, and the probability p_m of the mutation operator. Each chromosome has $|V|$ genes, where each gene represents a node in the graph G. The value assigned to each gene is a cluster identifier.

Algorithm 7: GAOD (LIU; ZENG, 2010).

```
  Input : $G(V, E)$, $generations$, $p_c$, $p_m$
1 initialize $pop$
2 for $g \leftarrow 1$ to $generations$ do
3     evaluate $pop$
4     $selpop \leftarrow selection(pop)$
5     $crosspop \leftarrow crossover(selectpop, p_c)$
6     $pop \leftarrow mutation(crosspop, p_m)$
```

The crossover operator takes two chromosomes. The first one is the source chromosome and the second one is the destination. Multipoint genes are randomly selected from the source and, their values replace the equivalent gene in the destination chromosome.

The mutation operator changes the cluster identifier of the genes with probability p_m. The new identifier assigned is obtained from an adjacent cluster of the node represented by the updated gene. Only chromosomes resulting from the crossover are mutated.

The results were reported for two benchmark graphs. The first was “Karate”, where the GAOD reached D = 7.8451. The second was “Football”, where the maximal D obtained was 43.3701. The experiments also confirmed that Modularity Density Maximization avoids the resolution limit.

2.2.3.2 Memetic Algorithm iMeme-Net

Gong et al. (2012) developed a memetic algorithm called iMeme-Net to detect clusters by using Modularity Density Maximization objective function. The iMeme-Net uses Label Propagation and Elitist strategies.

Algorithm 8 shows iMeme-Net. The population is initialized by using the Label Propagation presented in Algorithm 9. After that, all chromosomes are evaluated, and the best individuals are kept. iMeme-Net repeats four operations. The first operation mutates the population with p_m probability, by changing the label identifier of a node to one of its neighbor labels. This label assigns a cluster to the node. The second operation is the intensification procedure of Simulated Annealing of Algorithm 10. It is applied to the mutated population. The best resulting chromosomes are preserved at the fourth step. Then the population is updated with intensified and elite chromosomes.

Algorithm 9 shows the Label Propagation method used to initialize the population. Each chromosome is defined by labels assigned to each node of the graph. First, a unique label is assigned to each node. Then the labels are updated by using Equation (2.20). The function $\omega(n)$ is the set of neighbor nodes of n, and $nl(j)$ is the label of the cluster that j belongs to. This equation finds the most common label of the

Algorithm 8: iMeme-Net (GONG et al., 2012).

```
   Input  : $G(V,E)$, $popsize$, $generations$, $p_m$, $temperature$
1  $pop \leftarrow PGLP(popsize)$
2  $pop \leftarrow evaluateFitness(pop)$
3  $bestpop \leftarrow keepBestIndividual(pop)$
4  for $g \leftarrow 1$ to $generations$ do
5      $newpop \leftarrow neighborBasedMutate(popsize)$
6      $temppop1 \leftarrow ISACLocalSearch(newpop)$
7      $temppop2 \leftarrow elitismPreservation(temppop1)$
8      $pop \leftarrow UpdatePopulation(temppop1$ & $temppop2)$
9  return $bestDensityFound$, $partitions$
```

neighborhood.

$$nl(n) = argmax_r \sum_{j \in \omega(n)} \delta(l(j), r) \tag{2.20}$$

Algorithm 9: PGLP for iMeme-Net (GONG et al., 2012).

```
    Input  : $G(V,E)$, $iterations$, $popsize$
1   $pop \leftarrow createNewChromosomes(popsize)$
2   for $chromosome \in pop$ do
3       for $j \leftarrow 1$ to $iterations$ do
4           for $k \leftarrow 1$ to $|V|$ do
5               if $node[k].neighbor.size > 1$ then
6                   for $m \leftarrow 1$ to $node[k].neighbor.size$ do
7                       $node[k].label \leftarrow nl(k)$
8               else
9                   $node[k].label \leftarrow node[k].neighbor[1].label$
10  return $pop$
```

Algorithm 10 is a Simulated Annealing local search metaheuristic. First, the heuristic chooses a random chromosome Ω. Its set of neighbors Ω' is found, and D values of these neighbors are calculated using the fitness function. f is the fitness value of Ω, and f_{cmax} is the fitness value of the best neighbor of Ω. The current partition $chrom$ is replaced by the best neighbor if $f < f_{cmax}$, or with probability $e^{-|f - f_{cmax}|/\beta}$.

The constant β is equal to 0.16.

Algorithm 10: ISACLocalSearch for iMeme-Net (GONG et al., 2012).

```
   Input : $G(V,E), \beta, popsize, chromosomes$
 1 $chrom \leftarrow chromosomes[rand(popsize)]$
 2 $\Omega \leftarrow decode(chrom)$
 3 $\Omega' \leftarrow calculateNeighbors(\Omega)$
 4 $f \leftarrow fitness(\Omega)$
 5 $f's \leftarrow fitness(\Omega')$
 6 $f_{cmax} \leftarrow max(f's)$
 7 if $f < f_{cmax}$ then
 8     $chrom \leftarrow chrom(\Omega'_{cmax})$
 9 else
10     if $rand(1) < exp(-|f - f_{cmax}|/\beta)$ then
11         $chrom \leftarrow chrom(\Omega'_{cmax})$
```

Gong et al. (2012) fixed some parameters for the experiments with iMeme-Net. The population size was 100 chromosomes, the number of generations was 5, and the probability of mutation was 0.9. The experiments were carried out with the Modularity Density Maximization ratio $\lambda \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$ of Equation (2.8). The instances submitted to the tests were “Karate”, “Dolphins”, “Football”, “Politics Books” (also known as “Polbooks”). With λ equal to 0.5, the iMeme-Net found partitions for “Karate” with average D value equal to 7.845, for “Dolphins” 10.883, for “Football” 29.321, and “Polbooks” 20.160.

2.2.3.3 Hybrid Artificial Immune Network HAIN

Karimi-Majd, Fathian and Amiri (2014) created an artificial immune network heuristic called HAIN for the Modularity Density Maximization problem. The method can be seen in Algorithm 11. The antigens and antibodies are represented as an array with $|V|$ elements, where each position is assigned to the cluster of the respective node.

The method starts creating antibodies and antigens using Algorithm 12. This procedure creates a set of antibodies for each antigen. For each antibody, a random set of nodes and their neighbors receive the same cluster identifier. Then for each node, the mode of the identifier of its neighbors is selected. Each set of antibodies is assigned to an antigen that is the best element of the antibody set.

Algorithm 11: HAIN (KARIMI-MAJD; FATHIAN; AMIRI, 2014).

```
   Input : Adjacent matrix of the graph $G(V, E)$
1  $\{antibodies, antigens\}$ ← Algorithm 12
2  repeat
3      foreach $A \in antibodies \cup antigens$ do
           // It calculates the D value of partition represented
              by $A$
4          affinityCalculation($A$)
5      foreach $AG \in antigens$ do
6          $newAntibodies$ ←clonalSelectionAndExpansion($AG$, antibodies of $AG$)
7          foreach $AB \in newAntibodies$ do
8              affinityMaturation($AB$)
9          metaDynamics
10     suppressDuplicates
11     assureDiversity
12 until population converges
```

After the generation of antibodies and their respective antigens, HAIN repeats the following procedures until the generated partitions converge: (*i*) the affinity of all antibodies and antigens is calculated; (*ii*) the clonal selection and expansion is executed for each antibody; (*iii*) the affinity maturation procedure is executed for each new antibody generated by the last clonal selection and expansion; (*iv*) the procedure of "meta dynamics" is executed for each antigen; (*v*) the duplicates are suppressed; (*vi*) the diversity is assured.

Clonal selection and expansion procedure clone each antibody by using its affinity antigen, working as a parallel population-based metaheuristic. The procedure ranks the antibodies and generates $c_i = \lfloor \beta \times {}^{N_{AB}}/_{r_i} \rfloor$ identical antibodies, where β is a constant and r_i is the rank value of the respective antibody i. N_{AB} is the number of antibodies.

Affinity maturation uses a local search to intensify the partitions of each new antibody generated by clonal selection and expansion. This local search can be seen

Algorithm 12: Antibody and Antigen Generator (KARIMI-MAJD; FATHIAN; AMIRI, 2014).

```
   Input  : $G(V, E)$, $N_{AG}$, $N_{AB}$, $C_{max}$
1  $antigens$ ←generate a number of $N_{AG}$ empty antigens
2  foreach $AG \in antigens$ do
3      $antibodies$ ←generate a number of $N_{AB}$ empty antibodies
4      foreach $AB \in antibodies$ do
5          $k \leftarrow 1$
6          $nodes \leftarrow$ choose $C_{max}$ nodes at random
7          foreach $node \in nodes$ do
8              $AB[node] \leftarrow k$
9              foreach $neighbor \in N(node)$ do
10                 $neighbor[neighbor] \leftarrow k$
11             $k \leftarrow k + 1$
12         foreach $node \in nodes$ do
13             $AB[node] \leftarrow mode\big(N(i)\big)$
14         calculateAffinity(AB)
15     add $antibodies$ to the $AG$ pool
16     $AG \leftarrow$ select the best $AB \in antibodies$
```

in Algorithm 13. The procedure receives each new antibody and changes the cluster assignment of a node. The procedure has a 50% chance of moving a node which has the smallest contribution in the cluster internal degree, and a 50% chance of moving a node which gives the largest contribution to an adjacent cluster. $|E_v^{new}|$ is the number of internal edges of CR connected to v, $|\overline{E_v^{new}}|$ is the number of outside edges of cluster CR connected to v, C_v is the current cluster of v, $|C_v|$ is the number of nodes inside the cluster C_v, $|E_v^{old}|$ is the number of edges of C_v connected to v, and $|\overline{E_v^{old}}|$ is the number of outside edges of C_v connected to v.

The *metaDynamics* and *suppressDuplicate* are cleaning procedures. The first procedure eliminates all antibodies, where the affinity of the respective antigen is lower than the threshold. The second procedure removes duplicated antibodies.

The diversity component is applied to all antibodies. This method has two disjoint procedures. With 50% probability, the largest cluster is split randomly by composing

Algorithm 13: Antibody Maturation (KARIMI-MAJD; FATHIAN; AMIRI, 2014).

```
   Input : antibody $AB$
1  $r \leftarrow$ generate a random number using Uniform distribution
2  if $r > 0.5$ then
3      $CR \leftarrow$ choose a random cluster number from $AB$
4      $CRf \leftarrow$ find a node that belongs to $CR$ and minimizes the difference between the internal and external cluster degree
5      $CN \leftarrow$ find a new cluster for $CRf$ which maximizes the difference between the intenal and external cluster degree
6      assign cluster $CN$ to node $CRf$
7  else
8      $CR \leftarrow$ choose a random cluster
9      $\overline{CR_{nodes}} \leftarrow$ list of nodes which do not belong to $CR$
10     $CRf \leftarrow argmax_{v \in \overline{CR_{nodes}}} \left\{ \frac{|E_v^{new}| - |\overline{E_v^{new}}|}{|CR|+1} - \frac{|E_v^{old}| - |\overline{E_v^{old}}|}{|C_v|} \right\}$
11     assign cluster $CR$ to node $CRf$
```

two clusters with equal numbers of nodes. With the same chance, a random node is chosen and assigned the most common cluster of its neighbors.

Karimi-Majd, Fathian and Amiri (2014) tested classic graphs, finding the best D values: “Karate” has a D value of 3.995, “Dolphins” 6.0626, “Polbooks” 10.9576, “Football” 22.194, “Adjnoun” 3.8937, “Lesmis” 12.2737, “Celegans Neural” 10.1143, “Co-authorship” 386.9737, “Power grid” 212.5005. The time required for “Karate”, “Polbooks”, and “Co-authorship” was 0.3328, 7.5623, and 493.0172 seconds respectively.

2.2.4 Exact Algorithms

There are six exact algorithms for Modularity Density Maximization, and they are all solved by mathematical integer programming solvers. The first two models are non-linear integer programming, and the others are linear programming models. This section describes these models.

2.2.4.1 Non-Linear Mathematical Programming

The first Modularity Density Maximization exact algorithm was reported in Li et al. (2008), and it was a non-linear mathematical programming model. The objective function and the constraints can be seen in Equation (2.21) respectively. Value a_{ij} is the sum of all edge weights between nodes i and j. The value k is the number of clusters where the nodes could be assigned. The binary decision variables x_{il} have values equal to 1 if a node i is assigned to cluster l; otherwise their values are equal to 0. The first constraint requires that the number of clusters used must be between 1 and $|V|-1$. The second constraint specifies that every node must be assigned to a single cluster.

$$\text{maximize} \sum_{l=1}^{k} \frac{\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} a_{ij}x_{il}x_{jl} - \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} a_{ij}x_{il}(1-x_{jl})}{\sum_{i=1}^{|V|} x_{il}} \tag{2.21a}$$

$$\text{subject to: } 0 < \sum_{i=1}^{|V|} x_{il} < |V| \qquad \forall l \in [k] \tag{2.21b}$$

$$\sum_{l=1}^{k} x_{il} = 1, \qquad \forall i \in V \tag{2.21c}$$

$$x_{il} \in \{0,1\}, \forall i \in V \forall l \in [k] \tag{2.21d}$$

The second non-linear model was presented by Karimi-Majd, Fathian and Amiri (2014). This model was based on the work of Li et al. (2008), but the number of clusters is not required (k parameter). Karimi-Majd, Fathian and Amiri (2014) added an upper bound to the number of clusters, called C_{max}, which is equal to $^{|V|}/_{3}$, because no cluster can have less than three nodes. In this model, some clusters could have zero nodes, and it causes division by zero in the objective function. Therefore, a new objective function is presented by Karimi-Majd, Fathian and Amiri (2014), where the denominator is added to a variable $b_l \in \{0,1\}$ for each cluster l. The value of b_l is equal to 1 if the cluster l has no node assigned to it, otherwise b_l equals to 0. This objective function can be seen in Equation (2.22).

$$\text{maximize} \sum_{l=1}^{k} \frac{\sum_{i=1}^{|V|}\sum_{j=1}^{|V|} a_{ij}x_{il}x_{jl} - \sum_{i=1}^{|V|}\sum_{j=1}^{|V|} a_{ij}x_{il}(1-x_{jl})}{\sum_{i=1}^{|V|} x_{il} + b_l} \tag{2.22}$$

2.2.4.2 Linear Mathematical Programming

Costa (2015) presented four linear mathematical programming models for the Modularity Density Maximization problem. The first two models are mixed integer linear programming formulations; the other two models are based on binary decompositions. The following section presents the four linear mathematical programming exact models.

The first model is called MDL1 and is shown in Equation (2.23) below. It is generated by reformulating the non-linear model of Li et al. (2008). It is presented in Equation (2.21). The products of the binary variables x_{il} are linearized by introducing new variables W using Fortet inequalities. These W variables are used in the constraints (2.23d) and (2.23e). The variables s are defined to linearize the objective function by using the McCormick inequalities. The s variable values are bound with the constraints (2.23g) and (2.23h).

To remove the fraction part of the objective function, a new variable α_l is added to the model for each cluster in $[k]$. The variables α_l are bound between L_α and U_α. The lower bound L_α is $-{(d_{max_1}+d_{max_2})}/{2}$, where d_{max_1} and d_{max_2} are the two highest degrees of the instance. The value of U_α is defined by a non-linear maximization problem of Equation (2.24).

$$\text{maximize} \sum_{l\in[k]} \alpha_l \tag{2.23a}$$

$$\text{subject to: } 2 \leq \sum_{i=1}^{|V|} x_{il} \leq |V| - 2(k-1), \forall l \in [k] \tag{2.23b}$$

$$\sum_{l=1}^{k} x_{il} = 1, \forall i \in V \tag{2.23c}$$

$$W_{ijl} \leq x_{il}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.23d}$$

$$W_{ijl} \leq x_{jl}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.23e}$$

$$4 \sum_{\{i,j\}\in E} W_{ijl} - \sum_{i\in V} d_i x_{il} \geq \sum_{i\in V} s_{il}, \forall l \in [k] \tag{2.23f}$$

$$L_\alpha x_{il} \leq s_{il} \leq U_\alpha x_{il}, \forall i \in V, \forall l \in [k] \tag{2.23g}$$

$$\alpha_l - U_\alpha(1 - x_{il}) \leq s_{il} \leq \alpha_l - L_\alpha(1 - x_{il}), \forall i \in V, \forall l \in [k] \tag{2.23h}$$

$$x_{il} \in \{0,1\}, \forall i \in V, \forall l \in [k] \tag{2.23i}$$

$$W_{ijl} \in \mathbb{R}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.23j}$$

$$s_{il} \in \mathbb{R}, \forall i \in V, \forall l \in [k] \tag{2.23k}$$

$$\alpha_l \in [L_\alpha, U_\alpha], \forall l \in [k] \tag{2.23l}$$

$$U_\alpha = \text{maximize } \frac{4\sum_{\{i,j\}\in E} x_i x_j - \sum_{i\in V} d_i x_i}{\sum_{i\in V} x_i} \tag{2.24a}$$

$$\text{subject to: } 2 \leq \sum_{i=1}^{|V|} x_i \leq |V| - 2(k-1) \tag{2.24b}$$

$$x_i \in [0,1], \qquad \forall i \in V \tag{2.24c}$$

$$\begin{aligned}
\text{maximize} & \sum_{l\in[k]} 4\beta_l - \gamma_l && \text{(2.25a)}\\
\text{subject to: } & 2 \le \sum_{i=1}^{|V|} x_{il} \le |V| - 2(k-1), \forall l \in [k] && \text{(2.25b)}\\
& \sum_{l=1}^{k} x_{il} = 1, \forall i \in V && \text{(2.25c)}\\
& W_{ijl} \le x_{il}, \forall \{i,j\} \in E, \forall l \in [k] && \text{(2.25d)}\\
& W_{ijl} \le x_{jl}, \forall \{i,j\} \in E, \forall l \in [k] && \text{(2.25e)}\\
& \sum_{\{i,j\}\in E} W_{ijl} \ge \sum_{i\in V} s_{il}, \forall l \in [k] && \text{(2.25f)}\\
& \sum_{i\in V} d_i x_{il} \le \sum_{i\in V} T_{il}, \forall l \in [k] && \text{(2.25g)}\\
& L_\beta x_{il} \le s_{il} \le U_\beta x_{il}, \forall i \in V, \forall l \in [k] && \text{(2.25h)}\\
& \beta_l - U_\beta(1 - x_{il}) \le s_{il} \le \beta_l - L_\beta(1 - x_{il}), \forall i \in V, \forall l \in [k] && \text{(2.25i)}\\
& L_\gamma x_{il} \le T_{il} \le U_\gamma x_{il}, \forall i \in V, \forall l \in [k] && \text{(2.25j)}\\
& \gamma_l - U_\gamma(1 - x_{il}) \le T_{il} \le \gamma_l - L_\gamma(1 - x_{il}), \forall i \in V, \forall l \in [k] && \text{(2.25k)}\\
& x_{il} \in \{0,1\}, \forall i \in V, \forall l \in [k] && \text{(2.25l)}\\
& W_{ijl} \in \mathbb{R}, \forall \{i,j\} \in E, \forall l \in [k] && \text{(2.25m)}\\
& s_{il} \in \mathbb{R}, \forall i \in V, \forall l \in [k] && \text{(2.25n)}\\
& T_{il} \in \mathbb{R}, \forall i \in V, \forall l \in [k] && \text{(2.25o)}\\
& \beta_l \in [L_\beta, U_\beta], \forall l \in [k] && \text{(2.25p)}\\
& \gamma_l \in [L_\gamma, U_\gamma], \forall l \in [k] && \text{(2.25q)}
\end{aligned}$$

$$U_\beta = \text{maximize } \frac{\sum_{\{i,j\}\in E} x_i x_j}{\sum_{i\in V} x_i} \tag{2.26a}$$

$$\text{subject to: } 2 \leq \sum_{i=1}^{|V|} x_i \leq |V| - 2(k-1) \tag{2.26b}$$

$$x_i \in [0,1], \qquad \forall i \in V \tag{2.26c}$$

The second linear model is called MDL2, as shown in Equation (2.25). It is an alternative to MDL1. It divides the numerator of the non-linear model objective function of Li et al. (2008) into two parts. These parts are linearized separately, resulting in the objective function of Equation (2.25a). The variables s and T are added to the model to linearize the first and second parts of the objective function. Each variable β_l is bound with $[L_\beta, U_\beta]$, where $L_\beta = 0$ and U_β is defined by a non-linear maximization problem in Equation (2.26). Each γ_l is bound with $[L_\gamma, U_\gamma]$. The value of L_γ is equal to $^{(d_{min_1} + d_{min_2})}/_2$, where d_{min_1} and d_{min_2} are the two lowest degrees of the instance. The value of U_γ is equal to $^{(d_{max_1} + d_{max_2})}/_2$.

The first linearization based on binary decomposition is called MDB1 and is shown in Equation (2.27). MDB1 uses the model MDL1. The latter model was changed by a binary decomposition to reduce the number of variables. In MDB1, Costa (2015) replaced $\sum_{i\in V} x_{il}$ by $\sum_{h\in H} 2_h b_{hl}$, where $H = \{0, \dots, t_D\}$ and $t_D = \lceil log_2(|V| - 2|C| + 3) - 1\rceil$.

The second binary decomposition is called MDB2, as shown in Equation (2.28). The MDB2 model was based on MDL2 which has the objective function split into two parts.

$$\text{maximize} \sum_{l\in[k]} \alpha_l \tag{2.27a}$$

$$\text{subject to: } 2 \leq \sum_{i=1}^{|V|} x_{il} \leq |V| - 2(k-1), \forall l \in [k] \tag{2.27b}$$

$$\sum_{l=1}^{k} x_{il} = 1, \forall i \in V \tag{2.27c}$$

$$W_{ijl} \leq x_{il}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.27d}$$

$$W_{ijl} \leq x_{jl}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.27e}$$

$$4 \sum_{\{i,j\}\in E} W_{ijl} - \sum_{i\in V} d_i x_{il} \geq \sum_{h\in H} 2^h r_{hl}, \forall l \in [k] \tag{2.27f}$$

$$\sum_{i\in V} x_{il} = \sum_{h\in H} 2^h b_{hl}, \forall l \in [k] \tag{2.27g}$$

$$L_\alpha b_{hl} \leq r_{hl} \leq U_\alpha b_{hl}, \forall h \in H, \forall l \in [k] \tag{2.27h}$$

$$\alpha_l - U_\alpha(1 - b_{hl}) \leq r_{hl} \leq \alpha_l - L_\alpha(1 - b_{hl}), \forall h \in H, \forall l \in [k] \tag{2.27i}$$

$$x_{il} \in \{0,1\}, \forall i \in V, \forall l \in [k] \tag{2.27j}$$

$$W_{ijl} \in \mathbb{R}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.27k}$$

$$r_{hl} \in \mathbb{R}, \forall h \in H, \forall l \in [k] \tag{2.27l}$$

$$b_{hl} \in \{0,1\}, \forall h \in H, \forall l \in [k] \tag{2.27m}$$

$$\alpha_l \in [L_\alpha, U_\alpha], \forall l \in [k] \tag{2.27n}$$

$$\text{maximize} \sum_{l\in[k]} 4\beta_l - \gamma_l \tag{2.28a}$$

$$\text{subject to: } 2 \leq \sum_{i=1}^{|V|} x_{il} \leq |V| - 2(k-1), \forall l \in [k] \tag{2.28b}$$

$$\sum_{l=1}^{k} x_{il} = 1, \forall i \in V \tag{2.28c}$$

$$W_{ijl} \leq x_{il}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.28d}$$

$$W_{ijl} \leq x_{jl}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.28e}$$

$$\sum_{\{i,j\}\in E} W_{ijl} \geq \sum_{h\in H} 2^h r_{hl}, \forall l \in [k] \tag{2.28f}$$

$$\sum_{i\in V} d_i x_{il} \leq \sum_{h\in H} 2^h p_{hl}, \forall l \in [k] \tag{2.28g}$$

$$\sum_{i\in V} x_{il} = \sum_{h\in H} 2^h b_{hl}, \forall l \in [k] \tag{2.28h}$$

$$L_\beta b_{hl} \leq r_{hl} \leq U_\beta b_{hl}, \forall h \in H, \forall l \in [k] \tag{2.28i}$$

$$\beta_l - U_\beta(1-b_{hl}) \leq r_{hl} \leq \beta_l - L_\beta(1-b_{hl}), \forall h \in H, \forall l \in [k] \tag{2.28j}$$

$$L_\gamma b_{hl} \leq p_{hl} \leq U_\gamma b_{hl}, \forall h \in H, \forall l \in [k] \tag{2.28k}$$

$$\gamma_l - U_\gamma(1-b_{hl}) \leq p_{hl} \leq \gamma_l - L_\gamma(1-b_{hl}), \forall h \in H, \forall l \in [k] \tag{2.28l}$$

$$x_{il} \in \{0,1\}, \forall i \in V, \forall l \in [k] \tag{2.28m}$$

$$W_{ijl} \in \mathbb{R}, \forall \{i,j\} \in E, \forall l \in [k] \tag{2.28n}$$

$$b_{hl} \in \{0,1\}, \forall h \in H, \forall l \in [k] \tag{2.28o}$$

$$r_{hl} \in \mathbb{R}, \forall h \in H, \forall l \in [k] \tag{2.28p}$$

$$p_{hl} \in \mathbb{R}, \forall h \in H, \forall l \in [k] \tag{2.28q}$$

$$\beta_l \in [L_\beta, U_\beta], \forall l \in [k] \tag{2.28r}$$

$$\gamma_l \in [L_\gamma, U_\gamma], \forall l \in [k] \tag{2.28s}$$

The main results of the linear models can be seen in Table 2.1. The table shows the % gap to the optimal partition and the time required in seconds, where “t.l.” means that the time goes beyond the limit of 2 hours. The best models are MDB1 and MDB2, because they reduce the number of variables. The experiments were performed on a PC with 4 Intel Xeon E5-4620 CPU at 2.20 GHz (8 cores each, Hyper-Threading and Turbo Boost disabled), 128 GB RAM. For further investigations, Costa (2015) suggests the usage of these models as a starting point to derive other heuristics, and to create a Column Generation heuristic like in work reported by Aloise et al. (2010). The latter idea was used in this thesis, and it is reported in Chapter 5.

Table 2.1: Time and gap results when using exact methods MDL1, MDL2, MDB1, and MDB2.

Graph				**MDL1**		**MDL2**		**MDB1**		**MDB2**	
Name	$\|V\|$	$\|E\|$	D^*	**gap**	**T(s)**	**gap**	**T(s)**	**gap**	**T(s)**	**gap**	**T(s)**
Strike	24	38	8.86111	0	1	0	68	0	1	0	2
Galesburg F	31	63	8.28571	0	6	0	1905	0	3	0	2
Galesburg D	31	67	6.92692	0	13	14	t.l.	0	2	0	4
Karate	34	78	7.8451	0	7	19	t.l.	0	4	0	4
Korea 1	35	69	10.9667	69	t.l.	189	t.l.	0	8	0	19
Korea 2	35	84	11.143	116	t.l.	246	t.l.	0	15	0	36
Mexico	35	117	8.71806	10	t.l.	112	t.l.	0	58	0	12
Sawmill	36	62	8.62338	0	3744	93	t.l.	0	10	0	10
Dolphins small	40	70	13.0519	86	t.l.	296	t.l.	0	121	0	435
Journal index	40	189	17.8	50	t.l.	190	t.l.	0	67	0	228

Source: Adapted from Costa (2015).

3 THEORETICAL CONTRIBUTIONS

This chapter presents proofs about Modularity Density Maximization failures when it is used as a prioritizer in a constructive search. An alternative prioritizer based on graph density is also presented, and its failure of detecting star-shaped modules is discussed. These results were published in European Journal of Operational Research; early in 2017. Theoretical contributions achieved in this research for the Modularity Maximization problem are reported in Appendix A.

First, we show that function D is not good for a constructive search, and an alternative priority function based on cluster density is presented. Some heuristics created in this thesis use the alternative prioritizer during the constructive phase. We call the heuristic searches which merge two disjoint clusters by using a priority criterion at each iteration as coarsening mergers. Our new coarsening merger heuristic is presented in Chapter 4.

Three lemmas and three theorems are presented in this chapter. The first three lemmas and theorem present three different function D prioritizers, and they explain the reason why function D is not good for a coarsening merger heuristic. The second theorem shows that an alternative two-phase prioritizer does not merge some cliques. The latter theorem shows that our "alternative prioritizer" fails to identify star-shaped modules.

The following two functions are defined to support these lemmas and theorems. Function $D_{cluster}(c)$ measures the cluster c contribution to the total D value of a partition.

$$D_{cluster}(c) = \frac{4|E_c| - \sum_{v \in c} d_v}{|c|} \tag{3.1}$$

The $\Delta D(c', c'')$ function is used to obtain the increased value when merging the clusters c' and c'' where $E_{\{c',c''\}}$ is the set of edges which have an endpoint node in c' and another in c''. The values $k_{c'}$ and $k_{c''}$ are the sum of all degrees from nodes of clusters c' and c'' respectively. This delta function is used to avoid recalculations of the

Modularity Density Maximization value for an entire partition.

$$\begin{aligned}\Delta D(c', c'') &= -D_{cluster}(c') - D_{cluster}(c'') \\ &+ \frac{4(|E_{c'}| + |E_{c''}| + |E_{\{c',c''\}}|) - k_{c'} - k_{c''}}{|c'| + |c''|}.\end{aligned} \tag{3.2}$$

3.1 Function D Fails as Prioritizer

The following three lemmas and the theorem are used to prove that function D cannot be used as a prioritizer for coarsening merger heuristics because it merges cliques. This characteristic does not respect the modular property of graph clustering problems. The modular property is an important constraint of graph clustering problems as defined by Radicchi et al. (2004) and Newman and Girvan (2004).

Lemma 1. *Coarsening merger heuristics can merge nodes from different cliques with the same size by using ΔD function (Equation* (3.2)*) as prioritizer.*

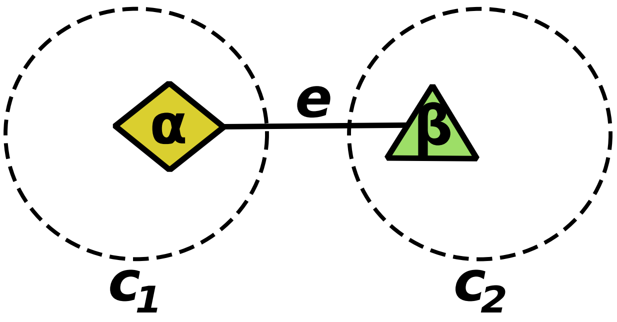
Proof. By contradiction, we suppose that all the coarsening merger heuristics that use the ΔD function as prioritizer do not merge two modular clusters.

Suppose an undirected graph composed of cliques c_1 and c_2. These cliques have the same number of nodes, where $n = |c_1| = |c_2|$. The two cliques are connected by an edge $e = \{\alpha, \beta\}$, where node $\alpha \in c_1$ and node $\beta \in c_2$. These two cliques can be seen in Figure 3.1. There are other nodes inside c_1 and c_2, but only α and β are featured to better understand the explanation about the lemmas and theorems.

At the start of a coarsening merger heuristic, suppose that each node belongs to a singleton cluster, so the starting number of clusters is equal to $2n$. The heuristic chooses the highest ΔD value to merge a pair of connected clusters. The options are:

(*i*) merge singleton clusters of two nodes which are inside one of the cliques c_1 or c_2, where these nodes are different from α and β, generating the following value to

Figure 3.1: Two cliques of equal size connected by a single edge.



prioritize:

$$\frac{4-2n+2}{2}+2n-2=n+1; \tag{3.3}$$

(*ii*) merge a singleton cluster of a node from c_1 with the singleton cluster α, or merge a singleton cluster of a node from c_2 with the singleton cluster β, resulting in the priority value:

$$\frac{4-2n+1}{2}+2n-1=n+\frac{3}{2}; \tag{3.4}$$

(*iii*) merge singleton clusters of α and β, resulting in the priority value:

$$\frac{4-2n}{2}+2n=n+2. \tag{3.5}$$

The highest gain is given by merging the singleton clusters of nodes α and β, proving that coarsening merger heuristics can merge clusters with nodes from different cliques when using ΔD function (Equation (3.2)).

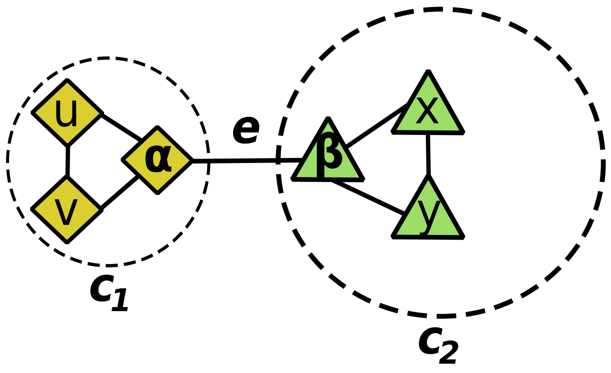
□

Lemma 2. *A coarsening merger heuristic can merge nodes from two cliques with different sizes when using $D_{cluster}$ function (Equation* (3.1)*) as prioritizer.*

Proof. Suppose, by contradiction, that all coarsening merger heuristics do not merge nodes from cliques with different sizes when using $D_{cluster}$ function (Equation (3.1)) as prioritizer.

Let c_1 and c_2 be two cliques. Clique c_1 has n_1 nodes, and clique c_2 has n_2 nodes, where $n_1 < n_2$. The cliques are connected by an edge $e = \{\alpha, \beta\}$, where the endpoints are $\alpha \in c_1$ and $\beta \in c_2$. These cliques can be seen in Figure 3.2. There are other nodes inside cliques c_1 and c_2, but only α, u, v, β, x and y are featured to support the explanation about lemmas and theorems.

Figure 3.2: Cliques c_1 and c_2 with n_1 and n_2 nodes respectively, where $n_1 < n_2$.



By assuming the use of the same start partition from Lemma 1, a coarsening merger heuristic will iteratively merge the singleton clusters from c_1, until c_1 is itself a cluster. To understand this, suppose the following gain and penalty parts of $D_{cluster}(c)$ function (Equations (3.1) and (3.6)):

$$D_{cluster}(c) = \underbrace{\frac{4|E_c|}{|c|}}_{gain} - \underbrace{\frac{k_c}{|c|}}_{penalty} . \tag{3.6}$$

From iteration 1 to $n_1 - 1$, the merging will be made only with clusters which are inside the clique c_1, and the last node to be merged is α. This behavior happens because the generated gain is greater than or equal to any other merge option, and the absolute penalty part for merging subcliques of c_1 will be lower than for merging singleton clusters from c_2. The average degree from any subset of c_1 will be lower than any subset of nodes from c_2.

At iteration n_1, there is a cluster with all nodes of c_1 and n_2 singleton clusters composed of nodes from c_2. The total number of clusters is $n_2 + 1$ at this iteration. By supposing that a coarsening merger heuristic does not merge nodes from cliques with different sizes when using the $D_{cluster}$ function prioritizer, so no singleton cluster from c_2 should be merged to the c_1 cluster. At this iteration, the merge options are:

(*i*) merge the cluster c_1 with singleton cluster β, resulting in the priority value:

$$\frac{2n_1(n_1-1)+4-n_1(n_1-1)-1-n_2}{n_1+1}; \tag{3.7}$$

(*ii*) merge two singleton clusters from c_2 which are not composed of the node β (for example, merge singleton clusters x and y of Figure 3.2), resulting in the priority value:

$$\frac{4-2n_2+2}{2} = -n_2+3; \tag{3.8}$$

(*iii*) merge the cluster $\{\beta\}$ with another singleton cluster from c_2, resulting in the priority value:

$$\frac{4-2n_2+1}{2} = -n_2+\frac{5}{2}. \tag{3.9}$$

The priority value produced by the merging of clusters c_1 and $\{\beta\}$ is larger than any other merge option. To show that, this priority value is simplified as follows:

$$\begin{aligned} \frac{2n_1(n_1-1)+4-n_1(n_1-1)-1-n_2}{n_1+1} &= \\ \frac{n_1^2-n_1+3-n_2}{n_1+1} &> \\ \frac{(n_1+1)(n_1-2)-n_2}{n_1+1} &= n_1-2-\frac{n_2}{n_1+1}. \end{aligned} \tag{3.10}$$

As $n_1 < n_2$, the lower bound of $D_{cluster}$ function by merging cluster c_1 and singleton

$\{\beta\}$ is larger than any other option at this iteration for all $n \geq 3$, where 3 is the size of the smallest possible clique, as below (this implies $n_1 = 3$, $n_2 = 4$):

$$n_1 - 2 - \frac{n_2}{n_1 + 1} > -n_2 + 3. \tag{3.11}$$

This implies that the coarsening merger heuristic can merge nodes from cliques with different sizes, proving the lemma by contradiction.

□

For the following lemma, we consider a prioritizer which uses two factors from Equation (3.6). The first is the gain factor, and the second is the penalty factor. The best merge option for the first factor has the highest value. The penalty factor is only used if there is more than one highest merge option with the same priority value. For this factor, the merge option with the lowest absolute value is chosen.

Lemma 3. *A coarsening merger heuristic can merge nodes from two cliques with different sizes when using the gain and penalty of Function* (3.6) *as two distinct factors of prioritizer.*

Proof. To prove this Lemma, similar arguments used in Lemma 2 are applied. The following explanation uses the same cliques c_1 and c_2 from Figure 3.2, which are connected by a single edge $\{\alpha, \beta\}$, where $\alpha \in c_1$ and $\beta \in c_2$.

Before the iteration n_1, all nodes from c_1 belong to the same cluster. This happens because the highest priority values of the gain factor are about merging all subset nodes of c_1 until all nodes from this clique form a single cluster. At iteration n_1, the gain factor merges cluster c_1 and the singleton cluster $\{\beta\}$ because this option results in a higher priority value than any other merge option between two singleton clusters from c_2 as can be seen in Equation (3.12), where w and z are two nodes of c_2.

$$\underbrace{\frac{2n_1(n_1 - 1) + 4}{n_1 + 1}}_{c_1 \cup \{\beta\}} > \underbrace{\frac{4}{2}}_{\{w \in c_2\} \cup \{z \in c_2\}} . \tag{3.12}$$

By contradiction, we prove that a coarsening merger heuristic can merge nodes from two cliques with different sizes when using the gain and penalty of Function (3.6) as two separated factor prioritizers.

□

Theorem 4. *A coarsening merger heuristic can merge cliques when using Modularity Density Maximization D as prioritizer.*

Proof. This theorem is proved by using Lemmas 1, 2, and 3. Lemma 1 shows that cliques with the same size can be merged by the prioritizer ΔD, that is presented in Equation (3.2). This prioritizer is often used to choose the movement that best generates the higher improvement in the objective function. Lemma 2 shows that cliques with different sizes can be merged when using the Function $D_{cluster}$ (Equation (3.1)) as prioritizer. This function is the objective function that measures the contribution of a cluster for a partition. Finally, Lemma 3 shows that an extension of the D function can merge cliques with different sizes. These lemmas show that Functions ΔD, $D_{cluster}$, and Function (3.6) can merge cliques when they are used as prioritizers.

□

3.2 Alternative Prioritizer for D

As the Modularity Density Maximization objective Function D fails as a prioritizer for coarsening merger heuristics, we have created a new prioritizer which uses a modular property and a penalty factor. This prioritizer selects the resulting merge option that maximizes Function (3.13) (gainD). When there are two or more equal maximal values, the second factor is applied by selecting the merge option which minimizes Function (3.14) (penaltyD). Here, this prioritizer is called “alternative density”.

$$gainD(c) = \frac{|E_c|}{(|c|^2 - |c|)/2}. \tag{3.13}$$

$$penaltyD(c) = \frac{k_c}{|c|}. \tag{3.14}$$

Theorem 5. *A coarsening merger heuristic does not merge nodes from different cliques of Figure 3.1 and 3.2 when using "alternative density" as prioritizer.*

Proof. The proof is divided into two parts. The first part shows that two cliques of Figure 3.1 with equal size are not merged. The second part shows that two cliques of Figure 3.2 with different sizes are not merged. For simplicity, let us consider that the coarsening merger heuristic starts with a partition where each node belongs to a singleton cluster.

To prove that two cliques of Figure 3.1 are not merged by a coarsening merger heuristic which uses the "alternative density" prioritizer, we use the two cliques from that figure. The cliques c_1 and c_2 have n nodes. The edge $e = \{\alpha, \beta\}$ connects the cliques c_1 and c_2. As the first factor is the relative density of edges inside the clusters to be merged, all merge options are the pairs of singleton nodes connected by an edge. Because they have the same $gainD$ value, the second factor is applied by selecting to merge the singleton clusters which have the minimal average degree. They are the two singleton clusters which do not contain the nodes α and β. In the next iterations, the clusters that are inside of each clique are merged. At the final iterations, $\{\alpha\}$ merges with $c_1 \backslash \{\alpha\}$, and $\{\beta\}$ merges with $c_2 \backslash \{\beta\}$, then the partition has two clusters c_1 and c_2. The cliques of the same size are not merged when the coarsening merger uses the "alternative density" as prioritizer.

The second part of the proof is about "alternative density" not merging two cliques of Figure 3.2 that have different sizes. The cliques are c_1 with n_1 nodes and c_2 with n_2 nodes, where $n_1 < n_2$. The cliques are connected by the edge $e = \{\alpha, \beta\}$, where $\alpha \in c_1$, and $\beta \in c_2$. The initial iterations merge each pair of clusters inside $c_1 \backslash \{\alpha\}$, because they have full edges density and the average degree is smaller than the rest of the merge options from c_2. At iteration $n_1 - 1$, the merging will be made between the singleton cluster α and cluster $c_1 \backslash \{\alpha\}$. After, the merging of two clusters connected by

e is not prioritized because the merging of c_1 with any other cluster does not result in a full density (first factor). So, each pair of clusters inside clique c_2 is merged.

The two proof parts show that a coarsening merger heuristic that uses the "alternative density" prioritizer does not merge the cliques of Figures 3.1 and 3.2.

□

3.3 Alternative Prioritizer and Star-shaped Modules

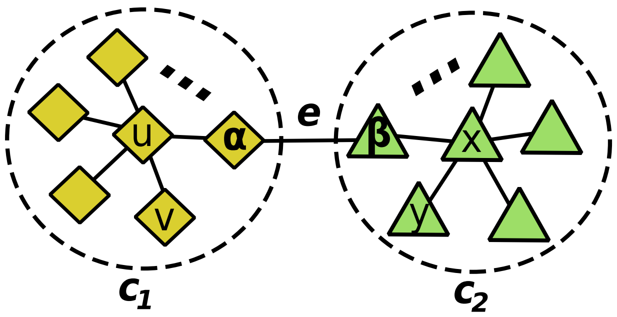
"Alternative density" does not merge cliques of Figures 3.1 and 3.2 as Theorem 5 shows. However, there is a counter-intuitive behavior. It merges nodes from different star-shaped modules as prioritizer, as shown in Theorem 6. Star-shaped modules are structures found in some real and artificial graphs (ZHANG; QIU; ZHANG, 2010).

Theorem 6. *A coarsening merger heuristic merges nodes from different star-shaped modules of Figure 3.3 when using "alternative density" as prioritizer.*

Proof. By contradiction, suppose that "alternative density" does not merge nodes from different star-shaped modules when it is used as prioritizer by a coarsening merger heuristic. To support this theorem, we use Figure 3.3. In this figure, there are two star-shaped modules c_1 and c_2 with n_1 and n_2 number of nodes respectively. Each one of these modules is composed of n nodes, $n-1$ edges, and $n-1$ border nodes are connected to a central node. The central nodes are called u and x in modules c_1 and c_2, respectively. There is an edge e connecting the border nodes α of c_1 and β of c_2. The other border nodes are called as v and y in the figure.

At the start of a coarsening merger heuristic, suppose that each node belongs to a singleton cluster. As *gainD* (Function (3.13)) is equal for all the merge options, we choose the merge option that leads to the minimal *penaltyD* value (Function (3.14)). The priority values of the merge options available are:

Figure 3.3: An undirected graph composed of two star-shaped modules c_1 and c_2 which have n_1 and n_2 number of nodes, respectively.



(i) merge the singleton clusters of nodes α and u in c_1, resulting in the priority value:

$$\frac{n_1 - 1 + 2}{2} = \frac{n_1 + 1}{2}; \tag{3.15}$$

(ii) merge the singleton clusters of nodes β and x in c_2, resulting in the priority value:

$$\frac{n_2 - 1 + 2}{2} = \frac{n_2 + 1}{2}; \tag{3.16}$$

(iii) merge the singleton clusters of nodes u and v in c_1, resulting in the priority value:

$$\frac{n_1 - 1 + 1}{2} = \frac{n_1}{2}; \tag{3.17}$$

(iv) merge the singleton clusters of nodes x and y in c_2, resulting in the priority value:

$$\frac{n_2 - 1 + 1}{2} = \frac{n_2}{2}; \tag{3.18}$$

(v) or merge the singleton clusters of nodes α and β, resulting in the priority value:

$$\frac{2 + 2}{2} = 2. \tag{3.19}$$

For $n_1, n_2 > 4$, the merge option with the minimal value is about to merge the singleton clusters of nodes α and β that belong to different star-shaped modules. So, one proves by contradiction that “alternative density” merges nodes from different star-shaped modules.

□

4 CONTRIBUTIONS TO HEURISTICS

This chapter presents novel multilevel, constructive, and hybrid heuristics. The first five heuristics are inspired by works reported in Clauset, Newman and Moore (2004), Blondel et al. (2008), Rotta and Noack (2011). The eighth heuristic was developed by mixing two of our scalable local search heuristics. Experiments designed to compare the new with existing methods are reported at the end of this chapter. All our heuristics do not have parameters that could require a hold-out set of the experiments. Additional results are reported in the ground truth analyses of Chapter 6.

The results reported in this Chapter were published in European Journal of Operational Research; in May 2017.

4.1 Constructive and Multilevel Heuristics

This section presents the algorithm design of new constructive and multilevel heuristics for Modularity Density Maximization. It is divided into three categories that are coarsening merger, moving node, and multilevel heuristics. They are inspired by other Modularity Maximization heuristics which solved graphs with tens of thousands of nodes. These results for Modularity Maximization heuristics are described in Clauset, Newman and Moore (2004), Blondel et al. (2008), Rotta and Noack (2011). The following three subsections present the categories and describe the design details of each heuristic.

4.1.1 Coarsening Merger

The coarsening merger heuristic (CM) was inspired by CNM (CLAUSET; NEWMAN; MOORE, 2004). Iteratively, this heuristic merges the pair of clusters which has the highest priority value. At the end of $|V| - 1$ iterations, the partition is composed of a single cluster with all nodes. This heuristic uses the “alternative density” prioritizer described in Chapter 3.

As proved by Theorem 4, Function D cannot be used in this kind of heuristic because it can merge cliques. In contrast, the "alternative density" uses the edge density (*gainD*, Equation (3.13)) and the degree proportion (*penaltyD*, Equation (3.14)). The "alternative density" was selected because it does not merge cliques of Figures 3.1 and 3.2 as described in Theorem 5.

The coarsening merger heuristic is described in Algorithm 14. The main data structure used by this heuristic is a Fibonacci heap that stores all options of each merge between two clusters. A Fibonacci heap is used to recover quickly the merge option which has the maximal priority value. In this heuristic, the Fibonacci heap uses as prioritizer the "alternative density", where the merge option that maximizes the value of gain Equation (3.13) (*gainD*) is prioritized. This best merge option is obtained by *heap.top*. When two or more merge options have the same *gainD* (Equation (3.13)) value, the one of them that has the minimal value of *penaltyD* (Equation (3.14)) is chosen. When a merge option results in a cluster with no edge, its evaluation is done by using only the penalty Equation (3.14) (*penaltyD* as prioritizer).

The heuristic starts with a partition composed of $|V|$ singleton clusters, where each node $v \in V$ belongs to a cluster (line 1). Then a copy of that partition is called *best* (line 2). The structure *best* is used to store the highest D value partition found during the coarsening merger search. After that, the Fibonacci heap is initialized in the procedure *maxHeapFibonacci* (line 3). After the heap is initialized, each merge option between each pair of adjacent clusters is inserted into the heap with its prioritized value (lines 4 to 6). After that, the lists *pairs* and *levels* are initialized (lines 7 and 8). The list *pairs* stores all pairs of clusters that are chosen to be merged, and the list *levels* stores all obtained partitions. This storing happens in the order of the following loop iterations. These data structures will be used in the multilevel density heuristics described in Section 4.1.3. After the initialization of *partition*, a loop is started that repeats $|V| - 1$ times (lines 9 to 16). Each iteration starts requesting the best merge option, following the "alternative density" criterion (line 10). The Fibonacci heap retrieves the best merge

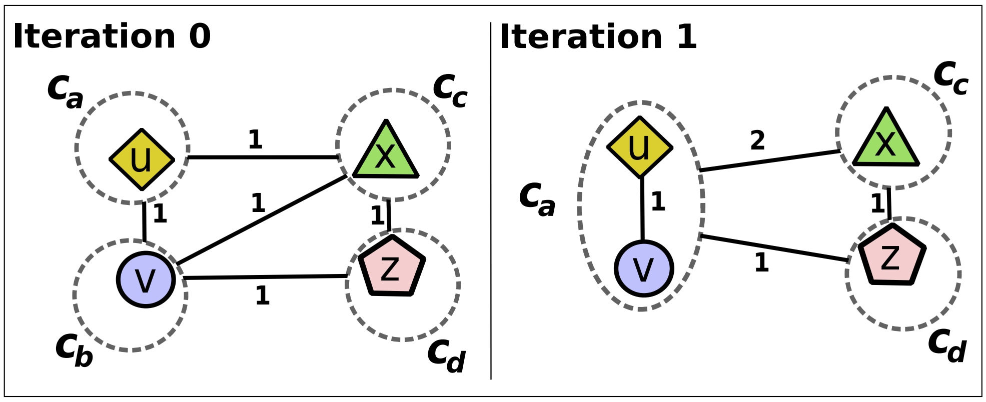
Algorithm 14: Coarsening merger heuristic.

```
   Input  : $G(V,E)$
1  $partition \leftarrow \big\{\{v\} : \forall v \in V\big\}$
2  $best \leftarrow partition$
3  $heap \leftarrow maxHeapFibonacci()$
4  foreach $\{u,v\} \in E$ do
5      $val \leftarrow \big(gainD(\{u\} \cup \{v\}), penaltyD(\{u\} \cup \{v\})\big)$
6      $heap.push\big(u, v, val\big)$
7  $pairs \leftarrow list()$
8  $levels \leftarrow list()$
9  for $l \leftarrow 1$ to $|V| - 1$ do
10     $\{u,v\} \leftarrow heap.top()$
11     $partition \leftarrow partition.merge(u,v)$
12     $updateMerge(heap, u, v)$
13     if $D(best) < D(partition)$ then
14         $best \leftarrow partition$
15     $pairs.append(\{u,v\})$
16     $levels.append(partition)$
17 return $(best, levels, pairs)$
```

option in constant time, represented here as the pair of clusters u and v. Then the clusters u and v are merged in the partition by the procedure *partition.merge* (line 11). After that, the heap is updated, considering the new merged cluster. At this step, all adjacent clusters are considered in the new option to merge with the new merged cluster. Then the current partition is compared with the best one (lines 13 to 14). If the current partition has a better D value, the best partition is updated. After that, the pair of clusters merged at the end of the list *pairs* is inserted, and the current partition is inserted at the end of the list *levels* (lines 15 and 16). Finally, the partition with the best D value, and the lists *levels* and *pairs* are returned (line 17).

Next, the merging method *partition.merge(u,v)* is described in detail. The merging method receives two clusters u and v. The cluster with the greater number of nodes remains, and the other is merged with the first. Let us call the first cluster as c_a and the second as c_b. When they are merged, all nodes from c_b are transferred to c_a. The transferring of edges is made by verifying if the cluster c_a already has an edge for the same endpoint cluster. If there is such an edge, the weight of this edge is increased by the same weight of the edge in c_b. If there is no such edge, the edge is transferred from c_b to c_a. This method is exemplified in Figure 4.1. The total complexity of the merging is $O(|c_b| + |E_{c_a}||E_{c_b}|)$, where E_c is the number of edges in cluster c.

Figure 4.1: Example of the CM heuristic execution.



The procedure "updateMerge" updates the current partition merge options in the heap. Considering $c_{u,v}$ the resulting cluster from the merging of clusters u and v, all options to merge any other cluster with u or v in the heap are updated to $c_{u,v}$. As the Fibonacci Heap requires amortized constant time to update, then the expected number of operations for "updateMerge" is $O(|E_{c_{u,v}}|)$.

4.1.2 Moving Nodes

In this work, moving node heuristics are local searches that try to move all nodes from their clusters to others. We use two different moving node heuristics. The main difference between them is at the end of the movement phase, where the resulting clusters are coarsened or not. The coarsening phase happens after the most internal loop of Algorithm 16 (lines 15 to 17). These methods are presented in Algorithms 15 and 16.

To support the heuristic explanation, the following Δ functions are presented. They are used in the heuristics described in this section. The $\Delta D_{node}(v, c)$ function is used to identify the increase in the value from D when moving the node v from c_v to cluster c. E_c^v and $E_{c_v}^v$ are the sets of edges between the node v and nodes from clusters c and c_v respectively. Clusters c' and c_v' represent c and c_v after the movement respectively. Equations (4.1) and (4.2) show the simplification of $D_{cluster}(c') - D_{cluster}(c)$ and $D_{cluster}(c_v') - D_{cluster}(c_v)$ which are used to define $\Delta D_{node}(v, c)$ function (Equation (4.3)).

$$\begin{aligned} D_{cluster}(c') - D_{cluster}(c) &= \\ \frac{4|E_c| + 4|E_c^v| - k_c - d_v}{|c| + 1} + \frac{-4|E_c| + k_c}{|c|} &= \\ \frac{|c|\big(4|E_c| + 4|E_c^v| - k_c - d_v\big)}{|c|^2 + |c|} &+ \\ \frac{(|c| + 1)\big(- 4|E_c| + k_c\big)}{|c|^2 + |c|} &= \\ \frac{|c|\big(4|E_c^v| - d_v\big) - 4|E_c| + k_c}{|c|^2 + |c|}&. \end{aligned} \tag{4.1}$$

$$\begin{aligned} D_{cluster}(c_v') - D_{cluster}(c_v) &= \\ \frac{4|E_{c_v}| - 4|E_{c_v}^v| - k_{c_v} + d_v}{|c_v| - 1} + \frac{-4|E_{c_v}| + k_{c_v}}{|c_v|} &= \\ \frac{|c_v|\big(4|E_{c_v}| - 4|E_{c_v}^v| - k_{c_v} + d_v\big)}{|c_v|^2 - |c_v|} &+ \\ \frac{(|c_v| - 1)\big(- 4|E_{c_v}| + k_{c_v}\big)}{|c_v|^2 - |c_v|} &= \\ \frac{|c_v|\big(- 4|E_{c_v}^v| + d_v\big) + 4|E_{c_v}| - k_{c_v}}{|c_v|^2 - |c_v|}&. \end{aligned} \tag{4.2}$$

$$\begin{aligned}\Delta D_{node}(v,c) &= D_{cluster}(c') - D_{cluster}(c) \\ &\quad + D_{cluster}(c'_v) - D_{cluster}(c_v) \\ &= \frac{|c|\big(4|E_c^v| - d_v\big) - 4|E_c| + k_c}{|c|^2 + |c|} \\ &\quad + \frac{|c_v|\big(-4|E_{c_v}^v| + d_v\big) + 4|E_{c_v}| - k_{c_v}}{|c_v|^2 - |c_v|}.\end{aligned} \tag{4.3}$$

The $\Delta D_{coarse}(s,c)$ function measures the increase of the D function by considering that s is moved from c_s to c, where s is a subcluster from cluster c_s, and c is a cluster. E_c^s and $E_{c_s}^s$ are the sets of edges between the subcluster s and nodes from clusters c and c_s respectively. Clusters c' and c'_s represent c and c_s after the movement respectively. Equations (4.4) and (4.5) show the simplification of $D_{cluster}(c') - D_{cluster}(c)$ and $D_{cluster}(c'_s) - D_{cluster}(c_s)$ which are used to define $\Delta D_{coarse}(s,c)$ function (Equation (4.6)).

$$\begin{aligned}&D_{cluster}(c') - D_{cluster}(c) = \\ &\frac{4|E_c| + 4|E_c^s| - k_c - k_s}{|c| + |s|} + \frac{-4|E_c| + k_c}{|c|} \\ &= \frac{|c|\big(4|E_c| + 4|E_c^s| - k_c - k_s\big)}{|c|^2 + |c||s|} + \\ &\quad \frac{\big(|c| + |s|\big)\big(-4|E_c| + k_c\big)}{|c|^2 + |c||s|} \\ &= \frac{|c|\big(4|E_c^s| - k_s\big) - 4|E_c||s| + k_c|s|}{|c|^2 + |c||s|}.\end{aligned} \tag{4.4}$$

$$\begin{aligned}
&D_{cluster}(c_s') - D_{cluster}(c_s) = \\
&\frac{4|E_{c_s}| - 4|E_{c_s}^s| - k_{c_s} + k_s}{|c_s| - |s|} + \frac{-4|E_{c_s}| + k_{c_s}}{|c_s|} = \\
&\frac{|c_s|\big(4|E_{c_s}| - 4|E_{c_s}^s| - k_{c_s} + k_s\big)}{|c_s|^2 - |c_s||s|} + \\
&\frac{(|c_s| - |s|)\big(-4|E_{c_s}| + k_{c_s}\big)}{|c_s|^2 - |c_s||s|} = \\
&\frac{|c_s|\big(-4|E_{c_s}^s| + k_s\big) + 4|E_{c_s}||s| - k_{c_s}|s|}{|c_s|^2 - |c_s||s|}.
\end{aligned} \tag{4.5}$$

$$\begin{aligned}
\Delta D_{coarse}(s,c) &= D_{cluster}(c') - D_{cluster}(c) \\
&+ D_{cluster}(c_s') - D_{cluster}(c_s) \\
&= \frac{|c|\big(4|E_c^s| - k_s\big) - 4|E_c||s| + k_c|s|}{|c|^2 + |c||s|} \\
&+ \frac{|c_s|\big(-4|E_{c_s}^s| + k_s\big) + 4|E_{c_s}||s| - k_{c_s}|s|}{|c_s|^2 - |c_s||s|}.
\end{aligned} \tag{4.6}$$

4.1.2.1 Local Node Moving Heuristic

The first moving node heuristic presented in this thesis was inspired by the method of Kernighan and Lin (1970). This heuristic is called "local node moving" (LNM) and is seen in Algorithm 15. First, the heuristic starts copying the input partition to the structure *best* that stores the best partition found during the heuristic execution (line 1). At the second step, a flag called *improvement* is set as *true* (line 2). Then the list *randomV* receives all nodes in a random order (line 3). After that, an external loop starts and is executed until the flag *improvement* is not *true* (lines 4 to 17). The first step of each iteration of the external loop sets the flag *improvement* as *false* (line 5). Then another loop is performed, and at each iteration, it treats a node from the random list *randomV* (lines 6 to 14).

At each iteration, the best gain to move a node to another cluster is stored (lines 13 to 14), and if the gain exists, v is moved to the cluster that provides the best gain. This gain is computed by the function $\Delta D_{node}(v, c)$ of Equation (4.3). The value of the best gain is stored in the variable ΔD_v, and corresponding cluster c_v, where v is the node to be moved. This search for the best movement option is done in the internal loop of lines 9 to 12. In this internal loop, one tries to move each node to its adjacent clusters, which are given by the set $C_{N(v)}$. After the attempts to move all nodes, the current partition is compared with the best one found during the search (lines 15 to 17). If the current partition is now the best partition, the flag *improvement* is set as *true*, and a new iteration is performed in the external loop. Finally, when the external loop stops, the best partition is returned (line 18).

Each iteration is bound by $O(|V||E|)$ operations because all node movements are tried, and the movement gain is evaluated for all neighborhood clusters. Each node movement gain is calculated in constant time by storing the number of internal edges, the nodes, and the total degree of nodes for each cluster. Equation (4.3) is used for the gain evaluation.

Two versions of the "local node moving" are reported. We call LNM the version where the initial partition is composed of singleton nodes (each node is alone in a cluster). The version CM+LNM uses the initial partition generated by the heuristic CM (Algorithm 14).

4.1.2.2 Move and Coarse Node Heuristic

At each iteration, the "move and coarse nodes" (MCN) heuristic tries to improve the partition value by changing all nodes from their clusters. This phase is called "level phase". At the end of each iteration, each cluster is considered as a single node, and a new iteration is done. This process repeats until no improvement is found. When the nodes of a cluster are united, the graph is updated to consider these changes. This heuristic was inspired by the Louvain method (BLONDEL et al., 2008).

Algorithm 15: Local node moving.

```
Input  : $G(V, E), partition$
1  $best \leftarrow partition$
2  $improvement \leftarrow true$
3  $randomV \leftarrow random(V)$
4  while $improvement$ do
5      $improvement \leftarrow false$
6      foreach $v \in randomV$ do
7          $\Delta D_v \leftarrow 0$
8          $c_v \leftarrow \emptyset$
9          foreach $c \in C_{N(v)}$ do
10             if $\Delta D_v < \Delta D_{node}(v, c)$ then
11                 $\Delta D_v \leftarrow \Delta D_{node}(v, c)$
12                 $c_v \leftarrow c$
13         if $c_v \neq \emptyset$ then
14             in $partition$, move node $v$ to cluster $c_v$
15     if $D(best) < D(partition)$ then
16         $best \leftarrow partition$
17         $improvement \leftarrow true$
18 return $best$
```

Algorithm 16 shows how this heuristic works. First, the input partition is copied to the structure *best* that stores the best partition found by the heuristic (line 1). Then a flag *improvement* is set to *true* (line 2). This flag is used as stop criterion to the external loop that stops when no improvement is found (lines 3 to 21). After that, the list of nodes $randomV$ receives nodes from V in a random order (line 3). At each iteration of the external loop, the level phase is started (lines 6 to 18), and then the coarsening phase is done (lines 19 to 22).

The level phase performs a local search similar to LNM (Algorithm 15). A loop repeats until no movement that improves the current partition is found. This movement is counted by the variable *moves*. At each iteration of the loop between lines 8 and 17, one tries to move each node v of the *randomV* to another adjacent cluster. The set of adjacent clusters of v is represented by $C_{N(v)}$. If there is a movement to change v to a

Algorithm 16: Move and coarse nodes.

```
   Input  : $G(V, E), partition$
1  $best \leftarrow partition$
2  $improvement \leftarrow true$
3  $randomV \leftarrow random(V)$
4  while $improvement$ do
5      $improvement \leftarrow false$
       // level phase
6      repeat
7          $moves \leftarrow 0$
8          foreach $v \in randomV$ do
9              $\Delta D_v \leftarrow 0$
10             $c_v \leftarrow \emptyset$
11             foreach $c \in C_{N(v)}$ do
12                 if $\Delta D_v < \Delta D_{coarse}(v, c)$ then
13                     $\Delta D_v \leftarrow \Delta D_{coarse}(v, c)$
14                     $c_v \leftarrow c$
15             if $c_v \neq \emptyset$ then
16                 $moves \leftarrow moves + 1$
17                 in $partition$, move node $v$ to cluster $c_v$
18     until $moves = 0$
       // coarsening phase
19     if $D(best) < D(partition)$ then
20         $best \leftarrow partition$
21         $improvement \leftarrow true$
22         clusters are coarsened as nodes in $partition$ and graph $G(V, E)$
23 return $best$
```

cluster c that improves the D function, the change is done. Equation (4.6) (ΔD_{coarse}) is used to calculate the gain of each movement. The level phase repeats until no movement is done, and then the coarsening phase starts. If the current partition is better than the best-stored partition, the best value is updated (line 20), the flag *improvement* is set as *true*, and the coarsening process is made (line 22). The coarsening process updates the graph instance by considering each cluster as a single node, rebuilding the graph instance with meta-nodes. When no better solution is found, the heuristic returns the best partition found and stops the search.

The loop between lines 8 and 17 of the level phase requires $O(|V||E|)$ operations, but the total time depends on the number of the best partitions found by the heuristic and how many times the coarsening phase is run.

The coarsening phase unites the nodes from each cluster as a single node. At the beginning of the level phase, the graph has singleton clusters where each cluster is a node. At the level phase, these nodes are assigned to clusters that optimize the current partition. When no more node movements are done, the level phase stops and the coarsening phase is executed. Then the partition is used to identify the cluster of each node. Mergings are done to unite all nodes from the same cluster. The upper bound of operations is the number of coarsened nodes times the required operation by each merging. The time for a merging is described in Section 4.1.1.

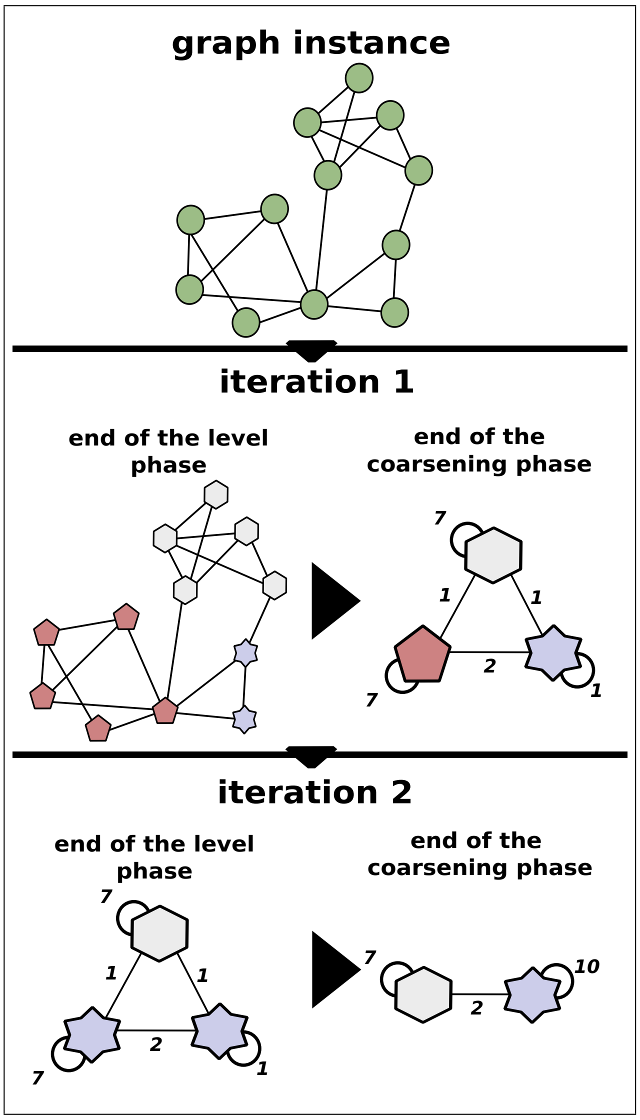
Figure 4.2 shows an example of MCN execution into two iterations. The first part shows the original graph. At each one of the next two iterations, it is shown the end of the level phases and the end of the coarsening phases. Before the first iteration, the instance graph can be seen without identified clusters. In the first iteration, the end of the level phase is shown, where each cluster is identified by a different shape, and the end of the first coarsening phase, where each cluster is transformed into a meta-node in the graph instance. In the second iteration, the two phases are performed using the updated instance. The values assigned to the lines are the number of edges in that graph region.

In this thesis, two versions of the "move and coarse nodes" are reported. We name as MCN the version where the initial partition is composed of singleton nodes (each node is alone in a cluster). The version CM+MCN uses the initial partition generated by the heuristic CM (Algorithm 14).

4.1.3 Multilevel Heuristics

Our multilevel heuristics use *levels* and *pairs* from the heuristic CM (Algorithm 14) to do exploitation. Each partition resulting of a merging from iterations of CM is called

Figure 4.2: The example used to illustrate the MCN heuristic divided into two iterations.



Source: Adapted from Blondel et al. (2008).

levels; the pairs of merged partitions are called *pairs* (see Section 4.1.1). We developed two multilevel heuristics for the Modularity Density Maximization problem. These heuristics and their two phases are described in Algorithm 17. They are constructive and refinement phases. The reason to do that are the results reported in Rotta and Noack

(2011) which reached high Modularity Maximization scored partitions.

Our two multilevel heuristics for Modularity Density Maximization use different prioritizers during the refinement phase. These prioritizers are used to select the cluster to which the node must be assigned. The heuristic called “multilevel density” (MD) applies the MCN (Algorithm 16) that uses the Modularity Density Maximization delta function ΔD_{coarse} (Equation (4.6)). The other is called “multilevel density modularity” (MDM) and uses the Louvain method which has as prioritizer the Function ΔQ_{coarse} (Equation (4.7)). The ΔQ_{coarse} equation is based on the Modularity Maximization problem, and it was used to test its behavior in multilevel methods for the Modularity Density Maximization problem.

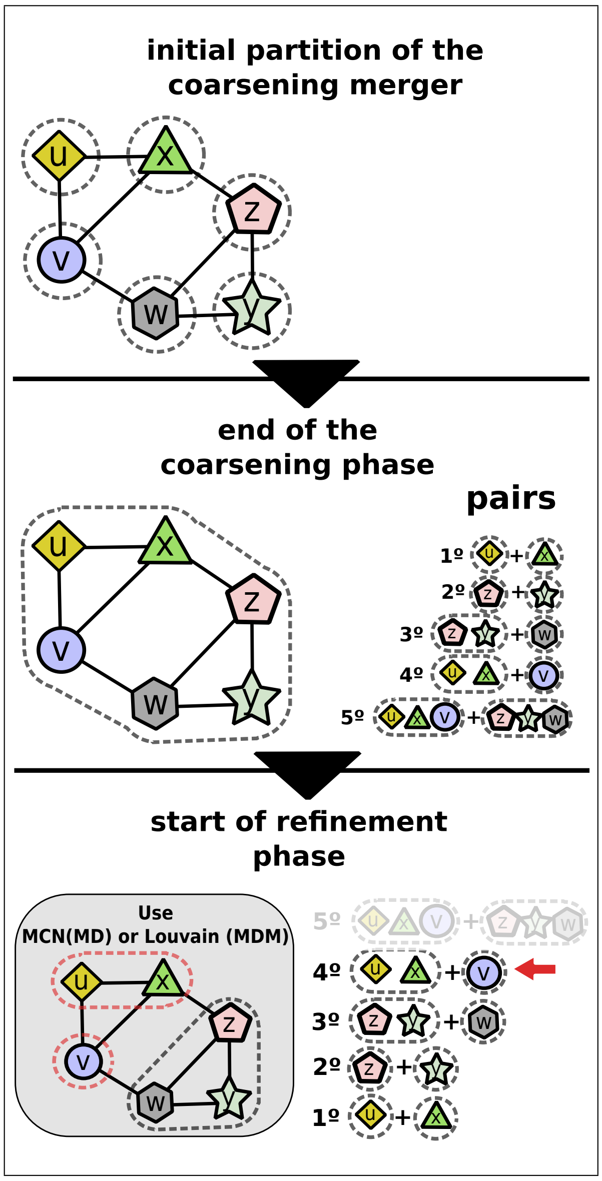
$$\Delta Q_{coarse}(s,c) = \frac{|E^s_c| - |E^s_{c_s}|}{|E|} + \frac{2k_s(k_{c_s} - k_c) - 2k_s^2}{4|E|^2}. \tag{4.7}$$

Algorithm 17: Multilevel density.

```
   Input : $G(V,E)$
   // constructive phase
1  $(best, levels, pairs) \leftarrow$ execute Algorithm 14 (G)
2  $numberLevels \leftarrow |levels|$
3  $partition \leftarrow levels_{numberLevels}$
   // refinement phase
4  for $l \leftarrow numberLevels - 1$ to $1$ do
5      separate $pairs_l$ as two clusters in $partition$
6      $partition \leftarrow$ execute Algorithm 16 or
7                      Louvain $(G, partition)$
8      if $D(best) < D(partition)$ then
9          $best \leftarrow partition$
10 return $best$
```

Our multilevel heuristics have a constructive phase that uses the Algorithm 14 (CM), passing the graph G as a parameter (line 1). Then the best partition, the *levels* partitions, and the list *pairs* is obtained. After that, the number of levels is stored in the variable

Figure 4.3: An example of the multilevel density heuristic in a graph of six nodes.


initial partition of the coarsening merger
U
X
Z
V
W
Y
end of the coarsening phase
pairs
1º U + X
2º Z + Y
3º Z Y + W
4º U X + V
5º U X V + Z Y W
start of refinement phase
Use MCN(MD) or Louvain (MDM)
5º U X V + Z Y W
4º U X + V
3º Z Y + W
2º Z + Y
1º U + X

numberLevels (line 2). Then the last partition of CM is set as the current partition (line 3). After the constructive phase, the refinement phase is started by repeating the loop between lines 4 and 8 for a number of times equal to $numberLevels - 1$, decreasing the variable l from $numberLevels - 1$ to 1. At each iteration, the two clusters of position l of the list *pairs* are set in the current partition (line 5). Then the current partition is used as input to the heuristic of Algorithm 16 (MCN) for MD version, or Louvain method (BLONDEL et al., 2008) for MDM version (line 6). After that, if the current partition is better than the best-found partition, the best partition is updated (lines 8 and 9). After the refinement phase, the best partition is returned, and the heuristic stops (line 10).

Figure 4.3 shows three steps of the multilevel density to understand better this heuristic. A graph with six nodes is submitted to the constructive phase, and the current partition is featured in each step (clusters are identified by dashed lines). The first one shows the starting partition at the beginning of the execution of CM. The second step shows the end of the constructive phase, where the best solution is on the left side, and the list *pairs* is on the right side. In the third step, the start of the refinement phase separates the merged clusters in the penultimate iterations during CM execution. After that step, the MCN (for MD version) or Louvain method is applied (for MDM version) on the current partition.

4.2 Hybrid Local Search

The HLSMD (Hybrid Local Search for Modularity Density) heuristic proposed in this thesis is seen in Algorithm 18. It is made up of two local search phases. In the first phase, the local search Move and Coarse Nodes (MCN) is applied. After that, the second phase is executed, in which the second local search Local Node Moving (LNM) is applied to the resulting partition from the MCN. The creation of HLSMD was motivated by MCN failures when finding good partitions for some of the tested instances, as is seen in Section 4.3.

This sequence was chosen because the MCN local search unites the nodes at each iteration to move entire subsets of nodes in the following iterations from their clusters. It is expected that LNM can improve the solution by changing clusters of each node for the resulting partition from MCN.

Algorithm 18: Hybrid local Search.

```
  Input : $G(V, E)$
1 $first \leftarrow MCN(G)$
2 $second \leftarrow LNM(G, first)$
3 if $D(first) > D(second)$ then
4     $best \leftarrow first$
5 else
6     $best \leftarrow second$
7 return $best$
```

4.3 Analysis of Results

In this section, we present methods and results of experiments performed with real graphs. The main objective is to compare our eight methods (CM, LNM, CM+LNM, MCN, CM+MCN, MD, MDM, HLSMD) among themselves and with CNM (CLAUSET; NEWMAN; MOORE, 2004), Louvain (BLONDEL et al., 2008), GAOD (LIU; ZENG, 2010), iMeme-Net (GONG et al., 2012), HAIN (KARIMI-MAJD; FATHIAN; AMIRI, 2014), and BMD-λ (COSTA et al., 2016).

CNM and Louvain are heuristics that treat Modularity Maximization (NEWMAN; GIRVAN, 2004). In essence, Modularity Maximization and Modularity Density Maximization are about the same problem, but they have different objective functions. Modularity Density Maximization was developed to avoid the resolution limit (FORTUNATO; BARTHÉLEMY, 2007) that happens in Modularity Maximization. We used the original versions of heuristics CNM and Louvain, so they applied the Modularity Maximization objective value to evaluate their solutions. For our comparisons, the Modularity Density Maximization objective function was calculated for all CNM and Louvain resulted partitions. The aim of obtaining this value is knowing if these methods or our heuristics lead to partitions with the highest Modularity Density Maximization objective value. As CNM and Louvain are scalable to instances with hundreds of thousands of nodes, we can compare results for the largest tested graphs.

The experiments were performed on a PC with an Intel Core i7 64 bits with 3.40GHz with 8192KB of cache memory and 8GB of RAM over Linux Ubuntu 14.04.1 LTS operating system. Each experiment was done by using a single thread. The language used was C++, with “GCC” compiler.

The presentation of results and the discussion are divided into three subsections: (*i*) set-up of experiments; (*ii*) gap to the optimal D value analysis; (*iii*) time required and amortized complexity analysis.

4.3.1 Set-up

The graphs tested are undirected and were selected from datasets of Batagelj and Mrvar (2006) and Leskovec and Krevl (2014). The first was chosen because it has popular graphs which are benchmarks for graph clustering problems, as can be seen in Xu, Tsoka and Papageorgiou (2007), Brandes et al. (2008), Agarwal and Kempe (2008), Liu and Zeng (2010), Aloise et al. (2013), Gong et al. (2012), Aloise et al. (2010), Djidjev and Onus (2013), Krzakala et al. (2013), Nascimento and Pitsoulis (2013), Newman (2013), Pizzuti (2012), Rotta and Noack (2011), Karimi-Majd, Fathian and Amiri (2014), Costa (2015). The second was used because it is a dataset collection for scalable heuristics (LESKOVEC; KREVL, 2014). This dataset collection has graphs with thousands of nodes. In this latter source, we chose the undirected and unweighted graphs with a limit of nodes equal to $500,000$. These graphs are representations of social, communication, and shopping networks. Higgs are the only directed graphs used in our experiments. They were chosen because of their number of nodes, which are similar to our largest chosen graphs, so they are converted to undirected graphs to complete our tests.

Table 4.1 shows details of the graphs used for the experiments. This table shows the number of nodes and edges, the source where each graph was obtained, and the Id column shows the identifier used as a graph reference for other tables in this document. All graph instances of Table 4.1 were submitted to all the heuristics tested. The heuristics were tested using 30 trials. In the tables of results, the bottom line named as “count” shows the number of the best results obtained by each heuristic.

Table 4.1: Real graph instances used in the experiments for our eight heuristics.

Id	Dataset name	Nodes	Edges	Source
1	Strike	24	38	(BATAGELJ; MRVAR, 2006)
2	Galesburg f	31	63	(BATAGELJ; MRVAR, 2006)
3	Galesburg d	31	67	(BATAGELJ; MRVAR, 2006)
4	Karate	34	78	(BATAGELJ; MRVAR, 2006)
5	Korea1	35	69	(BATAGELJ; MRVAR, 2006)
6	Korea2	35	84	(BATAGELJ; MRVAR, 2006)
7	Mexico	35	117	(BATAGELJ; MRVAR, 2006)
8	Sawmill	36	62	(BATAGELJ; MRVAR, 2006)
9	Dolphins	62	159	(BATAGELJ; MRVAR, 2006)
10	Lesmis	77	479	(BATAGELJ; MRVAR, 2006)
11	Polbooks	105	441	(BATAGELJ; MRVAR, 2006)
12	Adjnoun	112	425	(BATAGELJ; MRVAR, 2006)
13	Football	115	613	(BATAGELJ; MRVAR, 2006)
14	Jazz	198	2742	(BATAGELJ; MRVAR, 2006)
15	Celegansneural	297	3529	(BATAGELJ; MRVAR, 2006)
16	Celegans metabolic	453	2025	(BATAGELJ; MRVAR, 2006)
17	Email	1133	5451	(BATAGELJ; MRVAR, 2006)
18	Facebook combined	4039	88234	(LESKOVEC; KREVL, 2014)
19	Ca-grqc	5242	14496	(LESKOVEC; KREVL, 2014)
20	Ca-hepth	9877	25998	(LESKOVEC; KREVL, 2014)
21	Oregon1 010526	11174	23409	(LESKOVEC; KREVL, 2014)
22	Oregon2 010526	11461	32730	(LESKOVEC; KREVL, 2014)
23	Ca-hepph	12008	118521	(LESKOVEC; KREVL, 2014)
24	Ca-astroph	18772	198110	(LESKOVEC; KREVL, 2014)
25	Ca-condmat	23133	93497	(LESKOVEC; KREVL, 2014)
26	Email-enron	36692	183831	(LESKOVEC; KREVL, 2014)
27	Higgs-reply	38918	29895	(LESKOVEC; KREVL, 2014)
28	Brightkite edges	58228	214078	(LESKOVEC; KREVL, 2014)
29	Higgs-mention	116408	145774	(LESKOVEC; KREVL, 2014)
30	Gowalla edges	196591	950327	(LESKOVEC; KREVL, 2014)
31	Higgs-retweet	256491	327374	(LESKOVEC; KREVL, 2014)
32	Com-dblpgraph	317080	1049866	(LESKOVEC; KREVL, 2014)
33	Com-amazon	334863	925872	(LESKOVEC; KREVL, 2014)

4.3.2 Gap to the Best-known on Real Graphs

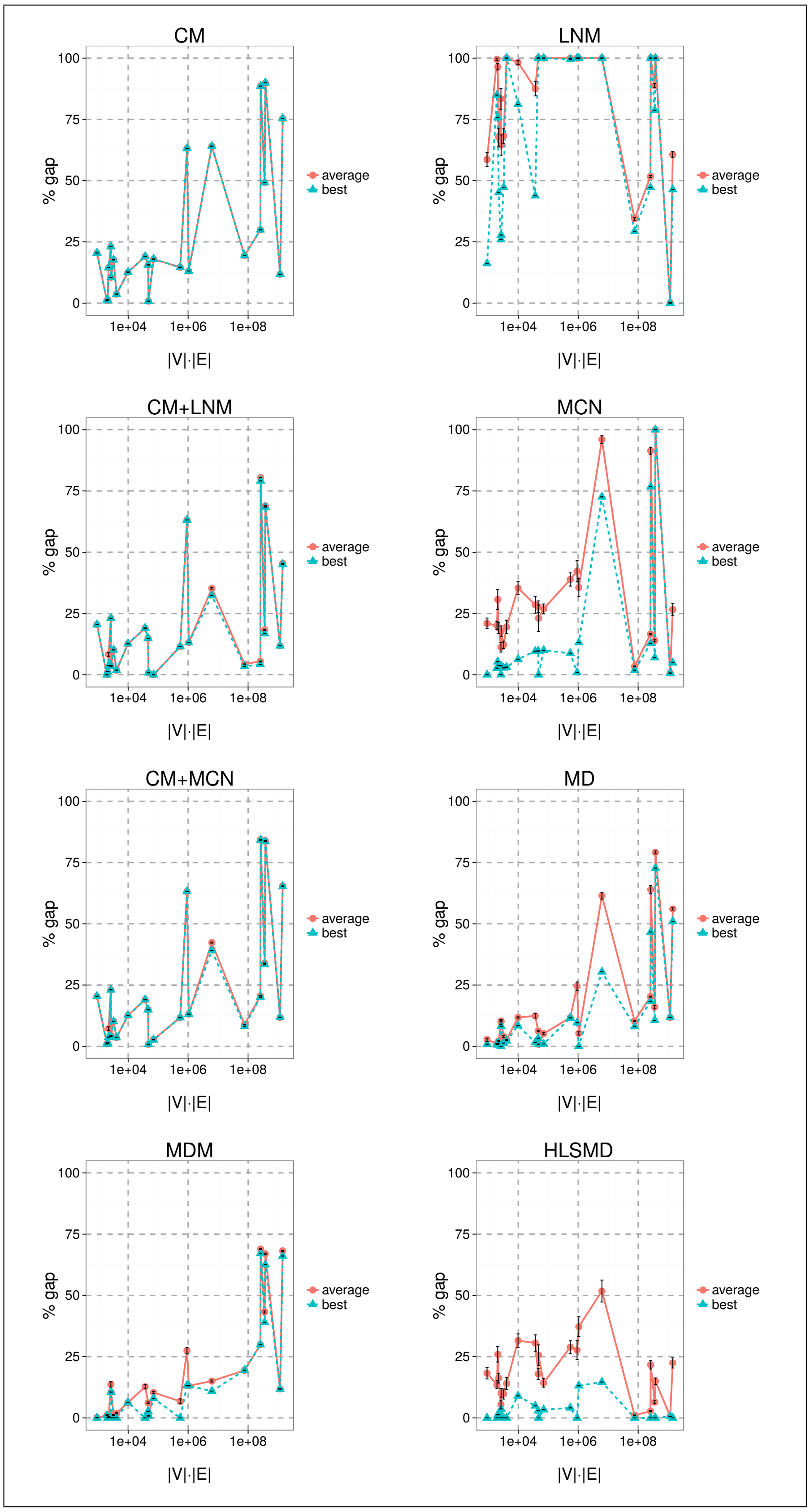
Table 4.2 compares the best values obtained by the heuristics tested. Featured values are the best-known D for each instance. The penultimate row shows the average gap to the best-known D values. Column D* shows the optimal values reported by Costa (2015).

The eight heuristics tested were compared with GAOD (LIU; ZENG, 2010), iMeme-Net (GONG et al., 2012), and HAIN (KARIMI-MAJD; FATHIAN; AMIRI, 2014) results. The CNM (CLAUSET; NEWMAN; MOORE, 2004) and Louvain (BLONDEL et al., 2008) found very different results when compared with other heuristics. This happens with CNM on “Adjnoun”, “Celegansneural”, “Ca-grqc”, “Ca-hepth”, “Ca-hepph”, “Ca-astroph”, “Ca-condmat”, “Higgs-reply”, and “Higgs-mention”. Louvain failed on “Adjnoun”, “Celegansneural”, “Com-dblpgraph”, and “Com-amazon”. When compared with CNM and Louvain, six of our novel heuristics surpass most of the best D values obtained.

The lowest gap is reported for GAOD, but the only known results are for two instances with 34 and 115 nodes. For most of the instances, the best results were obtained by HLSMD which are far from the best-known solution an average of 13.67%, considering the gap over the best partitions obtained by heuristics. This heuristic also reached the best D values for twelve instances tested, and it found six of eight known optimal partitions. When compared with iMeme-Net, and HAIN, the gap of HLSMD is better.

Figure 4.4 shows the gap of the best-known values for each of our eight heuristics. The size of each instance is represented as the product of the number of nodes and edges, composing the axis “$|V| \cdot |E|$”. A log scale is used in the $|V| \cdot |E|$ axis because of the high concentration of small instances. For the instances which the optimal value is known, we used the value obtained by Costa (2015), otherwise, the largest objective value obtained by the nine tested heuristics and the reported values for GAOD, iMeme-Net, HAIN, and BMD-λ in their papers. This figure shows that for all of our seven heuristics, the

Figure 4.4: Average gap (red) and the best gap (blue) with the standard error obtained to the best-known D value for each of our eight heuristics in the real graphs of Table 4.1.



gap grows as the instance graph grows in the number of nodes and edges. CM+LNM, CM+MCN, MD, MDM, and HLSMD presented the smallest gaps.

Table 4.3 compares the average D values among the heuristics tested. For probabilistic heuristics, the standard error is shown at the side of the D values. The best gap was obtained by MD, mainly for the largest instances. The smallest gaps were obtained by MD, MDM, CM+LNM, and HLSMD heuristics in the ordering of the best gaps. MD and CM+LNM obtained the best average D value for seven instances.

4.3.3 Temporal Analysis

To compare the time demanded by the eight heuristics tested, CNM, and Louvain, Figures 4.5 and 4.6 plots the average time and the standard error on a log scale, to show the relation between the size of instances and the required time for each heuristic. Time thresholds are printed as dashed lines for a millisecond, a second, ten seconds, a minute, ten minutes, an hour, and ten hours. These thresholds help to understand the time required by the heuristics to find the solutions.

Table 4.4 compares the average time in seconds of our tested heuristics CM, LNM, CM+LNM, MCN, CM+MCN, MD, MDM, HLSMD (reported in Sections 4.1 and 4.2). They are also compared with CNM (CLAUSET; NEWMAN; MOORE, 2004) and Louvain (BLONDEL et al., 2008) and the time reported for MDB2 (COSTA, 2015), HAIN (KARIMI-MAJD; FATHIAN; AMIRI, 2014), and BMD-λ (COSTA et al., 2016). For randomized heuristics, at the right-side of each time, there is the standard error. The Id column specifies the instance. The best results are marked in bold. The values "$<$ 0.001" are used for heuristics that required time less than 1 ms. The "-" values denote that there is not known a time result about that running.

The fastest heuristics are Louvain, CM, LNM and MCN which required less than a minute to execute the largest instances tested. For most of the instances, Louvain and MCN required less than 16 seconds for the largest instances. HLSMD found partitions to the largest instances in less than ten minutes. CNM and CM+LNM required about an

Table 4.2: Table for the comparison of the best D values for real graphs.

	MDB2		**GAOD**		**iMeme-Net**		**HAIN**		**BMD-λ**		**CNM**		**Louvain**		**CM**	
Id	**|C|**	**D***	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**
1	4	**8.861**	-	-	-	-	-	-	4	**8.861**	4	8.783	4	8.783	4	7.048
2	3	**8.286**	-	-	-	-	-	-	3	**8.286**	4	7.350	4	7.833	3	8.201
3	3	**6.927**	-	-	-	-	-	-	3	**6.927**	5	5.500	5	5.500	4	6.831
4	3	**7.845**	3	**7.845**	3	**7.845**	3	3.922	3	7.842	3	6.023	4	7.529	2	6.037
5	5	**10.967**	-	-	-	-	-	-	5	**10.967**	5	8.917	9	8.917	10	9.809
6	5	**11.143**	-	-	-	-	-	-	5	**11.143**	5	10.872	9	10.712	9	9.173
7	3	**8.718**	-	-	-	-	-	-	2	8.558	3	7.895	3	8.562	2	8.399
8	4	**8.623**	-	-	-	-	-	-	5	8.529	4	**8.623**	4	**8.623**	5	7.374
9	-	-	-	-	4	10.883	5	6.063	5	**12.125**	4	7.818	5	10.456	4	10.590
10	-	-	-	-	-	-	8	12.274	-	-	5	2.877	5	6.966	1	12.636
11	-	-	-	-	4	20.160	6	10.958	7	**21.965**	4	16.832	5	21.029	3	18.542
12	-	-	-	-	-	-	2	3.894	-	-	7	-5.996	6	1.563	1	7.589
13	-	-	10	43.370	8	29.321	11	22.194	-	-	7	33.135	9	42.023	10	36.379
14	-	-	-	-	-	-	-	-	-	-	4	39.115	4	39.291	2	42.446
15	-	-	-	-	-	-	-	-	-	-	226	-4932.870	6	-26.494	1	23.916
16	-	-	-	-	-	-	-	-	-	-	10	14.621	10	16.445	1	8.940
17	-	-	-	-	-	-	-	-	-	-	12	26.949	12	**35.592**	6	12.795
18	-	-	-	-	-	-	-	-	-	-	15	439.467	16	560.469	28	452.254
19	-	-	-	-	-	-	-	-	-	-	5057	-28395.500	395	847.234	757	1157.640
20	-	-	-	-	-	-	-	-	-	-	9581	-51029.000	477	864.041	1084	1172.680
21	-	-	-	-	-	-	-	-	-	-	37	57.046	37	64.520	26	15.319
22	-	-	-	-	-	-	-	-	-	-	68	**97.977**	34	87.476	21	14.996
23	-	-	-	-	-	-	-	-	-	-	11775	-236160.000	319	978.804	318	540.761
24	-	-	-	-	-	-	-	-	-	-	18620	-395620.000	328	884.886	290	545.375
25	-	-	-	-	-	-	-	-	-	-	22718	-185525.000	620	1278.840	1439	1812.920
26	-	-	-	-	-	-	-	-	-	-	1612	2815.710	1270	2322.730	1066	1626.400
27	-	-	-	-	-	-	-	-	-	-	30503	-33727.500	10699	13321.900	10787	13373.100
28	-	-	-	-	-	-	-	-	-	-	1718	**1900.830**	820	1208.930	576	695.947
29	-	-	-	-	-	-	-	-	-	-	109847	-261156.000	10661	21198.500	10865	21145.000
30	-	-	-	-	-	-	-	-	-	-	2842	3260.440	768	1710.940	344	199.802
31	-	-	-	-	-	-	-	-	-	-	14179	15900.900	13402	14900.000	13557	14838.400
32	-	-	-	-	-	-	-	-	-	-	3204	8995.930	247	992.399	17899	19607.100
33	-	-	-	-	-	-	-	-	-	-	1497	5969.950	262	1334.430	22598	34674.900
avg. gap		(0.0±0.0%)		(1.09±0.77%)		(13.08±6.3%)		(45.09±4.33%)		(0.22±0.19%)		(46.07±6.83%)		(34.25±5.39%)		(35.86±4.93%)
count		8		1		1		0		7		3		2		0

	LNM		**CM+LNM**		**MCN**		**CM+MCN**		**MD**		**MDM**		**HLSMD**	
Id	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**	**|C|**	**D**
1	6	7.426	4	7.048	4	**8.861**	24	7.048	4	8.783	4	**8.861**	4	**8.861**
2	8	1.250	3	**8.286**	3	8.062	3	8.201	3	8.201	3	8.201	3	**8.286**
3	7	1.680	4	6.841	3	6.558	4	6.831	3	6.886	3	**6.927**	4	6.671
4	5	5.809	2	6.037	3	7.842	2	6.037	3	**7.845**	3	7.842	3	**7.845**
5	12	7.917	10	10.591	10	10.567	39	10.531	10	10.067	10	9.809	9	**10.967**
6	12	5.872	9	10.014	10	10.833	9	10.014	9	10.994	9	**11.143**	9	**11.143**
7	7	-0.313	2	8.558	3	8.449	2	8.409	2	8.523	3	**8.718**	3	**8.718**
8	9	4.733	5	8.302	5	8.290	5	8.302	5	8.509	4	**8.623**	5	8.509
9	12	2.287	4	10.590	5	11.359	4	10.590	5	11.103	5	11.367	5	11.785
10	6	8.766	1	12.636	2	14.103	1	12.636	3	15.359	3	**15.606**	3	13.742
11	16	-2.997	3	18.703	6	19.835	3	18.686	5	21.230	5	21.361	6	20.961
12	13	-12.379	1	7.589	2	**7.651**	1	7.589	1	7.589	1	7.589	2	**7.651**
13	18	-4.198	10	**44.340**	12	39.911	10	43.159	11	43.916	8	40.777	12	40.667
14	15	0.251	2	44.003	8	45.350	2	43.894	3	43.950	4	**49.716**	4	46.167
15	22	-138.564	1	23.916	1	23.916	1	23.916	1	**27.525**	1	23.916	1	23.916
16	95	-138.183	1	8.940	17	24.055	1	8.940	13	21.929	9	21.051	15	**24.955**
17	127	-113.095	6	24.100	30	9.754	6	21.666	29	24.769	9	31.695	32	25.920
18	268	189.912	28	740.964	82	828.171	28	593.088	55	795.756	14	542.925	72	**875.074**
19	1245	1015.850	756	1387.410	894	**1409.860**	752	1319.740	827	1320.530	757	1157.640	891	1406.250
20	2170	883.768	1084	**1603.500**	1339	1460.200	1053	1338.040	1083	1366.390	1084	1172.680	1325	1427.330
21	1044	-91.217	26	28.020	515	31.260	23	21.249	226	**71.567**	38	43.996	516	34.703
22	1005	-239.495	21	46.969	258	-16.814	19	24.472	36	40.542	26	55.708	392	-0.039
23	1736	1177.010	318	1207.220	795	**2088.110**	318	762.494	842	1075.600	163	744.382	826	2060.010
24	2372	-1421.990	290	545.375	948	1909.920	290	545.375	295	1163.310	180	793.657	805	**2243.080**
25	4550	-163.533	1439	**3068.600**	1940	2762.480	1425	2270.500	1529	2395.840	1439	1812.920	1967	2765.220
26	5224	-774.428	1066	1630.520	2177	3110.160	1066	1629.090	1631	1631.330	1066	1626.400	1943	**3160.790**
27	13383	**15163.100**	10787	13373.100	13109	15069.900	10787	13373.100	10787	13373.100	10787	13373.100	13091	15073.000
28	9105	-336.309	576	792.180	1785	159.920	576	720.195	576	695.947	597	724.354	1804	-14.606
29	23568	34504.800	10865	21145.000	19514	29583.600	10812	21143.400	12834	**218910.000**	2298	176674.000	19514	29576.400
30	28510	-5271.990	344	517.059	7168	**3382.600**	154	223.019	344	199.802	544	1248.120	9521	3226.380
31	32176	23830.200	13557	14841.500	29991	23918.400	13557	14839.600	24186	18497.800	13557	14838.400	29944	**23937.700**
32	62472	11627.100	17779	**36070.200**	26405	35352.700	17179	26600.600	20140	29195.800	17899	19607.100	25893	35428.100
33	70350	-11574.700	22074	49203.100	39204	47848.400	21756	45998.600	29602	**49600.900**	22598	34674.900	39191	47868.200
avg. gap		(72.57±5.75%)		(24.62±4.71%)		(16.26±4.95%)		(29.96±4.88%)		(16.72±3.99%)		(21.87±3.82%)		(13.67±4.83%)
count		1		5		5		0		5		7		12

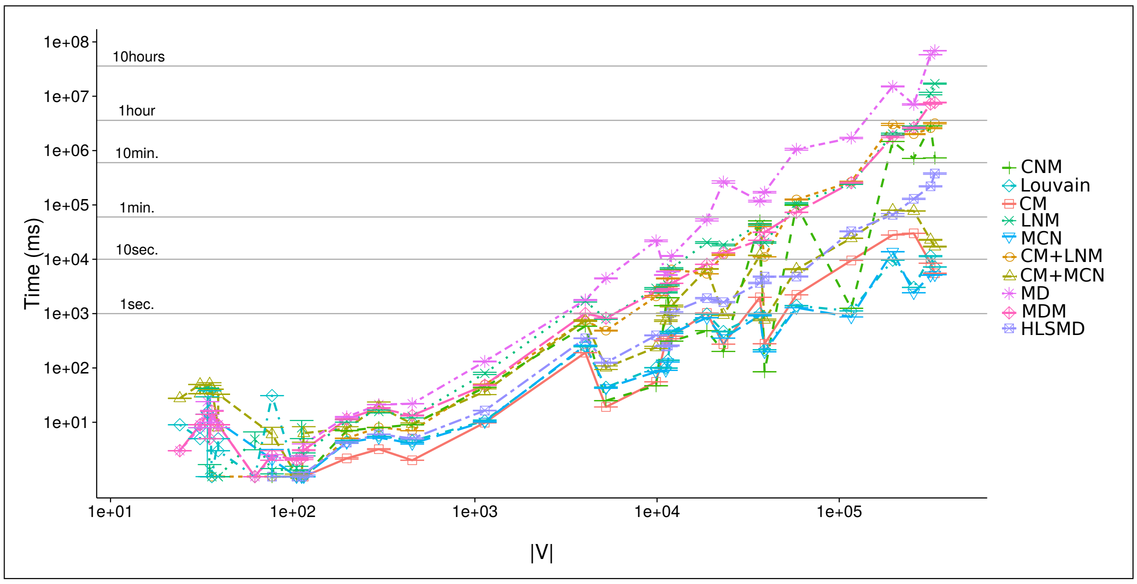
Table 4.3: Table for the comparison of the average D values for real graphs.

	CNM		Louvain		CM		LNM		CM+LNM	
Id	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**
1	4.0	8.783	4.0±0.0	8.783±9.729	4.0	7.047	7.7±0.11	3.647±0.256	4.0±0.0	7.047±6.486
2	4.0	7.3	4.0±0.0	7.833±4.864	3.0	8.2	9.57±0.13	-2.496±0.397	3.0±0.0	**8.285±6.486**
3	5.0	5	5.0±0.0	5±0.	4.0	6.831	8.3±0.13	-0.974±0.299	4.0±0.0	6.84±3.243
4	3.0	6.022	4.0±0.0	7.529±6.486	2.0	6.036	6.8±0.19	.26±0.53	2.0±0.0	6.036±3.243
5	5.0	8.916	9.0±0.0	8.916±6.486	10.0	9.809	13.23±0.14	3.724±0.519	10.0±0.0	**10.554±0.005**
6	5.0	10.872	9.0±0.0	10.712±0.	9.0	9.173	13.13±0.14	3.047±0.576	9.0±0.0	10.014±6.486
7	3.0	7.894	3.0±0.0	**8.561±6.486**	2.0	8.398	8.27±0.17	-8.87±1.002	2.0±0.0	8.557±9.729
8	4.0	**8.623**	4.0±0.0	**8.623±0.**	5.0	7.373	10.33±0.12	2.586±0.394	5.0±0.0	7.908±0.062
9	4.0	7.817	5.0±0.0	10.455±3.243	4.0	10.59	14.03±0.21	-4.639±0.874	4.0±0.0	10.59±1.621
10	5.0	2.877	5.0±0.0	6.056±0.03	1.0	12.636	8.5±0.17	.791±0.689	1.0±0.0	12.636±6.486
11	4.0	16.832	5.0±0.0	**21.029±1.297**	3.0	18.542	20.27±0.29	-22.806±1.426	3.0±0.0	18.703±1.297
12	7.0	-5.996	6.0±0.0	1.563±1.621	1.0	**7.589**	16.8±0.28	-24.816±1.149	1.0±0.0	**7.589±6.486**
13	7.0	33.134	9.0±0.0	42.023±3.891	10.0	36.378	21.97±0.27	-52.907±3.335	10.0±0.0	**44.34±0.**
14	4.0	39.115	4.0±0.0	39.291±2.594	2.0	42.445	15.63±0.21	-23.687±2.591	2.0±0.0	44.002±3.891
15	226.0	-4932.8	6.0±0.0	-26.494±0.	1.0	23.915	25.33±0.39	-179.328±3.358	1.0±0.0	23.915±1.297
16	10.0	14.621	10.0±0.0	16.444±1.945	1.0	8.94	103.53±0.67	-169.749±2.973	1.0±0.0	8.94±6.486
17	12.0	26.94	12.0±0.0	**32.591±0.146**	6.0	12.794	134.97±0.74	-140.349±2.587	6.0±0.0	23.039±0.115
18	15.0	439.46	15.4±0.13	548.58±2.276	28.0	452.25	275.53±0.98	99.452±7.713	28.0±0.0	727.877±1.616
19	5057.0	-28395	391.43±0.15	829.852±0.751	757.0	1157.6	1255.37±1.24	941.533±8.522	756.0±0.0	1374.199±1.094
20	9581.0	-51029	476.43±0.25	855.205±1.286	1084.0	1172.6	2184.57±2.13	809.031±6.258	1084.0±0.0	**1582.598±2.657**
21	37.0	**57.046**	32.47±0.48	56.216±0.731	26.0	15.319	1068.87±2.31	-122.705±2.629	26.0±0.0	26.248±0.101
22	68.0	**97.97**	29.33±0.52	75.229±0.99	21.0	14.996	1028.97±2.45	-283.345±4.11	21.0±0.0	46.196±0.08
23	11775.0	-236160	315.07±0.49	947.558±3.214	318.0	540.76	1749.8±3.17	863.612±26.456	318.0±0.0	1197.8±0.979
24	18620.0	-395620	325.7±0.43	850.227±3.002	290.0	545.37	2392.83±2.82	-1704.12±27.789	290.0±0.0	545.37±0.
25	22718.0	-185525	619.67±0.35	1256.843±2.332	1439.0	1812.9	4593.9±3.46	-375.738±16.079	1439.0±0.0	**3050.93±1.453**
26	1612.0	**2815.7**	1244.4±3.04	2256.997±9.034	1066.0	1626	5273.37±4.65	-940.635±17.365	1066.0±0.0	1630.25±0.025
27	30503.0	-33727	10696.4±0.33	13316.503±0.656	10787.0	13373	13386.37±2.46	**15140.0±2.057**	10787.0±0.0	13373±9.963
28	1718.0	**1900.8**	738.33±5.24	1115.868±6.989	576.0	695.94	9135.4±5.88	-473.961±12.381	576.0±0.0	791.535±0.08
29	109847.0	-261156	10640.83±1.57	21147.2±4.042	10865.0	21145	23565.37±6.31	34401.086±8.728	10865.0±0.0	21145±0.
30	2842.0	**3260.4**	541.9±12.73	1254.862±26.641	344.0	199.8	28636.2±18.19	-5617.459±31.956	344.0±0.0	503.208±1.502
31	14179.0	15900	13376.43±2.47	14857.756±4.042	13557.0	14838	32131.73±7.33	23743.6±8.854	13557.0±0.0	14841.486±0.009
32	3204.0	8995.9	211.4±2.49	904.29±8.406	17899.0	19607	62660.7±17.51	10893.496±58.36	17779.17±0.26	**35987.933±9.023**
33	1497.0	5969.9	237.33±1.39	1214.9±6.948	22598.0	34674	70483.93±16.39	-12207.953±58.453	22075.07±0.34	49137.943±6.809
avg. gap		(43.24±7.12%)		(32.43±5.54%)		(32.92±4.87%)		(82.59±4.88%)		(21.66±4.53%)
count		6		4		1		1		7

	MCN		CM+MCN		MD		MDM		HLSMD	
Id	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**	**avg. |C|**	**avg. D**
1	4.9±0.18	7.004±0.189	24.0±0.0	7.047±6.486	4.2±0.07	8.62±0.056	4.0±0.0	**8.853±0.004**	4.73±0.22	7.224±0.204
2	5.23±0.15	6.625±0.135	3.0±0.0	8.2±9.729	3.0±0.0	8.2±9.729	3.0±0.0	8.2±9.729	4.6±0.21	7.329±0.136
3	4.47±0.21	4.784±0.289	4.0±0.0	6.831±8.107	3.7±0.08	**6.847±0.004**	3.97±0.03	6.834±0.003	4.07±0.17	5.691±0.16
4	3.43±0.1	6.966±0.149	2.0±0.0	6.036±3.243	3.63±0.09	7.062±0.081	3.07±0.17	6.772±0.082	3.77±0.08	**7.544±0.044**
5	10.93±0.12	9.026±0.239	33.2±2.12	10.489±0.009	10.0±0.05	9.837±0.011	10.0±0.0	9.809±3.243	10.53±0.15	9.93±0.161
6	9.87±0.17	9.782±0.162	9.0±0.0	10.014±6.486	9.73±0.08	10.7±0.047	8.97±0.03	**11.004±0.024**	9.63±0.14	10.572±0.131
7	3.13±0.15	7.013±0.233	2.0±0.0	8.4076±0.0006	2.0±0.0	8.486±0.007	2.77±0.08	8.552±0.023	3.23±0.15	7.329±0.27
8	6.43±0.2	6.975±0.193	5.0±0.0	8.001±0.062	5.1±0.07	8.435±0.022	4.07±0.04	8.573±0.01	6.47±0.15	7.343±0.141
9	8.3±0.24	7.834±0.315	4.0±0.0	10.59±1.621	4.5±0.13	10.698±0.031	5.0±0.0	**11.367±0.**	7.87±0.21	8.476±0.306
10	3.4±0.33	11.045±0.606	1.0±0.0	12.636±6.486	2.07±0.19	**13.668±0.137**	3.6±0.12	13.618±0.136	3.83±0.24	11.072±0.603
11	8.43±0.31	15.861±0.506	3.0±0.0	18.686±1.297	4.93±0.15	20.584±0.073	4.3±0.08	20.606±0.07	7.27±0.27	17.25±0.445
12	3.13±0.27	5.856±0.423	1.0±0.0	**7.589±6.486**	1.0±0.0	**7.589±6.486**	1.0±0.0	**7.589±6.486**	4.33±0.33	4.701±0.65
13	13.1±0.12	32.427±0.874	10.0±0.0	43.093±0.006	11.47±0.1	42.034±0.161	9.27±0.14	39.735±0.21	13.0±0.09	34.014±0.815
14	9.33±0.31	30.392±1.314	2.0±0.0	43.893±3.891	2.97±0.03	43.899±0.049	2.73±0.09	**46.35±0.476**	7.77±0.22	31.902±1.274
15	2.2±0.18	17.7217±1.0003	1.0±0.0	23.915±1.297	1.0±0.0	**26.073±0.176**	1.0±0.0	23.915±1.297	2.17±0.2	18.563±0.988
16	14.4±0.84	14.006±1.052	1.0±0.0	8.94±6.486	15.37±0.56	**18.31±0.41**	7.97±0.14	17.607±0.288	13.9±0.72	14.501±1.277
17	47.1±1.56	-40.99±5.778	6.0±0.0	20.523±0.065	9.27±1.29	13.717±0.48	8.33±0.12	30.241±0.142	47.33±1.43	-36.977±5.263
18	83.73±1.53	766.273±5.01	28.0±0.0	588.346±0.362	62.7±1.07	748.014±6.54	15.17±0.47	505.525±3.648	82.1±1.38	**811.953±8.026**
19	891.93±1.86	1387.7±2.265	752.07±0.12	1309.04±1.503	824.47±1.76	1288.06±3.137	757.0±0.0	1157.6±8.302	889.4±2.35	**1391.667±1.677**
20	1313.8±2.52	1395.893±5.227	1058.7±0.44	1328.32±0.933	1076.4±2.16	1334.071±3.217	1084.0±0.0	1172.6±0.	1312.4±2.69	1392.319±3.482
21	525.37±3.17	10.448±2.183	23.0±0.0	20.969±0.025	164.4±18.09	48.344±2.185	45.57±1.35	41.706±0.255	521.93±2.75	10.101±2.196
22	242.03±5.55	-75.634±4.828	19.0±0.0	23.947±0.041	48.13±4.17	31.012±0.817	22.7±0.8	49.128±0.48	252.93±9.06	-70.522±5.101
23	611.5±31.64	1610.76±50.83	318.0±0.0	758.168±0.446	527.67±52.74	964.258±9.452	156.63±0.42	698.195±4.202	514.83±25.66	**1629.4±30.373**
24	856.3±19.18	1732.02±18.902	290.0±0.0	545.37±0.	404.57±14.47	987.852±13.413	177.2±0.35	767.055±2.352	848.4±15.91	**1735.69±26.169**
25	1946.93±5.43	2547.691±17.184	1424.53±0.33	2259.251±1.164	1503.57±3.52	2290.162±7.427	1439.0±0.0	1812.9±2.075	1953.9±5.08	2594.441±14.245
26	1646.43±71.82	2129.851±77.384	1066.0±0.0	1629.02±0.006	1085.48±19.14	1626.5±0.167	1066.0±0.0	1626±1.245	1832.03±101.87	2418.74±74.318
27	13126.77±3.12	15052.6±1.555	10787.0±0.0	13373±9.963	10787.0±0.0	13373±9.963	10787.0±0.0	13373±9.963	13119.03±3.64	15055.066±1.594
28	1848.77±17.68	-484.12±41.074	576.0±0.0	719.681±0.04	576.0±0.0	695.94±0.	585.83±1.71	701.768±1.494	1845.33±14.23	-526.119±38.053
29	19478.73±8.75	29451.313±11.372	10811.07±0.04	21142.853±0.027	11194.47±472.72	**201597.133±2676.205**	2209.93±45.03	176299.034±42.201	19472.4±7.72	29451.086±12.113
30	5079.97±400.18	1546.177±215.968	137659.43±16435.23	219.318±0.309	344.0±0.0	199.8±1.037	462.73±7.18	1064.142±13.586	5891.77±494.7	1975.081±215.401
31	29947.77±9.43	**23844.486±7.381**	13557.0±0.0	14839.566±0.008	19930.57±797.82	16673.353±289.928	13557.0±0.0	14838±7.585	29939.17±6.3	23842.763±9.858
32	26319.23±48.39	34911.956±35.934	17180.6±1.78	26550±4.112	19912.1±16.34	28898.173±30.404	17899.0±0.0	19607±1.992	26324.0±39.14	34908.0±34.789
33	39238.03±16.59	47573.886±19.908	21750.87±1.89	45937.763±5.837	29662.43±37.9	**49287.083±25.03**	22598.0±0.0	34674±2.656	39222.5±17.41	47623.1±27.278
avg. gap		(28.56±5.22%)		(27.16±4.75%)		(17.68±4.14%)		(20.36±3.75%)		(25.62±5.36%)
count		1		1		7		5		5

hour to execute the largest instances. MDM is only faster than MD.

Figure 4.5: Average runtime according to the number of nodes for each heuristic tested.

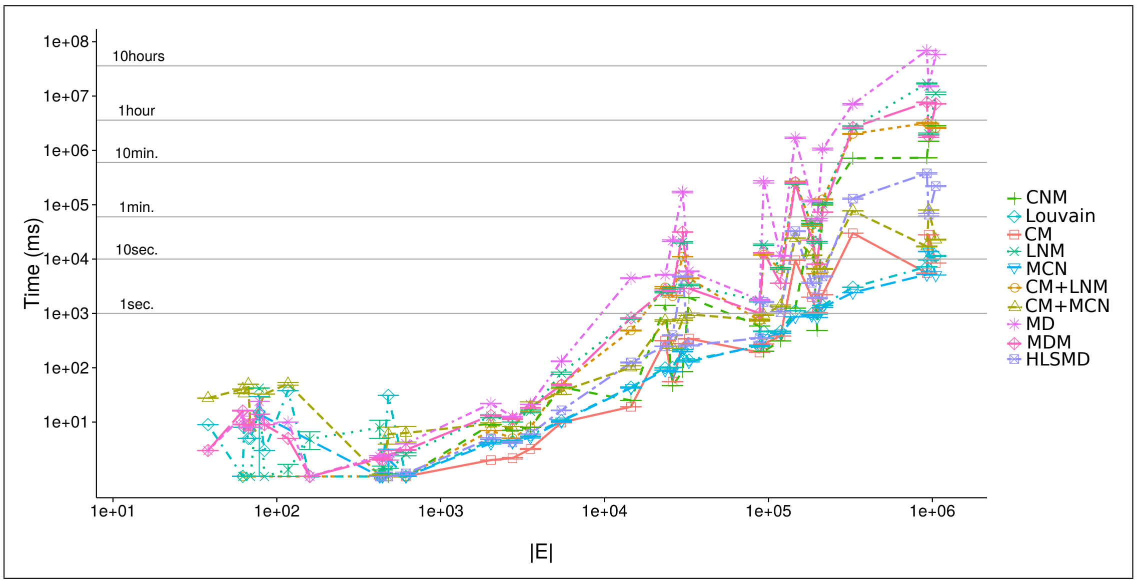


Louvain and MCN required almost the same time. Our results clearly suggested that our eight heuristics are faster than GAOD, iMeme-Net, and HAIN when compared with the results reported in Karimi-Majd, Fathian and Amiri (2014), where the fastest heuristic required 78 seconds for instances with less than 3,000 edges. The exact models of Costa (2015) required more than a minute to find optimal partitions for instances with 40 nodes, using a parallel architecture for its experiments. All our heuristics found partitions in less than a second for instances with less than 5,000 nodes. MCN, CM+MCN and HLSMD found solutions for graphs with at most 100,000 edges in less than seven seconds.

To confirm the time results, amortized complexity analysis was carried out using the linear regression model with the polynomial hypothesis to determine how many operations are required, considering the number of nodes and edges of the instances tested. Let $n = |V|$ and $m = |E|$, then the amortized time complexity is shown below:

- CM: $n^{0.78\pm0.09}m^{0.62\pm0.11}$;

Figure 4.6: Average runtime according to the number of edges for each heuristic tested.



- LNM: $n^{1.02\pm0.11}m^{0.07\pm0.09}$;
- CM+LNM: $n^{1.99\pm0.1}m^{0.14\pm0.11}$;
- MCN: $n^{0.19\pm0.05}m^{1.23\pm0.06}$;
- CM+MCN: $n^{0.41\pm0.07}m^{1.04\pm0.07}$;
- MD: $n^{1.92\pm0.09}m^{0.14\pm0.1}$;
- MDM: $n^{1.64\pm0.08}m^{0.2\pm0.09}$;
- HLSMD: $n^{1.22\pm0.07}m^{0.31\pm0.08}$.

The time required by the LNM, CM+LNM, MD, MDN, and HLSMD heuristics are strongly dependent on the number of nodes. MCN and CM+MCN are strongly dependent on the number of edges. These results emphasize the sublinear time required by CM for the largest instances tested.

Table 4.4: Average time in seconds of tested heuristics and results from literature for real graphs.

Id	**MDB2**	**HAIN**	**BMD-λ**	**CNM**	**Louvain**	**CM**	**LNM**	**CM+LNM**	**MCN**	**CM+MCN**	**MD**	**MDM**	**HLSMD**
1	2.3	-	2.3	< **0.001**	< **0.001**	< **0.001**	< **0.001**	< **0.001**	< **0.001**	.004±0.001	< **0.001**	< **0.001**	< **0.001**
2	2.6	-	2.6	< **0.001**	.0002±0.0002	< **0.001**	< **0.001**	< **0.001**	< **0.001**	.005±0.002	< **0.001**	< **0.001**	< **0.001**
3	5.0	-	2.9	< **0.001**	.0001±0.0001	< **0.001**	< **0.001**	< **0.001**	< **0.001**	.008±0.003	.0002±0.0002	< **0.001**	< **0.001**
4	4.7	.33	2.4	< **0.001**	.002±0.001	< **0.001**	.005±0.002	< **0.001**	.0004±0.0004	< **0.001**	< **0.001**	.0004±0.0004	< **0.001**
5	19.1	-	2.8	< **0.001**	.0001±0.0001	< **0.001**	3.33333±0.00003	< **0.001**	< **0.001**	.0013±0.0005	.0002±0.0002	< **0.001**	< **0.001**
6	36.3	-	2	< **0.001**	< **0.001**	< **0.001**	< **0.001**	< **0.001**	< **0.001**	.005±0.002	< **0.001**	< **0.001**	< **0.001**
7	13.3	-	3.3	< **0.001**	.001±0.001	< **0.001**	.00013±0.00007	< **0.001**	< **0.001**	.008±0.003	.0003±0.0003	.0001±0.0001	< **0.001**
8	10.7	-	1.8	< **0.001**	3.33333±0.00003	< **0.001**	< **0.001**	3.33333±0.00003	< **0.001**	.006±0.002	.003±0.001	.0005±0.0005	< **0.001**
9	-	-	17.6	< **0.001**	< **0.001**	< **0.001**	.002±0.001	< **0.001**	< **0.001**	< **0.001**	< **0.001**	.00033±0.00008	< **0.001**
10	-	-	-	< **0.001**	.001±0.001	< **0.001**	.0012±0.0001	.00026±0.00008	.0008±0.0004	< **0.001**	.0022±0.0002	.002±0.0004	< **0.001**
11	-	7.56	72.4	< **0.001**	.00033±0.00008	< **0.001**	.0014±0.0001	< **0.001**	< **0.001**	.00106±0.00004	.00203±0.00003	.00203±0.00003	< **0.001**
12	-	-	-	< **0.001**	< **0.001**	**3.33333**	.007±0.002	< **0.001**	< **0.001**	.00103±0.00003	.00233±0.00008	.00213±0.00007	< **0.001**
13	-	-	-	< **0.001**	< **0.001**	**.00013**	.0025±0.0001	< **0.001**	< **0.001**	.006±0.001	< **0.001**	.00306±0.00006	.00113±0.00006
14	-	-	-	< **0.001**	.004±0.0002	**.00216**	.0106±0.0005	.00506±0.00004	.0042±0.0001	.007±0.0005	.0125±0.0002	.011±0.0002	.0042±0.0001
15	-	-	-	< **0.001**	.0057±0.0002	**.003**	.015±0.0009	.00806±0.00004	.0052±0.0001	.019±0.003	.0211±0.0001	.0183±0.0002	.00603±0.00003
16	-	-	-	< **0.001**	.0045±0.0002	< **0.001**	.0128±0.0007	< **0.001**	.004±0.0001	.0093±0.0003	.0219±0.0001	.0133±0.0002	.005±0.0001
17	-	-	-	.04	.0106±0.0004	**.0098**	.079±0.003	.049±0.001	.01±0.0002	.036±0.004	.131±0.002	.0482±0.0008	.0164±0.0001
18	-	-	-	.58	.246±0.002	**.1912**	1.69±0.06	.73±0.01	.259±0.001	.74±0.02	1.77±0.04	1.006±0.009	.351±0.004
19	-	-	-	**.02**	.0439±0.0006	**.01913**	.8±0.02	.485±0.009	.0425±0.0004	.101±0.007	4.45±0.09	.825±0.001	.124±0.002
20	-	-	-	**.04**	.101±0.0008	**.055**	2.91±0.1	2.07±0.07	.0848±0.0005	.23±0.01	21.86±0.75	2.536±0.006	.397±0.007
21	-	-	-	1.4	.099±0.001	.3152	2.45±0.06	3.01±0.11	**.09±0.0001**	.74±0.02	5.14±0.08	2.692±0.008	.248±0.002
22	-	-	-	1.95	.138±0.001	.3448	3.34±0.1	4.43±0.08	**.128±0.0003**	.96±0.05	5.95±0.1	2.87±0.01	.258±0.006
23	-	-	-	**.31**	.467±0.005	.383	6.79±0.2	6.49±0.08	.431±0.002	1.38±0.05	11.5±0.2	3.6±0.02	1.05±0.03
24	-	-	-	**.48**	.98±0.01	1.027	20.34±0.7	5.393±0.001	.84±0.004	6.63±0.09	53.12±2.47	8.06±0.05	1.92±0.05
25	-	-	-	**.2**	.467±0.007	.2743	18.33±0.6	12.06±0.31	.353±0.001	.94±0.02	264.37±12.96	13.15±0.04	1.62±0.04
26	-	-	-	50.91	.99±0.01	1.993	43.9±1.51	42.03±2.03	**.92±0.01**	11.76±0.13	117.52±4.21	22.35±0.1	3.66±0.1
27	-	-	-	**.08**	.216±0.0008	.2785	20.33±0.62	11.0786±0.0008	.1981±0.0009	.76±0.02	171.24±5.4	31.47±0.39	4.78±0.11
28	-	-	-	98.65	1.39±0.01	2.211	107.07±2.94	125.9±2.67	**1.285±0.004**	6.58±0.14	1058±41.0	73.1±0.74	4.8±0.11
29	-	-	-	1.25	1.12±0.005	9.51	243.29±4.76	266.33±6.43	**.866±0.001**	24.35±0.21	1706.84±47.34	256.5±6.47	32.67±0.61
30	-	-	-	1469.9	**9.64±0.08**	27.953	2017.02±74.24	3033.73±126.56	13.77±0.12	79.98±0.42	15163.92±303.5	1794.43±65.17	65.63±3.49
31	-	-	-	716.21	3.03±0.01	30.01	2654.78±167.85	2009.44±40.7	**2.4±0.005**	77.33±0.52	7075.26±204.72	2631.89±76.81	129.34±3.36
32	-	-	-	2848.74	11.4±0.26	8.464	11228.06±576.36	2587.73±56.07	**5.09±0.01**	22.79±0.42	58088.12±282.65	7196.02±84.03	220.03±5.1
33	-	-	-	732.19	7.2±0.05	5.52	17056.63±403.22	3196.53±88.85	**5.2±0.01**	17.1±0.31	68860.14±667.5	7657.89±117.13	377.88±11.11
count	0	0	0	22	6	20	5	12	19	3	6	5	12

4.4 Chapter Summary

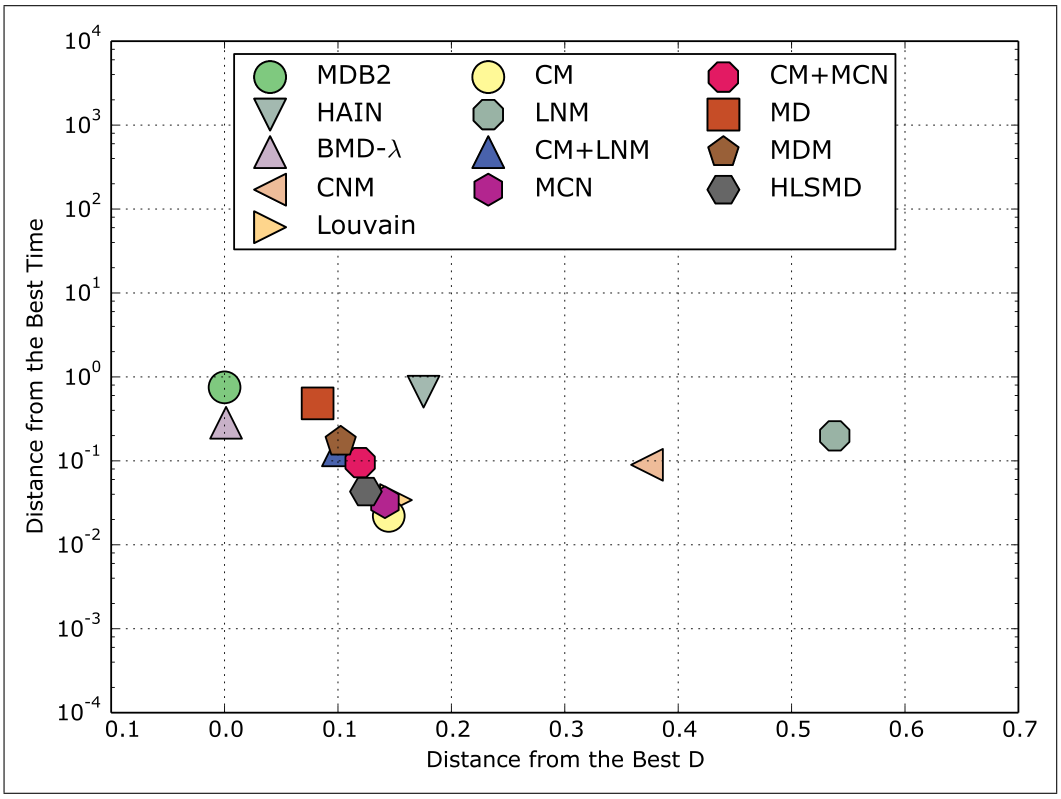
We presented and analyzed efficient heuristics for Modularity Density Maximization which solve instances with hundreds of thousands of nodes. Eight heuristics are compared with GAOD (LIU; ZENG, 2010), iMeme-Net (GONG et al., 2012), HAIN (KARIMI-MAJD; FATHIAN; AMIRI, 2014), and BMD-λ (COSTA et al., 2016) Modularity Density Maximization heuristics and CNM (CLAUSET; NEWMAN; MOORE, 2004) and Louvain (BLONDEL et al., 2008) Modularity Maximization heuristics. Our eight heuristics were also compared with Costa (2015) exact results to identify gaps in the optimal partitions. To show the scalability of our methods, they were tested using real datasets from the "Stanford Large Network Dataset Collection" (LESKOVEC; KREVL, 2014).

Our experiments suggest that six of our eight heuristics (CM+LNM, MCN, CM+MCN, MD, MDM, and HLSMD) find solutions with higher D value than GAOD, iMeme-Net, HAIN, and BMD-λ for some of the tested graphs (see Table 4.2). The experiments also suggest that these six heuristics were more scalable than HAIN and BMD-λ (Table 4.4).

With the selected dataset collections composed of real graphs, the fastest heuristics are Louvain, CM, MCN, and CM+MCN requiring at most 1m30s to run the largest tested instances. Louvain, CM, and MCN required at most 32 seconds.

Figure 4.7 can be used to compare results from the literature and tested heuristics. The axis "Distance from the best time" shows the relative distance from the best time obtained for each graph on a log scale, and the axis "Distance from the best D" is the relative distance from the best D value obtained. The closer the heuristic is zero on both axes, the better are the average time and D value.

Figure 4.7: Distance from the best time and the best D value, considering results from the literature and from our experiments.



5 ALGORITHMIC CONTRIBUTIONS

There are six exact known algorithms for Modularity Density Maximization. Li et al. (2008) have presented an exact binary nonlinear mathematical program (0-1 NLP) that is difficult to use with typical solvers. Karimi-Majd, Fathian and Amiri (2014) have reported an improvement to the Li et al. (2008) model in which the parameter of the number of clusters is no longer required. Costa (2015) has presented four different mixed-integer linear models converted from the 0-1 NLP. Instances with at most 40 nodes have been tested, and the results have indicated the difficulty in solving it exactly.

In this context, we developed six column generation methods for Modularity Density Maximization. They use the heuristic HLSMD described in Chapter 4 to generate the initial solution considering results of Table 4.2. In the experimental analysis, our column generations provided only integer solutions, so no other procedure (such as branch and price) was needed. Our six column generation methods are compared to the best exact models of Costa (2015). The obtained results suggest that two of our methods are state-of-the-art for exact Modularity Density Maximization. They have solved instances where the optimal solution had not previously been proved.

The remainder of this chapter is organized as follows. Section 5.1 explains how our column generation methods work, and general details about them are specified. Section 5.2 describes our two heuristics for the auxiliary problem of our column generations. Section 5.3 presents a detailed experimental analysis of the methods.

The results reported in this Chapter were published in the Computers & Operations Research journal; in 2017.

5.1 On Column Generation for Modularity Density Maximization

It is well-known that column generation is a technique used to solve mathematical linear programming problems with an exponential number of variables (ALOISE et al., 2010; NASH, 2013). Thus, for these problems, it is impossible to generate and store all variables, so the column generation can solve larger instances than other methods. The technique is based on Dantzig-Wolfe decompositions (DANTZIG; WOLFE, 1960).

In column generation, the linear problem is called the Master Problem (MP). A column generation starts by executing a Restricted Master Problem (RMP) that is a version of the MP with a reduced number of variables. Iteratively, the method tries to find a new variable that best optimizes the objective function for the MP by using a mathematical programming problem known as the Auxiliary Problem (AP). When no such variable is found, the procedure stops with the optimal solution for the MP. In this procedure, each variable is a column for linear programming problems.

The MP used in our column generations for Modularity Density Maximization can be seen in the linear problem of Equation (5.1) below. This problem was derived from the Modularity Maximization MP of Aloise et al. (2010). Each cluster is a subset of nodes, so there are $|T| = 2^{|V|}$ possible clusters, where $T = \left\{1, \dots, 2^{|V|}\right\}$. The variables z_t are binary in the original MP, but they were relaxed to obtain the dual problem. If $z_t = 0$, the cluster t does not belong to the solution; when $z_t = 1$, the cluster t belongs to the solution. The value of c_t is the contribution of cluster t to the objective function. This value is defined in Equation (5.2). The value a_{vt} is a binary number. If $a_{vt} = 1$, the node $v \in V$ belongs to the cluster t, and when $a_{vt} = 0$, the node v does not belong to this cluster. For all $u, v \in V$, the constant $w_{uv} = 1$ if $\{u, v\} \in E$, and $w_{uv} = 0$ if $\{u, v\} \notin E$. The constraint (5.1b) of the MP is required to define that each node belongs to one cluster.

$$\max \sum_{t \in T} c_t z_t \tag{5.1a}$$

$$\text{subject to: } \sum_{t \in T} a_{vt} z_t = 1, \forall v \in V \tag{5.1b}$$

$$z_t \geq 0, \forall t \in T. \tag{5.1c}$$

$$c_t = \frac{4 \sum_{u,v \in V : u<v} a_{ut} a_{vt} w_{uv} - \sum_{v \in V} d_v a_{vt}}{\sum_{v \in V} a_{vt}}. \tag{5.2}$$

The dual problem is necessary to obtain a new variable which has reduced cost. The dual problem of the linear model for Modularity Density Maximization (5.1) can be seen in the linear model (5.3) below. There is a dual variable λ_v for each constraint in the MP. For each variable in the MP, there is a constraint in the dual model.

$$\min \sum_{v \in V} \lambda_v \tag{5.3a}$$

$$\text{subject to: } \sum_{v \in V} a_{vt} \lambda_v \geq c_t, \forall t \in T \tag{5.3b}$$

$$\lambda_v \in \mathbb{R}, \forall v \in V. \tag{5.3c}$$

For the column generations reported in this thesis, we use two different auxiliary problems. They are AP-I and AP-II which are mixed-integer nonlinear problems.

To define AP-I, the constraints of the dual values are used to identify the negative reduced cost. The sequence of Derivation (5.4) leads to the definition of our first auxiliary problem. Inequation (5.4a) is the constraint for a new variable z_t of the RMP. This constraint has the constant c_t detailed in Inequation (5.4b). Inequation (5.4c) composes the derivation below.

$$\sum_{v\in V} a_{vt}\lambda_v \geq c_t \tag{5.4a}$$

$$\sum_{v\in V} a_{vt}\lambda_v \geq \frac{4\sum_{u,v\in V:u<v} a_{ut}a_{vt}w_{uv} - \sum_{v\in V} a_{vt}d_v}{\sum_{j\in V} a_{jt}} \tag{5.4b}$$

$$\sum_{j\in V}\sum_{v\in V} a_{jt}a_{vt}\lambda_v - 4\sum_{u,v\in V:u<v} a_{ut}a_{vt}w_{uv} + \sum_{v\in V} a_{vt}d_v \geq 0. \tag{5.4c}$$

AP-I can be described by the nonlinear model (5.5) below. It is a minimization problem, where the values of λ_v are given by the dual variables of a solved RMP. AP-I finds the value of the binary variables a_v. If the value of $a_v = 1$, the node v belongs to the new cluster of the new variable z_t for the RMP. If $a_v = 0$, the node v does not belong to the new cluster.

$$\begin{aligned} \min \sum_{j\in V}\sum_{v\in V} a_j a_v \lambda_v - 4\sum_{u,v\in V:u<v} a_u a_v w_{uv} + \sum_{v\in V} a_v d_v \\ \text{subject to: } a_v \in \{0,1\}, \forall v \in V. \end{aligned} \tag{5.5}$$

AP-II is derived from the linearization of AP-I, and was inspired by the model of Xu, Tsoka and Papageorgiou (2007) and the auxiliary problem of Aloise et al. (2010). AP-II is an integer nonlinear model, where $a_u a_v = x_e$ for each $e = \{u, v\} \in E$. If $x_e = 1$, the nodes of edge $e = \{u, v\}$ belong to the cluster; otherwise they do not belong to it. So, the AP-II objective function is

$$\min \sum_{j\in V}\sum_{v\in V} a_j a_v \lambda_v - 4\sum_{e=\{u,v\}\in E} x_e + \sum_{v\in V} a_v d_v, \tag{5.6}$$

adding the following constraints

$$x_e \leq a_u, \qquad \forall\{u,v\} \in E \tag{5.7a}$$

$$x_e \leq a_v, \qquad \forall\{u,v\} \in E \tag{5.7b}$$

$$x_e \in \{0,1\}, \qquad \forall e \in E \tag{5.7c}$$

$$a_v \in \{0,1\}, \qquad \forall v \in V. \tag{5.7d}$$

Algorithm 19 presents the general procedure of column generation used in our experimental analyses. First, the initial variables are generated by 30 runs of the heuristic HLSMD that is described in Section 4.2. The best solution obtained by HLSMD is used to define the initial variables. After that, the RMP is created, and the variables generated by HLSMD are inserted into it. After the creation of the RMP, variables are generated iteratively until the optimal solution is found. At each iteration, some of our column generations use a heuristic solver to find a variable instead of using the exact auxiliary problem. They use the heuristics AP-LS or AP-ILS which are described in Section 5.2. If no improved variable is found, one of the exact auxiliary problem solvers is used. They are described by the nonlinear models (5.5) (AP-I) or (5.7) (AP-II). The iterations are repeated until the exact auxiliary problem solvers cannot find an improved variable for RMP.

5.2 Heuristics for the Auxiliary Problems

The MILP solver used in our experiments works with some types of nonlinear problems (as quadratic), so it can be used to solve the auxiliary problems AP-I and AP-II. This solution is hard in time, so it could require too much time to generate a new variable. Thus, two heuristics for the auxiliary problem were created to generate new variables for the RMP quickly. Here, we call them AP-LS (Auxiliary Problem Local Seach) and AP-ILS (Auxiliary Problem Iterated Local Search).

To help the description of AP-LS and AP-ILS, some delta functions used in these

Algorithm 19: Column Generation Procedure

```
   Input  : $G(V, E)$
1  for $i \leftarrow 1$ to 30 do
2      $\{columns, D\} \leftarrow$ HLSMD($G$) // Algorithm 18
3  Create $RMP$ from the linear model (5.1) with $initialColumns$
4  while solution is not optimal do
5      $\{z, D, \lambda\} \leftarrow$ solve($RMP$)
6      $\{newColumn, value\} \leftarrow$ execute AP-LS or AP-ILS
7      if $value = 0$ then
8          $\{newColumn, value\} \leftarrow$ execute model AP-I or AP-II
9          if $value = 0$ then
               // optimal value for the MP
10             break
11     Add $newColumn$ to $RMP$
12 return $D$
```

heuristics are explained below. These delta functions help the heuristics calculate the objective value of each candidate solution quickly during the search. Equation (5.8) divides the objective function of the model AP-I into three parts.

$$\underbrace{\sum_{j\in V}\sum_{v\in V} a_j a_v \lambda_v}_{first} \underbrace{-4 \sum_{u<v:u,v\in V} a_u a_v w_{uv}}_{second} + \underbrace{\sum_{v\in V} a_v d_v}_{third}. \tag{5.8}$$

The simplification of the first part can be seen in Equations (5.9), (5.10), and (5.11). Suppose that S is a set of nodes that belongs to the cluster solution, so the first part can be rewritten as

$$\sum_{j\in V}\sum_{v\in V} a_j a_v \lambda_v = |S| \sum_{v\in S} \lambda_v. \tag{5.9}$$

Suppose that $k \notin S$ is a candidate node that is trying to enter the cluster S while the heuristic is searching for a neighbor of the current solution. Considering that node k

will belong to S, the first part will be

$$(|S|+1)\sum_{v\in S}\lambda_v+(|S|+1)\lambda_k. \tag{5.10}$$

So, the gain in the first part of the objective function will generate a value equal to

$$\sum_{v\in S}\lambda_v+\big(|S|+1\big)\lambda_k, \tag{5.11}$$

if k enters S.

The second part is simplified in Equation (5.12) below. Without the simplification, all nodes of the graph are verified with each other, using $\Theta\Big(\frac{|V|^2-|V|}{2}\Big)$ operations. The simplified second part is used to count how many edges exist between the node k and nodes of S.

$$-4\sum_{v\in S}a_v w_{kv}. \tag{5.12}$$

The third part is the sum of all degrees of nodes that belong to the current solution. The increase of the objective function of a node k joining the cluster is the degree of k.

The delta functions $\Delta D_{in}(S,k)$ and $\Delta D_{out}(S',k)$ are presented in Equations (5.13) and (5.14), respectively. They help the heuristics calculate quickly the objective value of each candidate solution during the search. The function $\Delta D_{in}(S,k)$ is used in the heuristics to calculate the increase of the objective function when the node k enters the cluster composed of nodes of S. The function $\Delta D_{out}(S',k)$ is used in heuristics to calculate the increase of the objective function when the node k exits the current cluster solution. Supposing that S is the current cluster solution, and k is exiting this cluster, we assume that $S'=S\backslash\{k\}$.

These two delta functions require $\Theta(|V|)$ operations to evaluate a candidate solution to compose a new variable for the column generation.

$$\Delta D_{in}(S,k) = \sum_{v\in S} \lambda_v + \big(|S|+1\big)\lambda_k - 4\sum_{v\in S} a_v w_{kv} + d_k. \tag{5.13}$$

$$\Delta D_{out}(S',k) = -\Delta D_{in}(S',k). \tag{5.14}$$

The solution representation used in AP-LS and AP-ILS is an array with $|V|$ binary values, where each value is related to a node of V. The value 1 means that a node belongs to the cluster solution, and the value 0 means that a node does not belong to the cluster solution.

The heuristic AP-LS is seen in Algorithm 20. This heuristic is a local search method, and the neighborhood strategy changes one binary value at a time. The current solution is called $cluster$ in Algorithm 20. API-LS requires the graph and the values of dual variables as parameters. The starting solution is generated by a random procedure. Iteratively, a different sequence of nodes defines the order that the procedure may change the values of the binary array. The heuristic stops when no improvement is found after $|V|$ iterations.

The AP-ILS is based on the Iterated Local Search metaheuristic over the heuristic AP-LS. It can be seen in Algorithm 21. The difference between AP-LS and AP-ILS is that the latter generates a perturbation in the current cluster solution when a local optimum is found. The perturbation is done by the method *shuffleCurrentSolution*. The parameter $factor \in [0,1]$ is the proportional number of nodes that will randomly join or leave the current cluster solution when a local optimum is found. AP-ILS tries to escape from $|V|$ local optimum solutions.

5.3 Experimental Analysis and Results

In the following, we present the main results obtained by our column generation methods. The experiments were carried out over ten classical instances used as

Algorithm 20: AP-LS Heuristic

```
Input : $G(V,E), \lambda \in \mathbb{R}^{|V|}$
1  $cluster \leftarrow$ randomBinaryVector()
2  $value \leftarrow$ calculateObjectValue($\lambda$)
3  $noImprovement \leftarrow 0$
4  while $noImprovement < |V|$ do
5      $improved \leftarrow false$
6      $randomNodes \leftarrow randomOrder(V)$
7      for $v \in randomNodes$ do
8          $S \leftarrow \{u \in cluster : cluster_u = 1\}$
9          if $cluster_v = 0$ then
10             $gain \leftarrow \Delta D_{in}(S, v)$
11         else
12             $gain \leftarrow \Delta D_{out}(S, v)$
13         if $gain > 0$ then
14             $cluster_v \leftarrow$ not $cluster_v$
15             $value \leftarrow value + gain$
16             $noImprovement \leftarrow 0$
17             $improved \leftarrow true$
18     if not $improved$ then
19         $noImprovement \leftarrow noImprovement + 1$
20 return $\{cluster, value\}$
```

benchmarks. The results are also compared with the state-of-the-art linear models of Costa (2015).

This section is divided into two more subsections that show details of experiments and the results over the time, the number of columns, and components effectiveness of each column generation method.

5.3.1 Experimental Set-up

As our master problem is not exact for Modularity Density Maximization, it is expected that after the execution, the variables lead to non-integer values. After the column generation process, one could execute a branch and price method. However, in

Algorithm 21: AP-ILS Heuristic

```
Input : $G(V,E), \lambda \in \mathbb{R}^{|V|}, factor$
1  $cluster \leftarrow$ randomBinaryVector()
2  $value \leftarrow$ calculateObjectValue($\lambda$)
3  $noImprovement \leftarrow 0$
4  while $noImprovement < |V|$ do
5      $improved \leftarrow false$
6      $randomNodes \leftarrow randomOrder(V)$
7      for $v \in randomNodes$ do
8          $S \leftarrow \{u \in cluster : cluster_u = 1\}$
9          if $cluster_v = 0$ then
10             $gain \leftarrow \Delta D_{in}(S, v)$
11         else
12             $gain \leftarrow \Delta D_{out}(S, v)$
13         if $gain > 0$ then
14             $cluster_v \leftarrow$ not $cluster_v$
15             $value \leftarrow value + gain$
16             $improved \leftarrow true$
17     if not $improved$ then
18         $noImprovement \leftarrow noImprovement + 1$
19         shuffleCurrentSolution($cluster$, $value$, $\lambda$, $factor$)
20 return $\{cluster, value\}$
```

all experiments, this procedure was not necessary because integer solutions are obtained at the end of the column generation. So, no other method was needed after the column generation.

The experiments were run on a PC with an Intel Core i7 64 bits with 3.40GHz with 8192KB of cache memory and 8GB of RAM over Linux Ubuntu 14.04.1 LTS operating system. Each experiment was done by using a single thread. The language used to program the column generations was C++, with "GCC" compiler. The solver IBM ILOG CPLEX 12.6 (IBM, 2015) was used to solve the mathematical programming model RMP and the exact auxiliary problems AP-I and AP-II.

The instances used for the experiments are retrieved from the Pajek datasets (BATAGELJ; MRVAR, 2006). They are all undirected and unweighted graph instances.

For comparison reasons, we used almost all instances of Costa (2015), except for "Dolphins small" and "Journal index" because we cannot find them in the instance repository. These graphs were selected because they were used in Costa (2015) and Costa et al. (2016). The number of nodes, edges, and the optimal value of D^* for each instance can be seen in Table 5.1. Some of our column generations proved the optimal D values for larger instances than the reported in the known literature. These instances are marked in bold.

Table 5.1: Graph instances used in the experiments from the repository Batagelj and Mrvar (2006).

Id	**Dataset name**	**Nodes**	**Edges**	**D^***
1	Strike	24	38	8.8611
2	Galesburg f	31	63	8.2857
3	Galesburg d	31	67	6.9269
4	Karate	34	78	7.8451
5	Korea1	35	69	10.9667
6	Korea2	35	84	11.1430
7	Mexico	35	117	8.7181
8	Sawmill	36	62	8.6234
9	**Dolphins**	**62**	**159**	**12.1252**
10	**Polbooks**	**105**	**441**	**21.9652**

We have developed six variations of column generations; we named them CGI, CGII, CGI+LS, CGII+LS, CGI+ILS, and CGII+ILS. All our column generation versions use the HLSMD to generate initial variables. CGI and CGII are column generation methods that do not use a heuristic to find a new variable. They only use the exact auxiliary problems AP-I and AP-II to find new variables respectively. In Tables 5.2, 5.4, 5.3, 5.5, and 5.6, they are referred as "(only CGI)" and "(only CGII)" respectively. CGI+LS uses the heuristic AP-LS, and when no further improved variables are found, AP-I is used. CGII+LS uses the same heuristic, but when no further improved variables are found, it solves AP-II. CGI+ILS uses the heuristic AP-ILS, and AP-I is applied when the heuristic does not find a new variable. CGII+ILS is almost like

CGI+ILS, except that it uses AP-II instead of AP-I.

In the following tables, the column “Id” shows the identifier of the respective instance of Table 5.1. At the side of each value, there is the standard error. The value “t.l.” means that all results for that method and instance reached the ten-hour limit. “-” means that there is no value known for that instance and method in the literature. A total of 30 trials were executed, and they generated the results of these tables. Our experiments ran on a single processor with a single thread.

The heuristic AP-ILS requires a parameter *factor*. This parameter defines the proportion of nodes that are changed when a local optimum is found. We tested this parameter with the values 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0. When the parameter *factor* was equal to 0.7, our column generation methods found the optimal solution in less time than using all the other factors. Even with other values for the *factor* parameter, our algorithms (CGI+ILS and CGII+ILS) required less time than Costa (2015)’s methods.

Table 5.2: Average time in seconds to find the optimal solution for each column generation presented here.

	Costa MDB			**CGI**			**CGII**	
Id	**1**	**2**	**(only CGI)**	**+LS**	**+ILS**	**(only CGII)**	**+LS**	**+ILS**
1	1.4	2.32	11.53±0.45 (30)	6.67±0.71 (30)	0.86±0.05 (30)	1.66±0.13 (30)	0.52±0.03 (29)	**0.12±0.009 (30)**
2	3.36	2.69	141.92±6.66 (30)	61.95±4.87 (30)	2.13±0.13 (30)	17.97±4.03 (30)	10.36±1.58 (29)	**0.51±0.04 (30)**
3	2.9	5.01	304.35±22.02 (30)	91.62±4.44 (30)	2.74±0.2 (30)	26.39±3.22 (27)	12.007±1.45 (27)	**0.69±0.07 (30)**
4	4.51	4.79	162.95±4.61 (30)	279.08±34.62 (30)	2.88±0.16 (30)	15.69±0.51 (30)	26.1±3.7 (20)	**0.6±0.02 (30)**
5	8.75	19.16	317.36±9.1 (30)	144.62±10.19 (30)	4.31±0.29 (30)	30.53±1.17 (30)	12.93±1.02 (29)	**1.22±0.06 (30)**
6	15.55	36.34	423.99±12.26 (30)	178.82±14.31 (30)	4.01±0.22 (30)	41.57±1.18 (30)	60.87±13.34 (25)	**1.51±0.08 (30)**
7	58.02	13.31	716.17±19.92 (30)	665.54±40.49 (30)	5.35±0.2 (30)	64.23±2.7 (30)	231.74±33.9 (13)	**1.84±0.12 (30)**
8	10.11	10.77	148.3±4.5 (30)	62.81±3.63 (30)	2.37±0.09 (30)	13.49±0.84 (30)	5.67±0.33 (29)	**0.57±0.02 (30)**
9	-	-	24271.32±881.82 (23)	20701.42±1432.8 (13)	1000.95±362.03 (30)	5666.03±0.0 (1)	t.l.	**787.41±279.99 (28)**
10	-	-	t.l.	t.l.	**1317.27±98.22 (20)**	t.l.	t.l.	1327.52±156.74 (13)

Table 5.3: D value obtained by the heuristic for initial variables.

		CGI						**CGII**				
	(only CGI)		**+LS**		**+ILS**		**(only CGII)**		**+LS**		**+ILS**	
Id	**Init.D**	**% D***	**Init.D**	**% D***	**Init.D**	**% D***	**Init.D**	**% D***	**Init.D**	**% D***	**Init.D**	**% D***
1	8.12±0.18	**8.39±2.01%**	8.58±0.13	3.18±1.51%	8.67±0.12	2.17±1.36%	8.65±0.09	**2.41±1.04%**	8.76±0.07	1.09±0.82%	8.69±0.12	1.93±1.35%
2	8.13±0.05	1.88±0.57%	8.16±0.06	1.53±0.76%	7.59±0.16	**8.38±1.99%**	8.16±0.06	1.55±0.75%	8.05±0.08	**2.83±0.92%**	8.12±0.06	1.96±0.74%
3	6.56±0.12	**5.33±1.81%**	6.77±0.06	2.22±0.81%	6.72±0.07	3.02±1.08%	6.76±0.06	2.41±0.95%	6.84±0.02	1.2±0.25%	6.71±0.12	**3.17±1.81%**
4	7.84±0.0	0.0±0.0%	7.82±0.01	**0.27±0.18%**	7.83±0.002	0.13±0.12%	7.84±0.0	0.0±0.0%	7.83±0.01	**0.19±0.19%**	7.84±0.0	0.0±0.0%
5	10.57±0.0	**3.65±0.0%**	10.58±0.02	3.51±0.16%	10.59±0.01	3.44±0.13%	10.59±0.02	**3.42±0.19%**	10.59±0.01	3.42±0.13%	10.6±0.02	3.33±0.17%
6	10.99±0.0	**1.34±0.0%**	11.02±0.01	0.41±0.14%	11.08±0.01	0.53±0.12%	11.11±0.01	0.31±0.1%	11.08±0.01	**0.59±0.13%**	11.11±0.01	0.31±0.1%
7	8.72±0.0	0.0±0.0%	8.69±0.01	**0.37±0.13%**	8.7±0.009	0.18±0.1%	8.62±0.002	0.24±0.11%	8.72±0.0	0.0±0.0%	8.69±0.01	**0.31±0.12%**
8	8.33±0.06	3.37±0.68%	8.34±0.04	3.23±0.47%	8.29±0.06	**3.83±0.73%**	8.38±0.03	2.86±0.33%	8.36±0.04	3.02±0.49%	8.32±0.05	**3.52±0.61%**
9	11.78±0.0	2.81±0.0%	11.63±0.07	4.02±0.58%	11.34±0.1	**6.51±0.84%**	11.66±0.0	3.79±0.0%	t.l.	t.l.	11.42±0.11	**5.8±0.89%**
10	t.l.	t.l.	t.l.	t.l.	21.53±0.05	**1.99±0.23%**	t.l.	t.l.	t.l.	t.l.	21.42±0.06	**2.12±0.26%**

Table 5.4: The number of columns generated during the search for optimal solutions of the tests that did not reach the ten-hour limit.

	CGI			CGII		
Id	**(only CGI)**	**+LS**	**+ILS**	**(only CGII)**	**+LS**	**+ILS**
1	78.77±4.13	**68.63±8.28**	94.13±7.17	70.37±4.12	**58.34±2.91**	99.17±5.76
2	193.2±7.02	**178.47±10.82**	307.83±33.24	**192.13±13.06**	209.79±20.86	218.8±16.71
3	246.67±7.76	**173.87±9.03**	274.07±13.03	243.15±13.48	**182.89±8.18**	260.33±17.12
4	**182.0±0.0**	368.5±46.03	235.47±6.35	**193.0±0.0**	290.85±19.67	232.73±5.16
5	280.0±0.0	**267.2±15.6**	330.53±8.57	351.9±7.99	**279.93±11.08**	329.37±6.98
6	349.0±0.0	**346.4±24.4**	394.13±12.95	391.43±5.34	**309.96±27.64**	378.5±9.72
7	**368.0±0.0**	576.6±28.2	422.2±9.73	**362.87±9.25**	505.92±27.37	426.67±13.67
8	183.53±2.47	**141.8±6.09**	233.17±8.48	204.17±8.02	**146.93±5.07**	213.5±6.45
9	**1979.0±0.0**	19278.77±1573.41	6336.97±1084.06	**1292.0±0.0**	t.l.	5258.75±1129.4
10	t.l.	t.l.	**3633.6±166.03**	t.l.	t.l.	**3736.85±153.09**

Table 5.5: The percentage of columns generated during the search for optimal solutions of the tests that did not reach the ten-hour limit.

	CGI							
	(only CGI)		+LS			+ILS		
Id	**Init.**	**Exact**	**Init.**	**Exact**	**Heur.**	**Init.**	**Exact**	**Heur.**
1	6.63±0.62%	**93.37±0.62%**	6.66±0.36%	23.22±1.26%	**70.11±1.49%**	4.66±0.24%	0.14±0.07%	**95.19±0.25%**
2	1.69±0.02%	**98.31±0.02%**	1.91±0.08%	27.11±0.97%	**70.98±0.98%**	1.52±0.09%	0.02±0.02%	**98.46±0.09%**
3	1.31±0.006%	**98.69±0.006%**	1.93±0.08%	33.61±0.92%	**64.46±0.94%**	1.28±0.04%	0.08±0.03%	**98.63±0.05%**
4	1.63±0.0%	**98.37±0.0%**	1.05±0.08%	33.93±1.16%	**65.02±1.15%**	1.3±0.03%	0.03±0.02%	**98.67±0.04%**
5	3.12±0.0%	**96.87±0.0%**	3.85±0.19%	24.13±0.59%	**72.01±0.64%**	2.93±0.07%	0.12±0.03%	**96.95±0.09%**
6	2.79±0.0%	**97.21±0.0%**	2.91±0.17%	23.48±0.77%	**73.61±0.86%**	2.34±0.07%	0.06±0.02%	**97.6±0.08%**
7	0.81±0.0%	**99.19±0.0%**	0.53±0.04%	31.9±0.81%	**67.56±0.83%**	0.62±0.02%	0.01±0.009%	**99.29±0.02%**
8	2.95±0.13%	**97.04±0.13%**	3.74±0.14%	28.28±0.86%	**67.98±0.89%**	2.33±0.14%	0.01±0.01%	**97.66±0.14%**
9	0.25±0.0%	**99.75±0.0%**	0.03±0.003%	5.48±0.69%	**94.48±0.69%**	0.25±0.04%	0.06±0.02%	**99.62±0.05%**
10	t.l.	t.l.	t.l.	t.l.	t.l.	0.2±0.01%	0.18±0.02%	**99.62±0.03%**

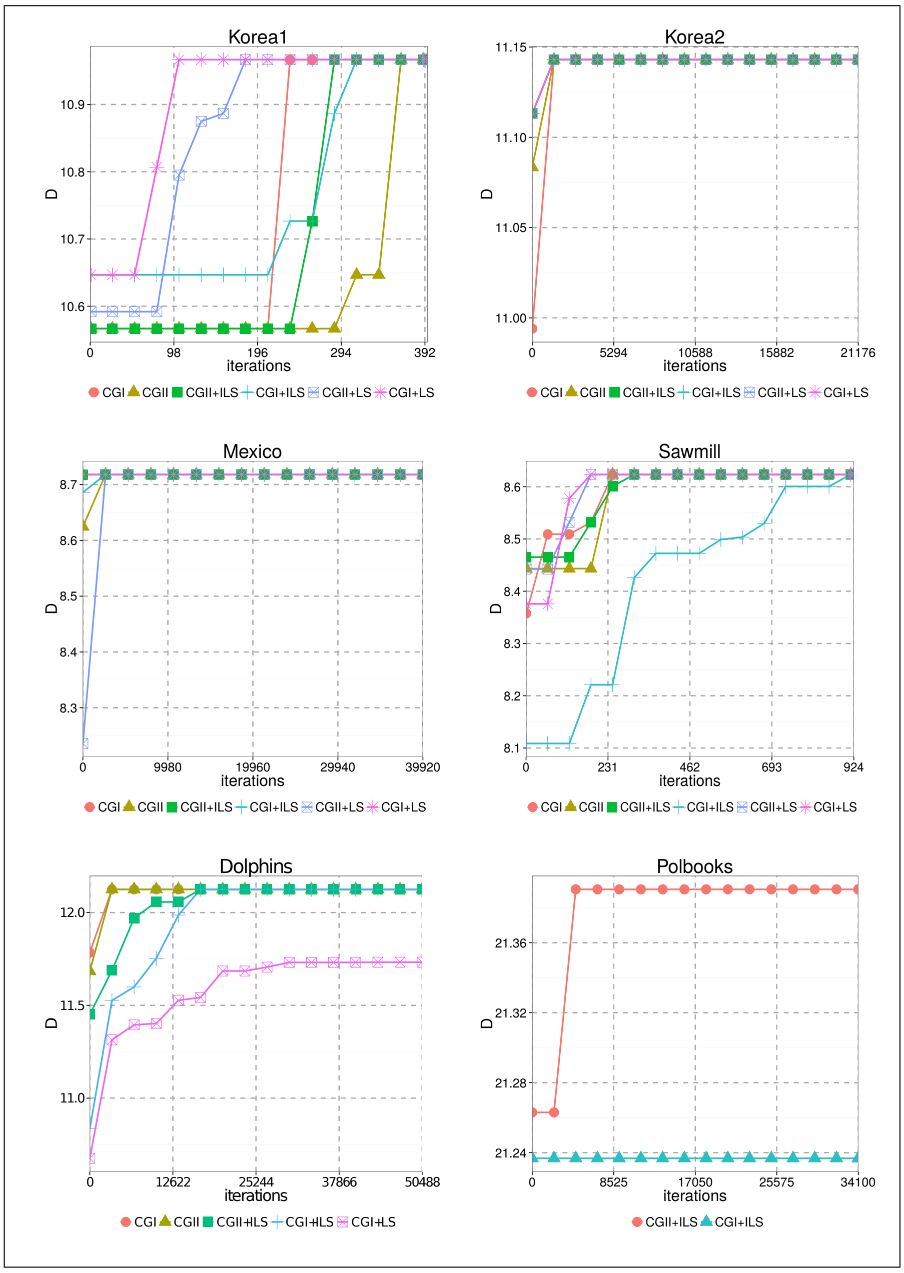
	CGII							
	(only CGII)		+LS			+ILS		
Id	**Init.**	**Exact**	**Init.**	**Exact**	**Heur.**	**Init.**	**Exact**	**Heur.**
1	5.86±0.26%	**94.13±0.26%**	6.97±0.32%	24.13±1.18%	**68.82±1.37%**	4.27±0.23%	0.13±0.06%	**95.6±0.23%**
2	1.71±0.05%	**98.28±0.05%**	1.74±0.07%	25.21±0.73%	**73.04±0.73%**	1.58±0.08%	0.03±0.02%	**98.38±0.08%**
3	1.42±0.05%	**98.58±0.05%**	1.83±0.02%	33.62±0.87%	**64.55±0.84%**	1.32±0.04%	0.09±0.04%	**98.59±0.06%**
4	1.54±0.0%	**98.46±0.0%**	1.15±0.09%	34.34±1.15%	**64.51±1.15%**	1.29±0.03%	0.04±0.02%	**98.66±0.04%**
5	2.62±0.08%	**97.3±0.08%**	3.54±0.14%	24.69±0.77%	**71.77±0.82%**	2.87±0.07%	0.21±0.04%	**96.92±0.08%**
6	2.28±0.03%	**97.72±0.03%**	3.2±0.19%	22.41±0.72%	**74.38±0.74%**	2.4±0.06%	0.08±0.02%	**97.51±0.08%**
7	0.81±0.03%	**99.19±0.03%**	0.62±0.04%	34.98±0.97%	**64.39±0.96%**	0.69±0.03%	0.03±0.01%	**99.28±0.03%**
8	2.63±0.09%	**97.37±0.09%**	3.66±0.16%	28.1±0.83%	**68.24±0.88%**	2.51±0.02%	0.08±0.03%	**97.41±0.11%**
9	0.38±0.0%	**99.61±0.0%**	t.l.	t.l.	t.l.	0.28±0.04%	0.08±0.02%	**99.64±0.05%**
10	t.l.	t.l.	t.l.	t.l.	t.l.	0.19±0.01%	0.2±0.03%	**99.6±0.04%**

Table 5.6: The proportional time required during the search for optimal solutions of the tests that did not reach the ten-hour limit. Each column generation version is divided into its components.

	CGI										
		(only CGI)			**+LS**				**+ILS**		
Id	**Init.**	**RMP**	**Exact**	**Init.**	**RMP**	**Exact**	**Heur.**	**Init.**	**RMP**	**Exact**	**Heur.**
1	0.03±0.0006%	0.02±0.007%	**99.88±0.007%**	0.06±0.005%	0.12±0.02%	**99.72±0.02%**	0.02±0.001%	0.4±0.02%	1.24±0.22%	**93.82±0.45%**	4.53±0.32%
2	0.003±0.0001%	0.03±0.003%	**99.96±0.003%**	0.008±0.0006%	0.05±0.008%	**99.94±0.008%**	0.008±0.0003%	0.25±0.01%	3.33±0.52%	**82.03±1.62%**	14.39±1.2%
3	0.002±0.0%	0.03±0.0009%	**99.97±0.0009%**	0.005±0.0003%	0.03±0.002%	**99.96±0.002%**	0.005±0.0001%	0.19±0.009%	1.99±0.16%	**87.02±0.73%**	10.8±0.58%
4	0.003±0.0%	0.03±0.0008%	**99.96±0.0008%**	0.002±0.0003%	0.03±0.003%	**99.97±0.003%**	0.005±0.0003%	0.18±0.006%	1.33±0.06%	**88.25±0.44%**	10.24±0.39%
5	0.002±0.0%	0.04±0.0007%	**99.96±0.0007%**	0.005±0.0003%	0.03±0.001%	**99.96±0.002%**	0.007±0.0003%	0.15±0.009%	1.75±0.11%	**84.4±0.94%**	13.62±0.83%
6	0.001±0.0%	0.04±0.0007%	**99.96±0.0007%**	0.004±0.0003%	0.03±0.002%	**99.95±0.002%**	0.007±0.0003%	0.17±0.007%	2.42±0.18%	**79.38±1.08%**	17.95±0.82%
7	0.0009±0.0%	0.02±0.0004%	**99.98±0.0004%**	0.001±0.0%	0.02±0.0002%	**99.97±0.001%**	0.003±0.0001%	0.13±0.005%	2.34±0.13%	**84.5±0.59%**	13.02±0.47%
8	0.003±0.0%	0.03±0.001%	**99.96±0.001%**	0.002±0.0005%	0.03±0.002%	**99.95±0.002%**	0.008±0.0002%	0.24±0.009%	1.65±0.11%	**85.18±0.67%**	12.92±0.57%
9	<0.0001%	0.02±0.0004%	**99.98±0.0004%**	<0.0001%	0.79±0.13%	**99.12±0.13%**	0.01±0.001%	0.02±0.003%	32.45±5.22%	**42.76±5.41%**	24.77±1.85%
10	t.l.	t.l.	t.l.	t.l.	t.l.	t.l.	t.l.	0.003±0.0002%	5.35±0.84%	**81.24±1.66%**	13.41±0.93%

	CGII										
		(only CGII)			**+LS**				**+ILS**		
Id	**Init.**	**RMP**	**Exact**	**Init.**	**RMP**	**Exact**	**Heur.**	**Init.**	**RMP**	**Exact**	**Heur.**
1	0.22±0.009%	0.58±0.07%	**99.2±0.07%**	0.78±0.04%	1.17±0.07%	**97.79±0.11%**	0.26±0.01%	3.85±0.12%	11.16±0.96%	41.41±1.26%	**43.57±1.29%**
2	0.03±0.002%	0.2±0.006%	**99.76±0.006%**	0.08±0.008%	0.38±0.02%	**99.45±0.03%**	0.07±0.005%	1.25±0.05%	8.08±0.62%	40.46±1.56%	**50.21±1.09%**
3	0.02±0.001%	0.18±0.007%	**99.79±0.008%**	0.06±0.006%	0.22±0.02%	**99.58±0.02%**	0.06±0.004%	0.98±0.05%	8.94±0.62%	39.29±2.5%	**50.79±2.02%**
4	0.03±0.0007%	0.23±0.004%	**99.74±0.003%**	0.04±0.006%	0.25±0.02%	**99.65±0.03%**	0.06±0.006%	1.08±0.03%	7.82±0.17%	31.12±1.08%	**59.9±0.98%**
5	0.02±0.0009%	0.29±0.006%	**99.69±0.006%**	0.06±0.005%	0.4±0.02%	**99.44±0.03%**	0.09±0.006%	0.67±0.03%	7.42±0.24%	33.23±2.06%	**58.59±1.84%**
6	0.01±0.0005%	0.24±0.004%	**99.74±0.004%**	0.04±0.008%	0.19±0.02%	**99.72±0.04%**	0.04±0.007%	0.54±0.02%	7.41±0.28%	36.35±1.99%	**55.62±1.75%**
7	0.002±0.0004%	0.15±0.003%	**99.84±0.003%**	0.005±0.001%	0.08±0.009%	**99.91±0.01%**	0.01±0.002%	0.42±0.02%	8.78±0.41%	41.71±1.99%	**49.002±1.67%**
8	0.04±0.002%	0.26±0.006%	**99.69±0.006%**	0.12±0.008%	0.38±0.02%	**99.38±0.03%**	0.11±0.006%	1.22±0.04%	6.85±0.21%	34.38±1.35%	**57.54±1.2%**
9	0.0002±0.0%	0.03±0.0%	**99.96±0.0%**	t.l.	t.l.	t.l.	t.l.	0.02±0.004%	17.92±4.32%	**62.98±4.05%**	19.07±1.52%
10	t.l.	t.l.	t.l.	t.l.	t.l.	t.l.	t.l.	0.003±0.0004%	4.77±0.97%	**80.87±2.86%**	14.36±2.1%

Figure 5.1: Average D value of five tests with the largest instances for each iteration.



5.3.2 Analysis of results

This section analyzes the tables and a figure which are used to report experimental results of our column generations. Table 5.2 shows the average elapsed time in seconds for each column generation that found the optimal solution. At the side of each value, there is its standard error. The number of tests that reached the optimal values before ten-hour limit is presented in parentheses. For comparison, the column "Costa MDB" shows the time reported in Costa (2015) for the best linear programming models for Modularity Density Maximization problem in there. They are MDB1 and MDB2. The experiments of Costa (2015) were performed on a PC with 4 Intel Xeon E5-4620 CPU at 2.20 GHz (8 cores each, Hyper-Threading and Turbo Boost disabled), 128 GB RAM.

In Table 5.2, CGI-ILS and CGII+ILS obtained the best results. CGII+ILS was the fastest for most of the tested instances that did not reach the time limit, but CGI-ILS solved more trials. Only for 10 ("Polbooks"), it did not find the optimal solution for some trials. The results suggest that the component AP-ILS accelerated the time of column generation methods CGI and CGII. All column generations that used the AP-ILS heuristic for the auxiliary problem improved on state-of-the-art time reported in Costa (2015) for the exact solving of Modularity Density Maximization.

Table 5.4 shows the average number of columns and the standard error of its trials. This information is important to understand which is the most effective variable generator for each auxiliary model CGI and CGII. Although CGI+ILS and CGII+ILS are the fastest methods, they generated the largest number of columns, so some of these columns are unnecessary to find the optimal solution. For the instance 9 ("Dolphins"), CGI+LS generated more than nine times the number of columns generated by CGI. CGI+LS and CGII+LS found the optimal solution using less number of columns than CGI+ILS and CGII+ILS in average.

To analyze the contribution of each one of the components of our column generation versions, Tables 5.3, 5.5, and 5.6 show the quality of the initial solutions and the effort for reaching the optimal value, the percentage of columns generated, and the percentage of time required by each component.

Table 5.3 shows the initial D value obtained by the heuristic HLSMD (Section 4.2) and the distance to the optimal value ($\%D^*$). The largest percentage is marked in bold for each instance. This distance is calculated using

$$\frac{(D^* - D) \cdot 100}{D^*}, \tag{5.15}$$

where D is the value obtained by HLSMD, and D^* is the optimal value. Even for the largest tested instances, HLSMD found values close to the optimal value which helped the column generation process. For some instances, the heuristic for initial variables found the optimal values for all tests. For instances 1 ("Strike"), 2 ("Galesburg f"), 3 ("Galesburg d"), and 9 ("Dolphins"), HLSMD found the furthest solutions from the optimal values, even when it is compared with the results of 10 ("Polbooks") that is the largest tested instance.

Table 5.5 shows the percentage of columns obtained by each component of the tested column generations. AP-ILS was the component that generated more than 95% of columns. The column generations with AP-ILS component used the auxiliary problems to generate less than 0.25% of the columns, even for the largest instances. CGI-ILS and CGII-ILS had a similar behavior when considering this proportion for each instance. When using AP-LS in CGI-LS and CGII-LS, the exact auxiliary problem is often solved. Comparing the results of this table with Table 5.4, the component AP-LS generated less columns than AP-ILS, and the column generations that use AP-LS depended on more of the exact problem (AP-I or AP-II) to find variables.

Table 5.6 shows the percentage of the time required by each component. Each column generation version is divided into its components. The time required by the heuristic for initial variables is labeled as "Init.", the exact auxiliary problem is labeled as "Exact", and the heuristic for the auxiliary problem is identified as "Heur.". The largest values are featured for each column generation version. The CGI column generations spent more time solving the exact auxiliary models. Except for CGI+ILS and CGII+ILS, the algorithms used more than 98% of the time to solve the exact auxiliary problem. With component AP-ILS, the algorithms demanded less time as can be seen in Table 5.2. When comparing CGI+ILS and CGII+ILS, the RMP demanded more time for CGII+ILS, except for the two largest instances. For the methods CGI+ILS and CGII+ILS and the instance 9 (“Dolphins”), the RMP demanded more time than for other instances.

Comparing the results of Tables 5.5 and 5.6, CGI+LS and CGII+LS demanded more than 99% of the time in the execution of the exact algorithm, and their heuristic generated more than 64% of the columns, so the experiments suggested that the component AP-LS is less effective than AP-ILS.

Figure 5.1 shows the average D value of five tests obtained by each iteration for the largest tested instances. Some tests did not reach optimal solutions because of our ten-hour limit, so some lines did not achieve the optimal value. The figure also confirmed the analysis of Table 5.4, where CGI and CGII found the variables of optimal solutions in the first iterations because they use the exact auxiliary problems. AP-ILS found the best variables. For the auxiliary problem, the heuristic AP-ILS surpassed the AP-LS results. It suggests that the AP-ILS helps to escape from local optimum solutions.

We tested our methods with the “Adjnoun” (112 nodes and 425) and “Football” (115 nodes and 613) instances from Batagelj and Mrvar (2006). They did not reach the optimal value within a time limit of 100 hours.

5.4 Chapter Summary

Four of our column generation methods used heuristics for the auxiliary problem and two of them obtained better, improved times over the results from Costa (2015). Using a simple experimental set-up, our methods also proved that the results for the instances 9 (“Dolphins”) and 10 (“Polbooks”) found in Costa et al. (2016) are optimal for the first time. These proofs are larger than the ones solved by the best models reported in Costa

(2015). The column generations CGI+ILS and CGII+ILS obtained the best time results when finding the optimal solution. For the tested instances, CGI+ILS did not reach the ten-hour limit, except for instance 10 (“Polbooks”). The results obtained with both CGI+ILS and CGII+ILS suggest that they are the state-of-the-art for exact solving of the Modularity Density Maximization problem when comparing results of Costa (2015) and our column generation methods.

Tables 5.5 and 5.6 show the importance of heuristic to find variables where AP-ILS reduced the execution time of the exact auxiliary problems. Table 5.4 shows that heuristics for auxiliary problems could be improved because the methods CGI and CGII required less columns than AP-LS and AP-ILS for instance 9 (“Dolphins”). As CGI and CGII use exact solvers for the auxiliary problem, this shows that a more effective heuristic could converge to the optimal solution more quickly than the fastest method used so far.

6 GROUND TRUTH ANALYSES

This chapter presents results about ground truth analyses performed with our heuristic and algorithms described in Chapters 4 and 5. These analyses aim to compare the results of the methods and the expected clustering, so the quality of different methods can be tested beyond the objective function. The chapter is divided into three sections. The first section shows the results of our eight heuristics over different λ parameters of the Modularity Density Maximization objective function when solving artificial graphs, using the LFR benchmark of Lancichinetti, Fortunato and Radicchi (2008). The second one also uses similar artificial graphs, but it is compared exact solutions from Modularity Maximization and Modularity Density Maximization solvers. We then summarize the results.

For the ground truth analyses, we used the “Matthews Correlation Coefficient” that takes account of true positives (N_{11}), true negatives (N_{00}), false negatives (N_{10}), and false positives (N_{10}) (MATTHEWS, 1975). The ϕ function is described in Equation (6.1). The resulting values of the ϕ function are bound between $[-1, 1]$. The larger the coefficient ϕ is, the stronger the correlation between the partition obtained by the heuristic and the correct partition is.

$$\phi = \frac{N_{00}N_{11} - N_{10}N_{01}}{\sqrt{(N_{11}+N_{01})(N_{11}+N_{10})(N_{00}+N_{01})(N_{00}+N_{10})}}. \tag{6.1}$$

Each N has the number of pairs of nodes, comparing the correct partitions and the obtained by the tested methods. They are described as follow:

- N_{00}: the number of pairs of nodes which are not in the same cluster in the correct partition generated by LFR, and which are not in the same cluster in the obtained partition by our tested method;
- N_{01}: the number of pairs of nodes which are not in the same cluster in the correct partition generated by LFR, but which are in the same cluster in the obtained partition by our tested method;

- N_{10}: the number of pairs of nodes which are in the same cluster in the correct partition generated by LFR, but which are not in the same cluster in the obtained partition by our tested method;
- N_{11}: the number of pairs of nodes which are in the same cluster in the correct partition generated by LFR, and which are in the same cluster in the obtained partition by our tested method.

6.1 Heuristic Analysis

In this section, results of ground truth analyses of our tested heuristics on artificial random graphs are reported. The tests were performed by using our eight heuristics, and the heuristics CNM and Louvain for the Modularity Maximization problem. The used random graph generator was LFR from Lancichinetti, Fortunato and Radicchi (2008) because it creates benchmarks for cluster detection problems, where the clusters are known, allowing that we can measure the quality of the tested heuristics to the expected clustering. All our eight heuristics were tested by using the λ quantifier of the Modularity Density Maximization objective function (Equation (2.8)).

All generated instances are undirected and unweighted. The graph instances are described in Table 6.1. The given name, the mixing parameter (μ), the number of nodes and edges are shown. The name is composed of the number of nodes followed by the mixing parameter used during each graph generation. These graphs have $100,000$ nodes and were generated in LFR with the parameters based on Nascimento and Pitsoulis (2013):

- average degree (k) equal to 15;
- maximum degree ($maxk$) equal to 50;
- mixing parameter (μ), where $\mu \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$;
- minus exponent for the degree sequence ($t1$) equal to 2;
- minus exponent for the cluster size distribution ($t2$) equal to 1.

The mixing parameter (μ) of LFR generator is the degree of overlapping among clusters. We used eight parameters to generate problems with increasing difficulty for detecting clusters. The higher the parameter is; the weaker the modular property of the clusters get.

Table 6.1: Details of the random graphs generated by the LFR benchmark.

Dataset name	μ	**Nodes**	**Edges**
100000-0.1	0.1	100000	765212
100000-0.2	0.2	100000	764900
100000-0.3	0.3	100000	766099
100000-0.4	0.4	100000	765240
100000-0.5	0.5	100000	767148
100000-0.6	0.6	100000	765365
100000-0.7	0.7	100000	764913
100000-0.8	0.8	100000	769003

Tables 6.2 shows the “Matthews Correlation Coefficient” (ϕ) for all tested heuristics and quantitative factors λ. CNM and Louvain do not have λ factors because they are Modularity Maximization heuristics. Each instance and heuristic was tested for five trials. The best ϕ values are featured in bold.

In our experiments, CNM and Louvain were worse than our eight heuristics. CM and LNM obtained $\phi > 0.8$ only for instance $\mu = 1$. CM+LNM presented high correlation $\phi > 0.9$ when using the $\lambda = 0.8$ for instances with $\mu \leq 0.6$. MCN obtained $\phi > 0.8$ only for instances with $\mu \leq 0.3$. For the same instances, CM+MCN was better because it reached $\phi > 0.9$; for instances with $\mu > 0.3$, it missed cluster structures in the graphs. MD obtained $\phi > 0.8$ only for instances with $\mu \in \{0.1, 0.2\}$; MDM obtained similar results only for instance with $\mu = 0.1$. HLSMD reached $\phi > 0.9$ for instances with $\mu \leq 0.3$. To help the understanding of these results, the ϕ values are shown in Figure 6.1. They are concentrated for each instance, so it is possible to follow the increasing difficulty of the μ parameter of the artificial graphs.

In Figure 6.2, plots are provided to understand the behavior of different parameters of instances (μ), the objective function (λ), and the correction (ϕ) for each graph.

Figure 6.1: Graphical results of our ground truth tests with CNM, Louvain, and our eight heuristics for each artificial instance.

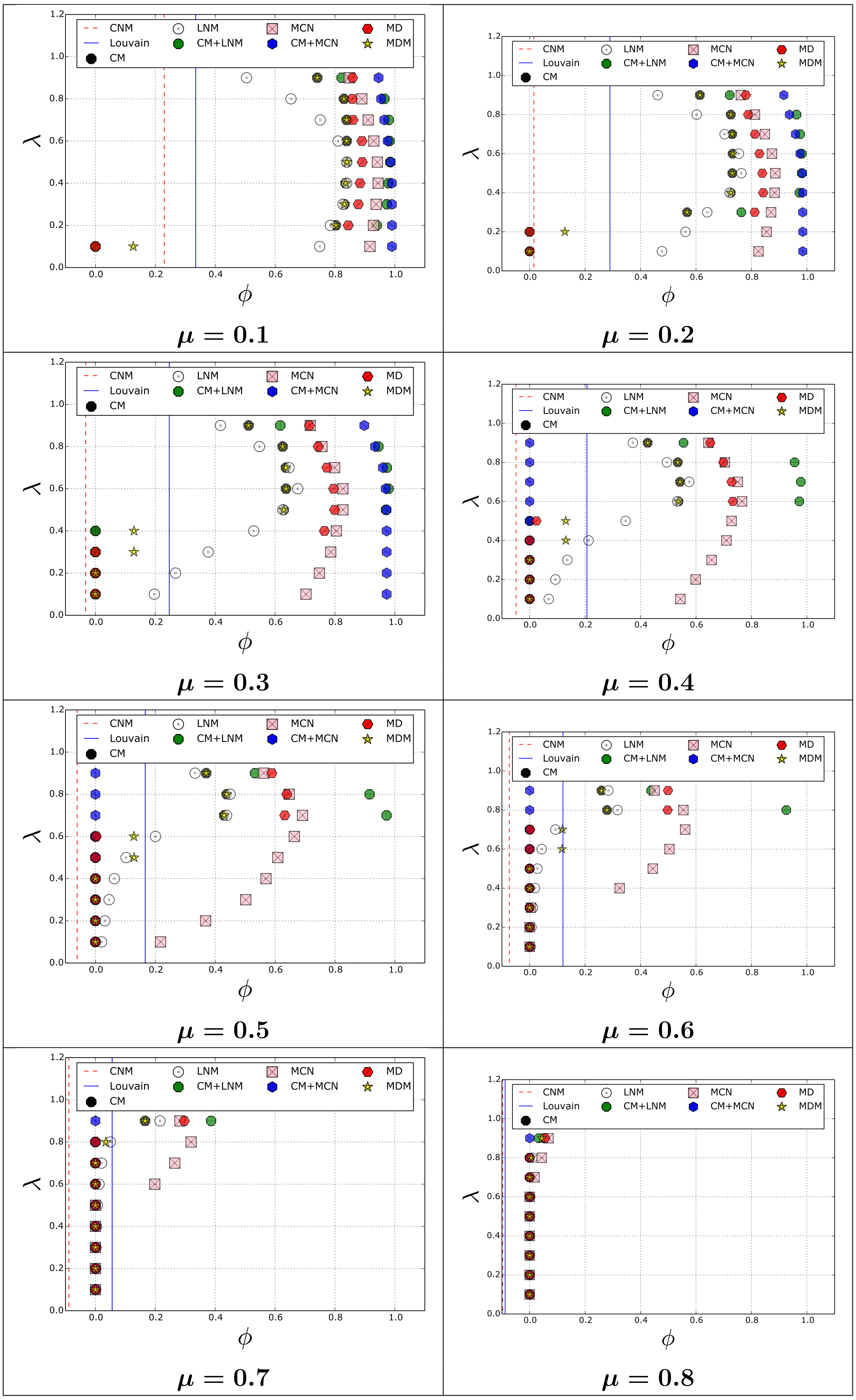


Figure 6.2: Graphical results of our ground truth tests for each one of our eight heuristics.

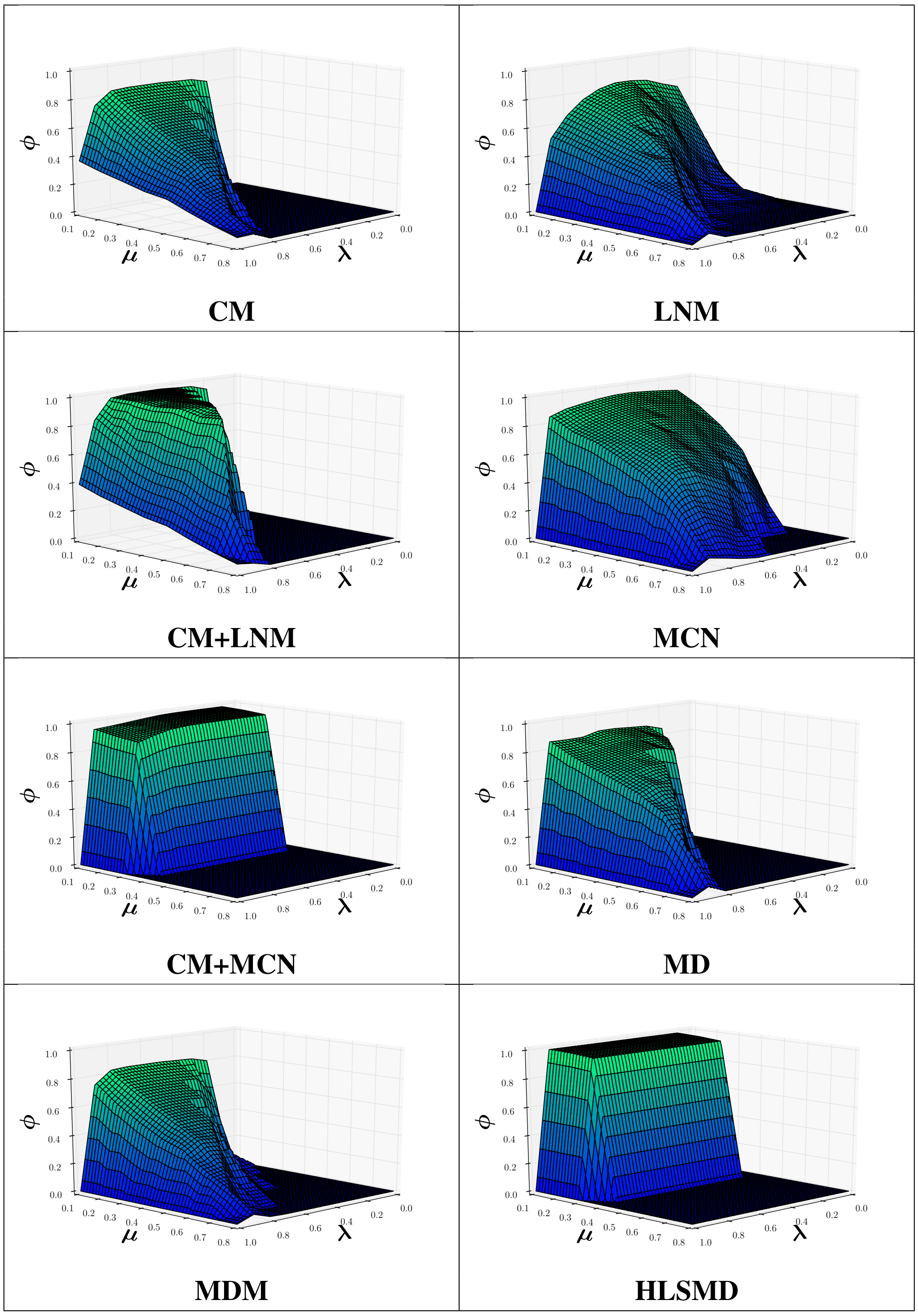


Table 6.2: Comparisons among ϕ values obtained by the tested heuristics in the ground truth analyses.

μ	**CNM**	**Louvain**	λ	**CM**	**LNM**	**CM+LNM**	**MCN**	**CM+MCN**	**MD**	**MDM**	**HLSMD**
0.1	.274	.362	0.1	.0	.749	.0	.916	**.99**	.0	.126	**.99**
			0.2	.802	.784	.939	.928	**.99**	.844	.802	**.99**
			0.3	.831	.826	.973	.937	**.99**	.877	.831	**.99**
			0.4	.835	.838	.978	.944	.989	.883	.835	**.99**
			0.5	.839	.839	.984	.941	.986	.89	.839	**.99**
			0.6	.839	.81	.982	.929	.976	.89	.839	**.99**
			0.7	.838	.75	.979	.91	.964	.86	.838	**.99**
			0.8	.83	.653	.965	.889	.954	.857	.83	**.99**
			0.9	.74	.504	.822	.847	.945	.858	.74	**.99**
0.2	.096	.324	0.1	.0	.477	.0	.825	**.985**	.0	.0	**.985**
			0.2	.0	.562	.0	.854	**.985**	.0	.127	**.985**
			0.3	.567	.64	.763	.87	.984	.811	.567	**.985**
			0.4	.725	.721	.973	.883	.984	.842	.725	**.985**
			0.5	.73	.763	.981	.886	.983	.839	.73	.984
			0.6	.732	.754	.981	.873	.975	.828	.732	.984
			0.7	.73	.703	.975	.848	.958	.812	.73	.984
			0.8	.725	.602	.962	.812	.937	.787	.725	**.985**
			0.9	.613	.461	.721	.761	.916	.779	.613	**.985**
0.3	.056	.288	0.1	.0	.196	.0	.703	.972	.0	.0	.973
			0.2	.0	.267	.0	.748	.972	.0	.0	.973
			0.3	.0	.376	.0	.785	.972	.0	.128	.972
			0.4	.0	.528	.0	.804	.972	.764	.128	.972
			0.5	.628	.625	.97	.826	.972	.798	.628	.972
			0.6	.636	.675	**.978**	.826	.969	.795	.636	.972
			0.7	.636	.646	.973	.798	.96	.772	.636	.972
			0.8	.625	.547	.944	.756	.933	.744	.625	.972
			0.9	.511	.417	.617	.717	.897	.714	.511	.972
0.4	.042	.255	0.1	.0	.068	.0	.543	.0	.0	.0	.0
			0.2	.0	.093	.0	.598	.0	.0	.0	.0
			0.3	.0	.135	.0	.656	.0	.0	.0	.0
			0.4	.0	.212	.0	.709	.0	.0	.13	.0
			0.5	.0	.346	.0	.728	.0	.024	.13	.0
			0.6	.536	.532	.972	.766	.0	.732	.536	.0
			0.7	.542	.575	**.978**	.75	.0	.728	.542	.0
			0.8	.534	.495	.955	.704	.0	.698	.534	.0
			0.9	.425	.372	.555	.644	.0	.651	.425	.0
0.5	.032	.221	0.1	.0	.02	.0	.217	.0	.0	.0	.0
			0.2	.0	.031	.0	.368	.0	.0	.0	.0
			0.3	.0	.045	.0	.502	.0	.0	.0	.0
			0.4	.0	.063	.0	.569	.0	.0	.0	.0
			0.5	.0	.102	.0	.608	.0	.0	.129	.0
			0.6	.0	.2	.0	.664	.0	.002	.128	.0
			0.7	.429	.438	**.972**	.691	.0	.632	.429	.0
			0.8	.438	.449	.915	.647	.0	.638	.438	.0
			0.9	.369	.332	.532	.561	.0	.588	.369	.0
0.6	.022	.183	0.1	.0	.003	.0	.0	.0	.0	.0	.0
			0.2	.0	.006	.0	.0	.0	.0	.0	.0
			0.3	.0	.011	.0	.004	.0	.0	.0	.0
			0.4	.0	.018	.0	.324	.0	.0	.0	.0
			0.5	.0	.027	.0	.444	.0	.0	.0	.0
			0.6	.0	.043	.0	.504	.0	.0	.116	.0
			0.7	.0	.093	.0	.56	.0	.001	.118	.0
			0.8	.279	.317	**.925**	.554	.0	.497	.279	.0
			0.9	.259	.283	.438	.45	.0	.499	.259	.0
0.7	.009	.129	0.1	.0	.001	.0	.0	.0	.0	.0	.0
			0.2	.0	.001	.0	.0	.0	.0	.0	.0
			0.3	.0	.001	.0	.0	.0	.0	.0	.0
			0.4	.0	.003	.0	.0	.0	.0	.0	.0
			0.5	.0	.006	.0	.0	.0	.0	.0	.0
			0.6	.0	.012	.0	.198	.0	.0	.0	.0
			0.7	.0	.02	.0	.264	.0	.0	.0	.0
			0.8	.0	.05	.0	.319	.0	.0002	.034	.0
			0.9	.165	.215	**.386**	.282	.0	.296	.165	.0
0.8	.002	.009	0.1	.0	.0002	.0	.0	.0	.0	.0	.0
			0.2	.0	.0003	.0	.0	.0	.0	.0	.0
			0.3	.0	.0004	.0	.0	.0	.0	.0	.0
			0.4	.0	.0006	.0	.0	.0	.0	.0	.0
			0.5	.0	.001	.0	.00001	.0	.0	.0	.0
			0.6	.0	.002	.0	.0	.0	.0	.0	.0
			0.7	.0	.004	.0	.017	.0	.0	.0	.0
			0.8	.0	.009	.0	.043	.0	.00005	.005	.0
			0.9	.052	.056	.032	**.067**	.0	.055	.043	.0

6.2 Algorithm Analysis

This section describes the ground truth analyses developed to compare exact solutions of Modularity Maximization and Modularity Density Maximization. The algorithm used for the latter problem was CGI+ILS (see Chapter 5). It was chosen because it is the fastest column generation that found optimal solutions most of the time. For the Modularity Maximization problem, we used the algorithm of Brandes et al. (2008).

We generated artificial graphs with 30, 40, 50, and 60 nodes by using the LFR (LANCICHINETTI; FORTUNATO; RADICCHI, 2008). Details about these graphs can be seen in Table 6.3. The parameters used are:

- average degree (k) equal to 3;
- maximum degree ($maxk$) equal to 8;
- mixing parameter (μ), where $\mu \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$;
- minimum number of nodes in each cluster ($minc$) equal to 5;
- maximum number of nodes in each cluster ($maxc$) equal to 15.

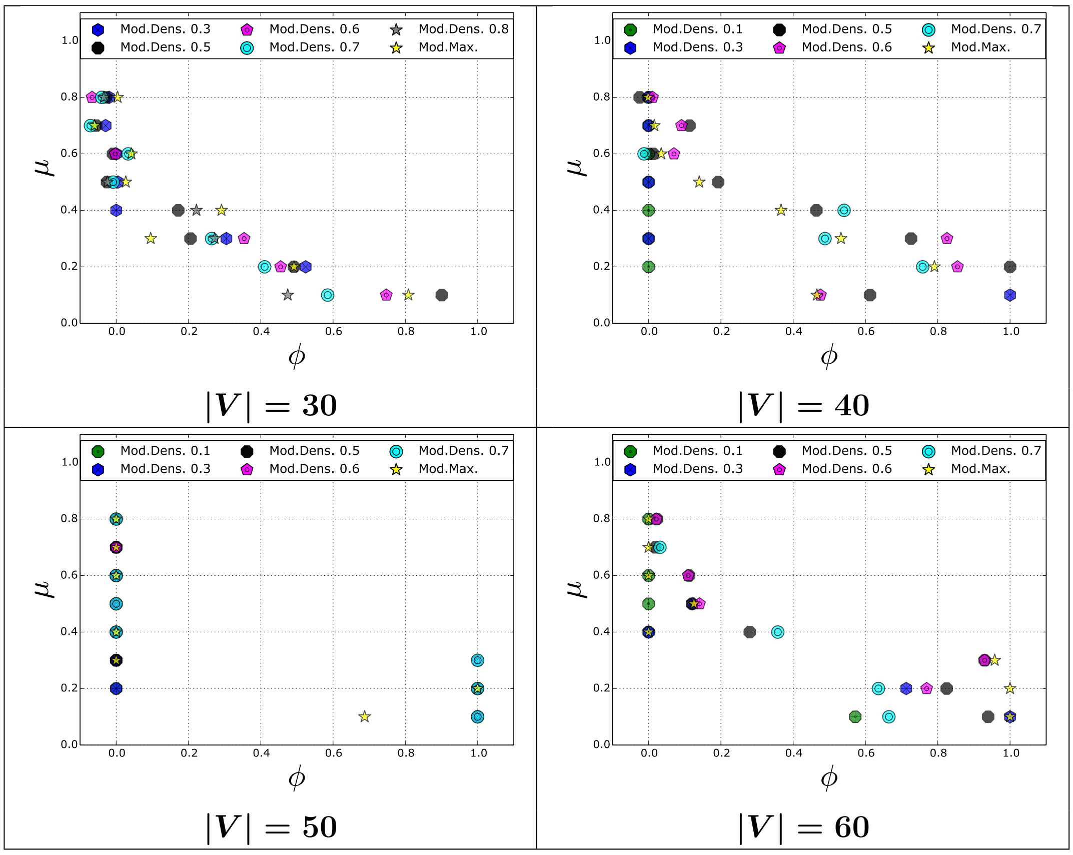
Tables 6.4, 6.5, 6.6 and 6.7 show the ϕ values obtained for Modularity Maximization and Modularity Density Maximization algorithm and heuristics (our eight heuristics of Chapter 4, CNM, and Louvain) for graphs with 30, 40, 50, and 60 nodes respectively. The value "-" means that the algorithm reached the ten-hour limit of execution. In our experiments, for instances with $\mu \leq 0.3$ obtained the best ϕ values. For instances with $\mu > 0.3$ the values are distant from 1, so the correlation is weak. Considering values with $\phi \geq 0.8$, Modularity Density Maximization obtained the best results. Modularity Maximization obtained solutions with $\phi > 0.8$ for instances with 40 and 60 nodes with $\mu \leq 0.3$. Our eight heuristics obtained $\phi > 0.8$ for instances with more than 30 nodes and $\mu \leq 0.2$.

A visual comparison between Modularity Maximization and Modularity Density Maximization algorithms is seen in Figure 6.3.

Table 6.3: Details of the small random graphs generated by using the LFR benchmark.

Dataset name	μ	**Nodes**	**Edges**
30-01	0.1	30	54
30-02	0.2	30	49
30-03	0.3	30	49
30-04	0.4	30	62
30-05	0.5	30	41
30-06	0.6	30	53
30-07	0.7	30	53
30-08	0.8	30	51
40-01	0.1	40	84
40-02	0.2	40	92
40-03	0.3	40	87
40-04	0.4	40	87
40-05	0.5	40	88
40-06	0.6	40	89
40-07	0.7	40	81
40-08	0.8	40	96
50-01	0.1	50	397
50-02	0.2	50	394
50-03	0.3	50	399
50-04	0.4	50	397
50-05	0.5	50	400
50-06	0.6	50	399
50-07	0.7	50	398
50-08	0.8	50	400
60-01	0.1	60	132
60-02	0.2	60	124
60-03	0.3	60	131
60-04	0.4	60	129
60-05	0.5	60	119
60-06	0.6	60	143
60-07	0.7	60	128
60-08	0.8	60	131

Figure 6.3: Graphical results of our ground truth tests over Modularity Maximization and Modularity Density Maximization algorithms by using instances with 30, 40, 50, and 60 nodes.



6.3 Chapter Summary

Our experiments showed two main results. The first one shows that our heuristics performed a better search for solutions with clustering closer to the expected partition than CNM and Louvain do. The second one is about the results over exact methods for 30, 40, 50, and 60 nodes, where Modularity Density Maximization algorithms presented better results than Modularity Maximization in most of the tests.

The λ parameter of the Modularity Density Maximization objective function helped the heuristics to find better solutions for different difficulties of the μ value. Identifying the best λ parameter is an important factor when working with Modularity Density Maximization algorithms and heuristics.

Table 6.4: Comparisons among ϕ values obtained in the ground truth analyses among the exact algorithms, CNM, Louvain, and our eight heuristics for graphs with 30 nodes.

	Algorithms						Heuristics							
μ	Brandes et al. (2008)	λ	CGI+ILS	CNM	Louvain	λ	CM	LNM	CM+LNM	MCN	CM+MCN	MD	MDM	HLSMD
0.1	.808	0.1	-	.792	**.9**	0.1	-	.453	.632	.816	.899	.0	.55	.899
		0.2	-			0.2	.55	.51	.632	.825	.899	.0	.55	.899
		0.3	-			0.3	.55	.588	.632	.814	**.9**	.0	.899	**.9**
		0.4	-			0.4	.792	.484	**.9**	.789	**.9**	.0	.899	**.9**
		0.5	**.9**			0.5	.792	.434	**.9**	.761	.744	.0	.899	**.9**
		0.6	.747			0.6	.656	.416	.747	.608	.792	.0	.685	**.9**
		0.7	.585			0.7	.578	.388	.585	.586	.792	.0	.578	**.9**
		0.8	.474			0.8	.589	.381	.507	.534	.792	.0	.589	**.9**
		0.9	-			0.9	.45	.385	.393	.376	.792	.0	.45	**.9**
0.2	.491	0.1	-	.407	.406	0.1	**.547**	.301	.0	.486	**.547**	.0	.0	.489
		0.2	-			0.2	.0	.337	.0	.388	.518	.0	.0	.518
		0.3	.524			0.3	.23	.329	.226	.464	.484	.0	.23	.45
		0.4	-			0.4	.419	.298	.45	.349	.45	.0	.385	.45
		0.5	.491			0.5	.419	.267	.45	.379	.489	.0	.421	.417
		0.6	.455			0.6	.422	.257	.492	.315	.417	.0	.404	.417
		0.7	.41			0.7	.411	.242	.369	.344	.415	.0	.408	.417
		0.8	-			0.8	.264	.246	.216	.294	.417	.0	.264	.434
		0.9	.258			0.9	.264	.243	.216	.254	.434	.0	.264	.417
0.3	.095	0.1	-	.19	.195	0.1	.33	.203	.0	.33	.196	.0	.0	.233
		0.2	-			0.2	.0	.228	.0	.35	.196	.0	.0	.233
		0.3	.304			0.3	.126	.233	.126	.287	.196	.0	.126	.238
		0.4	-			0.4	.126	.252	.165	.327	.212	.0	.126	.233
		0.5	.205			0.5	.197	.241	.227	.268	.212	.0	.154	.238
		0.6	.353			0.6	.197	.2	.227	.25	.221	.0	.165	.244
		0.7	.264			0.7	.137	.222	.135	.3	.235	.0	.137	.248
		0.8	.272			0.8	.091	.229	.071	**.381**	.227	.0	.091	.252
		0.9	-			0.9	.124	.196	.133	.257	.248	.0	.124	.252
0.4	.291	0.1	-	.121	.307	0.1	.0	.207	.0	.209	.005	.0	.0	.017
		0.2	-			0.2	.0	.282	.0	.251	-0.006	.0	.0	.005
		0.3	.0			0.3	.0	.326	.0	.205	.005	.0	.0	.017
		0.4	-			0.4	.0	.292	.0	.185	.005	.0	.0	.017
		0.5	.171			0.5	.088	.273	.017	.062	.005	.0	.088	.017
		0.6	-			0.6	.284	.224	.264	.194	.005	.0	.279	.017
		0.7	-			0.7	**.36**	.196	.212	.161	.017	.0	.32	.017
		0.8	.222			0.8	.34	.184	.142	.309	.005	.0	.34	.015
		0.9	.284			0.9	.322	.176	.33	.265	.017	.0	.322	.0005
0.5	.026	0.1	-	.026	-0.004	0.1	-0.016	-0.016	.0	-0.023	-0.003	.0	.0	-0.0002
		0.2	-			0.2	.0	-0.005	.0	.017	-0.008	.0	.0	-0.008
		0.3	.004			0.3	-0.041	-0.009	-0.035	.01	-0.008	.0	-0.041	-0.0002
		0.4	-			0.4	.018	.016	-0.014	.01	-0.008	.0	-0.012	-0.0002
		0.5	-0.025			0.5	-0.012	.029	-0.024	.006	-0.015	.0	.015	-0.01
		0.6	-			0.6	-0.012	.047	-0.024	.006	-0.02	.0	-0.007	-0.02
		0.7	-0.008			0.7	-0.014	**.061**	-0.008	.016	-0.036	.0	-0.014	-0.027
		0.8	-0.024			0.8	-0.014	**.061**	-0.008	.03	-0.045	.0	-0.014	-0.036
		0.9	-0.043			0.9	-0.017	.059	-0.038	.051	-0.036	.0	-0.017	-0.036
0.6	.041	0.1	-	-0.0007	.01	0.1	.033	.011	.0	.033	-0.036	.0	.0	-0.036
		0.2	-			0.2	.0	.022	.0	.034	-0.036	.0	.0	-0.036
		0.3	.0			0.3	.0	-0.007	.0	.011	-0.036	.0	.0	-0.036
		0.4	-			0.4	.0	.002	.0	.017	-0.036	.0	.0	-0.036
		0.5	-0.008			0.5	-0.04	.011	-0.036	**.092**	-0.036	.0	-0.04	-0.036
		0.6	-0.002			0.6	.009	-0.001	.007	.031	-0.036	.0	.009	-0.036
		0.7	.033			0.7	.037	-0.01	.021	.036	-0.036	.0	.017	-0.036
		0.8	-			0.8	-0.017	-0.013	-0.017	.021	-0.014	.0	-0.017	-0.014
		0.9	.007			0.9	-0.022	-0.022	-0.044	.004	-0.014	.0	-0.022	-0.014
0.7	-0.059	0.1	-	-0.059	-0.055	0.1	**.0**	-0.052	**.0**	-0.033	-0.042	**.0**	**.0**	-0.039
		0.2	-			0.2	**.0**	-0.058	**.0**	-0.031	-0.039	**.0**	**.0**	-0.039
		0.3	-0.029			0.3	**.0**	-0.045	**.0**	-0.036	-0.061	**.0**	**.0**	-0.039
		0.4	-			0.4	-0.033	-0.049	-0.033	-0.05	-0.056	**.0**	-0.033	-0.056
		0.5	-0.055			0.5	-0.049	-0.058	-0.056	-0.046	-0.056	**.0**	-0.055	-0.056
		0.6	-			0.6	-0.063	-0.07	-0.071	-0.054	-0.061	**.0**	-0.056	-0.061
		0.7	-0.071			0.7	-0.069	-0.065	-0.077	-0.064	-0.055	**.0**	-0.069	-0.055
		0.8	-			0.8	-0.075	-0.081	-0.079	-0.09	-0.058	**.0**	-0.075	-0.055
		0.9	-0.04			0.9	-0.09	-0.092	-0.087	-0.078	-0.059	**.0**	-0.09	-0.055
0.8	**.003**	0.1	-	-0.043	-0.02	0.1	-	-0.045	.0	-0.028	-0.022	.0	.0	-0.022
		0.2	-			0.2	.0	-0.043	.0	-0.02	-0.022	.0	.0	-0.022
		0.3	-0.019			0.3	.0	-0.046	.0	-0.033	-0.022	.0	.0	-0.018
		0.4	-			0.4	-0.014	-0.036	-0.03	-0.031	-0.022	.0	-0.014	-0.018
		0.5	-0.028			0.5	-0.014	-0.041	-0.026	-0.058	-0.026	.0	-0.016	-0.018
		0.6	-0.066			0.6	-0.038	-0.06	-0.015	-0.02	-0.03	.0	-0.032	-0.03
		0.7	-0.039			0.7	-0.06	-0.058	-0.046	-0.031	-0.034	.0	-0.021	-0.034
		0.8	-0.033			0.8	-0.06	-0.031	-0.046	-0.056	-0.034	.0	-0.06	-0.034
		0.9	-0.064			0.9	-0.044	-0.036	-0.046	-0.047	-0.029	.0	-0.044	-0.034

Table 6.5: Comparisons among ϕ values obtained in the ground truth analyses among the exact algorithms, CNM, Louvain, and our eight heuristics for graphs with 40 nodes.

	Algorithms						**Heuristics**							
μ	**Brandes et al. (2008)**	λ	**CGI+ILS**	**CNM**	**Louvain**	λ	**CM**	**LNM**	**CM+LNM**	**MCN**	**CM+MCN**	**MD**	**MDM**	**HLSMD**
0.1	.465	0.1	-	.438	.467	0.1	-	.274	.0	.602	**1.0**	.0	.0	**1.0**
		0.2	-			0.2	.0	.253	.0	.553	.478	.0	.0	.478
		0.3	**1.0**			0.3	.0	.255	.0	.649	.478	.0	.0	.478
		0.4	-			0.4	.344	.244	.478	.566	.478	.0	.496	.478
		0.5	.612			0.5	.344	.208	.478	.484	.478	.0	.519	.478
		0.6	.475			0.6	.232	.209	.348	.435	.478	.0	.543	.478
		0.7	-			0.7	.202	.205	.233	.305	.478	.0	.519	.478
		0.8	.279			0.8	.209	.204	.255	.29	.478	.0	.209	.478
		0.9	.22			0.9	.19	.208	.207	.198	.478	.0	.19	.478
0.2	.79	0.1	.0	.785	**1**	0.1	-	.484	.0	.756	**1.0**	.0	.0	**1.0**
		0.2	-			0.2	.0	.511	.0	.777	**1.0**	.0	.0	**1.0**
		0.3	-			0.3	.463	.505	.588	.764	**1.0**	.0	**1.0**	**1.0**
		0.4	-			0.4	.669	.481	.747	.684	**1.0**	.0	**1.0**	**1.0**
		0.5	**1.0**			0.5	.822	.464	**1.0**	.824	**1.0**	.0	**1.0**	**1.0**
		0.6	.854			0.6	.707	.43	.774	.722	**1.0**	.0	**1.0**	**1.0**
		0.7	.758			0.7	.707	.419	.774	.672	**1.0**	.0	**1.0**	**1.0**
		0.8	.531			0.8	.629	.395	.681	.67	**1.0**	.0	.629	**1.0**
		0.9	.463			0.9	.507	.376	.473	.38	**1.0**	.0	.507	**1.0**
0.3	.532	0.1	.0	.469	.465	0.1	.0	.336	.0	.347	.401	.0	.0	.401
		0.2	-			0.2	.0	.411	.0	.324	.401	.0	.0	.401
		0.3	.0			0.3	.0	.301	.0	.45	.401	.0	.0	.401
		0.4	-			0.4	.285	.309	.401	.399	.401	.0	.285	.401
		0.5	.726			0.5	.285	.264	.401	.371	.401	.0	.596	.401
		0.6	**.825**			0.6	.38	.274	.604	.568	.401	.0	.632	.401
		0.7	.488			0.7	.386	.273	.507	.342	.401	.0	.528	.401
		0.8	-			0.8	.386	.259	.503	.349	.401	.0	.613	.401
		0.9	-			0.9	.241	.24	.29	.28	.401	.0	.241	.401
0.4	.366	0.1	.0	.318	.422	0.1	.0	.276	.0	.404	.212	.0	.0	.212
		0.2	-			0.2	.0	.222	.0	.444	.212	.0	.0	.212
		0.3	-			0.3	.0	.277	.0	.375	.192	.0	.0	.204
		0.4	-			0.4	.0	.257	.0	.45	.204	.0	.0	.216
		0.5	.464			0.5	.142	.276	.204	.265	.216	.0	.165	.204
		0.6	-			0.6	.377	.266	.462	.465	.216	.0	.382	.192
		0.7	**.54**			0.7	.394	.234	.373	.366	.207	.0	.391	.207
		0.8	-			0.8	.352	.238	.409	.32	.207	.0	.352	.207
		0.9	-			0.9	.251	.263	.269	.282	.207	.0	.251	.207
0.5	.14	0.1	.0	.089	.259	0.1	**.278** .121	.0	.073	**.278**	.0	.0	**.278**	
		0.2	-			0.2	.0	.122	.0	.125	**.278**	.0	.0	**.278**
		0.3	.0			0.3	.0	.107	.0	.18	**.278**	.0	.0	**.278**
		0.4	-			0.4	.0	.114	.0	.136	**.278**	.0	.188	**.278**
		0.5	.192			0.5	.188	.108	.201	.131	.201	.0	.188	.201
		0.6	-			0.6	.109	.127	.157	.173	.201	.0	.082	.201
		0.7	-			0.7	.109	.16	.154	.1	.201	.0	.131	.201
		0.8	.264			0.8	.114	.134	.174	.128	.201	.0	.114	.201
		0.9	.164			0.9	.093	.179	.102	.128	.195	.0	.093	.201
0.6	.034	0.1	.0	.036	.036	0.1	.0	.015	.0	.034	.002	.0	.0	.002
		0.2	-			0.2	.0	.017	.0	.016	.002	.0	.0	.002
		0.3	-			0.3	.0	.051	.0	.035	.03	.0	.0	.032
		0.4	-			0.4	.0	.023	.0	.023	.036	.0	.0	.036
		0.5	.013			0.5	.033	.04	.054	.043	.05	.0	.033	.046
		0.6	.07			0.6	.021	.045	.024	.034	.046	.0	.037	.043
		0.7	-0.012			0.7	.042	.047	.045	.033	.046	.0	.03	.046
		0.8	-0.007			0.8	**.074**	.059	.045	.038	.05	.0	**.074**	.046
		0.9	.057			0.9	.043	.062	.043	.031	.043	.0	.043	.046
0.7	.015	0.1	.0	.104	.014	0.1	.0	.021	.0	.044	.018	.0	.0	.018
		0.2	-			0.2	.0	.047	.0	.048	.018	.0	.0	.018
		0.3	.0			0.3	.0	.027	.0	.064	.018	.0	.0	.018
		0.4	-			0.4	.0	.029	.0	.031	.018	.0	.0	.018
		0.5	**.113**			0.5	.056	.025	.018	.059	.018	.0	.107	.018
		0.6	.09			0.6	.056	.028	.018	.069	.018	.0	.107	.018
		0.7	-			0.7	.041	.016	.014	.061	.018	.0	.094	.018
		0.8	.034			0.8	.048	.005	.026	.05	.018	.0	.048	.018
		0.9	-			0.9	.039	.021	.024	.009	.018	.0	.039	.018
0.8	-0.001	0.1	.0	.005	**.074**	0.1	-	.037	.0	-0.00005	.0	.0	.0	.0
		0.2	-			0.2	.0	.044	.0	.004	.0	.0	.0	.0
		0.3	.0			0.3	.0	.054	.0	.02	.0	.0	.0	.0
		0.4	-			0.4	.0	.041	.0	.002	.0	.0	.0	.0
		0.5	-0.024			0.5	.0	.052	.0	.036	.0	.0	.0	.0
		0.6	.01			0.6	-0.016	.036	-0.025	.01	.0	.0	.061	.0
		0.7	-			0.7	-0.012	.048	-0.03	.02	.0	.0	.059	.0
		0.8	.032			0.8	.003	.024	.014	.022	.0	.0	.044	.0
		0.9	.055			0.9	.046	.037	.035	**.074**	.0	.0	.046	.0

Table 6.6: Comparisons among ϕ values obtained in the ground truth analyses among the exact algorithms, CNM, Louvain, and our eight heuristics for graphs with 50 nodes.

	Algorithms						Heuristics							
μ	Brandes et al. (2008)	λ	CGI+ILS	CNM	Louvain	λ	CM	LNM	CM+LNM	MCN	CM+MCN	MD	MDM	HLSMD
0.1	.687	0.1	-	**1**	**1**	0.1	-	.198	.0	.89	**1.0**	.0	.0	**1.0**
		0.2	-			0.2	**1.0**	.235	**1.0**	.905	**1.0**	.0	**1.0**	**1.0**
		0.3	-			0.3	**1.0**	.189	**1.0**	.89	**1.0**	.0	**1.0**	**1.0**
		0.4	-			0.4	**1.0**	.235	**1.0**	.889	**1.0**	.0	**1.0**	**1.0**
		0.5	**1.0**			0.5	**1.0**	.178	**1.0**	.905	**1.0**	.0	**1.0**	**1.0**
		0.6	**1.0**			0.6	**1.0**	.178	**1.0**	.89	**1.0**	.0	**1.0**	**1.0**
		0.7	**1.0**			0.7	**1.0**	.178	**1.0**	.889	**1.0**	.0	**1.0**	**1.0**
		0.8	.682			0.8	**1.0**	.185	**1.0**	.874	**1.0**	.0	**1.0**	**1.0**
		0.9	.422			0.9	.693	.178	.698	.569	**1.0**	.0	.693	**1.0**
0.2	**1**	0.1	.0	**1**	**1**	0.1	-	.196	.0	.723	.0	.0	.0	.0
		0.2	-			0.2	.0	.204	.0	.905	.0	.0	.0	.0
		0.3	.0			0.3	.0	.214	.0	.861	.0	.0	.0	.0
		0.4	-			0.4	.0	.202	.0	.722	.0	.0	**1.0**	.0
		0.5	**1.0**			0.5	.0	.205	.0	.905	.0	.0	**1.0**	.0
		0.6	**1.0**			0.6	.699	.205	**1.0**	.861	.0	.0	**1.0**	.0
		0.7	**1.0**			0.7	.699	.207	**1.0**	.673	.0	.0	**1.0**	.0
		0.8	-			0.8	.699	.209	**1.0**	.875	.0	.0	**1.0**	.0
		0.9	-			0.9	.364	.205	.414	.51	.0	.0	**1.0**	.0
0.3	.0	0.1	.0	**1**	**1**	0.1	-	.097	.0	.0	.0	.0	.0	.0
		0.2	-			0.2	.0	.13	.0	.0	.0	.0	.0	.0
		0.3	.0			0.3	.0	.108	.0	.0	.0	.0	.0	.0
		0.4	-			0.4	.0	.13	.0	.235	.0	.0	.0	.0
		0.5	.0			0.5	.0	.107	.0	.235	.0	.0	.0	.0
		0.6	**1.0**			0.6	.0	.116	.0	.235	.0	.0	.0	.0
		0.7	**1.0**			0.7	.0	.107	.0	.235	.0	.0	.257	.0
		0.8	.757			0.8	.02	.088	.624	.336	.0	.0	.495	.0
		0.9	.439			0.9	.138	.107	.347	.275	.0	.0	.478	.0
0.4	.0	0.1	.0	.13	.043	0.1	.0	.04	.0	.0	.0	.0	.0	.0
		0.2	-			0.2	.0	.08	.0	.0	.0	.0	.0	.0
		0.3	.0			0.3	.0	.05	.0	.0	.0	.0	.0	.0
		0.4	-			0.4	.0	.077	.0	.0	.0	.0	.0	.0
		0.5	.0			0.5	.0	.046	.0	.0	.0	.0	.0	.0
		0.6	.0			0.6	.0	.05	.0	.0	.0	.0	.0	.0
		0.7	.0			0.7	.0	.046	.0	.0	.0	.0	.0	.0
		0.8	.03			0.8	.0	.05	.0	.0	.0	.0	.045	.0
		0.9	**.165**			0.9	.021	.046	.095	.058	.0	.0	.045	.0
0.5	-	0.1	.0	**.025**	-0.027	0.1	-	-0.0007	.0	.0	.0	.0	.0	.0
		0.2	-			0.2	.0	-0.005	.0	.0	.0	.0	.0	.0
		0.3	.0			0.3	.0	-0.038	.0	.0	.0	.0	.0	.0
		0.4	-			0.4	.0	-0.005	.0	.0	.0	.0	.0	.0
		0.5	.0			0.5	.0	-0.029	.0	.0	.0	.0	.0	.0
		0.6	.0			0.6	.0	-0.035	.0	.0	.0	.0	.0	.0
		0.7	.0			0.7	.0	-0.029	.0	.0	.0	.0	.0	.0
		0.8	-0.008			0.8	-0.011	-0.033	-0.018	.0	.0	.0	-0.014	.0
		0.9	.004			0.9	-0.036	-0.029	-0.028	-0.017	.0	.0	-0.023	.0
0.6	**.0**	0.1	**.0**	-0.027	-0.026	0.1	**.0**	-0.027	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.2	-			0.2	**.0**	-0.035	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.3	**.0**			0.3	**.0**	-0.035	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.4	-			0.4	**.0**	-0.035	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.5	**.0**			0.5	**.0**	-0.031	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.6	**.0**			0.6	**.0**	-0.041	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.7	**.0**			0.7	**.0**	-0.031	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.8	-0.017			0.8	-0.017	-0.034	-0.018	**.0**	**.0**	**.0**	-0.016	**.0**
		0.9	-0.042			0.9	-0.016	-0.031	-0.027	-0.024	**.0**	**.0**	-0.019	**.0**
0.7	**.0**	0.1	**.0**	-0.017	-0.031	0.1	**.0**	-0.064	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.2	-			0.2	**.0**	-0.057	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.3	**.0**			0.3	**.0**	-0.058	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.4	-			0.4	**.0**	-0.05	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.5	**.0**			0.5	**.0**	-0.056	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.6	**.0**			0.6	**.0**	-0.049	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.7	-			0.7	**.0**	-0.056	**.0**	-0.005	**.0**	**.0**	**.0**	**.0**
		0.8	-0.022			0.8	-0.017	-0.057	-0.013	**.0**	**.0**	**.0**	-0.025	**.0**
		0.9	-0.041			0.9	-0.041	-0.056	-0.032	-0.027	**.0**	**.0**	-0.031	**.0**
0.8	**.0**	0.1	**.0**	-0.032	-0.03	0.1	**.0**	-0.068	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.2	-			0.2	**.0**	-0.066	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.3	-			0.3	**.0**	-0.066	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.4	-			0.4	**.0**	-0.066	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.5	**.0**			0.5	**.0**	-0.069	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.6	**.0**			0.6	**.0**	-0.067	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.7	**.0**			0.7	**.0**	-0.069	**.0**	**.0**	**.0**	**.0**	**.0**	**.0**
		0.8	-0.029			0.8	**.0**	-0.066	**.0**	**.0**	**.0**	**.0**	-0.014	**.0**
		0.9	-0.044			0.9	-0.034	-0.069	-0.042	-0.032	**.0**	**.0**	-0.028	**.0**

Table 6.7: Comparisons among ϕ values obtained in the ground truth analyses among the exact algorithms, CNM, Louvain, and our eight heuristics for graphs with 60 nodes.

	Algorithms							**Heuristics**						
μ	**Brandes et al. (2008)**	λ	**CGI+ILS**	**CNM**	**Louvain**	λ	**CM**	**LNM**	**CM+LNM**	**MCN**	**CM+MCN**	**MD**	**MDM**	**HLSMD**
0.1	**1**	0.1	.571	.966	**1**	0.1	-	.562	.458	.848	.811	.0	.458	.811
		0.2	-			0.2	.728	.54	.827	.868	.811	.0	.861	.811
		0.3	**1.0**			0.3	.728	.529	.827	.84	.811	.0	.895	.811
		0.4	-			0.4	.728	.508	.827	.847	.811	.0	.942	.811
		0.5	.939			0.5	.729	.487	.811	.859	.811	.0	.894	.811
		0.6	-			0.6	.729	.44	.811	.791	.811	.0	.931	.811
		0.7	.664			0.7	.604	.434	.668	.735	.811	.0	.604	.811
		0.8	.642			0.8	.564	.428	.642	.721	.811	.0	.564	.811
		0.9	.508			0.9	.553	.434	.586	.4	.79	.0	.553	.811
0.2	**1**	0.1	-	.593	.911	0.1	-	.487	.0	.495	.514	.0	.0	.49
		0.2	-			0.2	.0	.426	.0	.603	.481	.0	.0	.591
		0.3	.712			0.3	.375	.399	.427	.632	.474	.0	.751	.594
		0.4	-			0.4	.383	.404	.518	.669	.465	.0	.731	.681
		0.5	.824			0.5	.421	.386	.681	.576	.465	.0	.748	.686
		0.6	.769			0.6	.501	.38	.763	.608	.465	.0	.693	.742
		0.7	.635			0.7	.466	.37	.659	.572	.47	.0	.523	.808
		0.8	-			0.8	.44	.36	.518	.584	.465	.0	.44	.801
		0.9	.417			0.9	.328	.353	.35	.357	.475	.0	.328	.801
0.3	**.957**	0.1	-	.4	.716	0.1	-	.504	.0	.456	.586	.0	.0	.553
		0.2	-			0.2	.0	.473	.0	.509	.619	.0	.0	.601
		0.3	-			0.3	.0	.447	.0	.509	.619	.0	.0	.601
		0.4	-			0.4	.21	.485	.341	.579	.619	.0	.553	.619
		0.5	.929			0.5	.378	.442	.582	.543	.638	.0	.499	.619
		0.6	.929			0.6	.378	.434	.582	.488	.619	.0	.435	.601
		0.7	-			0.7	.357	.421	.455	.559	.601	.0	.357	.601
		0.8	.525			0.8	.407	.407	.533	.468	.619	.0	.407	.584
		0.9	.43			0.9	.331	.397	.369	.384	.601	.0	.331	.616
0.4	.0	0.1	.0	.259	**.425**	0.1	-	.226	.0	.234	.0	.0	.0	.0
		0.2	-			0.2	.0	.221	.0	.207	.0	.0	.0	.0
		0.3	.0			0.3	.0	.241	.0	.173	.0	.0	.0	.0
		0.4	-			0.4	.0	.207	.0	.206	.0	.0	.0	.0
		0.5	.279			0.5	.0	.234	.0	.225	.0	.0	.161	.0
		0.6	-			0.6	.148	.243	.214	.255	.0	.0	.18	.0
		0.7	.357			0.7	.148	.244	.196	.23	.0	.0	.148	.0
		0.8	-			0.8	.177	.227	.273	.276	.0	.0	.173	.0
		0.9	-			0.9	.18	.218	.209	.225	.0	.0	.18	.0
0.5	.125	0.1	.0	.08	.078	0.1	.0	.097	.0	.121	.069	.0	.0	.068
		0.2	-			0.2	.0	.092	.0	.115	.07	.0	.0	.07
		0.3	.12			0.3	.0	.054	.0	.075	.07	.0	.0	.07
		0.4	-			0.4	.071	.08	.05	.105	.07	.0	.081	.07
		0.5	.12			0.5	.09	.096	.064	.11	.063	.0	.136	.069
		0.6	.14			0.6	.125	.092	.149	.114	.074	.0	.13	.074
		0.7	-			0.7	.153	.094	.125	.134	.07	.0	.141	.07
		0.8	-			0.8	.149	.092	.138	.097	.07	.0	.149	.07
		0.9	.055			0.9	**.166**	.088	.154	.106	.066	.0	**.166**	.066
0.6	.0	0.1	.0	.085	.169	0.1	.0	.097	.0	.091	.001	.0	.0	.001
		0.2	-			0.2	.0	.095	.0	.063	.001	.0	.0	.001
		0.3	-			0.3	.0	.067	.0	.092	.001	.0	.0	.001
		0.4	-			0.4	.0	.065	.0	.086	.001	.0	.0	.001
		0.5	.112			0.5	-0.003	.092	.001	.081	.001	.0	-0.003	.001
		0.6	.109			0.6	.087	.093	.047	.124	.003	.0	.087	.005
		0.7	-			0.7	.098	.117	.095	.1	.005	.0	.088	.005
		0.8	.188			0.8	.116	.126	.143	.123	.005	.0	.116	.007
		0.9	**.205**			0.9	.127	.133	.1	.138	.007	.0	.127	.011
0.7	.0	0.1	-	.048	.053	0.1	.043 .035	.0	.012	.062	.0	.0	.043	
		0.2	-			0.2	.0	.053	.0	.024	.066	.0	.0	.058
		0.3	-			0.3	.0	.034	.0	.048	.066	.0	.0	**.071**
		0.4	-			0.4	.0	.043	.0	.037	**.071**	.0	.0	**.071**
		0.5	.018			0.5	.03	.047	**.071**	.016	**.071**	.0	.028	**.071**
		0.6	-			0.6	.026	.037	.033	.029	**.071**	.0	.031	**.071**
		0.7	.031			0.7	.034	.047	.031	.025	**.071**	.0	.034	**.071**
		0.8	-			0.8	.023	.038	.034	.042	**.071**	.0	.023	**.071**
		0.9	-			0.9	.04	.038	.04	.053	**.071**	.0	.04	**.071**
0.8	.0	0.1	.0	.02	**.039**	0.1	-	.015	.0	-0.005	-0.007	.0	.0	-0.006
		0.2	-			0.2	.0	.009	.0	-0.008	-0.006	.0	.0	-0.007
		0.3	-			0.3	.0	-0.001	.0	-0.01	-0.004	.0	.0	.001
		0.4	-			0.4	.0	-0.001	.0	.007	-0.001	.0	.0	.005
		0.5	.023			0.5	-0.031	-0.008	.011	-0.011	-0.004	.0	.011	.006
		0.6	.021			0.6	-0.027	-0.011	.028	.014	-0.003	.0	.001	.008
		0.7	-			0.7	-0.022	-0.012	-0.011	-0.004	-0.006	.0	-0.022	.007
		0.8	-			0.8	.002	-0.015	-0.013	.001	-0.006	.0	.002	.017
		0.9	-			0.9	.012	-0.016	.002	-0.008	-0.012	.0	.012	.002

7 CONCLUSIONS AND FUTURE WORK

In this work, we proposed heuristic searches which solve larger instances than the current Modularity Density Maximization heuristics do, and showed how close the obtained solutions are to the expected clustering results. As main contributions, we list the theoretical results about the prioritizers for constructive searches, the creation of eight new heuristics for Modularity Density Maximization that solve instances with hundreds of thousands of nodes, and the column generation algorithms.

The theoretical contributions helped in understanding which prioritizers can be used in constructive heuristics. We showed that Modularity Density Maximization fails as a prioritizer, so an alternative prioritizer was suggested. It was based on the density of edges inside each cluster to preserve the modular property. We showed that this alternative prioritizer does not merge some types of adjacent cliques. We also proved that this prioritizer presents problems when detecting star-shaped modules. These results helped the development of our eight heuristics.

Our eight new heuristics for Modularity Density Maximization are CM, LNM, CM+LNM, MCN, CM+MCN, MD, MDM, and HLSMD. The heuristics which found the lowest gap for average and the best D value were CM+LNM, MD, MDM, and HLSMD. The fastest heuristics are CM, LNM, MCN, and CM+MCN. HLSMD reached the best partitions for the largest graphs tested in at most 10 minutes. Except for CM, all our heuristics surpass the other Modularity Density Maximization heuristic solvers found in the literature for some instances. Our ground truth analyses showed that CM+LNM was the only tested heuristic that maintained high-quality solutions for instances with difficulty parameter $\mu < 7$. It means that even the instance with 60% of the edges mixed with nodes of different modules, CM+LNM led to good solutions. CM+MCN and HLSMD showed high quality for instances with $\mu \leq 3$. Hence, the results suggest that CM+LNM, CM+MCN and HLSMD are the state-of-the-art heuristics for Modularity Density Maximization.

We developed six column generation methods that used our HLSMD heuristic.

Two of them surpassed the exact algorithm results from the literature: CGI+ILS and CGII+ILS. They use an Iterated Local Search to find good columns faster than the exact auxiliary algorithm.

With the aim of understanding the differences between Modularity Maximization and Modularity Density Maximization optimal results, our ground truth analyses tested their exact methods. The optimal solutions proved to be far from the expected partition for instances with $\mu > 3$. The results also suggest that Modularity Density Maximization is better than Modularity Maximization.

The ground truth analyses also reinforced the idea that the λ parameter is an important factor for Modularity Density Maximization heuristics. The results of those analyses showed that some λ values are good for an instance and worse for others when using the same heuristic.

Another result of this thesis is an answer to the problem related to the exponential number of suboptimal partitions for Modularity Maximization (GOOD; MONTJOYE; CLAUSET, 2010). This is detailed in A.

We conclude that the reached results are relevant, and they suggest improvements to the state-of-the-art for the Modularity Density Maximization problem. Our contributions are relevant computational methods for a number of problems in several science areas.

7.1 Future Works

In this section, we present future works related to this thesis.

7.1.1 Alternative Prioritizer and Star-shaped Modules

As it is shown in Section 3.3, our alternative prioritizer can fail to detect star-shaped modules. This behavior is a problem because this kind of module is expected to be seen as a cluster. In this structure, a central node is connected to other nodes which had degree 1, so all other nodes are dependent on this central node. In a social network context, this could mean that an important person is followed by other people who do not know each other. This cluster could represent a cluster of fans of an important person. As further work, we suggest the investigation of combined or new prioritizers that can be used fix this problem.

7.1.2 Signed Modularity Density Maximization

Signed networks can represent positive and negative relationships between nodes of a network. In the context of Modularity Density Maximization, some methods are discussed in Li, Liu and Liu (2014), but the results are about graphs with at most $1,800$ nodes. The methods of Li, Liu and Liu (2014) do not work with directed and weighted networks. The weight and direction can be used to represent more details about the relationships like how much a person hates or loves another one as can be seen in Slovene Parliamentary Parties Network (KROPIVNIK; MRVAR, 1996). As future research, we suggested that new methods can be applied to signed, directed and weighted networks with more than $100,000$ nodes. We started a research on this line, but the graphs used in the experiments are unweighted and undirected.

7.1.3 Parallel Constructive and Multilevel Heuristics

Some of our eight heuristics of Section 4 can be converted into parallel heuristics for improving scability. So, they can be tested to understand the best number of threads and computers to solve specific sizes of graphs. The heuristics could divide the search space among the threads and processors to accelerate or improve the search for better solutions as proposed in Costa et al. (2016).

7.1.4 Ground Truth Analyses for Different Graph Clustering Functions

As described in Introduction (Chapter 1) and Background (Chapter 2) chapters, there are several methods for graph clustering inspired on the Modularity optimization. It is important to understand what prioritizer or objective function can lead to the best solutions. For graph clustering problems, ground truth comparisons among heuristic and exact methods can help to understand the best method to use. In this thesis and our paper (SANTIAGO; LAMB, 2017), we compare heuristics for Modularity Maximization and Modularity Density Maximization that solved instances with hundreds of thousands of nodes. The results suggest that the heuristics based on Modularity Density Maximization have the best results.

REFERENCES

AGARWAL, G.; KEMPE, D. Modularity-maximizing graph communities via mathematical programming. **The European Physical Journal B**, v. 66, n. 3, p. 409–418, Nov. 2008.

ALOISE, D. et al. Column generation algorithms for exact modularity maximization in networks. **Physical Review E**, v. 82, n. 4, p. 046112, Oct. 2010.

ALOISE, D. et al. Modularity maximization in networks by variable neighborhood search. In: BADER, D. A. et al. (Ed.). **Contemporary Mathematics**. Providence: American Mathematical Society, 2013. v. 588, chap. Graph Partitioning and Graph Clustering, p. 113–127.

ARENAS, A.; FERNANDEZ, A.; GOMEZ, S. Analysis of the structure of complex networks at different resolution levels. **New Journal of Physics**, v. 10, p. 053039, 2008.

BARCZ, A. et al. The game-theoretic interaction index on social networks with applications to link prediction and community detection. In: **International Joint Conference on Artificial Intelligence**. Buenos Aires: [s.n.], 2015. p. 638–644.

BATAGELJ, V.; MRVAR, A. **Pajek datasets**. [S.l.], 2006. Available at: <http://vlado.fmf.uni-lj.si/pub/networks/data/>.

BLONDEL, V. D. et al. Fast unfolding of communities in large networks. **Journal of Statistical Mechanics: Theory and Experiment**, v. 2008, n. 10, p. P10008, Oct. 2008.

BRANDES, U. et al. On Modularity Clustering. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, v. 20, n. 2, p. 172–188, Feb. 2008.

CAFIERI, S.; HANSEN, P.; LIBERTI, L. Loops and multiple edges in modularity maximization of networks. **Physical review. E, Statistical, nonlinear, and soft matter physics**, v. 81, n. 4 Pt 2, p. 046102, 2010.

CALDERONI, F.; BRUNETTO, D.; PICCARDI, C. Communities in criminal networks: a case study. **Social Networks**, v. 48, p. 116–125, Jan. 2017.

CHAKRABORTY, T. et al. Constant communities in complex networks. **Scientific reports**, v. 3, p. 1825, 2013.

CHAN, E.; YEUNG, D. A convex formulation of modularity maximization for community detection. In: **Proceedings of International Joint Conference on Artificial Intelligence**. Barcelona: [s.n.], 2011. p. 2218–2225.

CHEN, M.; KUZMIN, K.; SZYMANSKI, B. K. Community detection via maximization of modularity and its variants. **IEEE Transactions on Computational Social Systems**, v. 1, n. 1, p. 46–65, Mar. 2014.

CHEN, M.; NGUYEN, T.; SZYMANSKI, B. K. On Measuring the Quality of a Network Community Structure. In: **International Conference on Social Computing (SocialCom), 2013**. Alexandria: IEEE, 2013. p. 122–127.

CLAUSET, A.; NEWMAN, M. E. J.; MOORE, C. Finding community structure in very large networks. **Physical Review E**, v. 70, n. 6, p. 066111-1 – 066111-6, Dec. 2004.

COSTA, A. **Some remarks on modularity density**. [S.l.], 2014. Available at: <http://arxiv.org/abs/1409.4063>.

COSTA, A. MILP formulations for the modularity density maximization problem. **European Journal of Operational Research**, Elsevier Ltd., v. 245, n. 1, p. 14–21, 2015.

COSTA, A. et al. Divisive heuristic for modularity density maximization. **Computers & Operations Research**, v. 71, p. 100–109, 2016.

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

DARST, R. K.; NUSSINOV, Z.; FORTUNATO, S. Improving the performance of algorithms to find communities in networks. **Physical Review E**, v. 89, n. 3, p. 032809, Mar. 2014.

DJIDJEV, H. N.; ONUS, M. Scalable and accurate graph clustering and community structure detection. **IEEE Transactions on Parallel and Distributed Systems**, v. 24, n. 5, p. 1022–1029, May. 2013.

EATON, E.; MANSBACH, R. A spin-glass model for semi-supervised community detection. In: **Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence**. Toronto: [s.n.], 2012. p. 900–906.

FERRARA, E. et al. Detecting criminal organizations in mobile phone networks. **Expert Systems with Applications**, Elsevier Ltd, v. 41, n. 13, p. 5733–5750, Oct. 2014.

FORTUNATO, S.; BARTHÉLEMY, M. Resolution limit in community detection. **Proceedings of the National Academy of Sciences of the United States of America**, v. 104, n. 1, p. 36–41, Jan. 2007.

FORTUNATO, S.; CASTELLANO, C. Community structure in graphs. **Computational Complexity**, p. 490–512, 2012.

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

GONG, M. et al. An improved memetic algorithm for community detection in complex networks. In: **2012 IEEE Congress on Evolutionary Computation**. Brisbane, QLD: IEEE, 2012. p. 1–8.

GOOD, B. H.; MONTJOYE, Y.-A. de; CLAUSET, A. Performance of modularity maximization in practical contexts. **Physical Review E**, v. 81, n. 4, p. 046106, Apr. 2010.

GRANELL, C.; GÓMEZ, S.; ARENAS, A. Hierarchical multiresolution method to overcome the resolution limit in complex networks. **International Journal of Bifurcation and Chaos**, v. 22, n. 07, p. 1250171-1 – 1250171-7, Jul. 2012.

GUIMERA, R.; AMARAL, L. Functional cartography of complex metabolic networks. **Nature**, v. 433, n. February, p. 895–900, 2005.

HRIC, D.; DARST, R. K.; FORTUNATO, S. Community detection in networks: structural communities versus ground truth. **Physical Review E**, v. 90, n. 6, p. 062805, dec 2014.

IBM. **IBM ILOG CPLEX Optimization Studio V12.6.3 documentation**. [S.l.]: IBM, 2015.

IZUNAGA, Y.; MATSUI, T.; YAMAMOTO, Y. **A doubly nonnegative relaxation for modularity density maximization**. Tsukuba, 2016. 1–15 p. Available at: <http://www.optimization-online.org/DB{_}HTML/2016/03/5368.h>.

JARUKASEMRATANA, S.; MURATA, T. Edge weight method for community detection in scale-free networks. **Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics**, ACM Press, New York, v. 1, n. c, p. 1–9, 2014.

JIA, S. et al. Anti-triangle centrality-based community detection in complex networks. **IET systems biology**, v. 8, n. 3, p. 116–25, Jun. 2014.

JIANG, J. Q.; MCQUAY, L. J. Modularity functions maximization with nonnegative relaxation facilitates community detection in networks. **Physica A: Statistical Mechanics and its Applications**, v. 391, n. 3, p. 854–865, 2012.

KARIMI-MAJD, A.-M.; FATHIAN, M.; AMIRI, B. A hybrid artificial immune network for detecting communities in complex networks. **Computing**, v. 97, n. 5, p. 483–507, 2014.

KERNIGHAN, B. W.; LIN, S. An efficient heuristic procedure for partitioning graphs. **Bell System Technical Journal**, v. 49, n. 2, p. 291–307, Feb. 1970.

KROPIVNIK, S.; MRVAR, A. An analysis of the Slovene parliamentary parties network. **Developments in Statistics and Methodology, Metodološki zvezki**, v. 12, p. 209–216, 1996.

KRZAKALA, F. et al. Spectral redemption in clustering sparse networks. **Proceedings of the National Academy of Sciences of the United States of America**, v. 110, n. 52, p. 20935–40, Dec. 2013.

LANCICHINETTI, A.; FORTUNATO, S. Consensus clustering in complex networks. **Scientific reports**, v. 2, p. 336, Jan. 2012.

LANCICHINETTI, A.; FORTUNATO, S.; RADICCHI, F. Benchmark graphs for testing community detection algorithms. **Physical Review E - Statistical, Nonlinear, and Soft Matter Physics**, v. 78, n. 4, p. 1–6, 2008.

LANCICHINETTI, A. et al. Finding statistically significant communities in networks. **PloS one**, v. 6, n. 4, p. e18961, Jan. 2011.

LEICHT, E.; NEWMAN, M. Community structure in directed networks. **Physical Review Letters**, v. 100, n. 11, p. 118703, Mar. 2008.

LESKOVEC, J.; KREVL, A. **Stanford Large Network Dataset Collection**. [S.l.], 2014. Available at: <http://snap.stanford.edu/data>.

LI, Y.; LIU, J.; LIU, C. A comparative analysis of evolutionary and memetic algorithms for community detection from signed social networks. **Soft Computing**, v. 18, n. 2, p. 329–348, Feb. 2014.

LI, Z. et al. Quantitative function and algorithm for community detection in bipartite networks. **American Journal of Operations Research**, v. 5, p. 421–434, Jan.2015.

LI, Z. et al. Quantitative function for community detection. **Physical Review E**, v. 77, n. 3, p. 036109, Mar. 2008.

LIU, J.; ZENG, J. Community detection based on modularity density and genetic algorithm. In: **Proceedings of International Conference on Computational Aspects of Social Networks**. Taiyuan: [s.n.], 2010. p. 29–32.

MATTHEWS, B. W. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. **BBA - Protein Structure**, v. 405, n. 2, p. 442–451, 1975.

MEO, P. D. et al. Enhancing community detection using a network weighting strategy. **Information Sciences**, Elsevier Inc., v. 222, p. 648–668, 2013.

MEO, P. D. et al. Mixing local and global information for community detection in large networks. **Journal of Computer and System Sciences**, Elsevier Inc., v. 80, n. 1, p. 72–87, Feb. 2014.

MEUNIER, D. et al. Modular structure of functional networks in olfactory memory. **NeuroImage**, Elsevier Inc., v. 95, p. 264–75, Jul. 2014.

MUFF, S.; RAO, F.; CAFLISCH, A. Local modularity measure for network clusterizations. **Physical Review E**, v. 72, n. 5, p. 056107, Nov. 2005.

NASCIMENTO, M. C.; PITSOULIS, L. Community detection by modularity maximization using GRASP with path relinking. **Computers & Operations Research**, Elsevier, v. 40, n. 12, p. 3121–3131, Dec. 2013.

NASH, S. G. **Encyclopedia of Operations Research and Management Science**. Boston, MA: Springer US, 2013.

NEPUSZ, T.; YU, H.; PACCANARO, A. Detecting overlapping protein complexes in protein-protein interaction networks. **Nature Methods 9**, p. 471–472, 2012.

NEWMAN, M. Mixing patterns in networks. **Physical Review E**, v. 67, n. 2, p. 026126, Feb. 2003.

NEWMAN, M. **Spectral community detection in sparse networks**. [S.l.], 2013. Available at: <http://arxiv.org/abs/1308.6494>.

NEWMAN, M.; GIRVAN, M. Finding and evaluating community structure in networks. **Physical Review E**, v. 69, n. 2, p. 026113, Feb. 2004.

NEWMAN, M. E. J. Analysis of weighted networks. **Physical Review E**, v. 70, n. 5, p. 056131, Nov.2004.

NEWMAN, M. E. J. Finding community structure in networks using the eigenvectors of matrices. **Physical Review E**, v. 74, n. 3, p. 036104, Sept. 2006.

PALLA, G. et al. Uncovering the overlapping community structure of complex networks in nature and society. **Nature**, v. 435, p. 814–818, Jun. 2005.

PARK, H.; LEE, K. Dependence clustering, a method revealing community structure with group dependence. **Knowledge-Based Systems**, Elsevier B.V., v. 60, p. 58–72, Apr. 2014.

PEI, Y.; CHAKRABORTY, N.; SYCARA, K. Nonnegative matrix tri-factorization with graph regularization for community detection in social networks. In: **International Joint Conference on Artificial Intelligence**. Buenos Aires: [s.n.], 2015. p. 2083–2089.

PIZZUTI, C. Boosting the detection of modular community structure with genetic algorithms and local search. In: **Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12**. New York, New York, USA: ACM Press, 2012. p. 226.

PONS, P.; LATAPY, M. Computing communities in large networks using random walks. **Journal of Graph Algorithms and Applications**, Springer Berlin Heidelberg, Berlin, v. 3733, n. 2, p. 191–218, 2005.

RADICCHI, F. et al. Defining and identifying communities in networks. **Proceedings of the National Academy of Sciences of the United States of America**, v. 101, n. 9, p. 2658–2663, 2004.

ROTTA, R.; NOACK, A. Multilevel local search algorithms for modularity clustering. **Journal of Experimental Algorithmics**, v. 16, n. 2, p. 2.1, May. 2011.

SANKAR, M. V.; RAVINDRAN, B.; SHIVASHANKAR, S. CEIL: a scalable, resolution limit free approach for detecting communities in large networks. In: **International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2015. p. 2097–2103.

SANTIAGO, R.; LAMB, L. C. Efficient modularity density heuristics for large graphs. **European Journal of Operational Research**, Elsevier B.V., v. 258, n. 3, p. 844–865, May 2017.

SCHMEJA, S. Identifying star clusters in a field: a comparison of different algorithms. **Astronomische Nachrichten**, v. 332, n. 2, p. 172–184, Feb. 2011.

SUN, P. G. Weighting links based on edge centrality for community detection. **Physica A: Statistical Mechanics and its Applications**, v. 394, p. 346 – 357, 2014.

TIAN, F. et al. Learning deep representations for graph clustering. In: **Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence**. [S.l.]: AAAI Press, 2014. p. 1293–1299.

TRAAG, V. A.; DOOREN, P. V.; NESTEROV, Y. Narrow scope for resolution-limit-free community detection. **Physical Review E**, v. 84, n. 1, p. 016114, Jul. 2011.

WAKITA, K.; TSURUMI, T. Finding community structure in mega-scale social networks. In: **Proceedings of the 16th International Conference on World Wide Web**. New York: ACM Press, 2007. p. 1275.

WANG, J. et al. Remarks on network community properties. **Journal of Systems Science and Complexity**, v. 21, n. 4, p. 629–636, Dec. 2008.

XIE, J.; KELLEY, S.; SZYMANSKI, B. K. Overlapping community detection in networks. **ACM Computing Surveys**, v. 45, n. 4, p. 1–35, Aug. 2013.

XU, G.; TSOKA, S.; PAPAGEORGIOU, L. G. Finding community structures in complex networks using mixed integer optimisation. **European Physical Journal B**, v. 60, p. 231–239, 2007.

YANG, S.; LUO, S. Quantitative measure for community detection in weighted networks. **Modern Physics Letters B**, v. 23, n. 27, p. 3209–3224, Oct. 2009.

ZHANG, J.; QIU, Y.; ZHANG, X. S. Detecting community structure: from parsimony to weighted parsimony. **Journal of Systems Science and Complexity**, v. 23, n. 5, p. 1024–1036, 2010.

ZHAO, Y. et al. A cellular learning automata based algorithm for detecting community structure in complex networks. **Neurocomputing**, Elsevier, v. 151, p. 1216–1226, 2015.

Appendices

A SUBOPTIMAL PARTITIONS FOR MODULARITY MAXIMIZATION

This section describes our contribution to explain why Modularity Maximization has an exponential number of high-quality solutions. This behavior was first reported in Good, Montjoye and Clauset (2010). The demonstrations made here are marginal contributions to the main line of this work. These results were presented at the IEEE Congress on Evolutionary Computation of 2016.

Figure A.1: The landscape of several searches in “Football”.

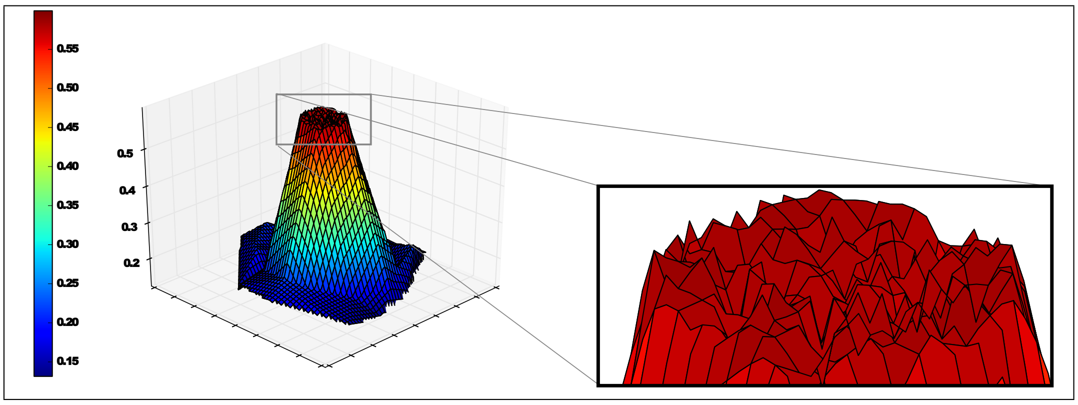


Figure A.1 represents the partitions collected in all of 30 restarts for each neighborhood in the “Football” instance. Observing the figure, the plateaus occupy a significant part of the explored area. This behavior is also reported in (EATON; MANSBACH, 2012; CAFIERI; HANSEN; LIBERTI, 2010; GOOD; MONTJOYE; CLAUSET, 2010; DARST; NUSSINOV; FORTUNATO, 2014). The data plotted to analyze the optimization landscape are collected from the Tabu Search stochastic local search. $\alpha.|V|$ value gives the iterations that a feature remains in the tabu list, where α is randomly selected from the interval between 0.1 and 0.9. The data were collected at each 100 iterations, with stop criteria of $10,000$ iterations, $1,000$ iterations without partition improvement, or when the execution reaches the best-known partition. A restart is made for each replication. The *1-neighborhood* strategy was used. To plot the optimization landscape and to show the degeneracy graphically, we used a method to compute the distance between two partitions and to depict each partition in the two-dimensional space by multidimensional scaling. The distance metric used was function Variation of Information (VI).

In the solution space for Modularity Maximization, a partition C can be represented as a set of disjoint clusters made up of nodes. The contribution of each node w in cluster c, for modularity function, can be seen in Equation (A.1). We divide Equation (A.1) into gain and penalty components.

$$q(w,c) = \underbrace{\sum_{u \in c \setminus \{w\}} \left(\frac{a_{uw}}{|E|} \right)}_{gain} - \underbrace{\sum_{u \in c \setminus \{w\}} \left(\frac{d_u d_w}{2|E|^2} \right) - \frac{d_w^2}{4|E|^2}}_{penalty} \tag{A.1}$$

The gain is the total positive contribution to the objective value. The gain factor represents the adjacency proportion from w connected to other nodes in cluster c. As the referred problem is a maximization, the larger the gain, the larger the solution value. The penalty factor is an equation that leads to the decreasing of the modularity value. It is a representation of an expected number of edges connected from w to any other node from cluster c.

A question to be answered is what feature of instance exploration can lead to more gain than penalty? The answer is seen in Proposition 1 here.

Proposition 1. *To obtain significant gain, a node must be in a partition where its internal cluster degree (d_w^c) is proportionally larger than the value of its expected edges in cluster c times the total number of edges.*

Proof. To prove this proposition, we assume that the gain value is significantly larger than that of the penalty, using Equation (A.1). Then we simplified the inequations in the sequence A.2 and A.3.

As d_w^c is the degree of node w inside cluster c. Inequation (A.4) shows that to change the partition to a higher objective value, the value of d_w^c must be larger than all degrees from $u \in c \backslash \{w\}$ multiplied by d_w and divided by four times the total number of edges.

$$\frac{\sum_{u \in c \backslash \{w\}} \left(a_{uw}\right)}{|E|} \gg \frac{\sum_{u \in c \backslash \{w\}} \left(d_u d_w\right)}{2|E|^2} + \frac{d_w^2}{4|E|^2} \tag{A.2}$$

$$\sum_{u \in c \backslash \{w\}} \left(a_{uw}\right) \gg \frac{2\sum_{u \in c \backslash \{w\}} \left(d_u d_w\right) + d_w^2}{4|E|} \tag{A.3}$$

$$d_w^c \gg \frac{2\sum_{u \in c \backslash \{w\}} \left(d_u d_w\right) + d_w^2}{4|E|} \tag{A.4}$$

□

Using Proposition 1, we claim that in every instance with a non-uniform degree distribution (like scale-free networks), the number of nodes of a specified low degree dictates the number of solutions in the plateau of suboptimal modularity. This is proved in Proposition 2. This proposition implies that the number of partitions is also exponential.

Proposition 2. *Nodes of low degree determine the number of partitions with suboptimal modularity.*

Proof. For Proposition 2, low degree nodes are formally defined. A low degree node is one which has a proportionally large distance from the highest degree in the graph. At this point, defining this proportion is necessary to understand Proposition 2.

Let the set V_{low} be made up of nodes of low degree that give a total gain of at most αQ_{max}. We define $V_{low} = \{w \in V : d_w \leq x\}$, where x is the upper bound for all w degrees, such that inequation (A.5) is always true. Suppose that Q_{max} is the optimal Q value.

$$\sum_{w \in V_{low}} gain(w, c_w) \leq \alpha Q_{max} \tag{A.5}$$

The degree of nodes has a positive correlation with the gain and penalty provided by the node itself for a solution value; then low degree nodes lead to small changes in modularity. If these changes are performed over the solution with Q_{max}, then we have suboptimal partitions given by changing low degree nodes.

Parameter $\alpha \in [0, 1]$ is the influence on modularity value. We want to find a degree x as upper bound, where α is small, to give a lower bound of how much partitions with suboptimal modularity at least $(1 - \alpha)Q_{max}$ can exist.

In inequation (A.6), the gain function is unveiled for simplification purposes.

$$\sum_{w \in V_{low}} \sum_{u \in c_w \setminus \{w\}} \left(\frac{a_{uw}}{|E|} \right) \leq \alpha Q_{max} \tag{A.6}$$

Considering that $\left(\sum_{u \in c_w \setminus \{w\}} a_{uw} \right) \leq d_w$, then we can assume that this substitution is the maximum gain possible for node w, where all its neighbors are inside the same cluster, as defined in inequation (A.7).

$$\sum_{w \in V_{low}} \left(\frac{d_w}{|E|} \right) \leq \alpha Q_{max} \tag{A.7}$$

Assuming that all nodes in V_{low} have at most a degree value x, we have inequation (A.8) and simplified inequation (A.9).

$$\sum_{w \in V_{low}} \left(\frac{x}{|E|} \right) \leq \alpha Q_{max} \tag{A.8}$$

$$|V_{low}|x \leq \alpha Q_{max}|E| \tag{A.9}$$

Inequation (A.9) defines $\alpha Q_{max}|E|$ as being upper bound to the summation of all degrees of nodes that belong to V_{low} to determine what nodes have almost no influence on the modularity value.

For the sake of clarity, we apply inequation (A.9) in the "Adjnoun" instance. Assuming suboptimal partitions with a modularity of at least $(0.9) \times Q_{max}$, we must define parameter α=0.1. As this instance has $|E|$ = 425 and Q_{max} = 0.313367 (ALOISE et al., 2013), we have $|V_{low}|x \leq$ 13.3180975. In this example, we could compose V_{low} with a group of nodes where the sum of all degrees is at most 13. All partitions generated by changing clusters of $w \in V_{low}$ will have a modularity value between $(0.9) \times Q_{max}$ and Q_{max}, so all these solutions will be suboptimal partitions.

By contradiction, we prove this proposition using set V_{low} and the degree upper bound on x. We will assume that the number of suboptimal solutions with a modularity value of at least $(1-\alpha)Q_{max}$ is not exponential. The number of possible partitions for an instance graph $G(V, E)$ is B_n, where $n = |V|$. Let $l = |V_{low}|$, the total number of partitions without V_{low} nodes is B_{n-l}. Then the number of possible partitions by varying the clusters of V_{low} nodes is $B_n - B_{n-l}$.

$$\begin{aligned}
B_n - B_{n-l} &= \\
\sum_{i=0}^{n-1}\left[\binom{n-1}{i}B_i\right] - \sum_{i=0}^{n-l-1}\left[\binom{n-l-1}{i}B_i\right] &= \\
\sum_{i=0}^{n-l-1}\left[\binom{n-l-1}{i}B_i\right] + \sum_{i=n-l}^{n-1}\left[\binom{n-1}{i}B_i\right] & \\
- \sum_{i=0}^{n-l-1}\left[\binom{n-l-1}{i}B_i\right] & \\
\geq \sum_{i=n-l}^{n-1}\left[\binom{n-1}{i}B_i\right] &
\end{aligned} \tag{A.10}$$

In equation list A.10 above, we have a simplification sequence resulting in a lower bound for the number of partitions that influence V_{low}; these solutions provide suboptimal modularity scoring. Clearly, the resulting lower bound is exponential, contradicting the assumption that the number of nodes of low degree does not determine the number of solutions with suboptimal modularity.

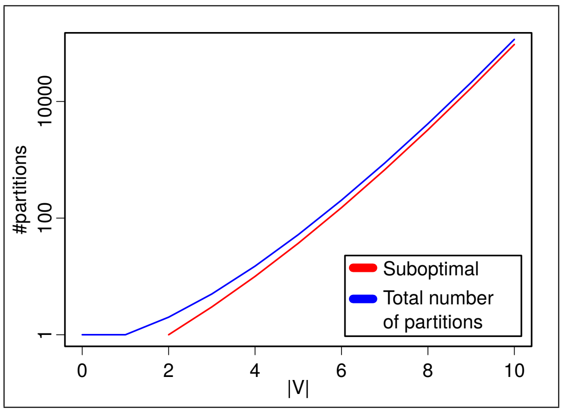
□

Figure A.2 shows the difference between the total number of partitions and the partitions with suboptimal solutions for instances with at most ten nodes. It is also assumed that $|V_{low}| = 0.1|V|$. One can see in this figure that there is a large number of suboptimal solutions, even if only 10% of nodes have small degree value.

A.1 Experimental Analysis

In order to validate Proposition 1 experimentally, we tested the same stochastic local search Tabu Search from Section A, mapping each iteration to classical instances described in Table A.1.

Figure A.2: Comparison between the total number of partitions and the partitions (blue) with suboptimal modularity value (red) for instances with at most ten nodes. The axis "#partitions" is in log scale.



The results are summarized in Figures A.3, A.4, A.5, A.6 and A.7, where the vertical axis ΔQ represents the modularity value difference between the last and the current partition at each iteration, and the horizontal axis "Internal Degree Difference" represents the difference between inside cluster degree of the last and the current partition. The colors represent the ΔQ modularity improvement (green denotes large improvements, red denotes small improvements). We can see that, the larger the value of the degree, the larger the modularity value of partitions.

Figure A.3: Degree difference in modularity gain in "Adjnoun".

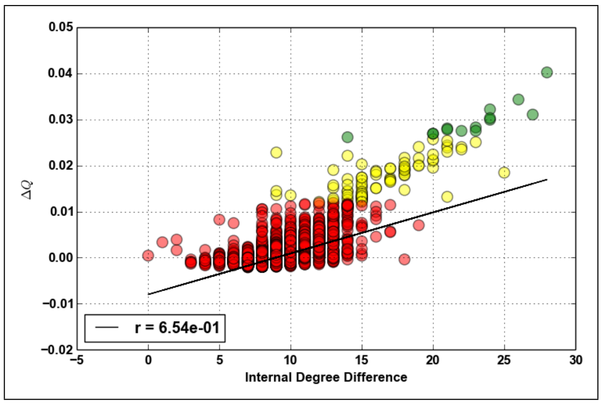


Figure A.4: Internal cluster degree and modularity gain in “Dolphins”.

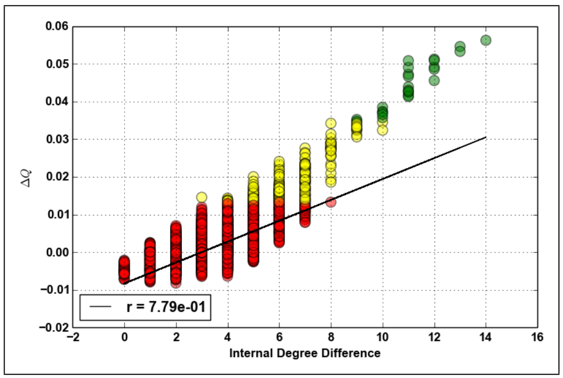
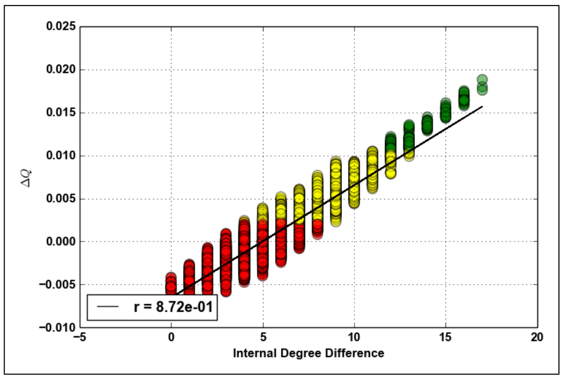


Figure A.5: Internal cluster degree and modularity gain in “Football”.



Another feature that can be observed in the figures is the concentration of partitions. Small ΔQ is associated with a high number of partitions. This behavior happens due to the small number of highest degree nodes. This observation agrees with Proposition 2.

The line “r” in the figures represents the linear approximation error. Table 4.1 shows the correlation between modularity value and inside cluster degree (Pearson/Spearman) for each classical instance. The large amount of small ΔQ in partition space influences the reading, but a positive correlation is also found.

Figure A.6: Internal cluster degree and modularity gain in “Karate”.

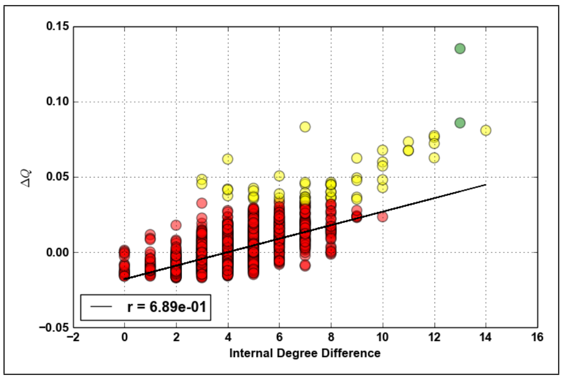
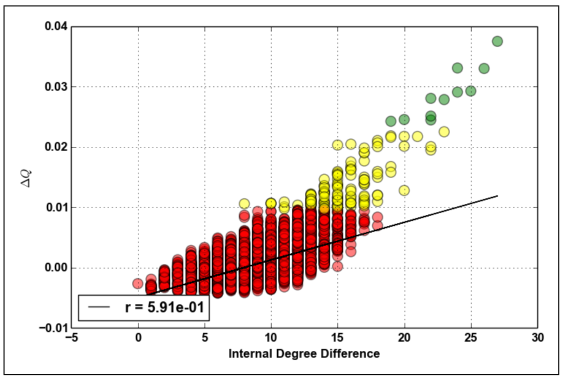


Figure A.7: Internal cluster degree and modularity gain in “Polbooks”.



A.2 On the Role of Influences

In order to show how the information provided by Proposition 1 and Proposition 2 could be used to locate higher modularity partitions, we changed two heuristics. The first is a Tabu Search and the second, the Louvain method (BLONDEL et al., 2008) that are scalable for instances with thousands of nodes. They have been changed to consider the degree influence. The following results show that the new versions obtained an improvement.

Table A.1: Classical Instances and Experimental Results

Graph	**Nodes**	**Edges**	**Q***	**Correlation (Pearson/Spearman)**
Adjnoun	112	425	.313	.656 / .609
Dolphins	62	159	.529	.779 / .796
Football	115	613	.605	.872 / .859
Karate	34	78	.420	.689 / .672
Polbooks	105	441	.527	.591 / .606

A.2.1 Adapted Tabu Search

The same Tabu Search from Section A is used on classical (see Table A.1) and random instances. The stop criteria is $10,000$ iterations or $1,000$ iterations without improvement from the current partition, and the number of replications was 30 (restarts).

The random instances are artificially created with a fixed number of clusters $\rho \in \{2, 4, 10, 20\}$ and number of nodes $n \in \{100, 200, 300, 400, 500\}$. Each cluster has $^n/_\rho$ nodes that are randomly connected, with internal edge density of 80%. The frequency of edges connecting two nodes from different clusters are $\beta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. The total number of random instances is 320.

Two experiments have been carried out over the described scenario: (*i*) using a randomly generated initial partition for each start, also denoted as “without advantage” or “non-adv”, and (*ii*) using as initial partition the one generated by Algorithm 22, which uses information advantages given by the propositions, also labelled as advantage or “adv”. The “non-adv” initial partition algorithm is a constructive stochastic method. The partition S is empty at the beginning, and a list l is created with all nodes. At each iteration, a random node is removed from l and inserted into partial solution S with all its neighbors. These neighbors are also removed from l. This procedure is repeated until all nodes from the graph are in S.

Algorithm 22 starts splitting the sorted nodes by decreasing degree order into the lists $leaders$ and $followers$. The $leaders$ list is composed of highest degree nodes and

$followers$ is composed of the remaining nodes. This is made to assign nodes with a small degree to the nodes in the $leaders$ list (lines 5 to 10). The remaining nodes from $followers$ are chosen by their highest degree rank, and their clusters are composed of their adjacencies (lines 11 to 16).

Algorithm 22: Initial partition generator "adv" algorithm.

```
   Input : $G(V, E)$, $S$ all nodes from $V$ sorted by decreasing degree
   // leaders and followers keep $S$ order
1  $\bar{x} \leftarrow \sum_{v \in V} \left( \frac{d_v}{|V|} \right)$
2  $leaders \leftarrow [v \in S | d_v \geq \bar{x}]$
3  $followers \leftarrow V \setminus leaders$
4  $Partition \leftarrow \emptyset$
5  foreach $l \in leaders$ do
6      $cluster \leftarrow \{l\}$
7      foreach $u \in N(l) \cap followers$ do
8          $cluster \leftarrow cluster \cup \{u\}$
9          $followers \leftarrow followers \setminus \{u\}$
10     $Partition \leftarrow Partition \cup \{cluster\}$
11 foreach $f \in followers$ do
12     $cluster \leftarrow \{f\}$
13     foreach $u \in N(f) \cap followers$ do
14         $cluster \leftarrow cluster \cup \{u\}$
15         $followers \leftarrow followers \setminus \{u\}$
16     $Partition \leftarrow Partition \cup \{cluster\}$
17 return $Partition$
```

The modularity value of the results was tested using the following two hypotheses:

- H_0^{TS}: the modularity value computed by using the initial partition from "adv" (Algorithm 22) is equal to the obtained value by the random initial partition algorithm "non-adv";
- H_1^{TS}: the modularity value computed by using the initial partition from "adv" (Algorithm 22) is better than the obtained value by random initial partition algorithm "non-adv".

We have used the Wilcoxon Signed Rank hypothesis test on modularity scoring results, paired test using H_1^{TS} as the alternative. With very high significance the null hypothesis (H_0^{TS}) is rejected, and H_1^{TS} is maintained. For the experiments, we show that Propositions 1 and 2 can be used in MM heuristics to locate suboptimal partitions, by spending more time exploring the high scoring plateau. The *p-value* obtained was less than 3.15×10^{-5}.

A.2.2 Adapted Louvain Heuristic

In this subsection, we report the experiments with the adapted version of Louvain method that was first introduced in (BLONDEL et al., 2008). We selected this heuristic because it is scalable for instances with thousands of nodes.

The adapted version of the Louvain method is presented in Algorithm 23, and it is very similar to the Louvain method. It also has two phases, where the first phase will take the instance and try to move the nodes from its cluster to another that leads to an improvement in the modularity value. When no improvement is found, the second phase melts each cluster as a single meta-node, thus changing the instance.

The difference between our adapted and the original version of the Louvain method is in the first phase. Instead of passing to the second phase after no improvement is found, our adapted version order the nodes by the number of no internal edges. This order is described in Algorithm 24. Our idea is first to move nodes which have the largest number of adjacencies outside of their clusters. Only after no improvement is found in this extension of the first phase that the second phase is run.

In the experiments, the original and adapted version have been run 30 times (number of replications). The best partitions achieved were collected and used for comparison purposes. The instances used in the experiments can be seen in Table 4.1. They are classical instances of (BATAGELJ; MRVAR, 2006) and undirected, unweighted instances of the “Stanford Large Network Dataset Collection” (LESKOVEC; KREVL, 2014).

Algorithm 23: Adapted Louvain Method.

```
   Input  : $G(V,E), partition$
1  $best \leftarrow partition$
2  $improvement \leftarrow true$
3  while $improvement$ do
4      $improvement \leftarrow false$
       // begin of first phase
5      for $i \leftarrow 1$ to $2$ do
6          if $i = 1$ then
7              $orderedV \leftarrow randomOrder(V)$
8          else
9              $orderedV \leftarrow notIntDegreeOrder(G, best)$
10         repeat
11             $moves \leftarrow 0$
12             foreach $v \in orderedV$ do
13                 $\Delta Q_v \leftarrow 0$
14                 $c_v \leftarrow \emptyset$
15                 foreach $c \in C_{N(v)}$ do
16                     if $\Delta Q_v < \Delta Q(v,c)$ then
17                         $\Delta Q_v \leftarrow \Delta Q(v,c)$
18                         $c_v \leftarrow c$
19                 if $c_v \neq \emptyset$ then
20                     $moves \leftarrow moves + 1$
21                     in $partition$, move node $v$ to cluster $c_v$
22         until $moves = 0$
       // begin of second phase
23     if $Q(best) < Q(partition)$ then
24         $best \leftarrow partition$
25         $improvement \leftarrow true$
26         clusters are coarsened as nodes in $partition$ and graph $G(V,E)$
27 return $best$
```

Algorithm 24: Algorithm $notIntDegreeOrder$.

```
  Input  : $G(V,E), partition$
1 $temp \leftarrow list()$
2 foreach $v \in V$ do
3     $ni \leftarrow d_v - d_v^c$
4     $temp \leftarrow temp \cup \{(v, ni)\}$
5 order $temp$ by descending of $ni$
6 $ordered \leftarrow list()$
7 foreach $(v, ni) \in temp$ do
8     $ordered \leftarrow ordered \cup \{v\}$
9 return $ordered$
```

The best modularity value found in experiments for each instance was tested using the following two hypotheses:

- H_0^{LA}: the best modularity value resulting from the adapted Louvain method is equal to the best-obtained value by the original Louvain method;
- H_1^{LA}: the best modularity value resulted from the adapted Louvain method is better than the best-obtained value by the original Louvain method.

The hypothesis test used was the same of described in the previous subsection. With very high significance the null hypothesis (H_0^{LA}) is rejected, and H_1^{LA} is maintained. The *p-value* obtained was less than 0.002.

A.3 Chapter Summary

We illustrated experimental tests that confirmed our analytical results. The experiments have corroborated the importance of the highest degree nodes on the modularity value and also illustrated the small impact of small degree nodes w.r.t. the objective value of a partition. The work reported in this thesis leads to future developments in algorithms and heuristics for MM optimization problems and methods, by providing ideas that contribute toward the fast convergence of solutions, in particular in the region of suboptimal partitions.

B *RESUMO*

"Modularity Density Maximization" é um problema de particionamento de grafos que evita a resolução limite do problema denominado "Modularity Maximization". A tese apresentada neste documento tem como objetivo resolver instâncias maiores que as heurísticas para Modularity Density Maximization e demonstrar a proximidade das soluções obtidas em relação às esperadas. A primeira contribuição é sobre as funções de prioridade para heurísticas deste problema. A segunda é a criação de oito heuristicas para Modularity Density Maximization. Nossas heurísticas são comparadas com os resultados ótimos da literatura, e as heurísticas denominadas GAOD, iMeme-Net, HAIN, e BMD-λ. Os resultados também foram comparados com CNM e Louvain que são heurísticas para a Modularity Maximization que podem resolver instâncias com milhares de vérticas. Os testes foram realizados usando grafos da "Stanford Large Network Dataset Collection", e os experimentos demonstraram que nossas oito heurísticas encontraram soluções para grafos com centenas the milhares de vértices. Os resultados demonstram que cinco de nossas heurísticas melhoraram o estado-da-arte de heurísticas para a Modularity Density Maximization. Nossos seis geradores de colunas são a terceira contribuição. Estes métodos usam algoritmos ou heurísticas como resolvedores de problema auxiliar. Comparações entre as gerações de colunas propostas e os algoritmos do estado-da-arte foram realizadas. Os resultados demonstraram que (i) dois de nossos métodos melhoraram o estado-da-arte sobre algoritmos em termos de tempo, e (ii) nossos geradores de colunas provam o valor ótimo para instâncias maiores que os resolvedores atuais conseguem. Conclui-se que os resultados são relevantes, e eles sugerem avanços no estado-da-arte para o problema da Modularity Density Maximization.

Palavras-chave: *Maximização de modularidade por densidade. particionamento. busca heurística. heurísticas multinível. busca local. geração de colunas.*