MARIO COSTA LEVORATO JUNIOR

EFFICIENT SOLUTIONS TO THE CORRELATION CLUSTERING PROBLEM

NITERÓI

2015

MARIO COSTA LEVORATO JUNIOR

EFFICIENT SOLUTIONS TO THE CORRELATION CLUSTERING PROBLEM

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização.

UNIVERSIDADE FEDERAL FLUMINENSE

Orientador: Yuri Abitbol de Menezes Frota

NITERÓI

2015

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

L666 Levorato Junior, Mario Costa
Efficient solutions to the correlation clustering problem / Mario Costa Levorato Junior. – Niterói, RJ : [s.n.], 2015.
91 f.

Dissertação (Mestrado em Computação) – Universidade Federal Fluminense, 2015.
Orientador: Yuri Abitbol de Menezes Frota.

1. Metaheurística. 2. Rede social. 3. Grafo. I. Título

CDD 005.136

MARIO COSTA LEVORATO JUNIOR

EFFICIENT SOLUTIONS TO THE CORRELATION CLUSTERING PROBLEM

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Algoritmos e Otimização.

Aprovada em dezembro de 2015.

BANCA EXAMINADORA

Prof. Yuri Abitbol de Menezes Frota(Orientador), IC-UFF

Prof ª. Lúcia Maria de Assumpção Drummond, IC-UFF

Prof. Luiz Satoru Ochi, IC-UFF

Prof ª. Cristiana Barbosa Bentes, UERJ

Niterói
2015

Este trabalho é dedicado à minha familia e a todos aqueles que contribuíram para que eu pudesse chegar até aqui.

Acknowledgements

The author would like to thank all the professors who contributed to the development of this work, in particular, Yuri Abitbol de Menezes Frota, Lúcia Maria de Assumpção Drummond and Rosa Maria Videira de Figueiredo, for their guidance and relaxed, thoughtful insights.

Special thanks to Petrobras, to Carlos Eduardo Cabral da Cunha for his precious help with the computer cluster, and also to Prof. Luiz Satoru Ochi for his valuable contributions to the improvement of the algorithms.

Resumo

Um dos desafios enfrentados por pesquisadores de redes sociais consiste na avaliação do equilíbrio em redes sociais de sinais, onde interações positivas (amizade) e negativas (antagonismo) estão presentes. O nível de equilíbrio de um grupo social pode ser utilizado como ferramenta de estudo pelos pesquisadores de redes sociais para saber de que forma (e se) determinado grupo evolui para um possível estado de equilíbrio. Neste sentido, uma rede social pode ser representada através de um grafo de sinais e a solução de problemas de *clustering* definidos sobre grafos de sinais pode ser utilizada como um critério para medição do nível de equilíbrio em redes sociais. Tal medida pode ser obtida por meio da solução ótima para o Problema de Correlação de Clusters (*Correlation Clustering* ou CC), assim como uma variação do mesmo, conhecida como *Relaxed Correlation Clustering* (RCC) *problem*. Contudo, resolver tais problemas não se traduz em tarefa fácil, especialmente quando é necessário analisar grandes instâncias de rede. Este trabalho visa contribuir para a solução eficiente de ambos os problemas por meio do desenvolvimento de versões sequenciais e paralelas das metaheurísticas GRASP e ILS. Ao aplicarmos estes algoritmos, foi possível realizar, de forma eficiente, a medição do equilíbrio estrutural em grandes redes sociais do mundo real.

Palavras-chave: Metaheurísticas. Grafo de sinais. Correlação de Clusters. Rede Social. Equilíbrio Estrutural. GRASP. ILS. MPI. CUDA.

Abstract

One challenge for social network researchers is to evaluate balance in signed social networks, where positive (friendly) and negative (antagonistic) interactions take place. The degree of balance of a social group can be employed as a tool to study whether and how this group might evolve to a possible balanced state. The solution of clustering problems defined on signed graphs can be used as a criterion to measure the degree of balance in social networks. This measure can be obtained with the optimal solution of the Correlation Clustering (CC) problem, as well as in variation, being the Relaxed Correlation Clustering (RCC) problem. However, solving these problems is no easy task, especially when large network instances require analysis. This work contributes to the efficient solution of both problems by developing sequential and parallel versions of GRASP and ILS metaheuristics. Then, by incorporating our algorithms, we efficiently solve the problem of measuring the structural balance on large real-world signed social networks.

Keywords: Metaheuristics. Signed Graph. Correlation Clustering. Social Network. Structural Balance. GRASP. ILS. MPI. CUDA.

List of Figures

List of Tables

List of abbreviations and acronyms

ANSI	American National Standards Institute
CC	Correlation Clustering (problem)
CPU	Central Processing Unit
CSR	Compressed Sparse Row (format)
CUDA	Compute Unified Device Architecture
DM	Doreian-Mrvar (algorithm)
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GRASP	Greedy Randomized Adaptive Search Procedure
ILP	Integer Linear Programming
ILS	Iterated Local Search
MPI	Message Passing Interface
PVND	Parallel Variable Neighborhood Descent
RAM	Random-Access Memory
RCC	Relaxed Correlation Clustering (problem)
SIMD	Single Instruction Multiple Data
SM	Stream Multiprocessor
SRCC	Symmetric Relaxed Correlation Clustering (problem)
SRI	Symmetric Relaxed Imbalance
TSP	Travelling Salesman Problem
UNGA	United Nations General Assembly

USA	United States of America
USSR	Union of Soviet Socialist Republics
VND	Variable Neighborhood Descent
VNS	Variable Neighborhood Search

Contents

1 Introduction

Structural balance is acknowledged as a fundamental social process. It has been used to explain how feelings, attitudes and beliefs, which social actors direct towards each other, can promote the formation of stable (but not necessarily conflict-free) social groups. Structural balance theory states that the balance or equilibrium of a social system should follow the human tendency to preserve a cognitive consistency of hostility and friendship. The supporting principle, as proposed by Heider (HEIDER, 1946), is simple: “my friend’s friend is my friend, my friend’s enemy is my enemy, my enemy’s friend is my enemy, my enemy’s enemy is my friend". The absence of balance creates a stream of tension in the minds of group members that can eventually lead to the altering of opinion. Once balance is achieved, it tends to become stable, since no cognitive dissonance could change the state.

Determining the structural balance of a signed social network is a key aspect in the study of the structure and origin of tensions and conflicts in a network of individuals whose mutual relationships are characterized in terms of friendship and hostility. Structural balance theory was first formulated by Heider (HEIDER, 1946) with the purpose of describing sentiment relationship between those people pertaining to the same social group (like/dislike, love/hate, respect/disrespect, or trust/distrust). Signed graphs were then introduced by Cartwright et al. (CARTWRIGHT; HARARY, 1956), which formalized Heider’s theory that stated that a balanced social group could be partitioned into two groups (or clusters), being that all relationships (or edges) within clusters are positive (internal solidarity) and all those between clusters are negative (mutually hostile groups). Davis (DAVIS, 1967) later introduced a notion called "weak balance", that further generalizes social balance with an assertion that a balanced social group can be divided into two or more mutually antagonistic subgroups (or clusters), each having internal solidarity.

Structural balance theory has a multitude of applications. Investigating the temporal dynamics of communities has attracted an increasing amount of attention, with models attempting to forecast how networks evolve over time. Known works involve the dynamics of signed structures and group formation (DOREIAN; KRACKHARDT, 2001), including balance adjustment processes (ABELL; LUDWIG, 2009). Structural balance is also potentially useful to study similarity and correlation networks, such as those determined by common voting patterns, or alliances and disputes among nations (TRAAG; BRUGGEMAN, 2009; MACON; MUCHA; PORTER, 2012).

Over the last decades, signed graphs have shown to be a very attractive and discrete structure for social network researchers (DOREIAN; MRVAR, 1996a; INOHARA, 1998; YANG; CHEUNG; LIU, 2007; ABELL; LUDWIG, 2009; DOREIAN; MRVAR, 2009;

FACCHETTI; IACONO; ALTAFINI, 2011; ESMAILIAN; ABTAHI; JALILI, 2014). By using these structures, researchers face the challenge of measuring and evaluating balance within a social network. Differing criteria and different resolution approaches have been incorporated into the literature as an attempt to accomplish this task. However, most social network analysis is based on unsigned networks such as Facebook and Twitter (DUCH; ARENAS, 2005; NEWMAN, 2006; BRANDES et al., 2008), which contain only positive relationships (e.g. friend or trust) between users. For signed social networks, quantifying and evaluating balance is still challenging and problematic (DOREIAN; MRVAR, 2009; LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010; FACCHETTI; IACONO; ALTAFINI, 2011; SRINIVASAN, 2011; ESMAILIAN; ABTAHI; JALILI, 2014). This work is focused on the evaluation of structural balance within these networks. Our main contribution is to efficiently solve the problem of measuring the structural balance on large real-world signed social networks.

Clustering is the action of grouping individual elements based on their similarity. Clustering problems defined on signed graphs arise in many scientific areas, such as efficient document classification (BANSAL; BLUM; CHAWLA, 2002), detection of embedded matrix structures (GÜLPINAR et al., 2004), biological systems (DASGUPTA et al., 2007), community structure (TRAAG; BRUGGEMAN, 2009; MACON; MUCHA; PORTER, 2012), and image segmentation (KIM et al., 2014). The common element among these applications is the collaborative vs. conflicting environment in which they are defined. In particular, the clustering task in a signed network attempts to identify k antagonistic groups in the network, being that most entities within the same cluster are friends while most entities belonging to different clusters are enemies. Notice that, since this (weak) balance definition solely applies to signed networks, most traditional clustering algorithms for unsigned networks cannot be directly applied.

Moreover, with this in mind, the solution of clustering problems defined on signed graphs can be used as criterion to measure the degree of balance in social networks (DOREIAN; MRVAR, 1996a; DOREIAN; MRVAR, 2009; FIGUEIREDO; MOURA, 2013). One such instance is the Correlation Clustering (CC) problem, defined by Bansal et al. (BANSAL; BLUM; CHAWLA, 2002), which provides a measure for clustering a set of objects into the optimal number of clusters, without specifying a said number in advance. The main idea is that, in a signed network, there are some links that create imbalance. The number of such links can be expressed as an amount of frustration. Links that contribute to frustration are negative links within clusters and positive links between clusters. The CC objective function consists of minimizing that frustration. Apart from the CC problem, alternative measures to the structural balance and the clustering problems associated with them have also been previously discussed (DOREIAN; MRVAR, 2009; FIGUEIREDO; MOURA, 2013). Our intention is to evaluate the structural balance using the CC and a

relaxed version of this problem.

From a practical point of view, when solving the clustering problems treated in this paper, heuristic approaches are of prime interest, since large social networks[1] require analysis (KUNEGIS; LOMMATZSCH; BAUCKHAGE, 2009; LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010; FACCHETTI; IACONO; ALTAFINI, 2011). For example, large-scale online networks with two opposite kinds of relationships are very common nowadays. Slashdot (SLASHDOT, 1997), a technology-related news website, includes a feature which allows users to tag each other as friends or foes. On-line review websites such as Epinions (EPINIONS, 1999) allow users to either like or dislike other people's reviews and this behavior can be modeled as a signed network, where edge weights can be either greater or less than zero, representing, respectively, a positive or negative relationship. The definition of a measure to represent the balance/imbalance of a social network adds to itself a degree of approximation to the task of evaluating balance in a social network. Thus, it is imperative that the clustering problem associated with this measure be solved efficiently. This is particularly challenging when, as in social networks retrieved from on-line media, the size of the community is very big, of the order of 10^5 individuals or higher.

To our knowledge, until the development of this work, Zhang et al. (ZHANG et al., 2008) presented the only metaheuristic approach applied to the CC problem. It consists of a genetic algorithm for the CC problem, applied to document clustering. Since previous works on the CC problem have used greedy local search algorithms with success (VOTE/BOEM (ELSNER; SCHUDY, 2009) and Doreian Mrvar algorithm (DOREIAN; MRVAR, 1996b)), we have chosen to focus our research in the development of local search algorithms with a greedy component. Moreover, population-based algorithms (like genetic algorithms) usually have a high computational cost, while local search algorithms with a greedy component are faster when solving combinatorial problems on large-scale instances. A neighborhood search heuristic for a related problem (clique partitioning), developed by Brusco et al. (BRUSCO; KÖHN, 2009), has confirmed our choice.

In this work, we present a Greedy Randomized Adaptive Search Procedure (GRASP) (FEO; RESENDE, 1995) implementation capable of efficiently solving the problem in networks of up to 8000 vertices (DRUMMOND et al., 2013). Nonetheless, after observing the great amount of time spent on the processing of larger graphs, we saw an opportunity to extend this GRASP algorithm with a hybrid metaheuristic that can solve the problem faster. We then developed sequential and parallel algorithms based on the Iterated Local Search (ILS) (LOURENÇO; MARTIN; STÜTZLE, 2003) metaheuristic. By employing the proposed algorithms, we demonstrate the improvements of ILS over the GRASP procedure. We also extend the procedure to solve the Symmetric Relaxed

[1] We consider small networks those comprised of dozens of nodes, while medium-sized networks contain hundreds of elements and large-scale ones can have more than a hundred thousand nodes.

Correlation Clustering (SRCC) problem (DOREIAN; MRVAR, 2009), which provides an alternative measure of the structural balance of a social network.

By applying the solution provided by these heuristic approaches, we conducted an analysis of the relative imbalance of the signed networks available in Stanford Large Network Dataset Collection (LESKOVEC; KREVL, 2014). As previously mentioned, structural balance theory affirms that human societies tend to avoid tension and conflicted relationships. In a signed graph that represents a real-world social network, this translates into a level of balance greater than expected, when compared to a random signed graph of equivalent size (FACCHETTI; IACONO; ALTAFINI, 2011). Partial hints that real-world, currently available signed social networks are more balanced than expected are provided by (KUNEGIS; LOMMATZSCH; BAUCKHAGE, 2009; LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010; KUNEGIS et al., 2010; FACCHETTI; IACONO; ALTAFINI, 2011). Our analysis confirms that three well-known signed social networks (WikiElections, Epinions and Slashdot) are indeed extremely balanced, hence supporting previous related works.

We also present a historical analysis of the results obtained from the voting on resolutions in the United Nations General Assembly (UNGA), this based on the solutions for the CC and SRCC problems. Other works have also applied different signed network clustering methods to similar networks of international alliances and disputes. For example, Traag and Bruggeman (TRAAG; BRUGGEMAN, 2009) analyze international relations taken from the Correlates of War (STINNETT et al., 2002) data set, and Macon et al. (MACON; MUCHA; PORTER, 2012) attempts to identify voting groups in UNGA annual sessions using three different network representations.

This text is organized as follows. Chapter 1 presents the Correlation Clustering problem, including a mathematical formulation and a literature review. Chapter 2 describes sequential and parallel GRASP metaheuristics to solve the CC problem, while Chapter 3 introduces sequential and parallel ILS algorithms that bring an improvement over GRASP. Experimental results of ILS as well as a comparison with other available solution approaches are also available in this chapter. An improved local search procedure for the CC problem, based on CUDA technology, is explained in Chapter 4. Next, Chapter 5 introduces the SRCC problem, its application and an efficient solution method. Chapter 6 presents an analysis of structural balance on large real-world social networks, based on the solutions obtained by using our methodology. Finally, Chapter 7 presents our conclusions.

1.1 The Correlation Clustering Problem

Correlation Clustering (BANSAL; BLUM; CHAWLA, 2002) is a clustering technique stemming from the problem of document clustering in which, when given a large corpus of

documents such as web pages, one wants to group them into the optimal number of clusters, without specifying that number in advance. The basic problem consists of minimizing the number of unrelated pairs that are clustered together, in addition to the number of related pairs that are separate. In this section, we formally describe the CC problem and present a mathematical formulation, to be accompanied by a literature review.

1.1.1 Mathematical Formulation

Let $G = (V, E)$ be an undirected signed graph[2] (Figure 1) where V is the set of n vertices and E is the set of edges. In this text, a signed graph is allowed to have parallel edges (i.e. two edges associated with a given pair of vertices) but no loops. Also, we assume that parallel edges always have opposite signs[3]. For a vertex set $S \subseteq V$, let $E[S] = \{(i, j) \in E \mid i, j \in S\}$ denote the subset of edges induced by S. For two vertex sets $S, W \subseteq V$, let $E[S : W] = \{(i, j) \in E \mid i \in S, j \in W\}$. One observes that, by definition, $E[S : S] = E[S]$. Consider a function $s : E \to \{+, -\}$ that assigns a sign to each edge in E. An undirected graph G together with a function s is called a signed graph, denoted by $G = (V, E, s)$. An edge $e \in E$ is called negative if $s(e) = -$ and positive if $s(e) = +$. Let E^- and E^+ denote, respectively, the set of negative and positive edges in a signed graph. The negative graph density is defined as $d^- = |E^-|/|E|$, while the positive graph density is $d^+ = |E^+|/|E|$.

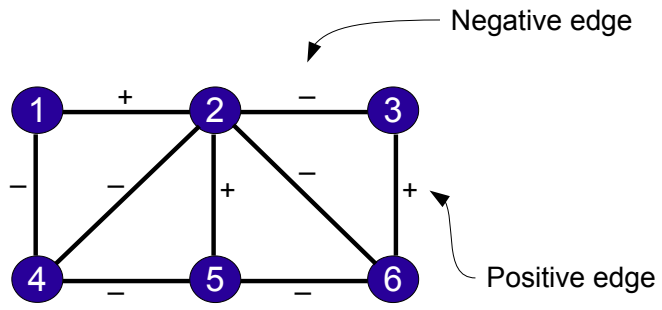


Figure 1 – Undirected signed graph.

A partition of V is a division of V into non-overlapping and non-empty subsets. Consider a partition $P = \{S_1, S_2, \ldots, S_l\}$ of V. The cut edges and the uncut edges related

[2] It is possible to solve the CC problem on directed graphs. One must first convert the directed graph to an undirected one: opposite arcs with the same sign are converted to a single edge whose weight is equal to the sum of the arcs' weights; opposite arcs with different signs become two parallel edges, each one with the original sign and weight of each arc.

[3] It is possible to use the same undirected graph to represent two possible relations. For example, in a given network, persons A and B may be evaluated by an external individual as collaborators (positive edge) or competitors (negative edge). In this case, if A and B are both seen as collaborators or competitors, there will be only one edge between A and B, representing a positive or negative behavior. On the other hand, if one behaves as collaborator and the other as competitor, there will be two parallel edges between them, one for each behavior, with opposite signs.

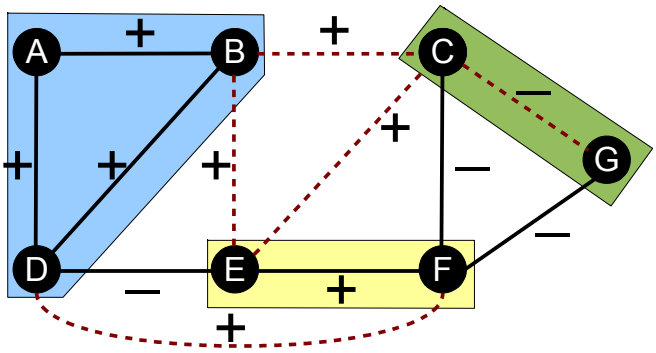
with this partition are defined, respectively, as the edges in sets $\cup_{1\leq i<j\leq l}E[S_i : S_j]$ and $\cup_{1\leq i\leq l}E[S_i]$. Let w_e be a nonnegative edge weight associated with edge $e \in E$. Also, for $1 \leq i, j \leq l$, let

$$\Omega^+(S_i, S_j) = \sum_{e\in E^+\cap E[S_i:S_j]} w_e \text{ and } \Omega^-(S_i, S_j) = \sum_{e\in E^-\cap E[S_i:S_j]} w_e.$$

The imbalance $I(P)$ of a partition P is defined as the total weight of negative uncut edges and positive cut edges, i.e.,

$$I(P) = \sum_{1\leq i\leq l} \Omega^-(S_i, S_i) + \sum_{1\leq i<j\leq l} \Omega^+(S_i, S_j). \tag{1.1}$$

Likewise, the balance $B(P)$ of a partition P can be defined as the total weight of positive uncut edges and negative cut edges. Clearly, $B(P) + I(P) = \sum_{e\in E} w_e$. Figure 2 presents an example of how to calculate the imbalance of an undirected signed graph. That being said, we are ready to provide a formal definition to the CC problem. Observe that this definition comprises the unweighted version of the problem.



$$I(P) = 1 + 4 = 5$$

Figure 2 – Example of imbalance I calculation for a given graph and partition P.

Problem 1.1.1 (CC problem) *Let $G = (V, E, s)$ be a signed graph and w_e be a nonnegative edge weight associated with each edge $e \in E$. The correlation clustering problem is the problem of finding a partition P of V such that the imbalance $I(P)$ is minimized or, equivalently, the balance $B(P)$ is maximized.*

The classical mathematical formulation for the CC problem is an integer linear programming (ILP) model proposed to uncapacitated clustering problems (which are also known as clique partitioning problems whenever the underlying graph is complete; see references in (MEHROTRA; TRICK, 1998)). In this formulation, a binary decision variable

x_{ij} is assigned to each pair of vertices $i, j \in V$, $i \neq j$, and defined as follows: $x_{ij} = 0$ if i and j are in a common set; $x_{ij} = 1$ otherwise. The model minimizes the total imbalance.

$$\text{minimize} \sum_{(i,j)\in E^-} w_{ij}(1 - x_{ij}) + \sum_{(i,j)\in E^+} w_{ij}x_{ij} \tag{1.2}$$

$$\text{subject to } x_{ip} + x_{pj} \geq x_{ij}, \qquad \forall\, i, p, j \in V, \tag{1.3}$$

$$x_{ij} = x_{ji}, \qquad \forall\, i, j \in V, \tag{1.4}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall\, i, j \in V. \tag{1.5}$$

The triangle inequalities (1.3) say that if i and p are in a same cluster as well as p and j, then vertices i and j are also in a same cluster. Constraint (1.4) written to $i, j \in V$ establishes that variables x_{ij} and x_{ji} assume always the same value in this formulation. Constraints (1.5) impose binary restrictions to the variables while the objective function (1.2) minimizes the total imbalance defined by equation (1.1). Even though this formulation is polynomial-sized, having $n(n-1)$ variables and n^3+n^2 constraints, notice that, according to constraints (1.4), half of the variables can be eliminated, which reduces both the number of variables and constraints of the formulation.

A set partitioning formulation (MEHROTRA; TRICK, 1998), proposed in the literature to uncapacitated clustering problems, could also be used in the solution of the CC problem. As we can expect, these two formulations are not appropriate solution approaches when time limit is a constraint in the solution process. The authors in (FIGUEIREDO; MOURA, 2013) report that the classical formulation begins to fail (time limit set to 1h) with random instances of 40 vertices and negative density (d^-) equal to 0.5.

1.1.2 Literature review

To the best of our knowledge, the CC problem, as defined in the previous section, was addressed for the first time in (DOREIAN; MRVAR, 1996a) (not under this name) where its heuristic solution was used as criteria for analyzing structural balance in social networks. The heuristic approach proposed by the authors is a simple greedy neighborhood search procedure that assumes a prior knowledge of the number of clusters in the solution. This heuristic is implemented in software Pajek (BATAGELJ; MRVAR, 2008). Motivated by the solution of a document clustering problem, Bansal et al. (BANSAL; BLUM; CHAWLA, 2002) formalized the unweighted version of the CC problem and also discussed its NP-completeness proof, whereas the weighted version of the problem was addressed in Demaine et al. (DEMAINE et al., 2006). Integer linear programming (ILP) can be used to solve the CC problem optimally, but only when the number of data points is small. Since it consists of a NP-hard minimization problem, the only available solutions for large instances are either heuristic or approximate. Predominately investigated from the viewpoint of constant factor approximation algorithms (BANSAL; BLUM; CHAWLA,

2002; SWAMY, 2004; CHARIKARA; GURUSWAMIB; WIRTHA, 2005; DEMAINE et al., 2006; GIOTIS; GURUSWAMI, 2006; AILON; CHARIKAR; NEWMAN, 2008), the problem has been applied in the solution of many applications, including portfolio analysis in risk management (HARARY; LIM; WUNSCH, 2003; HUFFNER; BETZLER; NIEDERMEIER, 2010), biological systems (DASGUPTA et al., 2007; HUFFNER; BETZLER; NIEDERMEIER, 2010), grouping of genes (BHATTACHARYA; DE, 2008), efficient document classification (BANSAL; BLUM; CHAWLA, 2002), detection of embedded matrix structures (GÜLPINAR et al., 2004) and community structure (TRAAG; BRUGGEMAN, 2009; MACON; MUCHA; PORTER, 2012). Additionally, some authors have focused on the detection of overlapping communities in signed graphs (ZHANG; WANG; ZHANG, 2007; BONCHI; GIONIS; UKKONEN, 2011). In particular, Bonchi et al. (BONCHI; GIONIS; UKKONEN, 2011) define an optimization problem that extends the framework of Correlation Clustering to allow overlaps.

A comparison of several heuristic strategies (greedy and local search methods) for the problem is presented in (ELSNER; SCHUDY, 2009) and applied to document clustering[4] and natural language processing (instances of $n = 1000$), to which ILP does not scale. In this context, the authors' recommended strategy for solving the CC Problem is a greedy algorithm called $VOTE/BOEM$, which can quickly achieve good objective values with tight bounds.

In Yang et al. (YANG; CHEUNG; LIU, 2007), the CC problem is known as community mining and an agent-based heuristic called FEC is proposed to its solution. In order to assess the performance of FEC, the authors present a method for generating random signed networks with controlled community structures, based on a set of community structure-conscious parameters. However, they only test their algorithm results against the Doreian Mrvar (DM) algorithm (DOREIAN; MRVAR, 1996b), based on the clusters listed in each solution, and without presenting a numerical measure of imbalance. Also, according to the authors, the success of the FEC algorithm is based on its capacity of extracting the communities initially defined by the aforementioned generation routine.

An approach based on genetic algorithms has been proposed in Zhang et al. (ZHANG et al., 2008) for the CC problem and applied to document clustering, although unfortunately, there is no explanation about how the genetic operators are applied and we were unable to obtain the program code or recreate the instances used in the experiments, making it difficult to understand and reproduce the proposed algorithm. As far as we are aware, before the development of this work, Zhang et al. (ZHANG et al., 2008) presented the only metaheuristic approach applied to the CC problem.

[4] The original data set from UCI Machine Learning Repository (BACHE; LICHMAN, 2013) consists of 20000 messages taken from 20 newsgroups. However, since the authors' bounding technique did not scale to the full dataset, they restricted their testbed to a subsample of 100 messages from each newsgroup, for a total of 2000 messages.

Finally, it is important to note that the term Correlation Clustering is also used to refer to a different correlation clustering problem, in which multidimensional data are analyzed aiming to spot clusters of objects in subspaces of the multidimensional space (KRIEGEL; KRÖGER; ZIMEK, 2009).

2 Solving the Correlation Clustering Problem by GRASP

A metaheuristic is a higher-level procedure or heuristic designed to find, generate or select a lower-level procedure or heuristic (partial search algorithm), in order to sample a set of solutions which is too large to be completely examined. Metaheuristics have been used successfully for solving hard combinatorial optimization problems as they can provide sub-optimal solutions in a reasonable time.

GRASP is a multi-start metaheuristic in which each iteration consists basically of two phases: construction and local search (FEO; RESENDE, 1995). The solutions generated by a GRASP construction procedure are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution.

To our knowledge, this is the first time the GRASP metaheuristic has been implemented to solve the CC Problem. First, it was developed as a sequential program. Then, in order to obtain performance improvements that would allow the program to scale to large network instances, we extended the algorithms to run tasks in parallel, using two well-known strategies.

2.1 Greedy Randomized Adaptive Search Procedure (GRASP) for the CC problem

We have designed and implemented a GRASP method which has been applied in many combinatorial optimization problems. The proposed heuristic (*GraspCC*) is based on a GRASP method for the maximum modularity problem (NASCIMENTO; PITSOULIS, 2013), which produced superior and robust (with respect to solution quality) solutions, better than other heuristics from the literature on the majority of the its cases. First, we have adapted the construction phase proposed by Nascimento et al. (NASCIMENTO; PITSOULIS, 2013), replacing the modularity gain function (used on unsigned graphs) by an imbalance (minimization) gain function, which makes more sense for signed networks. The second step was to improve the local search procedure, replacing the simple neighborhood traversal with a Variable Neighborhood Descent heuristic.

Our *GraspCC* is described in Algorithm 1. The parameter *iter* denotes the maximum number of iterations without improvement in the best solution found. The first task in each iteration of *GraspCC* is the construction of an initial solution in a greedy

randomized fashion. This task is performed in *ConstructivePhase* procedure described in Algorithm 2. In order to describe it, we need some additional notation.

Let $P = \{S_1, S_2, ..., S_l\}$ denote a *partial partition* (i.e., a partition of a proper subset of V). We define below a *function* $g : (V \setminus \bigcup_{1 \leq k \leq l} S_k) \rightarrow \mathbb{R}$, which will measure the impact on imbalance I of inserting a vertex i in the partial partition P.

$$g(i) = \min \left(I(P \cup \{i\}), \min_{\substack{1 \leq k \leq l \\ S_k \cup \{i\}}} I(P) \right). \tag{2.1}$$

This minimization function compares the cost of inserting vertex i in a new cluster ($g(i) = I(P \cup \{i\})$) with the cost of inserting it into one of the l clusters in P. Note that a vertex with a low cost function value will probably generate less imbalance if we add it to the partial partition P.

In this phase, the ordered set L_g is defined (line 3) as the set of vertices $V \setminus \bigcup_{1 \leq k \leq l} S_k$ ordered in decreasing order of function g (Figure 3). At each iteration, in lines 4-8, we randomly choose a vertex i among the first $\lfloor \alpha . \mid L_g \mid \rfloor$ vertices in this set and add it to the partial partition. Note that parameter α denotes the degree of randomness the construction phase will have. This process is repeated until a partition of V is obtained.

Algorithm 1: *GraspCC*

```
1  Input: G = (V, E) and α
2  Output: partition P*
3  P* = ∅;  I(P*) = ∞;  i = 0;
4  while (i ⩽ iter)
5     P = ConstructivePhase(G, α);
6     P = VariableNeighborhoodDescent(P, G);
7     if (I(P) < I(P*))
8        P* = P;  i = 0;
9     end if
10    i = i + 1;
11 end while
12 return P*;
```

There is no guarantee that the construction method returns a locally optimal solution with respect to some neighborhood. Therefore, the solution P obtained in *ConstructivePhase* could be improved by the local search procedure *VariableNeighborhoodDescent* (Algorithm 3).

The Variable Neighborhood Search (VNS) method, proposed by Mladenović and Hansen (MLADENOVIĆ; HANSEN, 1997), is a metaheuristic that explores distant neighborhoods of the current incumbent solution, and moves to a new solution if and only if an improvement is made. The main idea is to systematically explore differing neighborhood

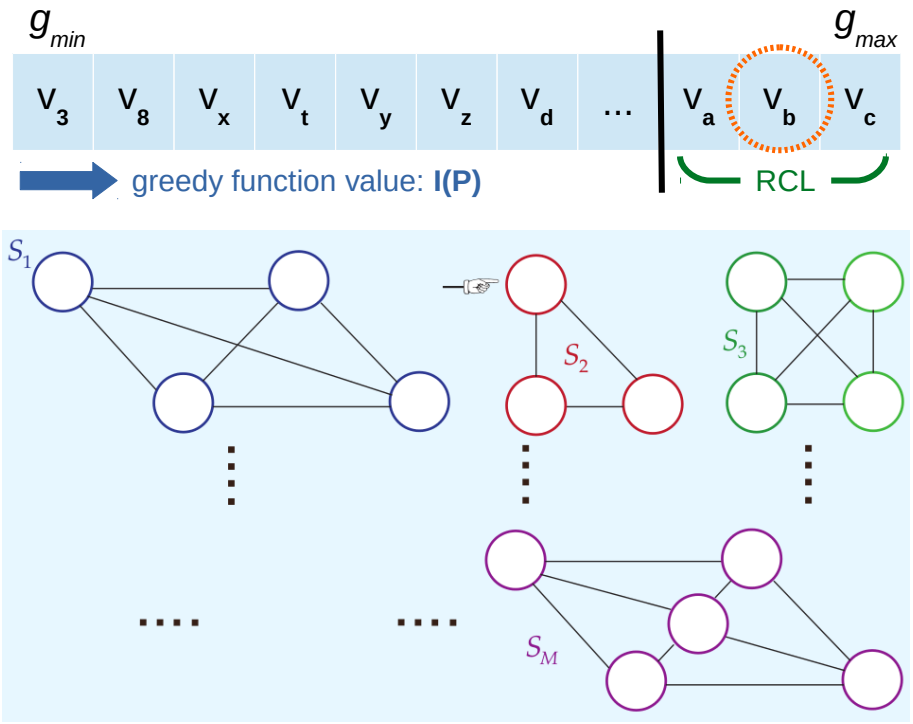


Figure 3 – Constructive phase.

structures, with the goal of escaping from local minima. Within this work, we apply the Variable Neighborhood Descent (VND) search, a variant of VNS. As illustrated in Figure 4, VND starts with the partition provided by the construction phase and iteratively compares the incumbent value $I(P)$ with the new value $I(\overline{P})$ obtained in the $r - neighborhood$, denoted by $N_r(P)$ and defined as the family of all partitions obtained by moving r vertices in P from one cluster into another. Figure 5 illustrates the neighborhood structures used in this work ($N_1(P)$, or 1-opt, and $N_2(P)$, also known as 2-opt[1](GENDREAU; POTVIN, 2010)), while Figure 6 examplifies a sequence of movements of the $N_1(P)$ neighborhood. If an improvement is obtained, r is returned to its initial value and the new incumbent updated (lines 7 and 8 in Algorithm 3). Otherwise, the next neighborhood is considered (line 12). The local search halts when no better partition is found in the most distant neighborhood of the current solution. Note that the neighborhood analyzes partitions with different number of clusters (i.e. a vertex can be moved into a new cluster or can be removed from a single vertex cluster).

The use of simpler neighborhoods (moving vertices between clusters) is better suited for the solution of large-scale real-world social networks, the focus of this work. For this reason, we dicarded any complex neighnorhood structure or more advanced processing

[1] A 1-optimal (or 1-opt) is a move which changes the value of one variable at a time, while a 2-optimal (2-opt) neighborhood is defined as a move which simply exchanges two assignments of the current solution(GENDREAU; POTVIN, 2010). In the CC problem, for example, a 2-opt move means moving 2 vertices in P from one cluster into another.

such as path relinking (LAGUNA; MARTI, 1999; RESENDE; RIBEIRO, 2005). Another strategy to optimize local search and save time was employing a first improvement (first descent) heuristic, which means that local search will stop whenever it finds an improvement of the current solution. This avoids the complete traversal of the r-neighborhood, which is very time-consuming.

Algorithm 2: *ConstructivePhase*

```
1  Input: G = (V, E) and α
2  Output: partition P
3  P = ∅; L_f = Order(V);
4  while (L_f ≠ ∅)
5     Choose vertex i randomly among the first ⌊α. | L_f |⌋ elements of L_f;
6     Update S_k = S_k ∪ {i} where k is the component in P that minimizes f;
7     L_f = L_f − {i};  Re-order(L_f);
8  end while
9  return P;
```

Algorithm 3: *VariableNeighborhoodDescent*

```
1  Input: G = (V, E) and a partition P
2  Output: partition P
3  r = 1;
4  while (r ≤ 2)
5    for all P̄ ∈ (N_r(P))
6      if (I(P̄) < I(P))
7        r = 1;
8        P = P̄;
9      end if
10   end for
11   if (P has not improved)
12     r = r + 1;
13   end if
14 end while
15 return P;
```

2.2 Parallel strategies to improve performance

In practice, interesting real-world optimization problems are often NP-hard, complex, and time consuming. Although the utilization of a sequential metaheuristic provides significant time reduction in the search process, execution time remains high for real-world problems arising in both academic and industrial domains. For example, when it comes to signed social networks, the instances generated from Wikipedia, Slashdot and Epinions websites, available in (LESKOVEC; KREVL, 2014), have thousands of nodes and in some

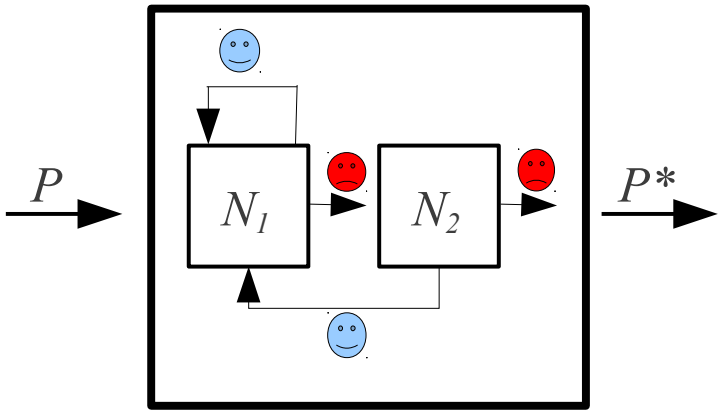


Figure 4 – Variable Neighborhood Descent procedure. N_1 stands for 1-opt neighborhood; N_2 stands for 2-opt neighborhood.

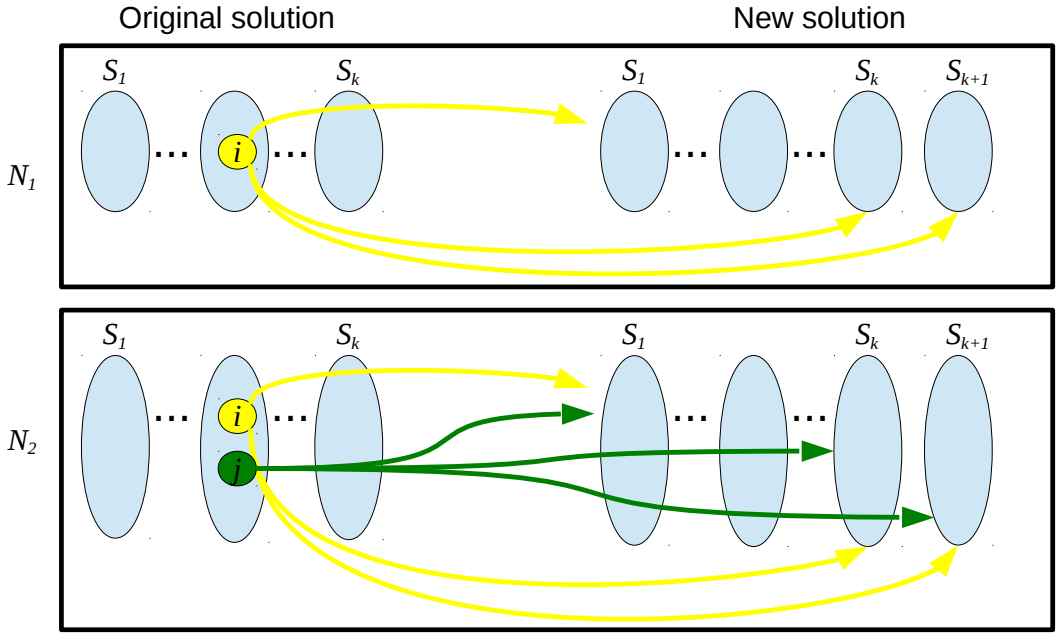


Figure 5 – Local search neighborhood structures used by GRASP. N_1 stands for 1-opt neighborhood; N_2 stands for 2-opt neighborhood.

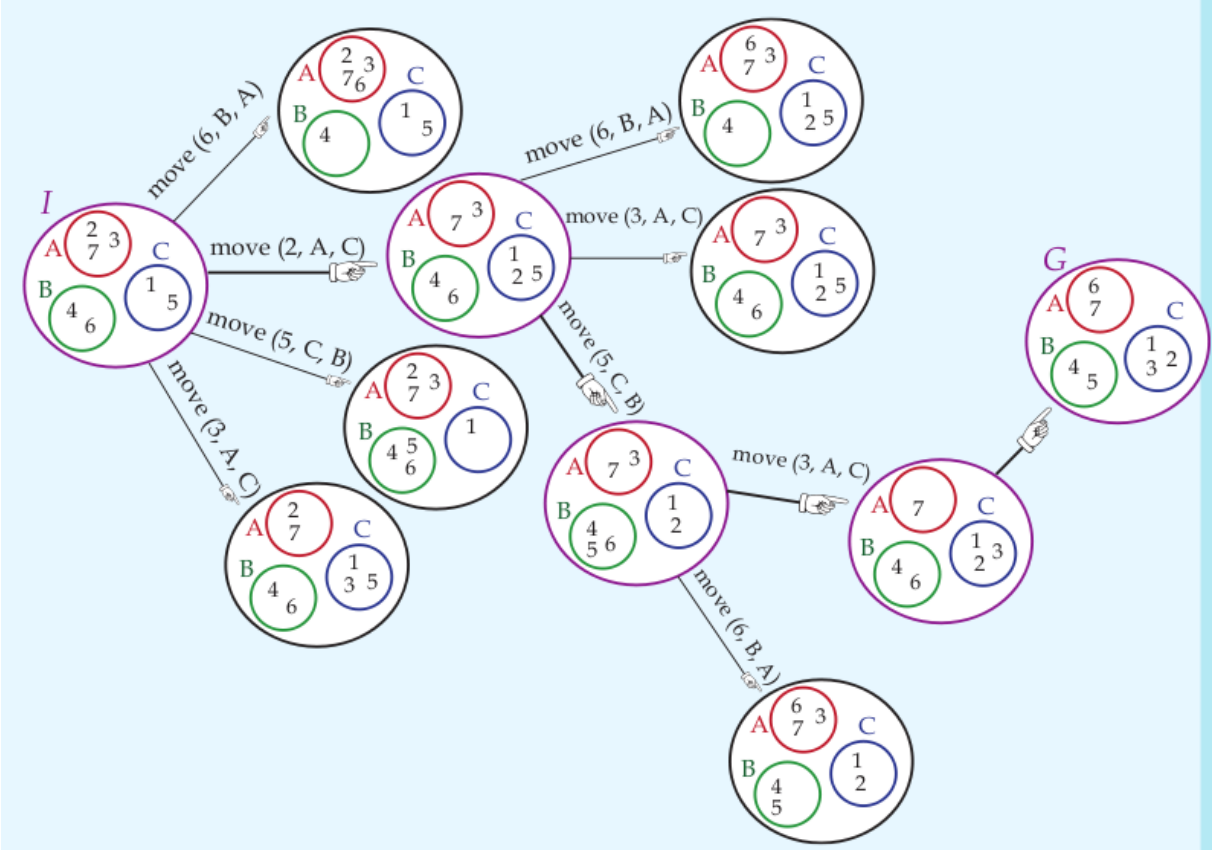


Figure 6 – Example of 1-opt (or N_1) neighborhood movements (FRINHANI et al., 2011).

cases almost a million relationships[2]. Therefore, parallelism presents as a natural way not to only reduce the search time, but also to improve the quality of the provided solutions.

2.2.1 Independent Parallel GRASP Algorithm

Our first strategy was to employ the well-known independent approach in the parallel program (GENDREAU; POTVIN, 2010). When p processors are used, a single (master) process reads the problem data and passes it to the remaining $p-1$ processes. The number of multistart iterations is then divided among the set of p processes. Each process executes a copy of the GRASP program and global search terminates once each individual search stops, i.e., when the maximum number of allotted iterations is completed.

Since we applied message passing for communication among processes, this scheme limits information exchange between processes only for problem input, detection of process termination and determination of best overall solution. It is also worth noting that various random seeds were used for each process in the construction phase (line 5 in Algorithm 2) to favor the exploration of different search spaces by the independent programs.

[2] Wikipedia adminship election data has 7,000 vertices and 100,000 edges, Epinions signed social network has 131,828 vertices and 841,372 edges and Slashdot Zoo signed social network (from February 21 2009) is comprised of 82,144 vertices and 549,202 edges.

2.2.2 Parallel Variable Neighborhood Descent (PVND) on MPI

Despite having several GRASP processes in parallel, which divides the number of alloted iterations, the previous parallelization strategy has an efficiency limit. Since our GRASP's main stopping criterion is the number of multistart iterations **without improvement**, increasing the number of simultaneous processes in order to reduce the quantity of iterations presents good results up to a certain level. Moreover, the previous algorithm still performs local search sequentially. When processing large network files, this stage proves to be very time-consuming, as can be seen in Figure 7.

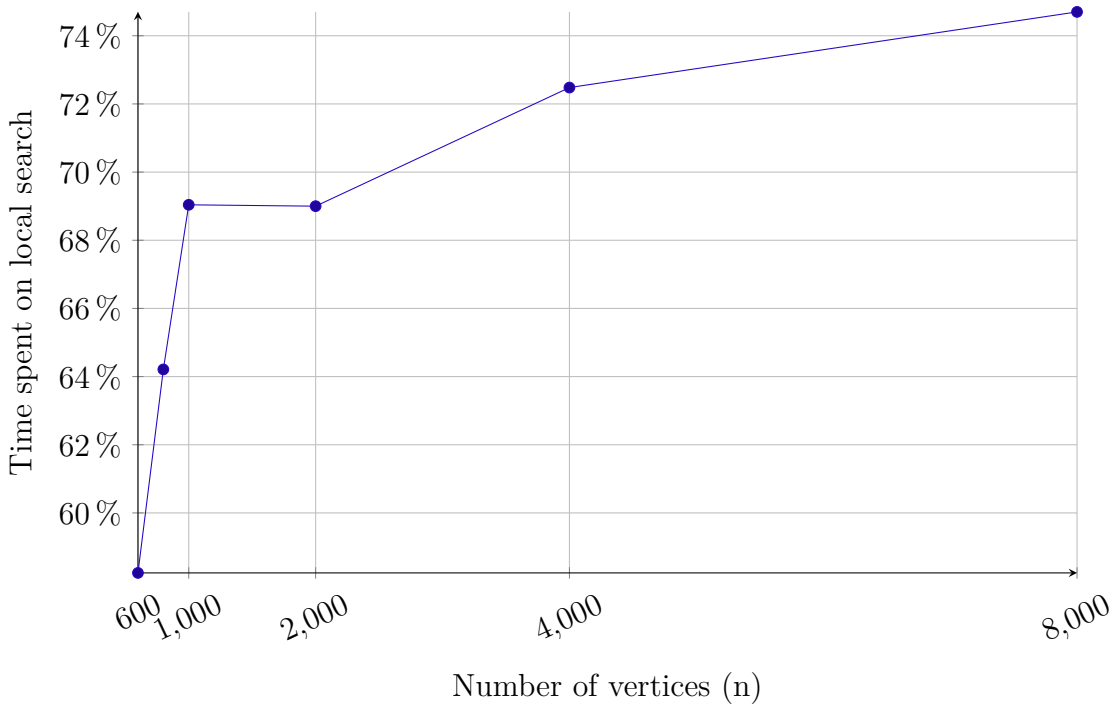


Figure 7 – Time spent on sequential local search, processed by sequential GRASP ($\alpha = 1.0$, r 1-opt, $iter = 400$) on Slashdot social network instances.

Another strategy for parallel implementation that, besides reducing the computation time, can also increase the exploration in the search space, consists of partitioning the search neighborhood and assigning each partition to a process running on a separate CPU (ALBA, 2005; CRAINIC; TOULOUSE, 2010). With this idea in mind, an interesting approach is extending the parallel metaheuristics to use the Parallel Variable Neighborhood Descent (PVND) (EKŞIOGLU; PARDALOS; RESENDE, 2002) algorithm inside its local search phase.

In this second approach of parallelizing the program, besides having a set of processes responsible for executing a copy of GRASP, there is an alternate process known as search slave, which is specialized in executing the local search (VND) algorithm. The only difference from the VND procedure, as previously explained, is the neighborhood traversal (line 5 in Algorithm 3), which is now executed in parallel, attempting to obtain a

balanced load among the processors. Therefore the search space of the $r - neighborhood$ of the current solution ($N_r(P)$) is divided among the available search slave processes in order to find the combination that minimizes the imbalance value $I(P)$. In the parallel search of the $N_1(P)$ neighborhood, for instance, each process will be in charge of moving a subset of vertices to another cluster, one vertex at a time. Search slave processes send a message to an GRASP master process when they either stop upon finding a better solution or complete their walk through the provided search space (in case no improved solution has been found). Finally, the GRASP master process will be in charge of gathering each search result produced in parallel, choosing the best one.

Search slave processes execute lines 5-12 in Algorithm 4, sending a message to a GRASP master process when they either stop upon finding a better solution or complete their walk through the provided search space. Finally, the GRASP master process will be in charge of gathering each search result produced in parallel, choosing the best one (lines 13-16).

Figure 8 exemplifies how Parallel GRASP with PVND traverses the search space. Suppose the algorithm runs with a total of 16 processes, out of which 4 are GRASP master processes (with ranks[3] 1, 5, 9 and 13), each one with a different random seed. For this reason, it is highly probable that each GRASP master will commence with a considerably different initial solution. Additionally, every process whose rank is different from the previous four is a Parallel VND search slave process. When Parallel VND is started, each GRASP master process splits the neighborhood of its current solution into four parts, dividing the neighborhood traversal between its three PVND search slaves and itself, resulting in four search tasks running in parallel. Division of the previously defined neighborhood $N_r(P)$ is done in a very simple way: each process is responsible for a specific range of clusters from which a vertex can be moved. A sample process allocation scheme is detailed in Figure 9.

2.3 Experiments and performance comparisons

After implementing the algorithms presented in the previous section, we have conducted extensive experiments in order to assess its performance against different sets of signed graph instances. The solutions obtained by our metaheuristics were then compared to results obtained from other solution techniques available in the literature. The following methods were implemented for comparison:

- **Integer Linear Programming (ILP) model**: can provide an exact solution to the CC problem, but is unable to process larger instances. Besides using the exact

[3] A rank is a unique process identifier, an integer r in the range $0 \leq r \leq p-1$, where p is the number of available processes.

Algorithm 4: *ParallelLocalSearch*

Input: $G = (V, E)$, partition P, neighborhood size r and number of search slave processes y
Output: partition P
Let $N_r^y(P) = N_r(P)$ equally divided in y pieces;
for each slave process y, send a message to execute the following block:
begin parallel block
  $P_y = P$
  for all $\overline{P} \in (N_r^y(P))$
    if $I(\overline{P}) < I(P_y)))$
      $P_y = \overline{P}$;
  end for
  Send P_y back to master process;
end parallel block
for each process y, receive a message containing P_y:
  if $(I(P_y) < I(P))$
    $P = P_y$;
return P;

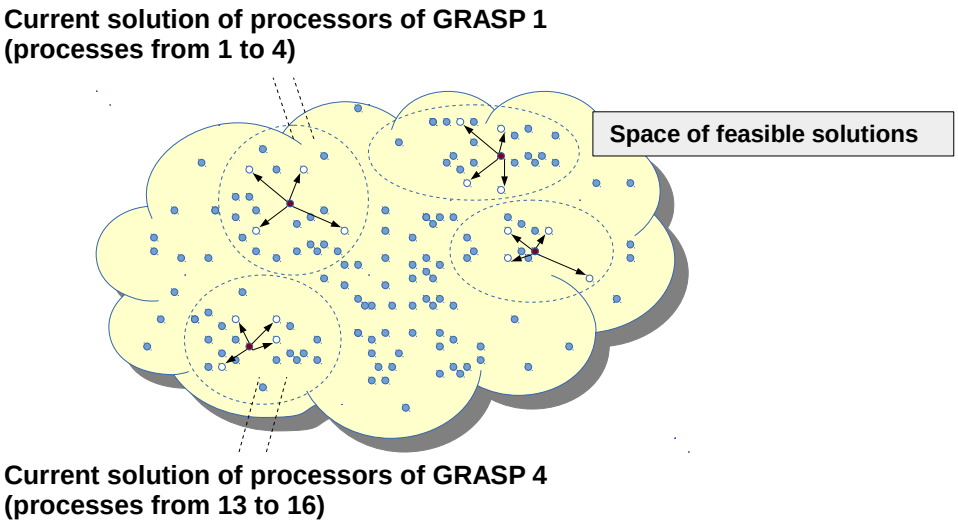


Figure 8 – Sample Parallel GRASP with PVND search space traversal

solution provided by the ILP model, we identify for which instance size the ILP formulation becomes very big and the solver is not able to return a good solution within the time limit. In these cases, the use of heuristics is necessary.

- **Doreian Mrvar Method, implemented in Pajek software**[4] (BATAGELJ; MRVAR, 2008): consists of a relocation algorithm for partitioning signed graphs. In

[4] The Pajek program is for analysis and visualization of large networks, which provides features such as cluster identification, decomposition of large networks, visualization tools, as well as an implementation of several efficient algorithms for analysis of large networks, having thousands or even millions of vertices.

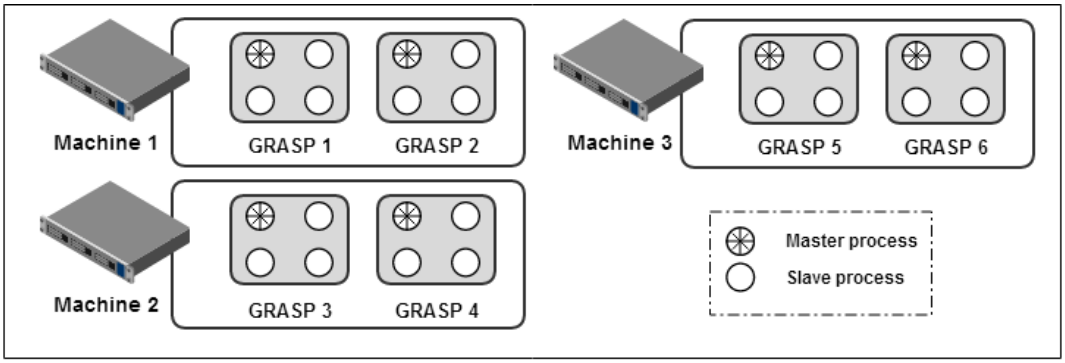


Figure 9 – GRASP with PVND machine-process deployment example. In this case, there are 6 GRASP master processes and 18 VND search slave processes. Since each machine has 8 processor cores, it will be able to host 8 processes at a time, for example GRASP 1 and GRASP 2 process groups.

simple terms, a given partition (randomly generated at each iteration) is optimized to get as much as possible positive edges inside clusters and negative edges between clusters. In the local optimization procedure, vertices can be moved from one cluster to another or pairs of vertices can be interchanged between clusters. For each such change, the criterion function is evaluated. If a relocating change leads to a decrease of the criterion function, the new partition is retained and the process continues until the criterion function cannot be lowered. If the procedure finds several optimal solutions, all of them are reported. More details are available in Doreian and Mrvar (DOREIAN; MRVAR, 1996b);

- **VOTE/BOEM heuristic** (ELSNER; SCHUDY, 2009): provides good correlation clustering solutions on datasets far too large for ILP to scale. The algorithm is composed of two phases: (1) the VOTE algorithm adds each vertex to the cluster that minimizes the correlation clustering objective; and (2) the best one element move (BOEM) algorithm repeatedly makes the most profitable and the best element move until a local optimum is reached.

As previously metioned, although an approach based on genetic algorithms has been proposed in Zhang et al. (ZHANG et al., 2008) for the CC problem and applied to document clustering, unfortunately, we were unable to implement the procedure and compare it to our algorithms. There was no explanation as to how the genetic operators were applied and we were also unable to obtain or recreate the instances used in the experiments.

Moreover, we were also unable to directly compare our results with the solutions of the FEC algorithm in Yang et al. (YANG; CHEUNG; LIU, 2007), even though we had generated and used, in our experiments, the random signed networks with controlled

community structures, as presented in their work. We verified that, since the FEC heuristic is focused on community detection, using a different criterion than minimizing imbalance, FEC does not present optimal (or close to optimal) solutions for the CC problem. In fact, when using the aforementioned random instances as reference, the imbalance measures of the best clustering configurations pointed by FEC are higher than the solutions generated by our algorithms.

We next present extensive computational results obtained with three benchmarks.

2.3.1 Computational environment

The algorithms described in the previous section were implemented in ANSI C++ and MPI (GROPP; LUSK; SKJELLUM, 1999) (OpenMPI) for message passing. All experiments were performed (with exclusive access) on a cluster with 42 nodes, each one with two Intel Xeon QuadCore 2.66GHz processors and 16Gb of RAM under Linux (Red Hat 5.3). The ILP formulation presented in this section is coded in Xpress Mosel 3.2.0 with solver Xpress Optimizer 21.01.00. The CPU time limit is set to 2 hours for the ILP formulation. Additionally, all heuristic outcomes represent the average of 25 independent runs and all confidence intervals (CI) were obtained through Student's t-test at a confidence level of 95%.

2.3.2 Test problems

Computational experiments were undertaken on (i) a set of 32 social networks from the literature, (ii) a set of 63 network instances that represent the United Nations General Assembly (UNGA) voting data and (iii) a set of 60 completely random instances. We will briefly describe these instances[5].

(i) This set of instances is composed by 22 small-sized instances normally used in block-modeling approaches to structural balance (BRUSCO, 2003; DOREIAN; MRVAR, 2009; FIGUEIREDO; MOURA, 2013) and 10 signed networks (with n varying from 200 to $10,000$ vertices) extracted from the large scale social network representing the technology-related news website Slashdot[6](LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010; FACCHETTI; IACONO; ALTAFINI, 2011).

(ii) We generated 63 medium-sized social networks based on UNGA voting records of

[5] All instances are available in <http://www.ic.uff.br/~yuri/files/CCinst.zip>.

[6] We have extracted subsets of Slashdot Zoo signed social network from February 21 2009 (LESKOVEC; KREVL, 2014), containing the first n vertices, where $200 \leq n \leq 10000$. Since the original Slashdot network is a signed digraph, before extracting the subsets, we have converted it into an undirected graph.

the separate annual sessions between 1946 and 2008[7]. These instances are weighted versions of UNGA signed graphs described in (FIGUEIREDO; FROTA, 2014). The set of vertices in each signed graph represents the set of countries in the associate annual voting session. The set of weighted positive/negative edges is defined as follows. For each pair of vertices (countries) i, j and for each resolution voted in the session, we totaled the weights associated with all pairs of votes from i and j: edge weights can be equal to 1.0 or 0.5; a positive edge means an agreement, while a negative edge represents a disagreement. Following an observation from Macon et al. (MACON; MUCHA; PORTER, 2012), we treat differently the disagreement (agreement) in a yes-no (yes-yes or no-no) pair of votes on a same resolution from a yes-abstain or no-abstain (abstain-abstain) pair. We normalize by the total number of votes in a session.

(iii) We generated random social networks with $n \in \{100, 200, 300, 400, 600\}$, varying network density $d = 2 \times |E|/(n^2 - n)$ and negative graph density defined here as $d^- = |E^-|/|E|$. For each value of n, we considered a set of 12 random instances having d and d^- ranging, respectively, in sets $\{0.1, 0.2, 0.5, 0.8\}$ and $\{0.2, 0.5, 0.8\}$. The random selection of graph edges follows a uniform distribution. Random networks are generally hard-to-solve instances, for not having a well-defined cluster structure. Moreover, since real-world signed networks are generally sparse graphs, random networks are important to assess the performance of the algorithm as the the graph's edge density grows. The largest random graph used in the experiments has such a size that the ILP solver would be unable to find a feasible solution within the specified time limit.

2.3.3 Methodology

We incorporated the instances in (i) and (ii) to parametrize the heuristics. Several tests were conducted with GRASP parameters varying according to the following range:

- Number of iterations without improvement ($iter$): 100, 200, 300, 400;
- $ConstructivePhase$ randomness factor (α): 0, 0.4, 0.8, 1.0;
- Neighborhood size (r): $r = 1$ and $r \leq 2$.

A high value for the α parameter results in the generation of more diverse initial solutions (highly random), while a low value like $\alpha = 0.4$ consists of a more greedy approach (more importance is given to the imbalance gain function).

[7] United Nations General Assembly Voting Data, by Anton Strezhnev and Erik Voeten, <http://hdl.handle.net/1902.1/12379>. Accessed in May 2014.

In order to apply the GRASP algorithm, the following configuration presented the best results:

SetPar	**Graph size**	Time limit	Alpha	Neighborhood	Number of iterations without improvement
A	$n < 200$	1 hour	$\alpha = 0.8$	$r \leq 2$	$iter = 400$
B	$n \geq 200$	2 hours	$\alpha = 1.0$	$r = 1$	$iter = 400$

When testing the independent parallel version of GRASP, if p is the number of simultaneous processes, the number of iterations ($iter$ parameter) is reduced to $iter/p$. This is due to the fact that the parallel procedure executes p independent metaheuristic procedures at the same time, which provides enough variability so that the aggregated results of parallel GRASP are quality-equivalent to those of the sequential GRASP.

Whenever it is required to assess the performance of parallel algorithms, two metrics are applied: speed-up $Su(p)$ measures the acceleration observed for the parallel algorithm when compared with its sequential version and efficiency $E(p)$ measures the average fraction of time along which each process is effectively used. Thus, $Su(p) = T(seq)/T(p)$, such that $T(seq)$ is the time required for the sequential algorithm and $T(p)$ the time required for the parallel algorithm run on p processors, and $E(p) = Su(p)/p$.

2.3.4 Exact solution by ILP formulation

As noted in the literature (MEHROTRA; TRICK, 1998), the linear relaxation of the ILP formulation described in Chapter 1 provides a very good representation of the problem which allows discovering the optimal solution for many instances by solving this linear relaxation. Experiments reported in Figueiredo and Moura (FIGUEIREDO; MOURA, 2013) confirm this assertion: the 22 small instances in set (i) were solved to optimality in some seconds. Also, in our experiments, the instances in set (ii) were solved to optimality by the ILP formulation in the root of the branch and bound tree. The results obtained on some of these instances with the ILP formulation are reported in Table 1. We can conclude that these signed networks are almost perfectly balanced with most part of the imbalance given by negative relations.

The drawback of the ILP approach appears when we attempt to solve larger instances. For most completely random instances in (iii), the solver is unable to find an optimal solution within the time limit (Table 2). On Slashdot-based instances, the ILP formulation becomes too big and the solver is unable to return any solution within the prescribed time limit.

2.3.5 Sequential GRASP

We have first compared the results obtained with Xpress and the sequential GRASP on UNGA (ii) and random (iii) instances, respectively. As displayed in Table 3, the GRASP

	Instance					Optimal Solution				Times (s)	
Session	n	$\|E^-\|$	$\|E^+\|$	$w(E^-)$	$w(E^+)$	k	$\%I$	$\%I^-$	$\%I^+$	$T(Xpress)$	$T(seq)$
2003	191	1119	16636	247.33	8247.60	2	0.09	0.47	0.08	281	51
2004	191	945	17121	315.15	8646.75	2	0.23	2.74	0.14	301	47
2005	192	1120	16566	251.28	8271.56	3	0.47	3.03	0.40	338	48
2006	192	886	17378	248.60	8873.92	2	0.32	1.59	0.28	223	49
2007	192	1109	17148	260.26	8646.04	2	0.51	4.18	0.40	262	59
2008	192	1055	17091	256.66	8854.55	2	0.40	5.05	0.27	178	56

Table 1 – Results obtained on UNGA instances with Xpress (ILP formulation) and the sequential GRASP algorithm. Number of vertices (n); number of negative ($|E^-|$) and positive ($|E^+|$) edges; sum of negative ($w(E^-)$) and positive ($w(E^+)$) weights; number of clusters (k) in the optimal solution $\bar{P}$; percentage imbalance ($\%I = 100 \times IP(\bar{P})/w(E)$); negative percentage imbalance ($\%I^- = 100 \times IP^-(\bar{P})/w(E^-)$) where $IP^-(\bar{P})$ is the sum of weights of negative edges in the optimal solution; positive percentage imbalance ($\%I^+ = 100 \times IP^+(\bar{P})/w(E^+)$) where $IP^+(\bar{P})$ is the sum of weights of positive edges in the optimal solution; and time in seconds spent by the Xpress software to solve the ILP formulation ($T(Xpress)$) and by the sequential GRASP procedure $SeqGRASP$ ($T(seq)$). $T(seq)$ is the average value after 25 executions of the heuristic.

program was able to solve the instances in (ii) faster than Xpress, with similar solution quality, and, for the instances in (iii), Xpress was not able to find an optimal solution to most networks of size bigger than 100 vertices within the 2-hour time limit (Table 2).

2.3.6 Comparison with existing heuristics

Our goal in the next subsections is to verify the limit of Doreian Mrvar and VOTE/BOEM heuristics. Notice that these heuristics are simplified versions of a local search heuristic, so they tend to have inferior performance when compared to metaheuristics. In order to run the tests, we chose to slice a huge real-world network into smaller sizes (Slashdot-based instances in set (i)).

2.3.6.1 Sequential GRASP *vs.* Doreian Mrvar Method

In this section we compared our GRASP procedure with Doreian Mrvar Method to solve some of the most challenging instances available in the related literature: the Slashdot instances in set (i).

The Doreian Mrvar Method (implemented in Pajek software) presents good-quality results (with respect to solution value and average time spent) for the small-sized literature instances in (i) and also for random social networks of up to 50 vertices (the results are available in Figueiredo and Moura (FIGUEIREDO; MOURA, 2013)). However, for larger instances, it presents poor performance.

For this benchmark, we have obtained average results of 25 executions. The software

				ILP		SeqGRASP					
n	**e**	**d**	**d-**	**BestSol**	**Time**	**Avg I(P)**	**CI[I(P)]**	**Gap %I(P)**	**Avg Time**	**CI(Time)**	**Gap Time**
100	990	0.1	0.2	198	71.49	198.00	0.00	0.00%	1.25	0.11	-70.24
			0.5	**292**	1,339.70	238.40	1.52	-18.36%	3.85	0.64	-1,335.85
			0.8	50	308.74	72.24	1.12	44.48%	6.06	0.76	-302.68
	1980	0.2	0.2	396	82.50	396.00	0.00	0.00%	1.19	0.09	-81.31
			0.5	**780**	933.03	586.08	1.27	-24.86%	5.73	0.62	-927.30
			0.8	**272**	709.02	226.24	1.16	-16.82%	9.49	1.06	-699.53
	4950	0.5	0.2	990	60.42	990.00	0.00	0.00%	1.27	0.17	-59.15
			0.5	**2234**	1,267.70	1,842.32	2.90	-17.53%	9.84	1.02	-1,257.86
			0.8	**858**	641.85	752.64	2.01	-12.28%	18.53	1.40	-623.32
	7920	0.8	0.2	1584	33.16	1,584.00	0.00	0.00%	1.53	0.32	-31.63
			0.5	**3624**	1,542.02	3,132.64	2.39	-13.56%	12.23	1.29	-1,529.79
			0.8	**1476**	689.52	1,325.20	1.30	-10.22%	28.21	3.52	-661.31
200	3980	0.1	0.2	**796**	812.28	792.00	0.00	-0.50%	2.56	0.31	-809.72
			0.5	**1990**	703.44	1,227.92	3.62	-38.30%	15.07	2.15	-688.37
			0.8	**3184**	1,289.60	462.32	1.81	-85.48%	27.36	3.25	-1,262.24
	7960	0.2	0.2	**1592**	994.13	1,592.00	0.00	0.00%	2.49	0.33	-991.64
			0.5	**3980**	1,001.72	2,870.08	4.76	-27.89%	25.30	3.98	-976.42
			0.8	**6368**	2,988.93	1,155.28	2.73	-81.86%	51.92	5.22	-2,937.01
	19900	0.5	0.2	**3980**	2,386.59	3,980.00	0.00	0.00%	3.04	0.22	-2,383.55
			0.5	**9606**	1,687.30	8,155.52	7.19	-15.10%	39.25	6.50	-1,648.05
			0.8	**3824**	2,395.06	3,410.80	3.52	-10.81%	96.79	13.23	-2,298.27
	31840	0.8	0.2	6368	1,449.56	6,368.00	0.00	0.00%	3.97	0.44	-1,445.59
			0.5	**15920**	2,272.33	13,615.04	6.59	-14.48%	54.16	10.89	-2,218.17
			0.8	**6236**	2,108.50	5,752.24	2.42	-7.76%	151.01	19.47	-1,957.49
300	8970	0.1	0.2	**1794**	862.03	1,794.00	0.00	0.00%	3.98	0.32	-858.05
			0.5	**4486**	859.98	3,078.72	4.79	-31.37%	36.53	3.99	-823.45
			0.8	**7176**	857.13	1,217.92	2.35	-83.03%	73.76	7.09	-783.37
	17940	0.2	0.2	**3588**	865.92	3,588.00	0.00	0.00%	4.17	0.21	-861.75
			0.5	**8970**	866.94	6,891.36	6.67	-23.17%	62.26	10.47	-804.68
			0.8	**14352**	862.98	2,864.32	2.77	-80.04%	137.69	15.34	-725.29
	44850	0.5	0.2	**8970**	861.98	8,970.00	0.00	0.00%	6.23	0.34	-855.75
			0.5	**22426**	879.82	19,119.76	8.92	-14.74%	110.13	24.89	-769.69
			0.8	**35880**	871.12	8,041.76	4.99	-77.59%	243.59	41.25	-627.53
	71760	0.8	0.2	**14352**	866.66	14,352.00	0.00	0.00%	9.18	0.65	-857.48
			0.5	**35880**	882.58	31,731.76	14.71	-11.56%	146.82	49.27	-735.76
			0.8	**57408**	879.75	13,326.48	4.17	-76.79%	398.19	71.99	-481.56
400	15960	0.1	0.2	**3192**	7,200.00	3,192.00	0.00	0.00%	5.63	0.24	-7,194.37
			0.5	**7980**	7,200.00	5,796.08	7.28	-27.37%	70.71	10.28	-7,129.29
			0.8	**12768**	4,675.04	2,342.40	4.02	-81.65%	162.51	23.72	-4,512.53
	31920	0.2	0.2	**6384**	4,711.29	6,384.00	0.00	0.00%	6.62	0.34	-4,704.67
			0.5	**15960**	4,819.54	12,836.16	10.38	-19.57%	102.82	19.05	-4,716.72
			0.8	**25536**	7,200.00	5,330.16	5.63	-79.13%	285.95	40.57	-6,914.05
	79800	0.5	0.2	**15960**	3,921.77	15,960.00	0.00	0.00%	11.96	0.70	-3,909.81
			0.5	**39900**	7,200.00	34,827.36	10.36	-12.71%	216.03	77.99	-6,983.97
			0.8	**63840**	4,970.73	14,639.68	6.38	-77.07%	550.07	116.59	-4,420.66
	127680	0.8	0.2	**25536**	7,200.00	25,536.00	0.00	0.00%	17.68	1.22	-7,182.32
			0.5	**63840**	7,200.00	57,417.76	14.16	-10.06%	327.69	142.55	-6,872.31
			0.8	**102144**	7,200.00	24,096.80	7.23	-76.41%	848.73	183.45	-6,351.27
600	35940	0.1	0.2	-	7,200.00	7,188.00	0.00	-	10.89	0.60	-7,189.11
			0.5	-	7,200.00	13,901.20	9.67	-	190.45	35.93	-7,009.55
			0.8	-	7,200.00	5,730.40	5.03	-	428.02	66.69	-6,771.98
	71880	0.2	0.2	-	7,200.00	14,376.00	0.00	-	14.62	0.52	-7,185.38
			0.5	-	7,200.00	30,136.32	10.40	-	298.51	91.20	-6,901.49
			0.8	-	7,200.00	12,615.12	5.88	-	797.49	146.57	-6,402.51
	179700	0.5	0.2	-	7,200.00	35,940.00	0.00	-	29.33	1.47	-7,170.67
			0.5	-	7,200.00	80,678.48	19.79	-	679.53	316.09	-6,520.47
			0.8	-	7,200.00	33,809.76	8.46	-	1,581.51	446.58	-5,618.49
	287520	0.8	0.2	-	7,200.00	57,504.00	0.00	-	42.74	2.94	-7,157.26
			0.5	-	7,200.00	132,075.52	26.65	-	1,042.45	566.78	-6,157.55
			0.8	-	7,200.00	55,163.20	10.00	-	2,210.26	736.74	-4,989.74
Average				**-**	**3,301.40**	**-**	**-**	**-23.62%**	**195.61**	**55.53**	**-3,105.78**

Table 2 – ILP and sequential GRASP (SeqGRASP) results for random instances in (iii). Instances not solved to optimality by ILP formulation are marked with bold values in column (ILP - BestSol); in that case this column exhibits the value of the best integer solution found in the time limit. Avg I(P) is the average imbalance and Avg Time is the average execution time (in seconds), after 25 executions of the heuristic. Gap %I(P)= $100 \times$ (Avg I(P)−BestSol)/BestSol. Gap Time is the gap between ILP and SeqGRASP execution times. CI[I(P)] and CI(Time) are the 95% confidence intervals of the imbalance and execution time, respectively.

Year	ILP		SeqGRASP					
	I(P)	Time	BestSol	CI(BestSol)	Gap %I(P)	Avg Time	CI(Time)	Gap Time
1946	9.3327	2.00	9.3378	1.18E-07	0.05%	2.57	0.30	0.57
1947	18.6904	6.00	18.6975	1.67E-07	0.04%	3.84	0.37	-2.16
1948	16.9734	6.00	16.9853	9.61E-08	0.07%	6.38	0.38	0.38
1949	37.7317	7.00	37.7484	4.99E-07	0.04%	3.54	0.29	-3.46
1950	25.0162	4.00	25.0279	1.92E-07	0.05%	3.25	0.32	-0.75
1951	58.9514	20.00	58.9600	3.85E-07	0.01%	4.04	0.39	-15.96
1952	46.0840	7.00	46.0990	2.72E-07	0.03%	4.73	0.33	-2.27
1953	31.2752	6.00	31.2880	0.00E+00	0.04%	3.65	0.34	-2.35
1954	32.8097	7.00	32.8227	4.71E-07	0.04%	4.57	0.54	-2.43
1955	13.1367	2.00	13.1468	1.36E-07	0.08%	3.96	0.40	1.96
1956	47.7567	17.00	47.7781	2.72E-07	0.04%	8.59	1.16	-8.41
1957	106.7950	75.00	106.8349	4.46E-07	0.04%	7.98	0.93	-67.02
1958	122.4970	112.00	122.5359	1.09E-06	0.03%	6.93	1.00	-105.07
1959	102.8430	93.00	102.8809	1.33E-06	0.04%	8.18	1.16	-84.82
1960	94.1976	129.00	94.2386	7.69E-07	0.04%	16.74	1.92	-112.26
1961	115.5650	262.00	115.6122	1.13E-06	0.04%	14.97	2.31	-247.03
1962	154.3600	358.00	154.4117	1.83E-06	0.03%	14.21	1.40	-343.79
1963	93.4542	84.00	93.4824	7.69E-07	0.03%	18.42	1.73	-65.58
1964	0.0000	17.00	0.0000	0.00E+00	0.00%	5.62	0.00	-11.38
1965	137.2510	73.00	137.2849	1.33E-06	0.02%	18.33	1.62	-54.67
1966	213.5970	849.00	213.6803	1.09E-06	0.04%	19.21	1.88	-829.79
1967	242.0090	1,205.00	242.0621	2.66E-06	0.02%	28.51	3.34	-1,176.49
1968	190.3440	1,001.00	190.4335	1.88E-06	0.05%	25.83	2.91	-975.17
1969	142.0640	359.00	142.1171	1.72E-06	0.04%	23.46	2.45	-335.54
1970	194.3390	338.00	194.4068	1.88E-06	0.03%	30.00	3.85	-308.00
1971	40.8834	33.00	40.9018	4.30E-07	0.05%	23.93	2.67	-9.07
1972	16.2836	25.00	16.2943	1.27E-07	0.07%	18.02	2.03	-6.98
1973	29.4791	59.00	29.5001	4.71E-07	0.07%	23.48	2.74	-35.52
1974	37.9123	38.00	37.9340	6.08E-07	0.06%	28.04	4.43	-9.96
1975	85.8126	214.00	85.8431	1.01E-06	0.04%	35.30	3.54	-178.70
1976	49.3427	134.00	49.3702	0.00E+00	0.06%	35.66	4.85	-98.34
1977	45.9462	69.00	45.9657	4.71E-07	0.04%	29.68	4.03	-39.32
1978	74.7211	183.00	74.7554	0.00E+00	0.05%	31.09	4.44	-151.91
1979	87.8995	169.00	87.9325	1.15E-06	0.04%	31.30	2.99	-137.70
1980	102.9330	237.00	102.9577	1.29E-06	0.02%	40.13	3.75	-196.87
1981	99.5351	285.00	99.5714	4.15E-07	0.04%	51.57	7.03	-233.43
1982	75.5698	229.00	75.6021	5.87E-07	0.04%	38.66	5.09	-190.34
1983	61.3197	169.00	61.3410	6.66E-07	0.03%	46.37	5.62	-122.63
1984	45.4756	277.00	45.4901	3.85E-07	0.03%	46.73	6.72	-230.27
1985	53.6096	220.00	53.6297	6.66E-07	0.04%	26.51	3.75	-193.49
1986	44.6501	71.00	44.6731	4.71E-07	0.05%	30.17	3.21	-40.83
1987	16.3129	53.00	16.3204	6.80E-08	0.05%	35.15	4.12	-17.85
1988	33.8870	72.00	33.9051	4.99E-07	0.05%	24.70	2.30	-47.30
1989	17.2942	57.00	17.3022	2.88E-07	0.05%	31.16	4.03	-25.84
1990	19.3817	70.00	19.3896	1.92E-07	0.04%	33.94	3.80	-36.06
1991	15.4498	130.00	15.4796	1.80E-07	0.19%	51.36	5.94	-78.64
1992	17.6703	104.00	17.7159	2.54E-07	0.26%	52.42	6.89	-51.58
1993	27.4587	122.00	27.4796	1.36E-07	0.08%	52.56	7.60	-69.44
1994	35.4362	240.00	35.4781	2.72E-07	0.12%	64.85	8.69	-175.15
1995	28.4099	125.00	28.4413	1.36E-07	0.11%	58.90	5.63	-66.10
1996	13.1945	115.00	13.2284	1.36E-07	0.26%	51.09	6.66	-63.91
1997	83.3901	271.00	83.4223	1.02E-06	0.04%	56.60	6.31	-214.40
1998	92.6719	339.00	92.6943	1.38E-06	0.02%	61.93	6.82	-277.07
1999	22.0806	160.00	22.1183	2.72E-07	0.17%	61.35	10.87	-98.65
2000	29.3329	296.00	29.3535	3.85E-07	0.07%	78.62	11.65	-217.38
2001	33.4920	516.00	33.5312	4.40E-07	0.12%	76.13	11.96	-439.87
2002	12.6935	242.00	12.7005	6.80E-08	0.06%	55.96	7.36	-186.04
2003	7.4469	281.00	7.4663	5.89E-08	0.26%	50.88	5.81	-230.12
2004	20.6182	301.00	20.6382	1.67E-07	0.10%	47.24	5.69	-253.76
2005	40.2684	338.00	40.2919	5.09E-07	0.06%	47.96	4.52	-290.04
2006	28.9418	223.00	28.9545	3.85E-07	0.04%	48.59	4.76	-174.41
2007	45.5292	262.00	45.5699	3.33E-07	0.09%	58.93	7.41	-203.07
2008	36.86	178.00	36.8889	4.30E-07	0.08%	55.50	6.85	-122.50
Average					**0.06%**			**-158.19**

Table 3 – Average % gap between ILP and sequential GRASP in solution values [Gap % I(P)] and execution time (Gap Time) in seconds. CI(BestSol) and CI(Time) are the 95% confidence intervals of the imbalance and execution time, respectively, after 25 independent executions of SeqGRASP.

Pajek (version 3.14)[8] was invoked with the following configuration: Doreian Mrvar Method, starting from a random partition (1-mode), 1000 repetitions, $\alpha = 0.5$, minimum number of vertices in clusters = 1.

Pajek's Doreian Mrvar Method accepts only one stopping criterion: number of iterations. Since an iteration of this method is fundamentally different from an iteration performed by GRASP, the number of iterations is not a sound basis for comparison of both algorithms. Therefore, as an alternative, we have set the time-limit parameter of GRASP to 2 hours and the number of repetitions of Pajek to 1000, which was enough for Pajek to either reach the same solution value of GRASP, or exceed the time limit with an inferior solution value. In this way, we were able to compare both solution values and execution times in a fair manner. As shown in Table 4, when it comes to these larger instances, the GRASP procedure can get to the same solution or to a better one at least 34 times faster than Pajek's Doreian Mrvar Method. We do not include Pajek execution data for instances with $n > 800$, as it was unable to find any solution within 2 hours.

2.3.6.2 Sequential GRASP *vs.* VOTE/BOEM

We have implemented the VOTE/BOEM heuristic for the CC Problem (here named $VOTE/BOEM$), comparing the obtained solutions with the previous results from our best sequential GRASP algorithm. Again, our goal was to solve the Slashdot instances in set (i). The time limit was set to 2 hours for both procedures.

As shown in Table 4, the imbalance results of the sequential GRASP are always better than $VOTE/BOEM$. For smaller instances, the faster execution times of $VOTE/BOEM$ are not surprising, since the heuristic is equivalent to only one multistart iteration of the GRASP metaheuristic. As the number of vertices increases, the GRASP procedure is still unable to finish earlier than VOTE/BOEM.

2.3.7 The parallel strategies

Since the Slashdot instances in (i) have shown the limits of the sequential version of GRASP, we have used them to assess the performance of the parallel GRASP algorithm. Three different solution methods were applied:

- sequential GRASP metaheuristic ($SeqGRASP$, using only one processor core);

[8] In order to solve the CC Problem with Pajek, we followed these steps: after loading the network file, we created a random initial partition using $1 - mode$. At this point we defined the number of clusters k of the solution. We have used the same number of clusters of the solution returned by GRASP. Finally, we called the resolution method from the menu (Network → Signed Network → Create Partition → Doreian Mrvar Method), specifying the number of repetitions (we set it to 1000), the alpha parameter (set to 0.5 to match the CC Problem objective function) and the minimum number of vertices in clusters (set to 1).

Instance					*SeqGRASP*			Doreian Mrvar Method			*VOTE/BOEM*		
n	$\|E^-\|$	$\|E^+\|$	$w(E^-)$	$w(E^+)$	k	BestSol	Time	k	BestSol	Time	k	BestSol	Time
200	62	825	62	825	6	45.0	1.82	3	45.2	61.80	11	47.0	0.42
300	82	981	82	981	10	54.0	3.54	2	54.0	125.00	14	57.0	0.50
400	87	1192	87	1192	9	57.0	4.88	2	57.0	454.40	14	61.0	0.44
600	156	1761	156	1761	10	109.0	8.78	3	119.0	2,740.60	12	111.0	1.14
800	371	2965	371	2965	21	240.0	18.54	4	244.8	7,381.40	33	245.8	2.09
1000	859	5132	859	5132	24	600.0	32.80	-	-	-	44	608.9	4.27
2000	3217	17598	3217	17598	55	2,187.0	169.50	-	-	-	105	2,264.2	43.86
4000	8664	40868	8664	40868	81	6,203.0	751.70	-	-	-	147	6,216.5	395.10
8000	22789	86916	22789	86916	195	16,083.0	2,891.84	-	-	-	375	16,115.8	3,475.91
10000	29805	109266	29805	109266	238	20,586.0	4,589.55	-	-	-	169	23,458.4	7,041.97

Table 4 – Sequential GRASP for the CC Problem, here named $SeqGRASP$, *vs.* Doreian Mrvar Method *vs.* $VOTE/BOEM$ results obtained on Slashdot signed graphs. Average number of clusters (k) in the solution; BestSol is the average value of the best solution found and Time is the average execution time (in seconds), after 25 executions of each algorithm.

- independent parallel algorithm with sequential local search ($ParGRASP/SeqVND$);
- parallel GRASP with parallel local search ($ParGRASP/ParVND$).

We conducted several experiments to find the optimal number of processes to be used in the parallel algorithms. This setting is closely related to the hardware configuration of the computer cluster used in the experiments. As previously explained in Figure 9, since each machine has 2 quad-core CPUs (8 processor cores), it can host 8 processes running in parallel. In $ParGRASP/ParVND$, we chose to group each GRASP master process together with its corresponding VND search slaves, in order to maximize the performance of message exchange between related processes. As seen in Figure 11, the parallel procedures with the best efficiency were:

- independent parallel algorithm with sequential local search ($ParGRASP/SeqVND$), using 8 cores;
- parallel GRASP with parallel local search ($ParGRASP/ParVND$), using 8 cores, where 2 cores run GRASP master processes and 6 cores run VND search slave processes, 3 for each master.

The parameters of the procedures used in the parallel approaches of GRASP are the same used for testing the sequential algorithms, except for the number of GRASP iterations. In $ParGRASP/SeqVND(8)$, the number of iterations is reduced from 400 to 50. This is due to the fact this parallel algorithm executes 8 GRASP procedures at the same time, which provides enough variability to make the 50-iteration parallel GRASP results quality-equivalent to the 400-iteration sequential GRASP. In $ParGRASP/ParVND(8)$,

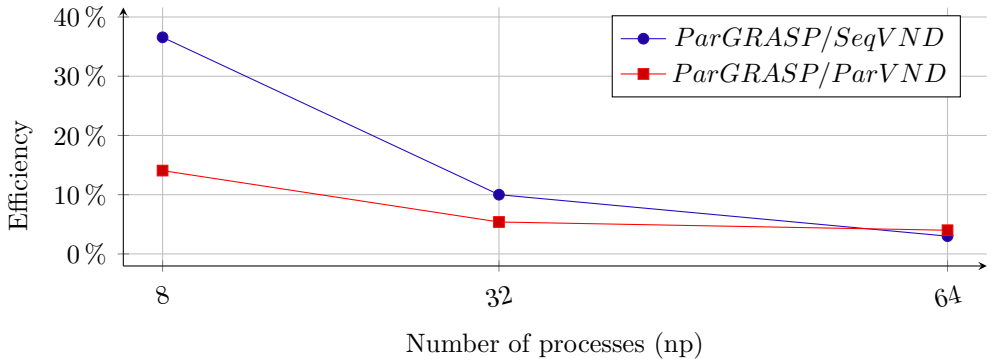


Figure 10 – Average efficiency of parallel GRASP with sequential local search ($ParGRASP/SeqVND$) and parallel GRASP with parallel local search ($ParGRASP/ParVND$) when solving Slashdot-based signed graphs, after 25 independent executions.

2 GRASP master processes run 200 iterations each. There are also 6 VND search slaves, 3 for each master.

Note that, in order to compare the sequential and parallel procedures, the following additional stopping criterion was applied: all procedures stop whenever the specified target solution value [Target I(P)] is found.

2.3.8 Sequential *vs.* Parallel GRASP

In this section, the Parallel GRASP algorithm results [$ParGRASP/SeqVND(8)$ and $ParGRASP/ParVND(8)$] are compared to the sequential version ($SeqGRASP$).

As seen in Table 5, the algorithm with the best efficiency was $ParGRASP/SeqVND$ (8 cores from one machine with 2 quad-core CPUs), with an average speed-up of 2.93 (minimum of 0.97 and maximum of 4.11) and efficiency of 37% (minimum of 12% and maximum of 51%). Remark that lower speed-up and efficiency values are related to smaller instances, whose solution takes just a few seconds. Also, we did not obtain linear speed-ups because of the random nature of the heuristics. The sequential algorithm reached the target solution values proportionately faster than their parallel counterparts.

The execution times of $ParGRASP/ParVND(8)$ were in average higher than the execution times of $ParGRASP/SeqVND(8)$. Therefore $ParGRASP/ParVND$, using the same number of processes, presented a speedup of x1.13 and an average efficiency of only 14%.

We also ran both parallel algorithms to solve the random networks in (iii) and the results shown in Figure 11 confirm the superiority of $ParGRASP/SeqVND(8)$, with average efficiency of 40% (minimum of 28% and maximum of 56%), while $ParGRASP/ParVND(8)$

Instance	SeqGRASP			ParGRASP/SeqVND(8)			ParGRASP/ParVND(8)		
	Target I(P)	Avg Time	CI(Time)	Avg Time	CI(Time)	Speedup	Avg Time	CI(Time)	Speedup
200	45	1.82	0.04	1.88	0.30	0.97	1.35	0.17	1.36
300	54	3.54	0.16	1.70	0.33	2.09	2.41	0.17	1.47
400	57	4.88	0.19	1.53	0.22	3.19	3.41	0.19	1.43
600	109	8.78	0.31	3.17	0.39	2.77	5.51	0.35	1.59
800	240	18.54	0.95	4.96	0.34	3.74	11.71	1.12	1.58
1000	600	32.80	1.40	7.98	0.75	4.11	23.62	1.99	1.39
2000	2187	169.50	19.02	41.83	6.84	4.05	263.74	26.29	0.64
4000	6203	751.70	94.42	218.89	68.64	3.43	2,049.40	199.34	0.37
8000	16083	2,891.84	599.95	1,152.27	534.20	2.51	3,924.46	482.53	0.74
10000	20586	4,589.55	660.33	1,918.06	958.57	2.39	6,654.05	448.19	0.69
Average						**2.93**			**1.13**

Table 5 – CC results obtained on Slashdot signed graphs by the use of the following approaches: Sequential GRASP ($SeqGRASP$), Parallel GRASP with sequential VND ($ParGRASP/SeqVND(np)$) and Parallel GRASP with parallel VND ($ParGRASP/ParVND(np)$), where np is the number of processes in parallel. Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time spent (in seconds) by each algorithm to reach the specified target solution value, after 25 independent executions. CI(Time) is the 95% confidence interval of the execution time, for each algorithm.

presented a maximum efficiency of only 12%. This degradation in speedup and efficiency is somewhat expected, since the $ParallelLocalSearch$ algorithm causes an overhead in the number of messages exchanged between processes.[9] Therefore, even in random networks, where higher edge density and complex structure increase the time spent by local search, the additional computational resources required to run parallel local search with message-passing are not worth the available acceleration and efficiency brought by this procedure.

The obtained computational results indicate that our independent parallel GRASP metaheuristic is an efficient approach for the heuristic solution of the CC problem, with $ParGRASP/SeqVND$ with 8 processes being the fastest configuration for networks of up to $10,000$ vertices.

[9] Please note that the loss of efficiency when using parallel local search is not caused by the lack of work for the VND search slaves. For example, when solving Slashdot instance of size $n = 10000$, local search performs more than 600 million neighborhood evaluations, and when solving random instances with $n = 600$, this number rises to 700 million.

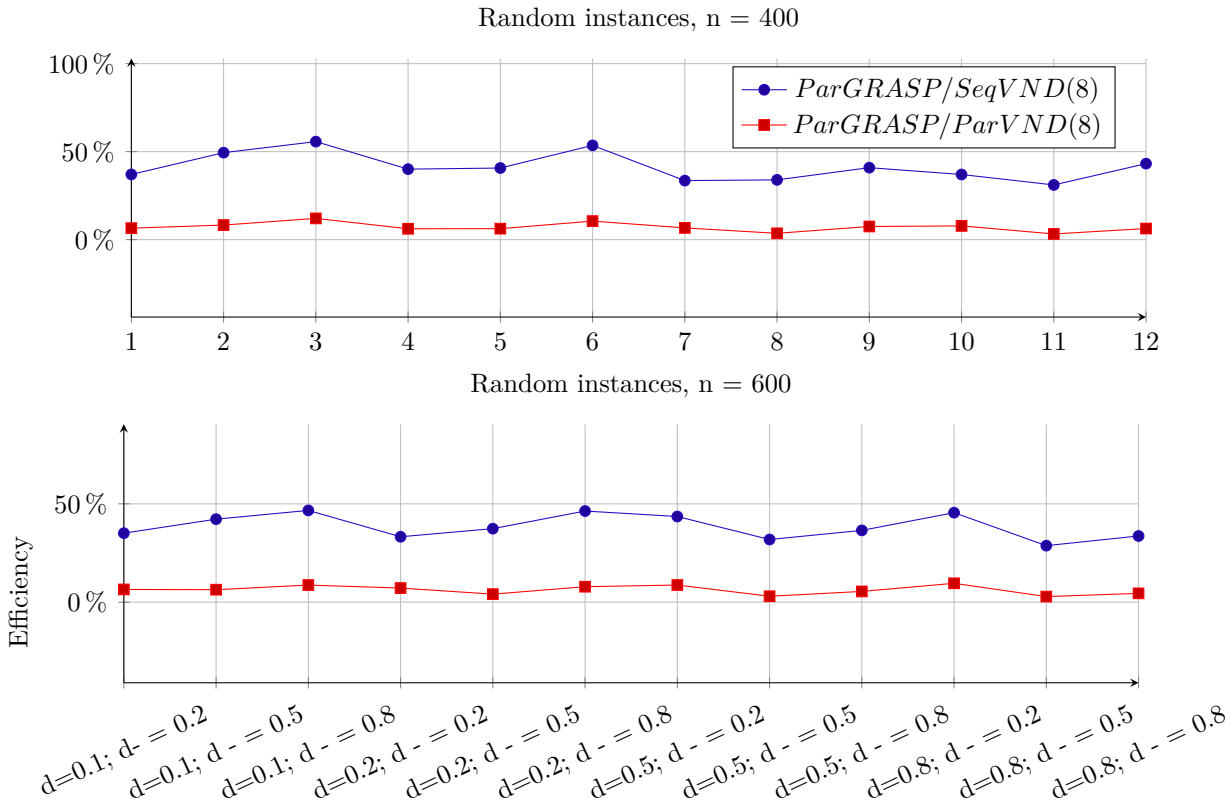


Figure 11 – Average efficiency of parallel GRASP with sequential local search ($ParGRASP/SeqVND(8)$) and parallel GRASP with parallel local search ($ParGRASP/ParVND(8)$) when solving random instances in (iii), after 25 independent executions.

3 Multistart Iterated Local Search (ILS) heuristic for the CC Problem

The Iterated Local Search (LOURENÇO; MARTIN; STÜTZLE, 2003) (ILS) belongs to a class of local search algorithms (as VNS (MLADENOVIĆ; HANSEN, 1997), IGS (RUIZ; STÜTZLE, 2007), GRASP (FEO; RESENDE, 1995) and ad hoc local-search schemes (BRUSCO; KÖHN, 2009)) proposed in the literature to efficiently investigate the solution space of combinatorial problems. The difference among all these methods is the philosophy used to walk in the solution space. ILS is a metaheuristic that aims to improve upon stochastic multi-restart search by sampling in a broader and distant neighborhood of candidate solutions, followed by a local search that refines solutions to their local optima. To reach these distant neighborhoods, it explores a sequence of solutions created as perturbations of the current best solution. The result is then refined using an embedded heuristic.

As mentioned in Chapter 1, the CC problem is related to the clique partitioning problem. The authors in Brusco et al.(BRUSCO; KÖHN, 2009) developed a neighborhood search heuristic (known as NS-R) for the clique partitioning problem, which is similar to the one proposed in this chapter. NS-R includes the basic principles of random perturbation and pursuit of a local optimum through a local search procedure also present in ILS.

Our ILS algorithm to solve the CC problem reuses the constructive and local search modules of GRASP, as well as its *multistart* nature. It also includes a procedure that creates perturbations in the best current solution. Like GRASP, ILS was implemented not only sequentially, but also using the parallel strategies presented in the previous chapter.

3.1 Multistart ILS algorithm

The proposed heuristic ($ILSMultiStartCC$) has the same multistart (MARTÍ, 2003) nature of GRASP. In a nutshell, *multistart* brings an alternative way to achieve diversification of solutions, restarting the ILS algorithm with a new initial solution whenever a search region has already been extensively explored.

Algorithm 5 summarizes the operation of ILS, which consists of 4 elements: (i) the $ConstructivePhase$, that generates an initial solution, (ii) the local search $VariableNeighborhoodDescent$, (iii) a procedure called $Perturbation$ to modify a solution at random, and (iv) an acceptance criterion, which defines from which solution the search will resume. The *iter* parameter represents the number of *multistart* iterations, that

is, the number of times ILS will be executed, returning the best solution based on all ILS executions. The value $iterMaxILS$ determines the number of iterations of the main ILS loop, where perturbation and local search phases will be applied. Finally, the parameter $perturbationMax$ limits the level of perturbations to be applied to a given solution. We next describe each step of the algorithm in detail.

The first task in each iteration of $ILSMultiStartCC$ is the construction of an initial solution in a greedy randomized fashion, the same way it is done in the GRASP algorithm. This task is performed in $ConstructivePhase$ procedure previously described in Algorithm 2 from Chapter 2, so it will not be listed here.

Algorithm 5: $ILSMultiStartCC$

```
1  Input: G = (V, E), α, iter, iterMaxILS and perturbationMax
2  Output: partition P*
3  P* = ∅;  I(P*) = ∞;  i = 1;
4  while (i ⩽ iter)
5      P = ConstructivePhase(G, α);
6      P = VariableNeighborhoodDescent(P, G);
7      j = 1;  t = 1;
8      repeat
9          P̄ = Perturbation(P, t);
10         P̄ = VariableNeighborhoodDescent(P̄, G);
11         if (I(P̄) < I(P))
12             P = P̄;  j = 1;  t = 1;
13         else
14             j = j + 1;
15             if (j ⩾ iterMaxILS)
16                 t = t + 1;  j = 1;
17             end if
18         end if
19     while (t ⩽ perturbationMax)
20     i = i + 1;
21     if (I(P) < I(P*))
22         P* = P;
23 end while
24 return P*;
```

There is no guarantee that the construction method will return a locally optimal solution with respect to some neighborhood. Therefore, the solution P, obtained in $ConstructivePhase$, could be improved by the local search procedure $VariableNeighborhoodDescent$, which consists of the same local search phase applied in GRASP. More details are available in Algorithm 3 from Chapter 2.

The next step of the algorithm is generating a perturbation whose level is proportional to the value of t. The perturbation, listed in Algorithm 6, simply consists of t

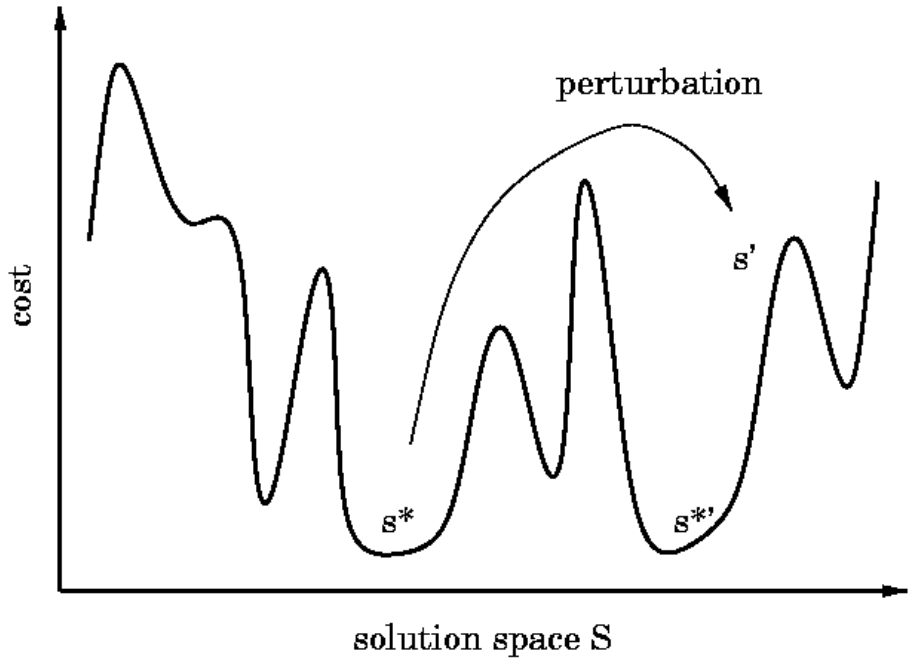


Figure 12 – ILS perturbation (GLOVER; KOCHENBERGER, 2003).

random movements of any vertex from one cluster to another, that is, a large random move (shuffle) of the current solution. The perturbation serves as mechanism for escaping from local optima (Figure 12).

Finally, a basic acceptance criterion was used in our ILS. We accepted only improving moves, that is, a move that forces the cost (imbalance) to decrease, which corresponds to a first-improvement descent in the set of solutions.

Algorithm 6: *Perturbation*

Input: $G = (V, E)$, a partition P and perturbation level t
Output: partition P
$i = 1$;
while $(i \leqslant t)$
  Choose a random cluster $c_x \in P$ and a random vertex $i \in c_x$;
  Choose a random cluster $c_y \in P$ such that $x \neq y$;
  Move vertex i from cluster c_x to cluster c_y;
  $i = i + 1$;
end while
return P;

As described by Den Besten et al. (BESTEN; STÜTZLE; DORIGO, 2001), the efficacy of the local search module is of extreme importance to ILS, since it influences not only the quality of the final solution of the metaheuristic, but also the total time spent. In turn, the perturbations should allow ILS to effectively escape from local optima, while at the same time avoiding the disadvantages of a random restart (i.e., should not be too

strong). The acceptance criteria, along with the perturbations, greatly influence the type of walk through the solution space and can be applied to algorithm tuning, alternating between intensification and diversification of the search. The challenge of designing an ILS algorithm is the need to find the best combination and configuration of its modules, so that the best overall performance is achieved.

3.1.1 Parallel ILS and Parallel Variable Neighborhood Descent

The same approaches previously used to parallelize the GRASP algorithm have been applied to improve ILS performance. With the independent approach in mind, in the parallel version of multistart ILS, the outer loop of the ILS-CC algorithm (line 4), responsible for its multistart nature, is replaced by i ILS processes running at the same time, each with a different random seed. Likewise, in the parallel ILS with parallel VND ($ParILS/PVND$), the $VariableNeighborhoodDescent$ procedure of ILS uses the $ParallelLocalSearch$ procedure (Algorithm 4 in Chapter 2) to divide search space traversal. The process allocation scheme for parallel ILS is the same employed in parallel GRASP, as depicted in Figure 9 from Chapter 2.

3.2 Improvements of Multistart ILS over GRASP algorithm

In the previous chapter, we have proposed sequential and parallel GRASP approaches for solving large network instances. However, it was observed that the sequential version of GRASP presented long processing times (more than 1 hour) for instances with more than 8000 vertices.

To obtain an improved heuristic, we have reused the constructive and local search modules of GRASP, as well as its *multistart* nature, and added a procedure that creates perturbations in the best current solution, creating the $ILSMultiStartCC$ algorithm, described in the previous section. As we shall see, the great advantage that this algorithm has over GRASP is the reduced execution time. You can invoke i local searches within ILS much more quickly than if the same i local searches were performed within GRASP's random restart framework. That's probably because fewer movements in the neighborhood are necessary to achieve local optimum from a perturbed solution than from a new solution originated by a greedy random algorithm. In other words, the potential power of ILS lies in its random and biased sampling (perturbations) of the set of local optima found in the local search step. According to Den Besten et al.(BESTEN; STÜTZLE; DORIGO, 2001), even with the simplest implementations of perturbations and acceptance criteria, ILS appears to be much more efficient than simple random restart. This superiority can be demonstrated, for example, by the metaheuristic execution time.

3.3 Experiments and performance comparisons

3.3.1 Computer environment and test problems

Like GRASP, the ILS algorithms previously described were also implemented in ANSI C++. All experiments, including those with GRASP, were conducted on the same computer environment described in Chapter 2, Section 2.3.1. All heuristic outcomes represent the average of 25 independent runs and all confidence intervals (CI) were obtained through Student's t-test at a confidence level of 95%.

Computational experiments were performed on the same sets of instances already described in Section 2.3.2 from Chapter 2. In addition to it, we have generated a fourth set of instances, composed of 45 random signed networks with a predefined community structure, according to Yang et al. (YANG; CHEUNG; LIU, 2007). The random signed network is defined as $SG(c, n, k, p_{in}, p_-, p_+)$, where c is the number of communities in the network, n is the number of nodes in each community, k is the degree of each node, p_{in} is the probability of each node connecting other nodes in the same community, p_- denotes the probability of negative links appearing within communities, and p_+ denotes that of positive links appearing between communities. The signs of the generated links within communities are positive, whereas those between communities are negative. Based on the network generation process, there will be $c \times n \times k \times p_{in} \times p_-$ negative links within communities and $c \times n \times k \times (1 - p_{in}) \times p_+$ positive links outside communities.

3.3.2 Methodology

We have used the instances in (i) and (ii) to parametrize the heuristics. After testing various combinations of parameters, including the maximum level of perturbation, the following configurations presented the best results for ILS, depending on the instance size:

SetPar	**Graph size**	Time limit	Alpha	Neighborhood	Iterations	ILS iterations	Perturbation level
A	$n < 300$	1 hour	$\alpha = 0.4$	$r = 1$	$iter = 10$	$iterMaxILS = 5$	$perturbMax = 3$
B	$n \geq 300$	2 hours	$\alpha = 1.0$	$r = 1$	$iter = 10$	$iterMaxILS = 5$	$perturbMax = 30$

In order to compare the ILS and GRASP algorithms, we reuse the GRASP configuration presented in Chapter 2. We also compare the ILS algorithm to the same solution methods listed in Section 2.3 of Chapter 2.

When testing the independent parallel version of ILS, if p is the number of simultaneous processes, the number of multistart iterations (*iter* parameter) is reduced to $iter/p$. This is due to the fact that the parallel procedure executes p independent metaheuristic procedures at the same time, which provides enough variability so that the aggregated results of parallel ILS are quality-equivalent to those of the sequential ILS.

As described in the previous chapter, whenever it is required to assess the performance of parallel algorithms, two metrics are applied: speed-up $Su(p)$ measures the acceleration observed for the parallel algorithm when compared with its sequential version and efficiency $E(p)$ measures the average fraction of time along which each process is effectively used. Thus, $Su(p) = T(seq)/T(p)$, such that $T(seq)$ is the time required for the sequential algorithm and $T(p)$ the time required for the parallel algorithm run on p processors, and $E(p) = Su(p)/p$.

3.3.3 Exact solution by ILP formulation

As previously observed in Chapter 2, the drawback of the ILP approach appears when we attempt to solve larger instances. For most completely random instances in (iii), the solver is unable to find an optimal solution within the time limit (Table 6), and on Slashdot-based instances, the ILP formulation becomes too big and the solver is unable to return any solution within the prescribed time limit.

3.3.4 Sequential ILS *vs.* sequential GRASP

We compared both SeqGRASP and SeqILS, in their best configurations. When solving all UNGA instances in (ii), within the same time limit (1h), SeqILS presented the same results of SeqGRASP. Since UNGA networks consist of extremely balanced instances, small perturbations only change the current solution to a very similar one, already visited before. In other words, a small number of perturbations wasn't enough to provide the kicks that ILS needed to escape from local optima and further explore the space of feasible solutions. One possible explanation lies in the fact that, as such instances are well balanced, both pseudo-random initial solutions and the perturbations led to the same point to be reoptimized. As Figure 13 shows, the sequential ILS procedure was always faster than GRASP. Considering the sum of the average execution times of UNGA instances, ILS took 1739 seconds less. This summarizes in an average of minus 27.60 seconds per instance after 25 independent runs of each algorithm.

When applying the set of completely random instances (iii) as input, we observe the superiority of SeqILS over SeqGRASP. As Table 6 shows, for 60 random instances with $100 \leq n \leq 600$, SeqILS running time was strictly better in 45 of these instances, while SeqGRASP found the target solution value earlier for 15 instances. Still, the average time to target of ILS was smaller than GRASP's: on average, ILS is 4 times faster than GRASP.

Additionally, when applying both metaheuristics to solve the random instances with predefined community structure in (iv), SeqILS was, on average, superior to SeqGRASP in execution time. As shown in Table 7, an analysis of the gap in average time to target

				ILP		Target	SeqGRASP		SeqILS			T(GRASP) /
n	**e**	**d**	**d-**	**BestSol**	**Time**	**I(P)**	**AvgTime**	**CI**	**AvgTime**	**CI**	**Gap time**	**T(ILS)**
100	990	0.1	0.2	198	71.49	198	0.00	0.00	0.00	0.00	0.00	0.63
			0.5	**292**	1,339.70	228	0.53	0.23	0.23	0.07	-0.30	2.31
			0.8	50	308.74	54	1.03	0.37	0.22	0.04	-0.81	4.66
	1980	0.2	0.2	396	82.50	396	0.00	0.00	0.00	0.00	0.00	0.95
			0.5	**780**	933.03	582	1.27	0.46	1.63	0.84	0.36	0.78
			0.8	**272**	709.02	208	1.80	0.62	0.34	0.09	-1.46	5.28
	4950	0.5	0.2	990	60.42	990	0.00	0.00	0.00	0.00	0.00	0.92
			0.5	**2234**	1,267.70	1838	0.99	0.46	0.96	0.38	-0.03	1.03
			0.8	**858**	641.85	732	2.29	0.77	0.49	0.12	-1.80	4.72
	7920	0.8	0.2	1584	33.16	1584	0.00	0.00	0.00	0.00	0.00	1.03
			0.5	**3624**	1,542.02	3128	2.80	0.97	4.36	2.19	1.56	0.64
			0.8	**1476**	689.52	1306	4.64	1.81	0.92	0.39	-3.73	5.07
200	3980	0.1	0.2	**796**	812.28	792	0.01	0.00	0.01	0.00	0.00	0.89
			0.5	**1990**	703.44	1212	1.94	0.79	1.25	0.57	-0.69	1.55
			0.8	**3184**	1,289.60	406	3.50	1.53	0.61	0.14	-2.89	5.71
	7960	0.2	0.2	**1592**	994.13	1592	0.01	0.00	0.01	0.00	0.00	0.98
			0.5	**3980**	1,001.72	2852	5.03	1.64	2.44	0.93	-2.60	2.07
			0.8	**6368**	2,988.93	1098	14.96	5.01	1.31	0.21	-13.65	11.42
	19900	0.5	0.2	**3980**	2,386.59	3980	0.01	0.00	0.01	0.00	0.00	1.10
			0.5	**9606**	1,687.30	8144	4.37	2.54	2.86	1.38	-1.50	1.53
			0.8	**3824**	2,395.06	3350	11.82	4.46	2.19	0.53	-9.63	5.39
	31840	0.8	0.2	6368	1,449.56	6368	0.01	0.00	0.01	0.00	0.00	1.11
			0.5	**15920**	2,272.33	13604	10.55	4.04	11.28	9.39	0.73	0.94
			0.8	**6236**	2,108.50	5698	38.76	12.92	2.28	0.46	-36.48	17.00
300	8970	0.1	0.2	**1794**	862.03	1794	0.01	0.00	0.01	0.00	0.00	0.96
			0.5	**4486**	859.98	3036	6.97	3.42	3.67	1.59	-3.30	1.90
			0.8	**7176**	857.13	1116	15.00	5.06	1.48	0.22	-13.52	10.14
	17940	0.2	0.2	**3588**	865.92	3588	0.01	0.00	0.01	0.00	0.00	0.88
			0.5	**8970**	866.94	6878	10.32	3.23	5.33	2.72	-5.00	1.94
			0.8	**14352**	862.98	2766	46.69	14.93	2.91	0.65	-43.77	16.03
	44850	0.5	0.2	**8970**	861.98	8970	0.02	0.00	0.02	0.00	0.00	0.92
			0.5	**22426**	879.82	19070	35.45	18.51	16.22	7.48	-19.23	2.19
			0.8	**35880**	871.12	7930	22.88	8.25	3.18	0.65	-19.70	7.19
	71760	0.8	0.2	**14352**	866.66	14352	0.02	0.00	0.02	0.00	0.00	0.96
			0.5	**35880**	882.58	31704	24.18	19.83	14.18	7.79	-10.00	1.71
			0.8	**57408**	879.75	13222	117.57	42.75	4.66	1.02	-112.91	25.21
400	15960	0.1	0.2	**3192**	7,200.00	3192	0.01	0.00	0.01	0.00	0.00	0.99
			0.5	**7980**	7,200.00	5750	13.16	5.91	6.37	3.97	-6.79	2.07
			0.8	**12768**	4,675.04	2210	26.67	9.02	3.01	0.66	-23.65	8.85
	31920	0.2	0.2	**6384**	4,711.29	6384	0.02	0.00	0.02	0.00	0.00	1.11
			0.5	**15960**	4,819.54	12764	32.10	13.72	10.44	5.45	-21.66	3.08
			0.8	**25536**	7,200.00	5190	29.84	10.09	3.20	0.54	-26.65	9.34
	79800	0.5	0.2	**15960**	3,921.77	15960	0.03	0.00	0.03	0.00	0.00	1.04
			0.5	**39900**	7,200.00	34744	33.24	21.13	14.33	8.52	-18.91	2.32
			0.8	**63840**	4,970.73	14490	62.23	22.03	6.32	1.41	-55.91	9.84
	127680	0.8	0.2	**25536**	7,200.00	25536	0.04	0.00	0.04	0.00	0.00	1.03
			0.5	**63840**	7,200.00	57356	77.12	61.04	15.89	8.82	-61.24	4.86
			0.8	**102144**	7,200.00	23926	109.30	34.33	8.18	1.47	-101.12	13.37
600	35940	0.1	0.2	-	7,200.00	7188	0.03	0.00	0.03	0.00	0.00	0.94
			0.5	-	7,200.00	13814	34.30	10.27	10.56	5.04	-23.74	3.25
			0.8	-	7,200.00	5496	60.01	21.99	6.27	1.06	-53.74	9.57
	71880	0.2	0.2	-	7,200.00	14376	0.04	0.00	0.04	0.00	0.00	0.98
			0.5	-	7,200.00	30024	80.22	59.71	21.91	11.28	-58.31	3.66
			0.8	-	7,200.00	12364	119.48	49.82	9.78	2.60	-109.70	12.22
	179700	0.5	0.2	-	7,200.00	35940	0.07	0.01	0.07	0.00	0.00	1.04
			0.5	-	7,200.00	80512	249.22	196.25	58.82	39.03	-190.40	4.24
			0.8	-	7,200.00	33552	93.98	36.65	12.46	3.63	-81.52	7.54
	287520	0.8	0.2	-	7,200.00	57504	0.10	0.01	0.10	0.01	0.00	1.03
			0.5	-	7,200.00	131892	171.05	174.45	24.91	9.77	-146.14	6.87
			0.8	-	7,200.00	54886	209.53	72.35	17.89	4.12	-191.64	11.71
Average				**-**	**-**	**-**	**29.79**	**-**	**5.26**	**-**	**-24.52**	**4.48**

Table 6 – ILP, sequential GRASP (SeqGRASP) and sequential ILS (SeqILS) results for random instances in (iii). Instances solved to optimality by ILP formulation are marked with bold values in column (ILP - BestSol); in the other case this column exhibits the value of the best integer solution found in the time limit. Target I(P) is the target imbalance value used as stopping criterion for each algorithm. AvgTime is the average execution time spent (in seconds) by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. Gap Time is the gap between SeqILS and SeqGRASP execution times. T(GRASP)/T(ILS)= [AvgTime(SeqGRASP)/AvgTime(SeqILS)].

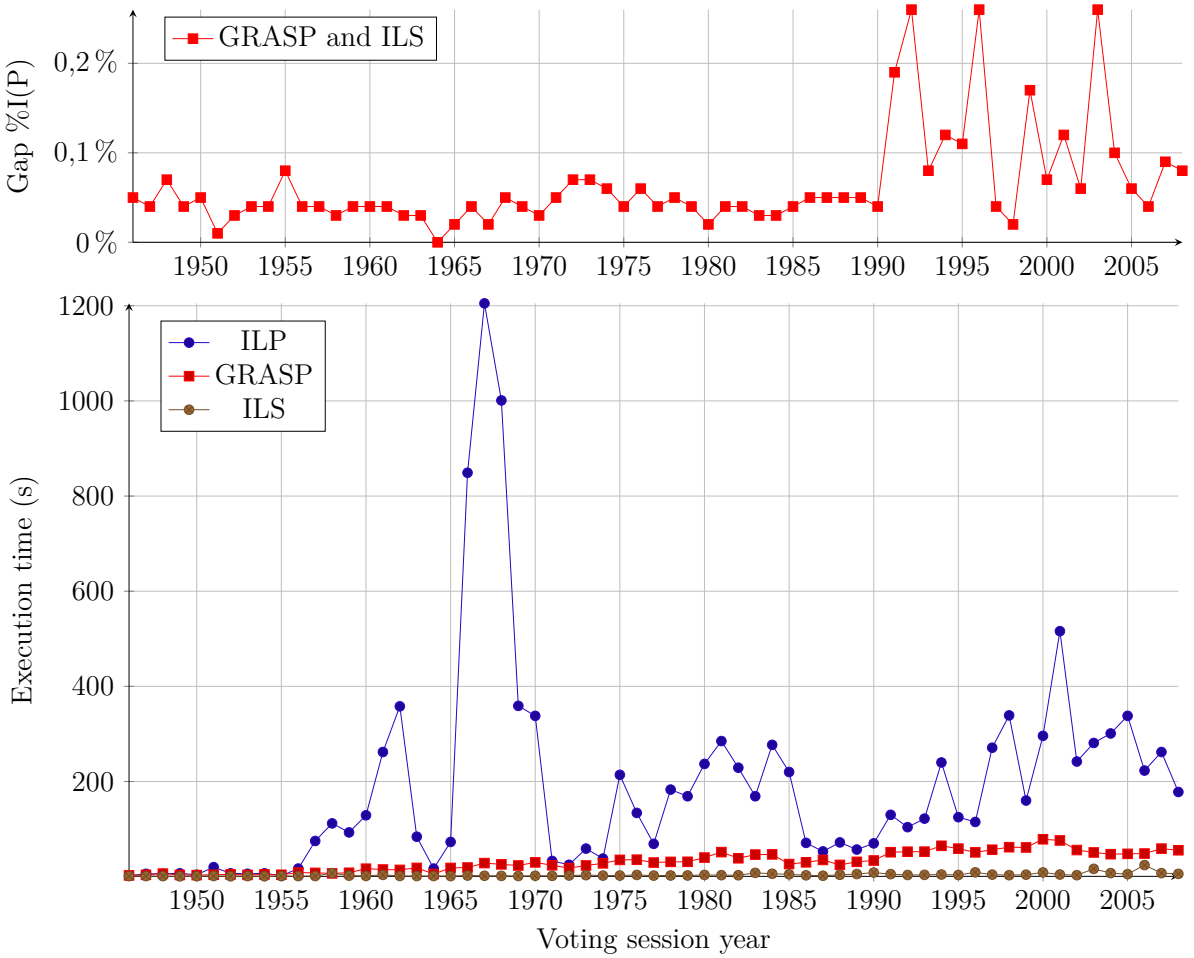


Figure 13 – First graph: average gap between ILP, sequential GRASP and sequential ILS solution values. Second graph: time spent (in seconds) on UNGA instances with ILP, sequential GRASP and sequential ILS. Average of 25 independent executions of each heuristic.

(column Gap time) between SeqILS and SeqGRASP indicates the superiority of SeqILS in 28 of 44 instances. On average, SeqILS is more than 2 times faster than SeqGRASP to find the specified target solution values.

Experiments with larger social networks were also performed with SeqGRASP and SeqILS. In this context, when solving Slashdot-based instances in (i), after running both algorithms for 2 hours, ILS has improved the solution quality on larger instances, with more than 2000 vertices (Table 8). Additionally, both SeqGRASP and SeqILS solve the CC problem with low variances in solution value, as seen in the confidence intervals (columns SeqGRASP-CI and SeqILS-CI in Table 8). This confirms the robustness of these heuristics, for the stability of the solution values returned by both algorithms.

If we change the stopping criterion to target solution value [Target I(P)], ILS also outperforms GRASP in average execution time when the instances have more than 2000 vertices (Table 9). In these cases, ILS can reach the target solution value up to 6 times

Instance								SeqGRASP		SeqILS			T(GRASP)
c	**n**	$\|E\|$	**k**	$\mathbf{p}_{in}$	$\mathbf{p}_{-}$	$\mathbf{p}_{+}$	**Target I(P)**	**Avg Time**	**CI(Time)**	**Avg Time**	**CI(Time)**	**Gap Time**	**/ T(ILS)**
4	16	64	16	0.2	0	0	0	0.73	0.26	0.63	0.33	-0.10	1.16
				0.7	0	0	0	0.03	0.01	0.08	0.04	0.05	0.38
4	32	128	20	0.7	0.6	0.6	80.8669	0.67	0.20	0.82	0.34	0.15	0.82
			24	0.7	0.2	0.2	62.8762	1.58	0.56	0.47	0.33	-1.11	3.34
4	64	256	24	0.7	0.2	0.2	99.8465	3.53	1.15	1.15	0.49	-2.38	3.07
4	64	256	24	0.8	0	0	0	1.12	0.41	0.52	0.36	-0.59	2.13
						0.2	86.2275	2.13	0.72	0.85	0.36	-1.28	2.50
						0.4	173.2103	0.69	0.30	0.69	0.30	0.00	1.00
						0.6	166.9959	0.01	0.00	0.18	0.08	0.17	0.08
						0.8	81.5059	0.01	0.00	0.09	0.03	0.08	0.06
						1	0	0.00	0.00	0.05	0.00	0.05	0.07
					0.2	0	49.6108	7.11	3.09	1.27	0.58	-5.84	5.61
						0.2	121.3758	4.06	1.61	0.71	0.28	-3.35	5.72
						0.4	191.5338	1.76	0.58	1.13	0.53	-0.62	1.55
						0.6	224.8849	0.35	0.15	1.42	0.73	1.06	0.25
						0.8	190.6345	1.09	0.38	1.09	0.39	-0.01	1.01
						1	142.315	0.11	0.04	0.59	0.27	0.48	0.19
					0.4	0	84.1122	0.87	0.28	1.61	1.25	0.74	0.54
						0.2	145.6284	7.23	2.69	2.42	1.02	-4.81	2.99
						0.4	202.3622	3.04	1.45	3.13	1.50	0.09	0.97
						0.6	253.4543	0.09	0.02	3.01	1.26	2.92	0.03
						0.8	251.4013	0.16	0.06	3.08	1.20	2.92	0.05
						1	212.095	1.19	0.41	2.23	1.07	1.05	0.53
					0.6	0	76.3863	0.15	0.03	0.59	0.35	0.44	0.25
						0.2	153.0779	6.59	1.99	4.60	1.68	-1.99	1.43
						0.4	193.5687	8.84	5.34	2.64	1.82	-6.21	3.36
						0.6	235.7613	0.36	0.10	6.47	2.93	6.11	0.06
						0.8	235.2052	10.06	4.57	6.34	2.76	-3.73	1.59
						1	270.1083	5.55	2.92	3.98	1.77	-1.57	1.40
					0.8	0	55.4669	0.08	0.01	0.27	0.02	0.19	0.30
						0.2	97.2707	10.30	4.82	2.03	0.82	-8.27	5.06
						0.4	140.8	13.23	9.35	3.22	1.45	-10.00	4.10
						0.6	187.4619	18.12	6.95	3.82	1.64	-14.30	4.74
						0.8	259.2399	24.09	9.48	6.03	3.00	-18.07	4.00
						1	269.8675	7.46	2.68	3.34	2.53	-4.12	2.24
					1	0	0	0.01	0.00	0.16	0.01	0.15	0.09
						0.2	27.2516	15.83	5.27	2.47	1.13	-13.36	6.41
						0.4	67.7761	6.41	1.96	1.18	0.52	-5.23	5.44
						0.6	117.656	15.69	8.45	3.14	1.29	-12.55	5.00
						0.8	174.2849	13.34	8.10	4.31	1.93	-9.03	3.10
						1	232.5436	10.16	5.03	4.43	2.36	-5.73	2.29
4	96	384	24	0.7	0.2	0.2	160.0418	11.08	5.61	1.06	0.61	-10.02	10.43
4	128	512	24	0.7	0.2	0.2	221.71	16.62	7.20	8.32	4.03	-8.30	2.00
25	30	750	20	0.6	0.3	0.3	414.3783	33.16	12.16	3.96	0.74	-29.20	8.37
Average							-	**6.02**	-	**2.26**	-	**-3.75**	**2.40**

Table 7 – Sequential GRASP (SeqGRASP) and sequential ILS (SeqILS) CC results for random instances with predefined community structure in (iv). Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI(Time) is the 95% confidence interval of the execution time, for each algorithm. Gap Time is the gap between SeqILS and SeqGRASP average execution times. T(GRASP)/T(ILS)= [AvgTime(SeqGRASP)/AvgTime(SeqILS)].

	SeqGRASP		SeqILS		
n	**Avg I(P)**	**CI**	**Avg I(P)**	**CI**	**Gap I(P)**
200	45	0	45	0	0
300	54	0	54	0	0
400	57	0	57	0	0
600	109	0	109	0	0
800	240	0	240	0	0
1000	600	0	600	0	0
2000	2,184.76	0.17	2,184.14	0.15	-0.62
4000	6,198.79	0.49	6,193.35	0.57	-5.44
8000	16,076.52	0.88	16,060.55	1.66	-15.97
10000	20,579.29	1.48	20,571.35	5.40	-7.94

Table 8 – Results obtained on Slashdot instances, with sequential GRASP (SeqGRASP) and sequential ILS (SeqILS). Number of vertices: n; Avg I(P): value of the best solution found after 2 hours of execution; CI is the 95% confidence interval of the solution value I(P), for each algorithm; Gap I(P)=[SeqILS Avg I(P)]−[SeqGRASP Avg I(P)].

faster than GRASP.

Figure 14 presents an additional comparison between both algorithms, based on Time-to-Target Plots (TTT-plots) (AIEX; RESENDE; RIBEIRO, 2007), used to analyze the behavior of stochastic algorithms. TTT-plots show, on the y-axis, the probability that an algorithm will find a solution at least as good as a given target solution value within a specific running time, displayed on the x-axis.

One can notice the improved behavior of SeqILS when compared to SeqGRASP. For example, when solving Slashdot $n = 8000$ instance, the probability of SeqILS finding the target solution of 16068 in 2000 seconds is almost equal to 100% while the same probability lies below 65% for SeqGRASP.

The above results lead us to conclude that the potential power of ILS lies in its random and biased sampling (perturbations) of the set of local optima found in the local search step. In most cases, i local searches within ILS can be invoked much faster than if the same i local searches were performed within GRASP's random restart framework. One possible explanation is that fewer movements in the neighborhood are necessary to achieve local optimum from a perturbed solution than from a new solution originated by a greedy random algorithm. In a nutshell, perturbations tend to be faster than simple random restart.

Moreover, SeqILS runs 10 (fixed) multistart iterations (10 constructions), while SeqGRASP runs at least 400 iterations, that is, a minimum of 400 constructions, a very time-expensive step when processing larger graphs, as seen in Figure 15.

Regarding the solution quality, besides random restart, the proposed ILS procedure

Slashdot	Target	SeqGRASP		SeqILS			T(GRASP)
n	**I(P)**	**Avg Time**	**CI**	**Avg Time**	**CI**	**Gap Time**	**/ T(ILS)**
200	45	0.01	0.00	0.02	0.01	0.01	0.45
300	54	0.06	0.02	0.31	0.18	0.25	0.19
400	57	0.19	0.09	1.60	1.19	1.41	0.12
600	109	0.38	0.13	1.52	0.62	1.14	0.25
800	240	0.97	0.28	2.27	0.87	1.30	0.43
1000	600	2.52	0.84	5.22	2.03	2.69	0.48
2000	2185	648.24	213.57	129.85	54.23	-518.38	4.99
4000	6196	1,069.74	609.50	335.80	190.80	-733.94	3.19
8000	16068	1,775.08	627.15	276.57	80.12	-1,498.51	6.42
10000	20580	3,336.05	1,095.30	2,613.07	739.97	-722.98	1.28
Average		**683.32**		**336.62**			**1.78**

Table 9 – Results obtained on Slashdot instances, with sequential GRASP (SeqGRASP) and sequential ILS (SeqILS). Number of vertices: n; Target I(P) is the target imbalance value used as stopping criterion for each algorithm. AvgTime is the average execution time spent (in seconds) by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. Gap Time is the gap between SeqILS and SeqGRASP execution times. T(GRASP)/T(ILS)= [AvgTime(SeqGRASP)/AvgTime(SeqILS)].

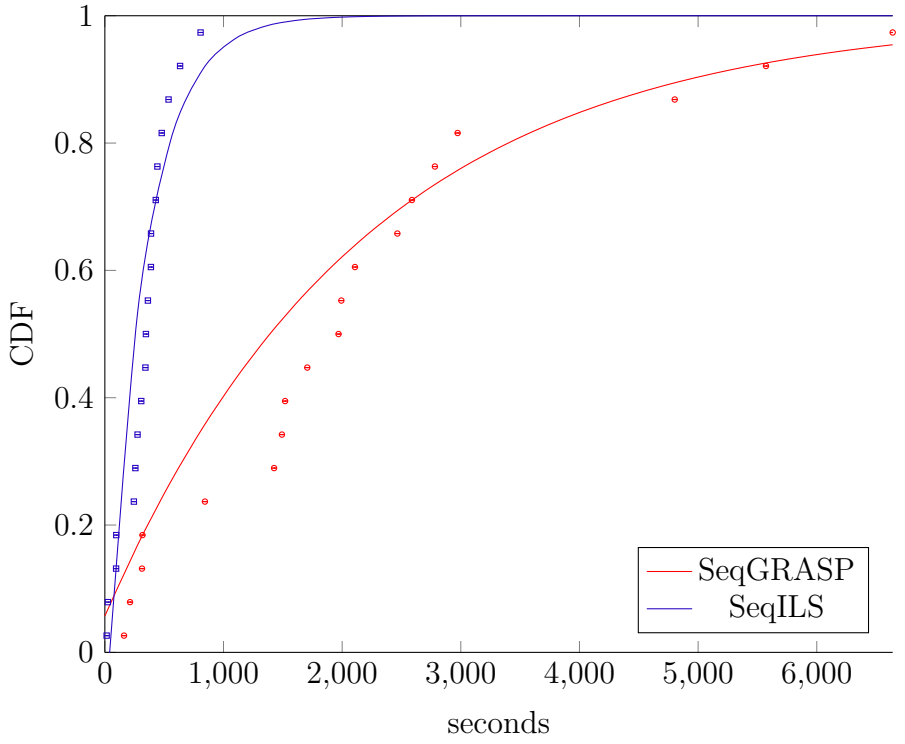


Figure 14 – Time-to-Target Plot (TTT-plot)(AIEX; RESENDE; RIBEIRO, 2007) for SeqGRASP and SeqILS algorithms when solving Slashdot $n = 8000$ instance to obtain the target solution value $I(P) = 16068$. TTT-plots show, on the y-axis, the probability that an algorithm will find a solution at least as good as a given target solution value within a specific running time, displayed on the x-axis.

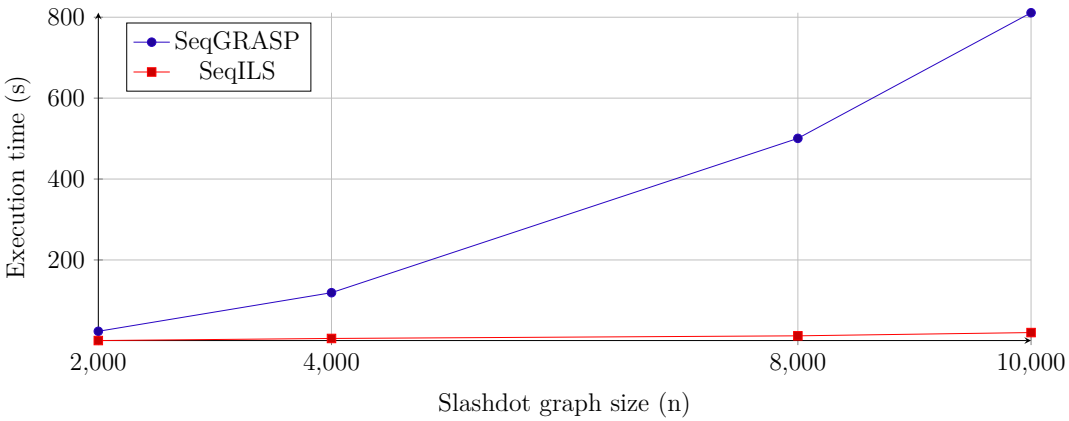


Figure 15 – Average construction time spent (in seconds) on Slashdot instances with SeqGRASP (400 iterations without improvement) and SeqILS (10 multistart iterations). Average of 25 independent executions of each heuristic.

achieves variability through perturbations as well, which explains the improved average solution quality of ILS when compared to GRASP.

3.3.5 Comparison with existing heuristics

Our goal in the next subsections is to verify the limit of Doreian Mrvar and VOTE/BOEM heuristics. Notice that these heuristics are simplified versions of a local search heuristic, so they tend to have inferior performance when compared to metaheuristics. In order to run the tests, we chose to slice a huge real-world network into smaller sizes (Slashdot-based instances in set (i)).

3.3.5.1 Sequential ILS *vs.* Doreian Mrvar Method

In this section we compared our ILS procedure with Doreian Mrvar Method to solve some of the most challenging instances available in the related literature: the Slashdot instances in set (i).

The Doreian Mrvar Method (implemented in Pajek software) presents good-quality results (with respect to solution value and average time spent) for the small-sized literature instances in (i) and also for random social networks of up to 50 vertices [the results are available in Figueiredo and Moura (FIGUEIREDO; MOURA, 2013)]. However, for larger instances, it presents poor performance.

For this benchmark, we have obtained average results of 25 executions. The software Pajek (version 3.14)[1] was invoked with the following configuration: Doreian Mrvar Method,

[1] In order to solve the CC Problem with Pajek, we followed these steps: after loading the network file, we created a random initial partition using $1 - mode$. At this point we defined the number of clusters k of the solution. We have used the same number of clusters of the solution returned by ILS. Finally, we called the resolution method from the menu (Network → Signed Network → Create Partition →

starting from a random partition (1-mode), 1000 repetitions, $\alpha = 0.5$, minimum number of vertices in clusters = 1.

Pajek's Doreian Mrvar Method accepts only one stopping criterion: number of iterations. Since an iteration of this method is fundamentally different from an iteration performed by ILS, the number of iterations is not a sound basis for comparison of both algorithms. Therefore, as an alternative, we have set the time-limit parameter of ILS to 2 hours and the number of repetitions of Doreian Mrvar Method to 1000, which was enough for this method to either reach the same solution value of ILS, or exceed the time limit with an inferior solution value. In this way, we were able to compare both solution values and execution times in a fair manner. As seen in Table 10, when it comes to these larger instances, our ILS procedure can get to the same solution or to a better one at least 13 times faster than Pajek's Doreian Mrvar Method. We do not include Doreian Mrvar Method execution data for instances with $n > 800$, as it was unable to find any solution within 2 hours.

3.3.5.2 Sequential ILS *vs.* VOTE/BOEM

We have implemented the VOTE/BOEM heuristic for the CC Problem (here named $VOTE/BOEM$), comparing the obtained solutions with the previous results from our sequential ILS algorithm. Again, our goal was to solve the Slashdot instances in set (i). The time limit was set to 2 hours for both procedures.

As shown in Table 10, the solution values obtained by sequential ILS are always better than $VOTE/BOEM$. For smaller instances, the faster execution times of $VOTE/BOEM$ are not surprising, since the heuristic is equivalent to only one multistart iteration of the GRASP metaheuristic proposed in Chapter 2. However, as the number of vertices increases, the ILS procedure outperforms VOTE/BOEM. Our sequential ILS algorithm becomes the fastest choice to solve the CC problem when the number of vertices is big ($n \geq 8000$).

3.3.6 Sequential ILS *vs.* Parallel ILS

We have also used the Slashdot instances in (i) to assess the performance of the parallel ILS algorithm. Three different solution methods were applied:

- sequential ILS metaheuristic ($SeqILS$, using only one processor core);
- independent parallel ILS with sequential local search ($ParILS/SeqVND$);

Doreian Mrvar Method), specifying the number of repetitions (we set it to 1000), the alpha parameter (set to 0.5 to match the CC Problem objective function) and the minimum number of vertices in clusters (set to 1).

Instance					*SeqILS*			Doreian Mrvar Method			*VOTE/BOEM*		
n	$\|E^-\|$	$\|E^+\|$	$w(E^-)$	$w(E^+)$	k	BestSol	Time	k	BestSol	Time	k	BestSol	Time
200	62	825	62	825	7	45.0	4.32	3	45.2	61.80	11	47.0	0.42
300	82	981	82	981	9	54.0	9.37	2	54.0	125.00	14	57.0	0.50
400	87	1192	87	1192	6	57.2	9.17	2	57.0	454.40	14	61.0	0.44
600	156	1761	156	1761	9	109.2	13.44	3	119.0	2,740.60	12	111.0	1.14
800	371	2965	371	2965	14	240.1	19.22	4	244.8	7,381.40	33	245.8	2.09
1000	859	5132	859	5132	18	600.1	30.20	-	-	-	44	608.9	4.27
2000	3217	17598	3217	17598	31	2,185.0	99.76	-	-	-	105	2,264.2	43.86
4000	8664	40868	8664	40868	44	6,195.2	528.44	-	-	-	147	6,216.5	395.10
8000	22789	86916	22789	86916	80	16,065.0	2,208.41	-	-	-	375	16,115.8	3,475.91
10000	29805	109266	29805	109266	95	20,580.4	4,277.18	-	-	-	169	23,458.4	7,041.97

Table 10 – Sequential ILS for the CC Problem, here named $SeqILS$, *vs.* Doreian Mrvar Method *vs.* $VOTE/BOEM$ results obtained on Slashdot signed graphs. Average number of clusters (k) in the solution; BestSol is the average value of the best solution found and Time is the average execution time (in seconds), after 25 executions of each algorithm.

- parallel ILS with parallel local search ($ParILS/ParVND$).

The parameters of the procedures used in the parallel approaches of ILS are the same used for testing the sequential algorithm, except for the number of ILS multistart iterations. Additionally, the number of processes used in the experiments was chosen following the best parallel results from the previous chapter. For independent parallel ILS ($ParILS/SeqVND$), in order to perfectly divide the number of multistart iterations of the original ILS procedure, 10 master processes execute one multistart iteration of ILS each.

In $ParILS/ParVND$, similarly to Parallel GRASP, we chose to group each ILS master process together with its corresponding VND search slaves. As previously explained in Chapter 2, since each machine has 2 quad-core CPUs (8 processor cores), $ParILS/ParVND$ was tested with 8 processes running in parallel, out of which 2 processes are ILS masters (each one running 5 ILS iterations) and the other 6 processes consist of VND search slaves, 3 for each master.

Finally, in order to compare the sequential and parallel procedures, the following additional stopping criterion was applied: all procedures stop whenever the specified target solution value [Target I(P)] is found.

On average, the independent parallel ILS ($ParILS/SeqVND$) with 10 processes is the best solution method. As shown in Table 11, it can solve the Slashdot instances in set (i) up to 6 times faster than sequential ILS, with an average efficiency of 49%.

On the other hand, the results obtained with parallel ILS with parallel local search ($ParILS/ParVND$) have no such performance improvement as one would expect. Using 8 processes to execute the algorithm (2 ILS masters and 3 VND search processes per master)

Instance		SeqILS		ParILS/SeqVND(10)				ParILS/ParVND(8)			
	Target I(P)	Avg Time	CI	Avg Time	CI	Speedup	Efficiency	Avg Time	CI	Speedup	Efficiency
200	45	4.32	0.17	2.68	0.66	1.61	16.13%	3.36	0.47	1.29	16.08%
300	54	9.37	5.83	1.67	0.18	5.63	56.29%	3.91	0.34	2.39	29.93%
400	57	9.17	0.48	1.67	0.35	5.49	54.86%	4.54	0.50	2.02	25.23%
600	109	13.44	3.81	3.33	0.90	4.04	40.36%	4.74	0.37	2.83	35.41%
800	240	19.22	0.93	4.33	1.13	4.44	44.38%	6.40	0.33	3.00	37.55%
1000	600	30.20	1.35	5.75	0.41	5.25	52.55%	9.49	0.46	3.18	39.78%
2000	2185	99.76	6.14	18.39	1.82	5.43	54.25%	57.24	2.30	1.74	21.79%
4000	6195	528.44	35.20	91.14	7.75	5.80	57.98%	330.80	15.25	1.60	19.97%
8000	16065	2,208.41	165.49	425.57	36.25	5.19	51.89%	590.40	12.92	3.74	46.76%
10000	20580	4,277.18	291.29	709.04	47.25	6.03	60.32%	1,012.47	32.03	4.22	52.81%
Average						**4.89**	**48.90%**			**2.60**	**32.53%**

Table 11 – CC results obtained on Slashdot signed graphs by the use of different metaheuristic approaches: Sequential ILS ($SeqILS$), Parallel ILS with sequential VND ($ParILS/SeqVND(np)$) and Parallel ILS with parallel VND ($ParILS/ParVND(np)$), where np is the number of processes in parallel. Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm.

resulted in an average speedup of 2.6 (average efficiency of 33%) for Slashdot instances. Similarly to what happened to Parallel GRASP in the previous chapter, the additional computational resources required to run parallel local search via message passing are not worth the available acceleration and efficiency brought by this procedure.

As noted in the previous chapter, this degradation in speedup and efficiency is expected, since the $ParallelLocalSearch$ algorithm causes an overhead in the number of messages exchanged between processes.[2]

3.3.7 Parallel ILS *vs.* Parallel GRASP

The last question that comes to mind is knowing which parallel metaheuristic presents the best efficiency and possibly the best solution values. First, when comparing the results over Slashdot instances in (i), $ParILS/SeqVND$ is up to 4 times faster than $ParGRASP/SeqVND$ when the stopping criterion is a specific target solution (Table 12).

To compare the behavior of $ParGRASP/SeqVND$ and $ParILS/SeqVND$ algorithms, we present a Time-to-Target Plot (TTT-plot) (AIEX; RESENDE; RIBEIRO, 2007) when solving Slashdot $n = 8000$ instance (Figure 16). The improved behavior of $ParILS/SeqVND$ is evident. For example, the probability of $ParILS/SeqVND$ finding

[2] Please note that the loss of efficiency when using parallel local search is not caused by the lack of work for the VND search slaves. For example, when solving Slashdot instance of size $n = 10000$ with $ParILS/ParVND$, local search performs more than 500 million neighborhood evaluations, and when solving random instances with $n = 600$, this number rises to 8 billion.

n	Target I(P)	ParGRASP/SeqVND(8)		ParILS/SeqVND(10)			T(GRASP) / T(ILS)
		Avg Time	CI	Avg Time	CI	Gap Time	
2000	2187	10.83	3.74	4.18	2.08	-6.64	2.59
4000	6203	64.03	37.36	20.33	3.55	-43.70	3.15
8000	16087	347.27	222.29	88.33	23.61	-258.94	3.93
10000	20589	985.85	786.81	246.33	199.92	-739.52	4.00
Average	**-**	**351.99**	**-**	**89.79**	**-**	**-262.20**	**3.42**

Table 12 – CC results obtained on Slashdot signed graphs by the use of the following metaheuristic approaches: Parallel GRASP with Sequential VND ($ParGRASP/SeqVND(np)$) and Parallel ILS with Sequential VND ($ParILS/SeqVND(np)$), where np is the number of processes in parallel. Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. T(GRASP)/T(ILS)= [AvgTime(ParGRASP/SeqVND)/AvgTime(ParILS/SeqVND)].

the target solution of 16087 in 400 seconds is almost equal to 100% while the same probability lies below 90% for $ParGRASP/SeqVND$.

Finally, for the random instances in (iii), as seen in Table 13, $ParILS/SeqVND$ is always faster than $ParGRASP/SeqVND$ when finding the specified target solution values, being on average 3 times faster.

3.4 Summary

Based on the previously presented results, the sequential ILS is the best algorithm to solve the UNGA instances in (ii). Moreover, the parallel ILS with sequential VND ($ParILS/SeqVND$) is the fastest and most efficient metaheuristic to solve the real-world Slashdot instances in (i) and the random instances in (iii).

Additionally, both GRASP and ILS are more efficient than the existing heuristic implemented in Pajek, but only ILS was able to outperform the VOTE/BOEM heuristic when solving larger networks like Slashdot.

The computational results indicate that the proposed multistart ILS metaheuristic consists of a fast and efficient approach to solve the CC Problem, outperforming, in processing time, the GRASP metaheuristic proposed earlier, with similar or improved solution quality.

A possible explanation for ILS superiority over GRASP on bigger instances lies on the fact that ILS executes more local searches and less initial solution constructions than GRASP. Since GRASP's random restart framework generally costs more when instances grow in size (building an initial solution becomes more costly), ILS outperforms GRASP and presents similar or improved solution quality when solving larger networks.

				Target	ParGRASP/SeqVND(8)		ParILS/SeqVND(10)			
n	**e**	**d**	**d-**	**I(P)**	**AvgTime**	**CI**	**AvgTime**	**CI**	**Gap time**	**Speedup**
100	990	0.1	0.2	198	0.00	0.00	0.00	0.00	0.00	1.16
			0.5	228	0.07	0.02	0.04	0.01	-0.02	1.55
			0.8	56	0.10	0.03	0.05	0.01	-0.06	2.18
	1980	0.2	0.2	396	0.00	0.00	0.00	0.00	0.00	1.21
			0.5	582	0.28	0.10	0.15	0.08	-0.14	1.91
			0.8	208	0.24	0.09	0.09	0.02	-0.15	2.71
	4950	0.5	0.2	990	0.00	0.00	0.00	0.00	0.00	1.08
			0.5	1838	0.22	0.07	0.13	0.04	-0.09	1.72
			0.8	734	0.76	0.19	0.18	0.04	-0.58	4.28
	7920	0.8	0.2	1584	0.00	0.00	0.00	0.00	0.00	1.17
			0.5	3124	0.42	0.16	0.40	0.15	-0.02	1.05
			0.8	1308	0.42	0.15	0.16	0.03	-0.26	2.62
200	3980	0.1	0.2	792	0.01	0.00	0.00	0.00	0.00	1.15
			0.5	1206	0.55	0.19	0.33	0.11	-0.22	1.67
			0.8	402	1.11	0.35	0.31	0.06	-0.80	3.53
	7960	0.2	0.2	1592	0.01	0.00	0.00	0.00	0.00	1.16
			0.5	2852	0.80	0.39	0.24	0.08	-0.56	3.29
			0.8	1098	1.08	0.31	0.35	0.08	-0.74	3.12
	19900	0.5	0.2	3980	0.01	0.00	0.01	0.00	0.00	1.14
			0.5	8140	1.43	0.61	0.65	0.30	-0.77	2.18
			0.8	3354	2.75	1.10	0.66	0.12	-2.09	4.17
	31840	0.8	0.2	6368	0.01	0.00	0.01	0.00	0.00	1.22
			0.5	13586	1.81	0.97	0.62	0.22	-1.18	2.89
			0.8	5696	4.70	1.46	0.93	0.18	-3.77	5.05
300	8970	0.1	0.2	1794	0.01	0.00	0.01	0.00	0.00	1.10
			0.5	3040	1.36	0.56	0.50	0.14	-0.86	2.72
			0.8	1122	1.71	0.67	0.60	0.11	-1.12	2.87
	17940	0.2	0.2	3588	0.01	0.00	0.01	0.00	0.00	1.15
			0.5	6862	1.53	0.58	0.67	0.30	-0.86	2.29
			0.8	2758	3.58	1.20	0.86	0.15	-2.72	4.18
	44850	0.5	0.2	8970	0.02	0.00	0.01	0.00	0.00	1.22
			0.5	19080	7.14	4.31	1.40	0.68	-5.73	5.08
			0.8	7928	6.88	1.96	1.27	0.16	-5.61	5.42
	71760	0.8	0.2	14352	0.02	0.00	0.02	0.00	0.00	1.16
			0.5	31698	7.84	4.15	2.33	1.05	-5.51	3.36
			0.8	13212	10.39	3.58	1.98	0.36	-8.41	5.25
400	15960	0.1	0.2	3192	0.01	0.00	0.01	0.00	0.00	1.10
			0.5	5746	3.19	1.50	0.91	0.33	-2.28	3.49
			0.8	2204	3.83	1.51	1.29	0.25	-2.55	2.98
	31920	0.2	0.2	6384	0.02	0.00	0.01	0.00	0.00	1.16
			0.5	12778	2.95	1.09	0.84	0.26	-2.11	3.52
			0.8	5180	11.15	3.50	1.59	0.37	-9.55	6.99
	79800	0.5	0.2	15960	0.03	0.00	0.03	0.00	0.00	1.13
			0.5	34764	9.59	5.49	3.00	0.94	-6.59	3.20
			0.8	14476	14.59	4.16	2.34	0.44	-12.25	6.23
	127680	0.8	0.2	25536	0.05	0.00	0.04	0.00	-0.01	1.16
			0.5	57342	22.78	11.94	5.58	2.52	-17.20	4.08
			0.8	23922	22.73	6.05	4.16	0.80	-18.58	5.47
600	35940	0.1	0.2	7188	0.03	0.00	0.02	0.00	0.00	1.13
			0.5	13802	6.72	3.94	1.28	0.37	-5.45	5.26
			0.8	5484	13.29	4.12	2.74	0.52	-10.55	4.85
	71880	0.2	0.2	14376	0.04	0.00	0.03	0.00	0.00	1.13
			0.5	30046	17.30	7.93	3.20	1.11	-14.10	5.40
			0.8	12362	15.20	5.64	3.76	0.75	-11.44	4.04
	179700	0.5	0.2	35940	0.08	0.00	0.06	0.01	-0.02	1.26
			0.5	80522	54.04	34.43	6.72	3.16	-47.33	8.05
			0.8	33524	34.81	9.99	7.68	1.74	-27.12	4.53
	287520	0.8	0.2	57504	0.11	0.01	0.09	0.01	-0.02	1.17
			0.5	132006	116.92	88.85	24.61	15.92	-92.30	4.75
			0.8	54872	58.45	15.50	11.43	2.45	-47.02	5.11
Average				**-**	**-**	**-**	**-**	**-**	**-6.15**	**2.94**

Table 13 – Parallel GRASP ($ParGRASP/SeqVND(8)$) and parallel ILS ($ParILS/SeqVND(10)$) results for random instances in (iii). Target I(P) is the target imbalance value used as stopping criterion for each algorithm. Avg Time is the average execution time (in seconds) spent by each algorithm to reach the specified target solution value, after 25 independent executions. CI is the 95% confidence interval of the execution time, for each algorithm. Gap time is the gap between parallel ILS and parallel GRASP execution times. Speedup= [AvgTime(ParGRASP)/AvgTime(ParILS)].

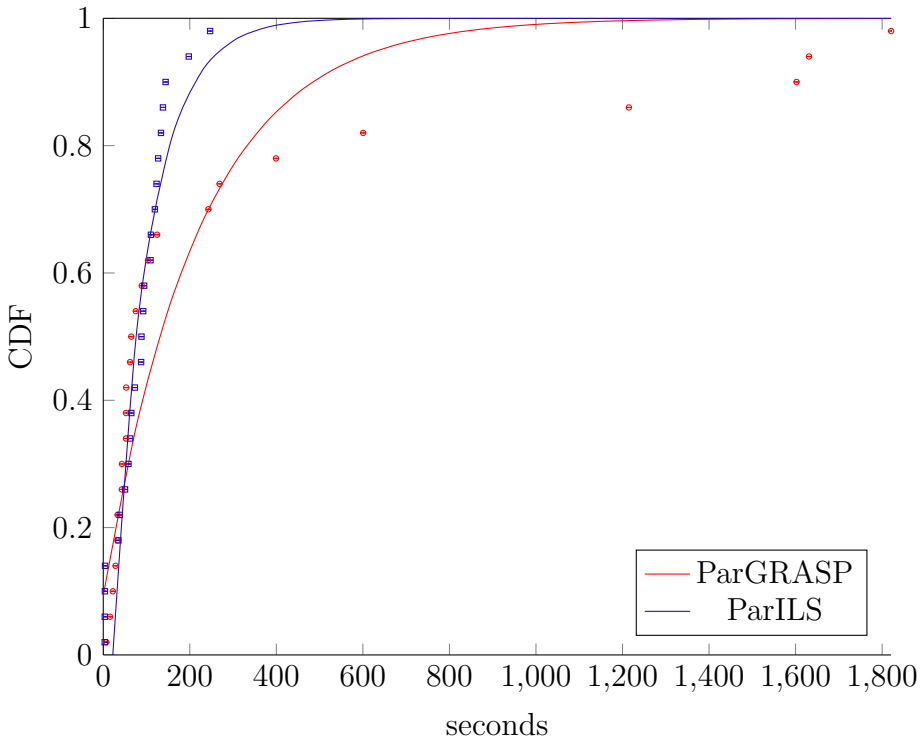


Figure 16 – Time-to-Target Plot (TTT-plot)(AIEX; RESENDE; RIBEIRO, 2007) for ParGRASP and ParILS algorithms when solving Slashdot $n = 8000$ instance to obtain the target solution value $I(P) = 16087$. TTT-plots show, on the y-axis, the probability that an algorithm will find a solution at least as good as a given target solution value within a specific running time, displayed on the x-axis.

Even better results can be obtained if the ILS modules are optimized. First, the acceptance criteria, defined to accept only solutions that improve the objective function value, can be modified in search of a better balance between intensification and diversification. It means they could accept solutions that are worse than the best current solution, so as to widen the search space exploration. In addition, the perturbation procedure can be adjusted to incorporate several kinds of problem-specific information, following a good practice that says that a good perturbation transforms an excellent solution into an excellent starting point for a local search.

4 Parallelizing local search in CUDA

An observation of the great amount of time spent on the local search phase of the aforementioned GRASP and ILS algorithms (Figure 17) reveals the possibility to enhance their performance by extending both of them with a new implementation of local search, that would be capable of solving the problem faster, without altering the behavior of the metaheuristic.

With this in mind, we developed an improved local search procedure for the CC problem, using the parallelism offered by GPGPU (General Purpose Graphics Processing Unit).

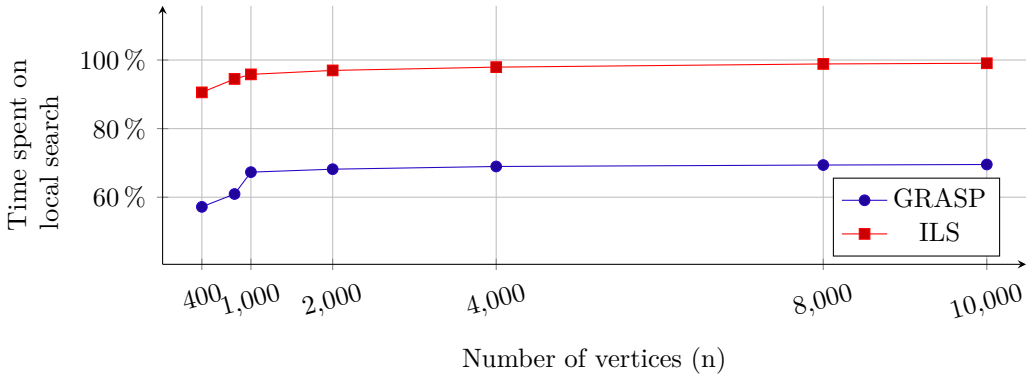


Figure 17 – Average time spent by sequential GRASP and ILS on r=1-opt local search on Slashdot-based signed graphs, after 25 independent executions.

4.1 Using General Purpose GPUs to solve optimization problems

The use of Graphics Processing Units (GPUs) has been extended to a wide range of application domains (e.g. computational science) thanks to the publication of the CUDA (Compute Unified Device Architecture) development toolkit (NVIDIA, 2015), which allows GPU programming in C-like language. When used as general-purpose computing devices, GPUs can efficiently accelerate many non-graphics programs, especially vector-and matrix-based codes that exhibit lots of parallelism with low synchronization requirements. Because their hardware is primarily designed to perform complex computations on blocks of pixels at high speed and with wide parallelism, GPU architectures differ substantially from conventional CPU hardware. Therefore, writing efficient programs to solve combinatorial optimization problems on GPUs is not a straightforward task and requires a huge effort not only at design but also at implementation level. Indeed, several challenges mainly related to the hierarchical memory management have to be dealt with. The major issues consist of

efficient distribution of data processing between CPU and GPU, thread synchronization, optimization of data transfer between the different memories, as well as the capacity constraints of these memories (LUONG; MELAB; TALBI, 2013).

Whenever parallel algorithms are applied to solve optimization problems, it is worth noticing that, in general, for distributed architectures, the global performance in metaheuristics is limited by high communication latencies. However, in GPU architectures, performance is bounded by memory access latencies. This being said, several works have already demonstrated the potential speedups when using GPUs to accelerate metaheuristics. For example, GRASP, ILS and evolutionary algorithms have already been adapted to use local search procedures implemented in GPGPU. Table 14 lists some results available in the literature.

Author	**Main contribution**	**Speedup**
Kruger et al. (KRÜGER et al., 2010)	Generic local search (memetic) algorithm	Between x70 and x120
Fujimoto et al. (FUJIMOTO; TSUTSUI, 2011)	Highly-parallel TSP solver for a GPU computing platform	Up to x24.2
Coelho et al.(COELHO et al., 2012)	Single Vehicle Routing Problem with Deliveries and Selective Pickups in CPU-GPU	From x2.73 to x16.23
Rocki et al.(ROCKI; SUDA, 2012)	Accelerating TSP 2 and 3-opt local search using GPU	From x3 to x26 compared to parallel CPU w/ 32 cores
Van et al.(LUONG; MELAB; TALBI, 2013)	GPU computing for parallel local search algorithms	From x0.5 up to x73.3
Pena et al.(PENA et al., 2014)	Parallel algorithm for Siting Observers on Terrain problem	More than x20

Table 14 – Speedups obtained when using GPGPUs to accelerate metaheuristics.

4.2 GPGPU architecture and the CUDA programming model

CUDA has made possible the development of algorithms to solve time-consuming problems using the large number of parallel multiprocessors as well as the high memory bandwidth provided by NVIDIA GPUs. To accomplish high-performance computing, it is necessary to develop parallel algorithms that are partially or totally executed on the GPU. The CUDA-enabled graphics cards are composed of multiple processors, more specifically, Single Instruction Multiple Data (SIMD) processors called Stream Multiprocessors (SMs), which allow the execution of multiple parallel threads. Thus, GPU processors can efficiently execute instructions involving operations with data parallelism, when the same operation is applied to different data.

Depending on the algorithm, GPUs can provide greater processing power than CPUs because they are specialized in performing parallel tasks involving many calculations. On the other hand, CPUs are optimized for execution flow control and data cache. The physical difference between both architectures can be visualized in Figure 18: GPUs dedicate most of their area for processing units (cores), while CPUs dedicate most of their area for execution control and data cache.

A CUDA application consists in code that is executed on CPU and functions (called kernels) that are executed on GPU. The GPU is able to do parallel processing by creating threads such that each thread may execute the kernel operations on different data. This way, the GPU is used as a coprocessor to perform certain tasks more efficiently than the CPU. The GPU processing units (CUDA cores) are grouped to share a single instruction unit, so that threads mapped on these cores execute the same instruction each cycle, but on different data. Each logical group of threads sharing instructions is called a warp. Moreover, threads belonging to different warps can execute different instructions on the same cores, but in a different time slot. In practice, CUDA cores are time-shared between warps, and a group of threads in a warp performs as a SIMD unit.

Moreover, modern GPU architectures relax SIMD constraints by allowing threads in a given warp to execute different instructions (i.e. if-then-else statements and loop-termination conditions). However, these varying instructions cannot be executed concurrently, since each SIMD unit must execute the same instruction on all cores. This way, the instructions are serialized in time, which can severely degrade performance. This situation is called (thread) divergence.

Another major concern about CUDA implementation which greatly impacts performance is memory access. Bottlenecks can appear not only during data transfer between host (CPU) and device (GPU) memory, but also during memory access on the device; namely, data locality is very important. Memory requests exhibiting spatial locality are maximally coalesced. For example, accesses to addresses i and $i + 1$ are served by a single memory fetch, as long as they are aligned. Depending on the accessed addresses, concurrent memory requests from multiple threads from a warp can exhibit undesired effects. Different threads writing to the same memory address will exhibit non-deterministic behavior (it is not possible to determine which value will be actually written). Non-coalesced memory requests (including atomic ones) will be serialized in a nondeterministic order. This last behavior, often called the scattering access pattern, greatly reduces memory throughput, since each memory request utilizes only a few bytes from each memory fetch.

The CUDA programming model includes the notion of shared memory and thread blocks, a reflection of the underlying hardware architecture as shown in Figure 19. All threads in a thread block can access the same shared memory, which provides lower latency and higher bandwidth access than global GPU memory but is limited in size. Threads in

a thread block may also communicate with each other via this shared memory.

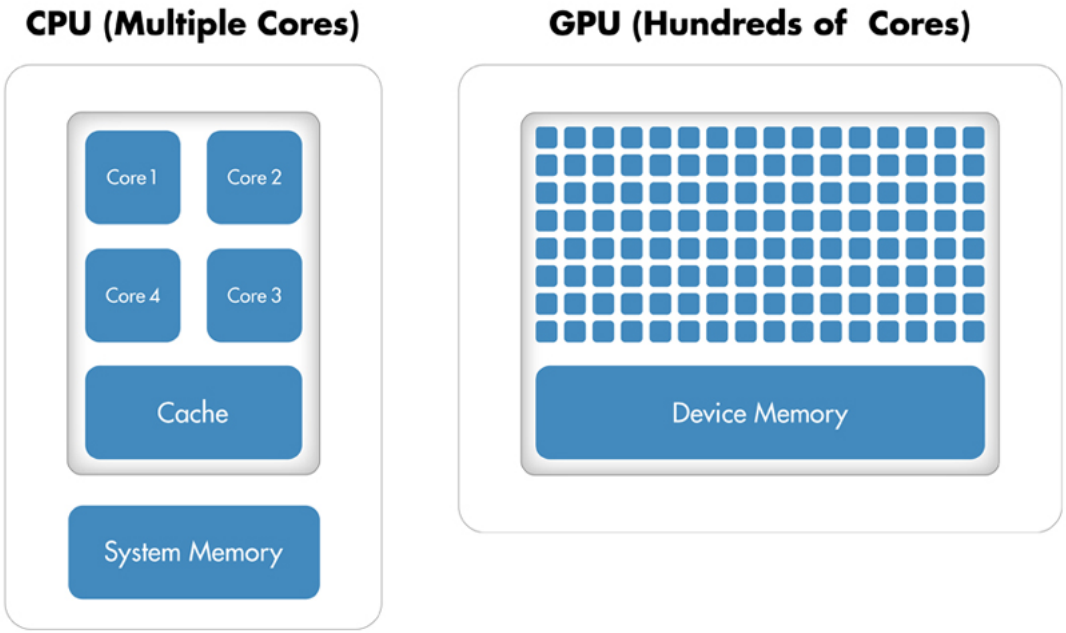


Figure 18 – Basic structure of a typical CPU (left) and GPU (right).

Memory type	Access latency	Size
Global	Medium	Big
Registers	Very fast	Very small
Local	Medium	Medium
Shared	Fast	Small
Constant	Fast (cached)	Medium
Texture	Fast (cached)	Medium

- Highly parallel multi-threaded many core
- High memory bandwidth compared to CPU
- Different levels of memory (different latencies)

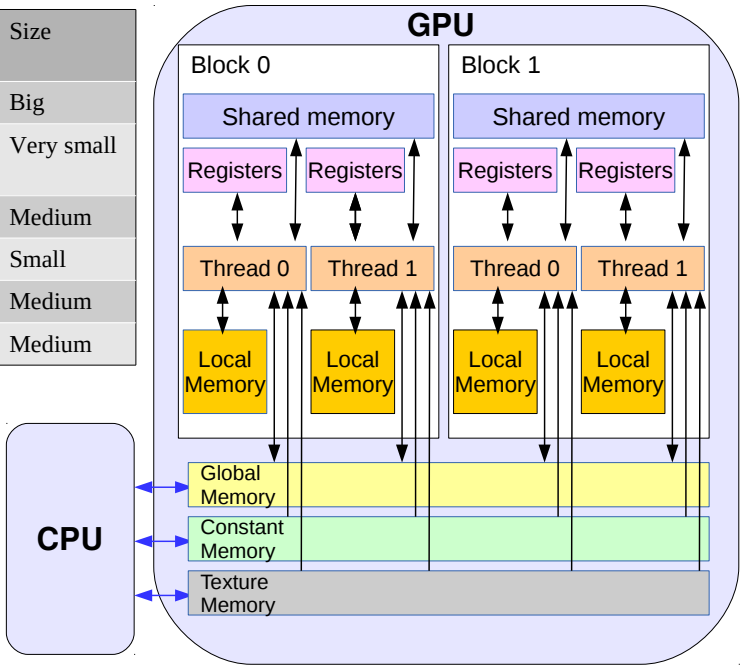


Figure 19 – GPU Memory Hierarchy (MELAB; TALBI et al., 2011).

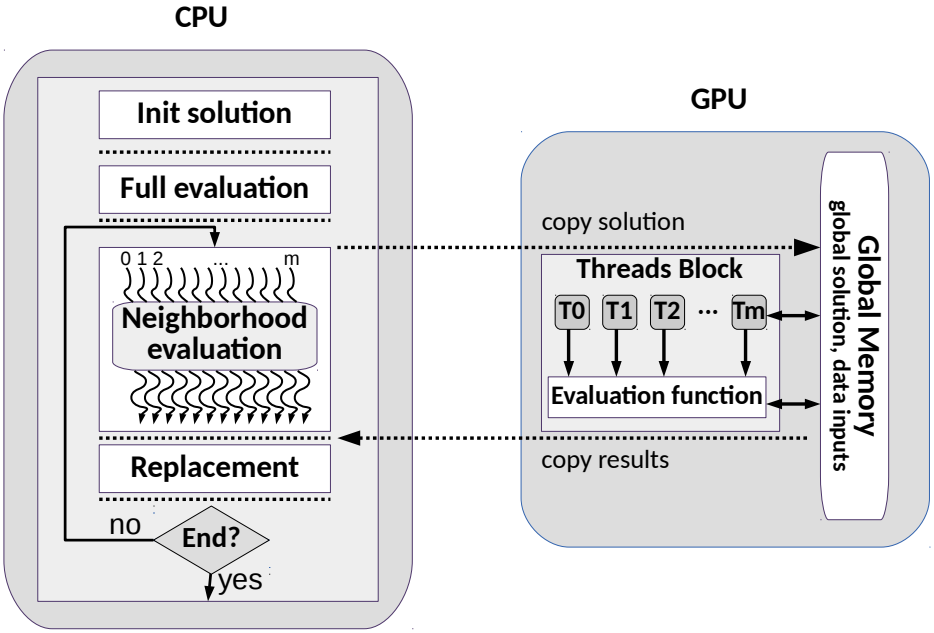


Figure 20 – CUDA local search parallelization scheme (MELAB; TALBI et al., 2011).

4.3 Modifying the search algorithm to run in the GPU

Our approach to parallelize the local search procedure followed the Iteration-level Parallel Model (LUONG; MELAB; TALBI, 2013). As can be seen on Figure 20, the evaluation of the neighborhood is made in parallel. At the beginning of each iteration, the master thread, that runs on the CPU, makes the current solution available to all threads of the GPU. Each of them evaluates a specific movement in the neighborhood of candidates, and the results are returned back to the master.

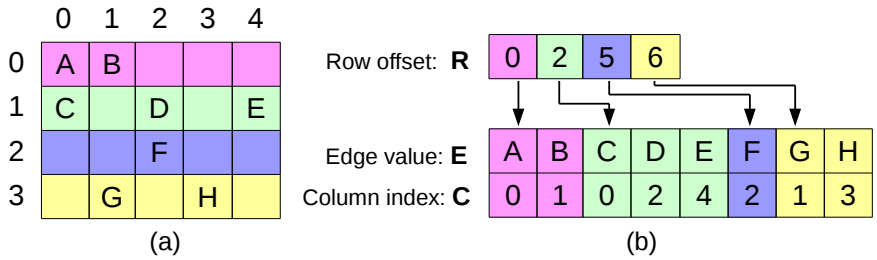


Figure 21 – Using the Compressed Sparse Row (CSR) format to store graph's adjacency matrix. This representation consists of two arrays. The column indices array is a concatenation of each vertex's adjacency list into an array of m elements. The row offsets array is an $n + 1$ element array that points at where each vertex's adjacency list begins and ends within the column indices array.

At this point, it is important to list some optimizations in the Correlation Clustering local search algorithm that have been applied for the code to run efficiently in the GPU.

First of all, the graph had to be stored in Compressed Sparse Row format (Figure 21), in order to save space and avoid unnecessary data transfers between host (CPU) and device (GPU) memory. Also, since it is impossible to store the graph in shared memory (as of 2015, shared memory size is limited to 112kB per multiprocessor), the graph is copied to the (slower) GPU global memory. It is then used to calculate matrices that contain the sum of edge weights between vertex i and every cluster k in the current solution. As we are processing a signed graph, there are two sum matrices: one for positive edges and the other for negative edges (*positiveSum* and *negativeSum*, respectively). These auxiliary matrices, also stored in GPU global memory, contain all the information needed to evaluate the imbalance of a new clustering configuration, without the need to traverse the graph, thus saving GPU memory accesses and execution time. It is worth noting that the use of these auxiliary matrices in CUDA local search consists of an improvement to the sequential local search algorithm, and the same structures have been applied in a second version of the local search that runs on the CPU, thus enhancing the original algorithms. All GRASP and ILS results listed in the previous chapters are related to this new version of local search.

4.4 CUDA local search kernel implementation

Algorithm 7: $1OptLocalSearchKernel$

```
1  Input: positiveSum[], negativeSum[], cluster[], currentImbalance, number of clusters (c), vertices
   (n)
2  Output: destImbalance[]
3  i = idx  mod n;          → The number of vertex i is derived from each thread's unique identifier
   (idx)
4  k2 = idx div n;          → Vertex i is being moved to cluster k2
5  if (i ≤ n and k2 ≤ c + 1)
6     k1 = cluster[ i ];    → obtains the cluster number of vertex i
7     /* calculates only the difference in positive and negative imbalance */
8     positiveSum = - positiveSum[ i + k2 × n ] + positiveSum[ i + k1 × n ];
9     negativeSum = - negativeSum[ i + k1 × n ] + negativeSum[ i + k2 × n ];
10    destImbalance[ idx ] = currentImbalance + positiveSum + negativeSum;
```

In order to test the viability of a parallel local search procedure for the CC problem running on GPU, we only implemented the neighborhood of size one, that is, moving one vertex at a time to a different cluster. Algorithm 7 presents the kernel pseudocode for CUDA CC 1-opt local search kernel and Figure 22 summarizes the work executed. Each thread running in the GPU (uniquely identified by idx) is responsible for calculating the delta of imbalance caused by moving a specific vertex i to a different cluster, for example, in the range k_1 to k_c. Afterwards, another kernel performs a reduction of the results, also in parallel, returning the best move for this specific local search.

Finally, whenever a vertex move is applied due to an improvement in imbalance, a third CUDA kernel is invoked to update the clustering configuration and the vertex-cluster edge-weight-sum auxiliary matrices (*positiveSum* and *negativeSum*) after a change in the clustering [1]. This update is a necessary step to allow a new execution of the local search procedure (new local search iteration), as long as the obtained clustering solution brings an improvement in imbalance.

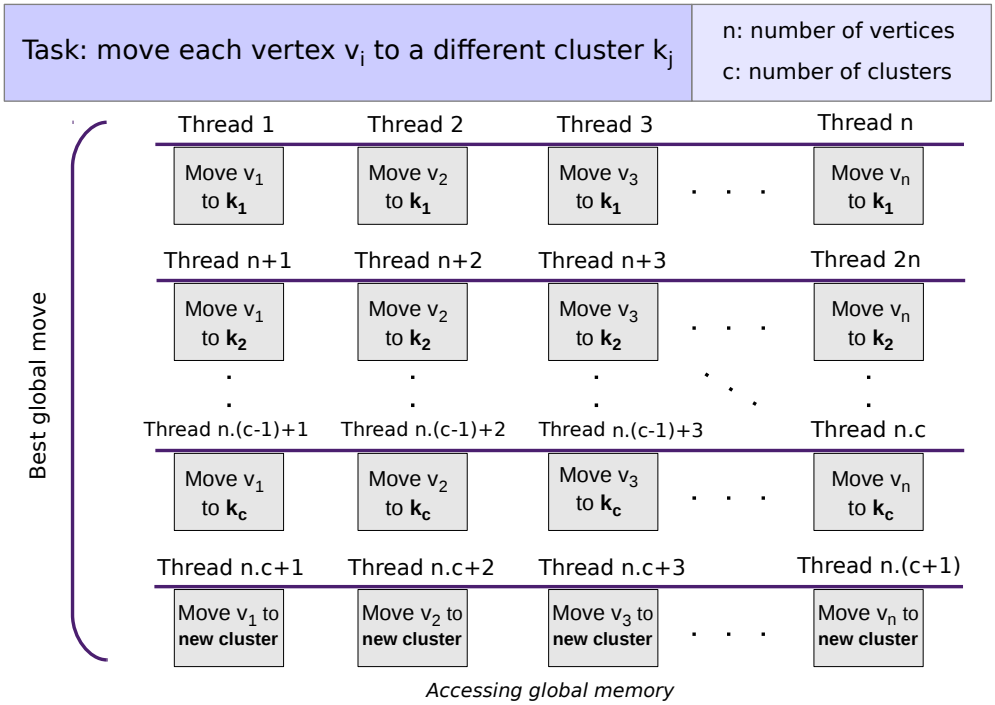


Figure 22 – GPU thread work representation for 1-opt local search. Each thread *idx* is responsible for moving vertex i to a different cluster, from k_1 to k_c, and to a new cluster $(k_c + 1)$.

4.5 Experiments performed with CUDA local search

The previously described algorithms were implemented in ANSI C++ and "C for CUDA V6.5" (NVIDIA, 2015) programming environment. All experiments were performed (with exclusive access) on a workstation with an Intel Core i7 QuadCore processor @3.40GHz (only one CPU core used), 32GB of RAM and NVIDIA Tesla K40 GPU (containing 12GB of memory and 2880 CUDA cores), under Ubuntu Linux 12.04. All heuristic outcomes are average results of 25 independent executions. Speedups are computed by dividing the sequential CPU time with the parallel time, which is obtained with the same CPU and the GPU acting as a co-processor.

[1] The update of the auxiliary matrices in the GPU is preferred, since it also benefits from CUDA parallelism.

Computational experiments were carried out on (i) a set of 24 random instances, and (ii) a set of 4 social networks from the literature. Next, we briefly describe these instances[2].

(i) We generated random social networks with $n \in \{400, 600\}$, varying network density $d = 2 \times |E|/(n^2 - n)$ and negative graph density defined here as $d^- = |E^-|/|E|$. For each value of n, we considered a set of 12 random instances having d and d^- ranging, respectively, in sets $\{0.1, 0.2, 0.5, 0.8\}$ and $\{0.2, 0.5, 0.8\}$.

(ii) This set of instances is composed by 4 signed networks extracted from the large scale social network representing the technology-related news website Slashdot (LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010; FACCHETTI; IACONO; ALTAFINI, 2011), containing the first n vertices, with $n \in \{2000, 4000, 8000, 10000\}$.

4.5.1 Sequential GRASP *vs.* Sequential GRASP with CUDA local search

In this section, we present the experiments performed with the sequential GRASP algorithm (SeqGRASP) in its best configuration, available in Chapter 2, and the sequential GRASP with CUDA Parallel Local Search (SeqGRASP/CUDALS), when solving random instances (Table 15) and Slashdot instances (Table 16). Both experiments used the following set of parameters:

Time limit	Alpha	Neighborhood	Number of iterations without improvement
2 hours	$\alpha = 1.0$	$r = 1$	$iter = 400$

4.5.2 Sequential ILS *vs.* Sequential ILS with CUDA local search

Here we list the results of the experiments performed with the sequential ILS algorithm (SeqILS) in its best configuration, available in Chapter 3, and the sequential ILS with CUDA Parallel Local Search (SeqILS/CUDALS), when solving random instances (Table 15) and Slashdot instances (Table 16). The following configuration was used in the ILS procedure.

Time limit	Alpha	Neighborhood	Iterations	ILS iterations	Perturbation level
2 hours	$\alpha = 1.0$	$r = 1$	$iter = 10$	$iterMaxILS = 5$	$perturbMax = 30$

4.5.3 Analysis of results

Our intent was to design an efficient parallelization strategy for the implementation of a parallel local search procedure for the Correlation Clustering problem on GPU. After applying the procedure, known as CUDALS, in existing GRASP and ILS procedures for

[2] all instances are available in http://www.ic.uff.br/∼ yuri/files/CCinst.zip.

the CC problem, our experimental results showed significant speedups, outperforming, in processing time, the equivalent algorithm with local search implemented in the CPU.

The GRASP/CUDALS algorithm presented an average speedup of x1.6 (up to x3) on random instances and x1.7 (up to x2) on Slashdot instances, while the ILS/CUDALS showed an average speedup of x13 (up to x47) on random instances and x6.2 (up to x10.3) on Slashdot instances. In both algorithms, the solution quality was equal or close to their sequential counterparts. Considering only the local search execution time, as Table 17 shows, it is also possible to measure the speedup obtained by CUDALS alone (column *LS Speedup*), when running inside the GRASP and ILS procedures. On Slashdot instances, in comparison with the sequential local search running on CPU, the CUDALS procedure presents an average speedup of x2.5 (maximum of x3.5) when running inside SeqGRASP and an average speedup of x7.3 (maximum of x12.5) when running inside SeqILS.

It is worth noting that the sublinear speedups obtained when using both procedures to solve random instances with $d^- = 0.2$ can be explained by their low negative edge density, which significantly reduces the local search space. As a consequence, these instances are always solved in a short time (less than a minute) and most of the time spent by CUDA local search is due to the overhead of memory copies between CPU and GPU.

An extension of this work could focus on improving the analysis of larger signed social networks. The numerical experience indicates that, in order to handle instances larger than Epinions (131,828 vertices and 841,372 edges) or Slashdot (82,144 vertices and 549,202 edges) networks, there is the need to develop better parallelization strategies. One possible approach is implementing a hybrid application, using the parallelism available both in CPU (multicore) and GPU (CUDA).

n	$\|E\|$	$\|E^+\|$	$\|E^-\|$	d	d^-	**SeqGRASP**		**SeqGRASP/CUDALS**				**SeqILS**		**SeqILS/CUDALS**			
						Avg I(P)	**Avg Time**	**Avg I(P)**	**Gap %I(P)**	**Avg Time**	**Speedup**	**Avg I(P)**	**Avg Time**	**Avg I(P)**	**Gap %I(P)**	**Avg Time**	**Speedup**
400	15960	12768	3192	0.1	0.2	3,192.00	2.44	3,192.00	0.00%	3.56	0.68	3,192.00	2.35	3,192.00	0.00%	15.19	0.15
400	15960	7980	7980	0.1	0.5	5,797.20	32.63	5,801.27	0.07%	26.48	1.23	5,693.33	69.06	5,775.10	1.44%	12.77	5.41
400	15960	3192	12768	0.1	0.8	2,341.27	65.12	2,361.93	0.88%	33.31	1.95	2,181.53	250.64	2,270.65	4.08%	12.35	20.29
400	31920	25536	6384	0.2	0.2	6,384.00	3.02	6,384.00	0.00%	3.85	0.78	6,384.00	3.36	6,384.00	0.00%	21.90	0.15
400	31920	15960	15960	0.2	0.5	12,838.07	49.06	12,844.60	0.05%	48.78	1.01	12,720.67	85.57	12,833.55	0.89%	19.74	4.33
400	31920	6384	25536	0.2	0.8	5,334.73	95.16	5,352.73	0.34%	53.72	1.77	5,159.07	432.98	5,265.55	2.06%	18.06	23.98
400	79800	63840	15960	0.5	0.2	15,960.00	5.41	15,960.00	0.00%	5.62	0.96	15,960.00	6.26	15,960.00	0.00%	41.22	0.15
400	79800	39900	39900	0.5	0.5	34,812.73	119.87	34,843.73	0.09%	107.51	1.12	34,690.60	156.70	34,853.74	0.47%	40.82	3.84
400	79800	15960	63840	0.5	0.8	14,638.87	268.17	14,659.80	0.14%	118.89	2.26	14,446.33	795.74	14,578.32	0.91%	37.36	21.30
400	127680	102144	25536	0.8	0.2	25,536.00	7.92	25,536.00	0.00%	7.58	1.04	25,536.00	10.70	25,536.00	0.00%	60.27	0.18
400	127680	63840	63840	0.8	0.5	57,415.87	200.92	57,448.47	0.06%	165.36	1.22	57,274.60	237.30	57,501.74	0.40%	57.96	4.09
400	127680	25536	102144	0.8	0.8	24,088.40	394.78	24,119.00	0.13%	203.28	1.94	23,896.27	1,149.82	24,043.29	0.62%	55.15	20.85
600	35940	28752	7188	0.1	0.2	7,188.00	5.07	7,188.00	0.00%	5.49	0.92	7,188.00	3.97	7,188.00	0.00%	21.55	0.18
600	35940	17970	17970	0.1	0.5	13,896.87	80.23	13,904.80	0.06%	53.00	1.51	13,754.47	145.41	13,883.94	0.94%	18.15	8.01
600	35940	7188	28752	0.1	0.8	5,730.53	211.86	5,760.27	0.52%	79.27	2.67	5,458.40	775.61	5,644.71	3.41%	17.33	44.75
600	71880	57504	14376	0.2	0.2	14,376.00	6.47	14,376.00	0.00%	6.61	0.98	14,376.00	6.23	14,376.00	0.00%	33.47	0.19
600	71880	35940	35940	0.2	0.5	30,124.93	142.37	30,144.27	0.06%	101.23	1.41	29,944.07	204.18	30,138.13	0.65%	30.03	6.80
600	71880	14376	57504	0.2	0.8	12,609.93	348.14	12,648.87	0.31%	125.61	2.77	12,333.07	1,297.92	12,543.48	1.71%	27.43	47.31
600	179700	143760	35940	0.5	0.2	35,940.00	14.00	35,940.00	0.00%	11.79	1.19	35,940.00	16.31	35,940.00	0.00%	65.35	0.25
600	179700	89850	89850	0.5	0.5	80,664.87	420.53	80,683.67	0.02%	259.77	1.62	80,434.13	421.37	80,757.16	0.40%	63.79	6.61
600	179700	35940	143760	0.5	0.8	33,802.07	929.33	33,845.00	0.13%	326.16	2.85	33,487.00	2,441.50	33,753.10	0.79%	57.10	42.76
600	287520	230016	57504	0.8	0.2	57,504.00	22.14	57,504.00	0.00%	17.86	1.24	57,504.00	31.36	57,504.00	0.00%	96.41	0.33
600	287520	143760	143760	0.8	0.5	132,050.60	636.54	132,102.33	0.04%	416.26	1.53	131,798.53	686.82	132,203.29	0.31%	94.88	7.24
600	287520	57504	230016	0.8	0.8	55,156.47	1,402.47	55,205.13	0.09%	472.92	2.97	54,829.20	3,413.36	55,112.19	0.52%	86.20	39.60
Average						**-**	**227.65**	**-**	**0.12%**	**110.58**	**1.57**	**-**	**526.85**	**-**	**0.82%**	**41.85**	**12.86**

Table 15 – SeqGRASP, SeqGRASP/CUDALS, SeqILS and SeqILS/CUDALS results for random instances in (i). Number of vertices: n; Avg I(P): average value of the best solution found; AvgTime: average time spent (in seconds) after 25 executions of each algorithm. Gap %I(P) is the % gap between sequential and CUDA-based local search.

Instance					SeqGRASP		SeqGRASP/CUDALS				SeqILS		SeqILS/CUDALS			
n	$\|E^-\|$	$\|E^+\|$	$w(E^-)$	$w(E^+)$	$AvgI(P)$	$AvgTime$	$AvgI(P)$	$Gap\%I(P)$	$AvgTime$	$Speedup$	$AvgI(P)$	$AvgTime$	$AvgI(P)$	$Gap\%I(P)$	$AvgTime$	$Speedup$
2000	3217	17598	3217	17598	2,186.77	70.19	2,186.62	-0.01%	55.65	1.26	2,196.73	29.95	2,196.52	-0.01%	18.64	1.61
4000	8664	40868	8664	40868	6,202.20	338.70	6,202.78	0.01%	169.20	2.00	6,211.33	151.75	6,216.84	0.09%	31.22	4.86
8000	22789	86916	22789	86916	16,081.83	1,346.06	16,084.34	0.02%	856.93	1.57	16,069.47	669.61	16,095.74	0.16%	83.18	8.05
10000	29805	109266	29805	109266	20,587.23	2,502.00	20,589.43	0.01%	1,272.11	1.97	20,590.70	1,130.71	20,616.52	0.13%	109.42	10.33
Average					**-**	**1,064.24**	**-**	**0.01%**	**588.47**	**1.70**	**-**	**495.50**	**-**	**0.09%**	**60.62**	**6.21**

Table 16 – SeqGRASP, SeqGRASP/CUDALS, SeqILS and SeqILS/CUDALS results for Slashdot instances in (ii). Number of vertices: n; Avg I(P): average value of the best solution found; AvgTime: average time spent (in seconds) after 25 executions of each algorithm. Gap %I(P) is the % gap between sequential and CUDA-based local search.

	Average SeqGRASP times			**Average SeqGRASP/CUDALS times**				
n	**Construction**	**Local search**	**Total**	**Construction**	**Local search**	**Total**	**LS Speedup**	**GRASP Speedup**
2000	23.65	46.54	70.19	20.61	35.04	55.65	1.33	1.26
4000	119.08	219.62	338.70	88.16	81.04	169.20	2.71	2.00
8000	500.58	845.48	1,346.06	516.22	340.70	856.93	2.48	1.57
10000	811.09	1,690.91	2,502.00	784.20	487.91	1,272.11	3.47	1.97
Average							**2.50**	**1.70**
	Average SeqILS times			**Average SeqILS/CUDALS times**				
n	**Construction**	**Local search**	**Total**	**Construction**	**Local search**	**Total**	**LS Speedup**	**ILS Speedup**
2000	0.79	29.16	29.95	0.79	17.86	18.64	1.63	1.61
4000	5.96	145.79	151.75	5.96	25.26	31.22	5.77	4.86
8000	12.45	657.17	669.61	12.45	70.74	83.18	9.29	8.05
10000	20.51	1,110.19	1,130.71	20.51	88.91	109.42	12.49	10.33
Average							**7.30**	**6.21**

Table 17 – Local search speedups obtained by SeqGRASP/CUDALS and by SeqILS/CUDALS on Slashdot instances in (ii). Number of vertices: n; Construction: average time spent on constructive phase; Local search: average time spent on local search phase; Total: average time spent on the whole algorithm; LS Speedup: speedup obtained only with the local search procedure; GRASP/ILS Speedup: speedup obtained considering the full algorithm execution time. Time is measured in seconds and all results were obtained after 25 independent executions of each algorithm.

5 Relaxed Correlation Clustering: an alternative measure for structural balance

As we noted in the introduction, other measures of structural balance are proposed in the literature. In Doreian and Mrvar (DOREIAN; MRVAR, 2009), the definition of a k-balanced signed graph was informally extended in order to include relevant processes (polarization, mediation, differential popularity and subgroup internal hostility) that originally were viewed as violations of structural balance. For example, the existence of a group of individuals who share only positive relationships with everyone in the network counts as imbalance in the CC Problem. Nonetheless, the individuals in this group could be construed as mediators (i.e. their relations probably won't change over time) and, as pointed in Esmailian et al. (ESMAILIAN; ABTAHI; JALILI, 2014), their relations should not be considered as a contribution to the imbalance of the network.

Using this new definition, the structural balance was generalized to a version labeled as relaxed structural balance (DOREIAN; MRVAR, 2009). This generalization gives rise to a new definition for the imbalance of a vertex partition. Let $P = \{S_1, S_2, \ldots, S_l\}$ be a l-partition of V. The relaxed imbalance $RI(P)$ of P is defined as

$$RI(P) = \sum_{1 \leq i \leq l} \min\{\Omega^+(S_i, S_i), \Omega^-(S_i, S_i)\} + \sum_{1 \leq i \neq j \leq l} \min\{\Omega^+(S_i, S_j), \Omega^-(S_i, S_j)\}. \quad (5.1)$$

Similarly to the CC Problem, the relaxed balance $RB(P)$ is defined in such a way that $RB(P) + RI(P) = \sum_{a \in A} w_a$. Consider a partition P and a cut (uncut) arc (i, j). The contribution of arc (i, j) for the relaxed imbalance $RI(P)$ depends on the sign of other cut (uncut) arcs. On the other hand, the contribution of arc (i, j) for the imbalance $I(P)$ depends only on its own sign. Remark that these two measures of imbalance are related in such a way that $RI(P) \leq I(P)$ for each partition P of V.

The definition of relaxed imbalance $RI(P)$ given by equation (5.1), and associated with the relaxed structural balance, has its roots in blockmodeling approaches (DOREIAN; MRVAR, 2009; BRUSCO et al., 2011). Besides, a redefinition of relaxed imbalance of a partition P that takes into account only symmetric relationships was presented by Figueiredo and Moura (FIGUEIREDO; MOURA, 2013), as stated below.

$$SRI(P) = \sum_{1 \leq i \leq l} \min\{{}^{\Omega^+(S_i,S_i),}_{\Omega^-(S_i,S_i)}\} + \sum_{1 \leq i < j \leq l} \min\{{}^{\Omega^+(S_i,S_j)+\Omega^+(S_j,S_i),}_{\Omega^-(S_i,S_j)+\Omega^-(S_j,S_i)}\}. \quad (5.2)$$

This definition of imbalance gives rise to a new graph clustering problem, the Symmetric Relaxed Correlation Clustering Problem, which will be studied here. The

problem is stated as follows.

Problem 5.0.1 (Symmetric RCC problem) *Let $G = (V, E)$ be a signed graph, w_e be a non-negative edge weight associated with edge $e \in E$ and k be an integer value satisfying $1 \leq k \leq n$. The symmetric relaxed correlation clustering problem is the problem of finding a l-partition P of V, with $l \leq k$, such that the symmetric relaxed imbalance $SRI(P)$ is minimized. Let us denote this minimal value by SRCC(G,k).*

It is worth noting that the Symmetric RCC (SRCC) problem is closely related with the CC problem but it is not a particular case nor is it a generalization. Actually, each feasible solution (a graph partition) of the SRCC problem is also feasible in the CC problem but the problems have different cost functions, i.e., there are different ways of evaluating the imbalance of a partition. The SRCC problem is intuitively as difficult as the CC problem and is indeed a NP-hard problem (FIGUEIREDO; MOURA, 2013).

Structural balance theory affirms that signed social networks tend towards balance, so as to avoid conflicting situations (e.g. cycles of negative parity). However, many empirical networks are unbalanced and it is not even possible to state that these networks have a tendency towards balance. The authors in (DOREIAN; MRVAR, 2009) argued that, in a social network, some subsets of elements can be viewed as sets of either negative or positive mediators and, as a consequence, the imbalance of a partition depends on this mediation rule. In other words, the relationship pertaining to a group of mediators should not be considered as a contribution to the imbalance of the network.

Suppose that a network has intermediaries positively linked to members of mutually hostile subgroups - in exactly balanced groups. This is shown in Figure 23-a with three group members, $\{A, B, D\}$, that are viewed as potential positive mediators. The sets $\{C, E, F\}$ and $\{G, H\}$ are mutually hostile, balanced subgroups.

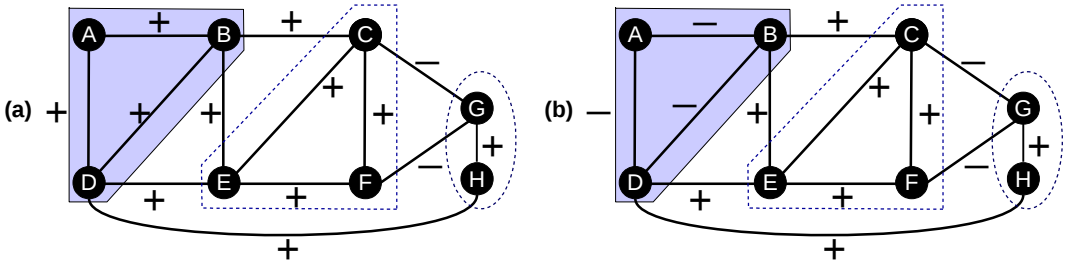


Figure 23 – Positive mediation (a) and negative mediation (b) examples.

Further exploring the mediation example, suppose that the mediators see themselves as competitors for performing mediation and have only negative ties between them because of this rivalry. The resulting network is shown in Figure 23-b, where (AB), (AD) and (BD) are negative edges, and this process is known as negative mediation.

The SRCC problem allows us to analyse mediation processes (positive and negative). That is not the case of the RCC problem, where mediation and differential popularity cannot be pointed out.

5.1 Literature review

To the best of our knowledge, the RCC problems have been proposed for only the evaluation of structural balance in social networks. However, these problems could be also used as an approach to efficient community detection. Two solution methods were initially proposed in the literature for RCC problems: a greedy heuristic approach (DOREIAN; MRVAR, 2009) and a branch-and-bound procedure (BRUSCO et al., 2011). Computational experiments with both procedures were reported over literature instances with up to 29 vertices (small literature instances listed in Chapter 2, item (i)) and for random instances with $|V| \in \{20, 40\}$ (DOREIAN; MRVAR, 2009; BRUSCO et al., 2011). For the branch-and-bound procedure, the values considered for k were $k \leq 7$ for literature instances and $k \in \{3, 5\}$ for the set of random instances.

Further on, Figueiredo and Moura (FIGUEIREDO; MOURA, 2013) presented the first mathematical formulation for the RCC and SRCC problems: a representatives ILP formulation, which is harder to solve than the CC ILP formulation. For the set of benchmark instances, the ILP formulation was able to solve the problem when $k = 2$, $k = 3$ (for some instances) and for higher values of k. The results presented for the branch-and-bound procedure in (BRUSCO et al., 2011) demonstrated that this approach was efficient in the solution of RCC instances with $k \leq 8$. At this point, no metaheuristic approaches have been applied to solve the RCC problems.

5.2 Experimental results

The RCC ILP formulation becomes numerically more difficult to solve with the symmetric definition and, as a consequence, the authors in (FIGUEIREDO; MOURA, 2013) were able to calculate the SRCC solution only for smaller networks, namely the literature instances in (i). We have applied these results to validate the quality of the solutions returned by our metaheuristics.

When adapting the ILS metaheuristic to solve the SRCC problem, instead of the CC problem, there is the need to change the objective function that evaluates the partition. When it comes to the SRCC problem, this evaluation becomes more costly, as it demands the solution of a series of minimization problems defined in equation 5.2. In order to reduce this computational cost, the algorithm manipulates matrices with precalculated imbalance values between clusters, and these matrices are incrementally

updated whenever an improved solution is found. The matrices are then used to analyze the solutions visited at each local search iteration. In this way, our ILS heuristic is able to provide an efficient solution to the SRCC Problem, applying the same methods (and corresponding configurations) used to solve the CC Problem, with one major advantage over exact methods: it can process larger graph files.

Therefore we have extended the metaheuristics explained in the previous chapters to deal with this new measure, modifying the applied objective function. Also, in order to provide SRCC measures for larger graph files, we ran the ILS procedure restricting the number of clusters in the SRCC solution in order to match the exact number of clusters (k) found by the CC solution previously obtained by the program. This way, we were able to compare the imbalance of the solutions returned by both ILS algorithms (CC and SRCC).

The computational effort to solve the SRCC problem using the Slashdot instances as input is greater when compared to the CC problem (Table 18). Note that the slow-down of SRCC decreases when $n \geq 4000$, since the algorithm exceeds the 2-hour time limit.

After solving the SRCC problem, one can observe that, when compared to the CC problem, the imbalance measures obtained with SRCC are sometimes smaller, as can be seen on Figure 24 (UNGA instances) and Table 19 (Slashdot-based instances). A reason why the CC problem tends to over-evaluate the imbalance is related to penalizing relationships associated with mediation processes that occur in the network.

n	**200**	**300**	**400**	**600**	**800**	**1000**	**2000**	**4000**	**8000**	**10000**
Time-CC	2.26	1.64	1.80	3.11	4.44	7.29	20.30	125.37	458.32	794.11
Time-SRCC	7.79	9.14	6.64	39.34	324.67	426.30	5262.02	7202.64	7202.95	7205.69
Slow-down	3.44	5.58	3.69	12.64	73.06	58.47	259.25	57.45	15.72	9.07

Table 18 – Difference in execution time (in seconds) when running the parallel ILS algorithm using CC and SRCC objective functions, respectively, over Slashdot-based instances listed in Chapter 2. Slow-down=Time-SRCC/Time-CC.

n	**200**	**300**	**400**	**600**	**800**	**1000**	**2000**	**4000**	**8000**	**10000**
CC	45	54	57.2	109.4	240	600	2201.4	6210.4	16074	20596.6
SRCC	23	11	23	19	83	226	755	3056	8215	10083
k	5	8	4	9	20	11	43	30	67	109

Table 19 – CC and SRCC imbalance measures of Slashdot-based instances listed in Chapter 2. Number of vertices: n; number of clusters in the solution: k.

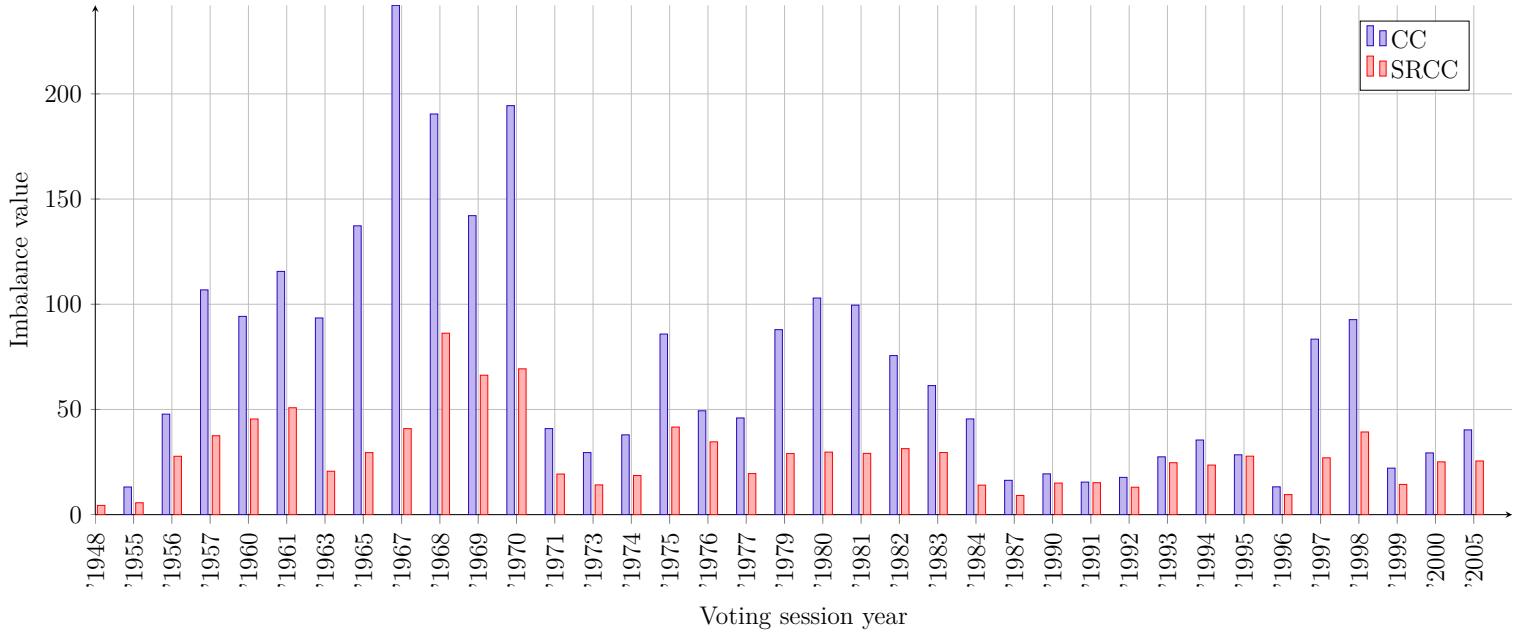


Figure 24 – CC and SRCC imbalance measures of UNGA instances listed in Section 2.3.2. In this graph, we only list the years when the CC and SRCC imbalance values differ. For a full report of these results, see the complementary material in <http://www.ic.uff.br/yuri/files/CCcomp.zip>.

6 Analysis of structural balance on real-world signed social networks

In this chapter, we apply the previously presented algorithms to efficiently solve the CC and SRCC problems, and analyze the obtained solutions to the United Nations General Assembly Voting Data and three large signed social networks (Epinions, Slashdot and Wikipedia) available in (LESKOVEC; KREVL, 2014).

6.1 The United Nations General Assembly (UNGA) voting data

As noted in the introduction, some authors have applied different signed graph clustering methods to networks of international alliances and disputes. Traag and Bruggeman (TRAAG; BRUGGEMAN, 2009) analyzed international relations taken from the Correlates of War (STINNETT et al., 2002) data set, while Macon et al. (MACON; MUCHA; PORTER, 2012) showed how three different network representations could be used to identify voting groups in UNGA annual sessions.

Likewise, the following analysis is based on the voting on resolutions in the United Nations General Assembly (UNGA), separated by year. Having applied the sequential ILS algorithm using the UNGA instances listed in Chapter 2 as input, we give a taste of social network analysis that can be done with the obtained CC and SRCC results.[1] We interpret the results obtained on graphs associated with UNGA votes, i.e. groups of countries that minimize imbalance, together with some historical facts.

6.1.1 UNGA CC results

According to our results, in 1946 (first UNGA voting session), the obtained CC solution has Poland, Czechoslovakia, Yugoslavia, Russia, Ukraine and Belarus together in the same group, reflecting the Soviet Union power in Eastern Europe. At the same time, USA and Cuba appeared together in another group, and this behavior persisted until 1953, the year of the Cuban Revolution (PÉREZ-STABLE, 1993). Later, in 1962, when the Cuban missile crisis took place (ALLISON, 1969), bipolarity was evident during the Cold War. Our CC results indicate two clusters: (i) USA, with most Latin American countries, Western Europe, Japan, Taiwan, India, Australia and other Pacific Countries; and (ii) Russia, with Cuba, Poland, Hungary, Czechoslovakia, Albania, Yugoslavia, Bulgaria,

[1] Appendix A lists the CC and SRCC results mentioned in this Chapter, including the groups of countries in each solution, separated by year. For a full report of these results, see the complementary material in <http://www.ic.uff.br/~yuri/files/CCcomp.zip>.

Ukraine and many African countries. By 1963, Cuba had moved towards a full-fledged Communist system modeled on the USSR.

Other interesting results are related to Apartheid in South Africa, as the country appears isolated in the CC clustering solution for the 1974 voting session. In that year a motion was passed to expel South Africa from the UN, but this was vetoed by France, the United Kingdom and the United States, all key trade associates of South Africa (NESBITT, 2004; MCGREAL, 2006; DOWDALL, 2009). The results obtained for the graph associated to 1974 also show an approximation of the USSR (Russia) and Arab states (Libya, Sudan, Iran, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman and Afghanistan), which appear in the same group. This behavior makes sense, since after the mid 1950's and throughout the remainder of the Cold War the Soviets unequivocally supported various Arab regimes in lieu of Israel (GOLANI, 1995; GOLAN, 2010).

Later on, from 1987 to 1991, during the period known as the First Intifada, characterized by Palestinian uprising against the Israeli occupation of the Palestinian Territories (SMITH, 2010), the CC results show that Israel is always in a small group while most other countries of the world form another group. The USA, however, always appears together with Israel and sometimes other countries like the UK, France and German Federal Republic also appear in the same group.

More recently, regarding the Gaza-Israel conflict, in 2006, Israel and USA appear together, isolated in a group, with the rest of the world in another group. This is probably due to the large-scale conventional warfare beyond the peripheries of the Gaza Strip (MEARSHEIMER; WALT, 2006), in addition to the 2006 Lebanon War (KREPS, 2007).

6.1.2 UNGA SRCC results

As explained in the previous section, an interesting analysis of the SRCC problem provides a different view over certain relations that were previously seen as violations to the balance on the network according the original CC problem definition. Since the SRCC problem does not count certain types of ties as violations (for considering some ties belonging to processes occurring in the network, such as positive and negative mediation), the values of imbalance tend to be even lower than those identified by the CC problem solver. Thus it is possible to extract and detect which actors possibly belong to a group of mediators. Taking the UNGA CC and SRCC solution values into account, as can be seen on Figure 24 in Section 5.2, it is possible to identify in which yearly voting sessions the SRCC relaxed imbalance is smaller than the CC imbalance measure. In these cases, there are relationships / ties that were not classified by SRCC as violations of the balance of

the network. For example, in the voting session of 1987, we were able to identify a group of countries that acted as positive mediators: Canada, Ireland, Netherlands, Belgium, Luxembourg, France, Spain, Portugal, German Federal Republic, Italy, Norway, Denmark, Iceland, Japan, Australia and New Zealand. This group of sixteen countries has 100% of internal positive relationships and 92% of external positive relationships. Besides this, there are two other clusters: the first one with the USA, Dominica, the UK and Israel, and another one with 138 countries. The year of 1987 is exactly when the First Intifada commenced, so probably the small group with Israel and the large group with most countries of the World are in opposite camps, subject to the mediation of a third group of countries.

The same holds true in the analysis of the SRCC results of 1990. In this case, the group of mediators was composed of the following twelve countries, out of which eleven were also present in the mediation group of 1987: Canada, Netherlands, Belgium, Luxembourg, Portugal, German Federal Republic, Poland, Italy, Norway, Denmark, Iceland and Japan. This group of twelve countries has 100% of internal positive relationships and 94% of external positive relationships. There are also two other groups. The second one comprising of nine countries (United States of America, Panama, United Kingdom, France, Sao Tome and Principe, Equatorial Guinea, Liberia, Israel and Cambodia) and the third one with 137 nations, including all Arab and Soviet states. The First Intifada lasted until the Madrid Conference in 1991, though some date its conclusion to 1993, with the signing of the Oslo Accords (MUNEM, 2008). With this in mind, an evidence that supports the existence of the aforementioned mediation group is the second phase of the Madrid Peace Process, launched by the United States and Russia in Moscow, in January 1992. Foreign ministers and delegates from thirty six countries - including representatives from the Middle East, Europe, Japan, China and Canada - were involved in the Israeli-Palestinian peace process.

6.2 Epinions, Slashdot and Wikipedia social media networks

In this section, we evaluate three large online signed social networks available in Stanford Large Network Dataset Collection (LESKOVEC; KREVL, 2014): (i) the social network of the technology-news website Slashdot[2], where a signed link indicates that one user likes or dislikes the comments of another; (ii) the voting network of Wikipedia (WikiElections[3]), where a signed link indicates a positive or negative vote by one user on the promotion to admin status of another; and (iii) the trust network of Epinions product review website[4], where users can indicate their trust or distrust of the reviews of others.

[2] Slashdot friends or foes network from Feb 21 2009: 82,144 vertices and 549,202 edges.
[3] Wikipedia adminship election data: 7,000 vertices and 100,000 edges.
[4] Epinions who-vote-whom network: 131,828 vertices and 841,372 edges.

Table 20 presents different imbalance measures found in the literature for the aforementioned social networks. Facchetti et al. (FACCHETTI; IACONO; ALTAFINI, 2011) defines a fraction called δ/m, which can be interpreted as a measure of the relative imbalance of the network, since it consists of the number of frustrated bipartite relationships divided by the total number of relationships of the network. In turn, Chiang et al. (CHIANG et al., 2013) shows the proportion of balanced 3-cycles ($P(C_{li})$), and Leskovec et al. (LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010) presents a measure of balanced undirected triads.

Method	Measure	Slashdot	Wikipedia	Epinions
(FACCHETTI; IACONO; ALTAFINI, 2011)	δ/m	14.76%	14.20%	7.17%
(CHIANG et al., 2013)	$1 - P(C_{li})$	13.35%	21.65%	9.50%
(LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010)	% unbalanced triads	8.80%	9.10%	5.90%
ILS-CC	$\%I(P)$	14.04%	14.32%	7.89%
ILS-SRCC	$\%RI(P)$	13.90%	5.37%	7.81%

Table 20 – Comparison of imbalance measures from different authors and our results (ILS-CC and ILS-SRCC).

By applying the best parallel version of the ILS algorithm, we provide an analysis of the relative imbalance of these networks, which corresponds with the results from three different works that have also shown how these networks are extremely balanced when compared to random networks of equivalent size.

According to Fachetti et al. (FACCHETTI; IACONO; ALTAFINI, 2011), the fraction of imbalance for the undirected Slashdot network is 14.76%, which is compatible with the results obtained by $ILS - CC$. Our metaheuristic has reached a solution with a relative imbalance ($\%I(P)$) of 14.04% (Table 20). Remark that the divergence in these values is expected and is probably due to the different measures applied. $ILS - CC$ managed to fully process Wikipedia network, resulting in a relative imbalance ($\%I(P)$) of 14.32%, while the paper's measure points to the value of 14.20%. Moreover, the relative imbalance of Epinions network obtained by $ILS - CC$ was 7.89%, being very close to the paper's result (7.17%).

Additionally, according to Chiang et al. (CHIANG et al., 2013), the three former social networks exhibit high levels of balance - we show the corresponding values of imbalance ($1 - P(C_{li})$) in Table 20. Even though this measure is not the same objective function of the CC problem, it still indicates that these networks are indeed extremely balanced, which is in accordance with the result of our work. Also remark that the same work presents another measure called $P_0(C_{li})$, which consists of the expected probability of finding balanced 3-cycles in an equivalent random network. The values of P_0 are a lot smaller than the values of P (P_0 equals an average of 60%). This leads to the conclusion that, in random networks of equivalent size, imbalance measures tend to be much higher than the actual imbalance of the three aforementioned social media networks.

Finally, a third work by Leskovec et al. (LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010) confirms that the analysed social media networks have high relative balance (i.e. low relative imbalance). In the corresponding line of Table 20, we display the percentage of unbalanced triads, based on the measures provided by the authors.

In summary, the above comparisons indicate that the measure applied in (FACCHETTI; IACONO; ALTAFINI, 2011) matches our relative imbalance measure derived from the CC problem objective function, while the measures available in (CHIANG et al., 2013) and (LESKOVEC; HUTTENLOCHER; KLEINBERG, 2010) underevaluate the imbalance of the networks, as they are based only on unbalanced triangles.

Great part of the imbalance measure of the networks is due to unaccounted mediation relationships. Although $ILS-SRCC$ was not able to detect mediation processes occurring in Slashdot and Epinions networks, the Wikipedia network does indeed present a great quantity of relationships associated with mediation and this translates into a lower imbalance measure ($\%RI(P)$), as shown in Table 20. This points to relative values of (relaxed) imbalance even lower than those found by the CC problem solver. Thus, if we factor the SRCC results into consideration, the Wikipedia network is still more balanced than the measures from Fachetti et al. (FACCHETTI; IACONO; ALTAFINI, 2011) and also the CC problem.

7 Discussion and concluding remarks

The CC and SRCC consist of NP-hard combinatorial optimization problems. We have contributed to their efficient solution by incorporating the GRASP and ILS metaheuristics, enabling the analysis of structural balance on large-scale real-world social networks. We have demonstrated the potential of social network analysis over UNGA voting session data, Wikipedia election history, Slashdot friends or foes network and Epinions who-trust-whom online social network. To the best of our knowledge, this is the first work to list CC and SRCC measures of large signed social networks.

The multistart ILS metaheuristic, proposed in Chapter 3 to solve these problems, is an improvement over the GRASP approach to solve the CC Problem, either by outperforming, in processing time, the GRASP metaheuristic proposed in Chapter 2, or by improving the solution quality. In fact, our tests have shown that the sequential version of ILS is more efficient than the existing heuristics (Pajek - Doreian Mrvar Method, and VOTE/BOEM).

Additionally, the parallel ILS with sequential VND, which offers substantial speedups over the sequential algorithm, is the most efficient metaheuristic to solve larger instances, such as those based on real-world signed social networks (Wikipedia, Slashdot and Epinions), as well as completely random and random instances with a predefined community structure. A comparison with the VOTE/BOEM heuristic has also revealed that the parallel ILS algorithm is able to outperform it when solving larger networks like Slashdot. A possible improvement of this work could be combining the parallel local search procedure for the Correlation Clustering problem on GPU (CUDAVND) with parallel ILS, in a hybrid application, applying the parallelism available both in CPU (multicore) and GPU (CUDA). Moreover, the development of an ILS procedure which benefits from data parallelism could also bring interesting results, allowing the processing of even larger graph instances (more than half a million vertices).

Regarding the Symmetric RCC Problem, we were able to calculate the Symmetric Relaxed Imbalance (SRI) of all network instances cited in the experiments by applying the same parallel ILS metaheuristic used to solve the CC Problem.

The multistart ILS heuristic can also be incorporated in the efficient solution of instances from other problems. The CC problem can be applied in several areas, in which large instances probably require solving. With this idea in mind, practitioners interested in these applications can apply our heuristic. In future works, these methods can be used to analyse other real networks that represent a social group in different periods of time. On top of this, it is possible to generalize our heuristic to manage other ways of evaluating the

imbalance of the network, which implies solving other relaxed versions of the CC problem.

Bibliography

ABELL, P.; LUDWIG, M. Structural balance: a dynamic perspective. *Journal of Mathematical Sociology*, v. 33, p. 129–155, 2009. 1, 2

AIEX, R. M.; RESENDE, M. G.; RIBEIRO, C. C. Ttt plots: a perl program to create time-to-target plots. *Optimization Letters*, Springer, v. 1, n. 4, p. 355–366, 2007. 13, 14, 42, 43, 47, 50

AILON, N.; CHARIKAR, M.; NEWMAN, A. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, ACM, v. 55, n. 5, p. 23, 2008. 8

ALBA, E. *Parallel metaheuristics: a new class of algorithms*. [S.l.]: John Wiley & Sons, 2005. v. 47. 17

ALLISON, G. T. Conceptual models and the cuban missile crisis. *American political science review*, Cambridge Univ Press, v. 63, n. 03, p. 689–718, 1969. 69

BACHE, K.; LICHMAN, M. *UCI Machine Learning Repository*. 2013. Disponível em: <http://archive.ics.uci.edu/ml>. 8

BANSAL, N.; BLUM, A.; CHAWLA, S. Correlation clustering. In: *Proceedings of the 43rd annual IEEE symposium of foundations of computer science*. Vancouver, Canada: [s.n.], 2002. p. 238–250. 2, 4, 7, 8

BATAGELJ, V.; MRVAR, A. *Pajek Wiki*. 2008. URL http://pajek.imfm.si/. Accessed on May 2014. 7, 19

BESTEN, M. D.; STÜTZLE, T.; DORIGO, M. Design of iterated local search algorithms. In: *Applications of Evolutionary Computing*. [S.l.]: Springer, 2001. p. 441–451. 35, 36

BHATTACHARYA, A.; DE, R. K. Divisive correlation clustering algorithm (dcca) for grouping of genes: detecting varying patterns in expression profiles. *bioinformatics*, Oxford Univ Press, v. 24, n. 11, p. 1359–1366, 2008. 8

BONCHI, F.; GIONIS, A.; UKKONEN, A. Overlapping correlation clustering. In: IEEE. *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. [S.l.], 2011. p. 51–60. 8

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

BRUSCO, M. An enhanced branch-and-bound algorithm for a partitioning problem. *British Journal of Mathematical and Statistical Psychology*, v. 56, p. 83–92, 2003. 21

BRUSCO, M. et al. Two algorithms for relaxed structural balance partitioning: linking theory, models and data to understand social network phenomena. *Sociological Methods & Research*, v. 40, p. 57–87, 2011. 63, 65

BRUSCO, M. J.; KÖHN, H.-F. Clustering qualitative data based on binary equivalence relations: neighborhood search heuristics for the clique partitioning problem. *Psychometrika*, Springer, v. 74, n. 4, p. 685–703, 2009. 3, 33

CARTWRIGHT, D.; HARARY, F. Structural balance: A generalization of heider's theory. *Psychological Review*, v. 63, p. 277–293, 1956. 1

CHARIKARA, M.; GURUSWAMIB, V.; WIRTHA, A. Clustering with qualitative information. *Journal of Computer and System Sciences*, v. 71, p. 360–383, 2005. 8

CHIANG, K.-Y. et al. Prediction and clustering in signed networks: A local to global perspective. *arXiv preprint arXiv:1302.5145*, 2013. 72, 73

COELHO, I. M. et al. The single vehicle routing problem with deliveries and selective pickups in a cpu-gpu heterogeneous environment. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. [S.l.], 2012. p. 1606–1611. 52

CRAINIC, T. G.; TOULOUSE, M. Parallel meta-heuristics. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2010. p. 497–541. 17

DASGUPTA, B. et al. Algorithmic and complexity results for decompositions of biological networks into monotone subsystems. *BioSystems*, v. 90, p. 161–178, 2007. 2, 8

DAVIS, J. Clustering and structural balance in graphs. *Human Relations*, v. 20, p. 181–187, 1967. 1

DEMAINE, E. D. et al. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, v. 361, p. 172–187, 2006. 7, 8

DOREIAN, P.; KRACKHARDT, D. Pre-transitive balance mechanisms for signed networks*. *Journal of Mathematical Sociology*, Taylor & Francis, v. 25, n. 1, p. 43–67, 2001. 1

DOREIAN, P.; MRVAR, A. A partitioning approach to structural balance. *Social Networks*, v. 18, p. 149–168, 1996. 1, 2, 7

DOREIAN, P.; MRVAR, A. Structural balance and partitioning signed graphs. *Developments in data analysis*, p. 195–208, 1996. 3, 8, 20

DOREIAN, P.; MRVAR, A. Partitioning signed social networks. *Social Networks*, v. 31, p. 1–11, 2009. 1, 2, 4, 21, 63, 64, 65

DOWDALL, A. T. *The birth and death of a tar baby: Henry Kissinger and southern Africa*. Tese (Doutorado) — University of Missouri–Columbia, 2009. 70

DRUMMOND, L. et al. Efficient solution of the correlation clustering problem: An application to structural balance. In: DEMEY, Y.; PANETTO, H. (Ed.). *On the Move to Meaningful Internet Systems: OTM 2013 Workshops*. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 8186). p. 674–683. ISBN 978-3-642-41032-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-41033-8_85>. 3

DUCH, J.; ARENAS, A. Community detection in complex networks using extremal optimization. *Physical review E*, APS, v. 72, n. 2, p. 027104, 2005. 2

EKŞIOGLU, S. D.; PARDALOS, P. M.; RESENDE, M. G. Parallel metaheuristics for combinatorial optimization. In: *Models for Parallel and Distributed Computation*. [S.l.]: Springer, 2002. p. 179–206. 17

ELSNER, M.; SCHUDY, W. Bounding and comparing methods for correlation clustering beyond ilp. In: *ILP'09 Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*. [S.l.: s.n.], 2009. p. 19–27. 3, 8, 20

EPINIONS. *Website*. 1999. URL http://www.epinions.com. Accessed on March 2015. 3

ESMAILIAN, P.; ABTAHI, S. E.; JALILI, M. Mesoscopic analysis of online social networks: The role of negative ties. *Physical Review E*, APS, v. 90, n. 4, p. 042817, 2014. 1, 2, 63

FACCHETTI, G.; IACONO, G.; ALTAFINI, C. Computing global structural balance in large-scale signed social networks. In: *Proceedings of the National Academy of Sciences of the United States of America*. [S.l.: s.n.], 2011. v. 108, p. 20953–20958. 1, 2, 3, 4, 21, 58, 72, 73

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995. 3, 11, 33

FIGUEIREDO, R.; FROTA, Y. The maximum balanced subgraph of a signed graph: Applications and solution approaches. *European Journal of Operational Research*, v. 236, n. 2, p. 473 – 487, 2014. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221713010205>. 22

FIGUEIREDO, R.; MOURA, G. Mixed integer programming formulations for clustering problems related to structural balance. *Social Networks*, Elsevier, v. 35, n. 4, p. 639–651, 2013. 2, 7, 21, 23, 24, 44, 63, 64, 65

FRINHANI, R. M. et al. Grasp with path-relinking for data clustering: a case study for biological data. In: *Experimental Algorithms*. [S.l.]: Springer, 2011. p. 410–420. 13, 16

FUJIMOTO, N.; TSUTSUI, S. A highly-parallel tsp solver for a gpu computing platform. In: *Numerical Methods and Applications*. [S.l.]: Springer, 2011. p. 264–271. 52

GENDREAU, M.; POTVIN, J. *Handbook of Metaheuristics*. Springer, 2010. ISBN 9781441916655. Disponível em: <http://books.google.com.br/books?id=xMTS5dyDhwMC>. 13, 16

GIOTIS, I.; GURUSWAMI, V. Correlation clustering with a fixed number of clusters. In: ACM. *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*. [S.l.], 2006. p. 1167–1176. 8

GLOVER, F.; KOCHENBERGER, G. A. *Handbook of metaheuristics*. [S.l.]: Springer Science & Business Media, 2003. 13, 35

GOLAN, G. *Yom Kippur and After: The Soviet Union and the Middle East Crisis*. [S.l.]: Cambridge University Press, 2010. v. 19. 70

GOLANI, M. The historical place of the czech-egyptian arms deal, fall 1955. *Middle Eastern Studies*, Taylor & Francis, v. 31, n. 4, p. 803–827, 1995. 70

GROPP, W.; LUSK, E.; SKJELLUM, A. *Using MPI: Portable Parallel Programming with the Message-passing Interface*. MIT Press, 1999. ISBN 9780262571326. Disponível em: <http://books.google.com.br/books?id=xpBZ0RyRb-oC>. 21

GÜLPINAR, N. et al. Extracting pure network submatrices in linear programs using signed graphs. *Discrete Applied Mathematics*, v. 137, p. 359–372, 2004. 2, 8

HARARY, F.; LIM, M.; WUNSCH, D. C. Signed graphs for portfolio analysis in risk management. *IMA Journal of Management Mathematics*, v. 13, p. 1–10, 2003. 8

HEIDER, F. Attitudes and cognitive organization. *Journal of Psychology*, v. 21, p. 107–112, 1946. 1

HUFFNER, F.; BETZLER, N.; NIEDERMEIER, R. Separator-based data reduction for signed graph balancing. *Journal of Combinatorial Optimization*, v. 20, p. 335–360, 2010. 8

INOHARA, T. On conditions for a meeting not to reach a deadlock. *Applied Mathematics and Computation*, v. 90, p. 1–9, 1998. 1, 2

KIM, S. et al. Image segmentation usinghigher-order correlation clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 36, n. 9, p. 1761–1774, 2014. 2

KREPS, S. E. The 2006 lebanon war: Lessons learned. *Parameters*, US ARMY WAR COLLEGE, v. 37, n. 1, p. 72, 2007. 70

KRIEGEL, H.-P.; KRÖGER, P.; ZIMEK, A. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, ACM, v. 3, n. 1, p. 1, 2009. 9

KRÜGER, F. et al. Speedups between× 70 and× 120 for a generic local search (memetic) algorithm on a single gpgpu chip. In: *Applications of Evolutionary Computation*. [S.l.]: Springer, 2010. p. 501–511. 52

KUNEGIS, J.; LOMMATZSCH, A.; BAUCKHAGE, C. The slashdot zoo: mining a social network with negative edges. In: *WWW'09 Proceedings of the 18th international conference on World wide web*. [S.l.: s.n.], 2009. p. 741–750. 3, 4

KUNEGIS, J. et al. Spectral analysis of signed graphs for clustering, prediction and visualization. In: SIAM. *SDM*. [S.l.], 2010. v. 10, p. 559–559. 4

LAGUNA, M.; MARTI, R. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, INFORMS, v. 11, n. 1, p. 44–52, 1999. 14

LESKOVEC, J.; HUTTENLOCHER, D.; KLEINBERG, J. Signed networks in social media. In: *CHI'10 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2010. p. 1361–1370. 2, 3, 4, 21, 58, 72, 73

LESKOVEC, J.; KREVL, A. *SNAP Datasets: Stanford Large Network Dataset Collection*. 2014. <http://snap.stanford.edu/data>. 4, 14, 21, 69, 71

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. *Iterated local search*. [S.l.]: Springer, 2003. 3, 33

LUONG, T. V.; MELAB, N.; TALBI, E.-G. Gpu computing for parallel local search metaheuristic algorithms. *Computers, IEEE Transactions on*, IEEE, v. 62, n. 1, p. 173–185, 2013. 52, 55

MACON, K.; MUCHA, P.; PORTER, M. Community structure in the united nations general assembly. *Physica A: Statistical Mechanics and its Applications*, v. 391, p. 343–361, 2012. 1, 2, 4, 8, 22, 69

MARTÍ, R. Multi-start methods. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 355–368. 33

MCGREAL, C. Brothers in arms-israel's secret pact with pretoria. *The Guardian*, v. 7, 2006. 70

MEARSHEIMER, J. J.; WALT, S. M. The israel lobby and us foreign policy. *Middle East Policy*, Wiley Online Library, v. 13, n. 3, p. 29–87, 2006. 70

MEHROTRA, A.; TRICK, M. A. Cliques and clustering: A combinatorial approach. *Oper. Res. Lett.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 22, n. 1, p. 1–12, fev. 1998. ISSN 0167-6377. Disponível em: <http://dx.doi.org/10.1016/S0167-6377(98)00006-6>. 6, 7, 23

MELAB, N.; TALBI, E.-G. et al. Gpu-based multi-start local search algorithms. In: *Learning and Intelligent Optimization*. [S.l.]: Springer, 2011. p. 321–335. 14, 54, 55

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100, 1997. 12, 33

MUNEM, D. E. B. A. *Canada and Peace in the Middle East*. 2008. URL http://www.palestine1.net/canp-e.htm. Accessed on Jan 2015. 71

NASCIMENTO, M. C.; PITSOULIS, L. Community detection by modularity maximization using grasp with path relinking. *Computers Operations Research*, 2013. Available online on March 2013. 11

NESBITT, F. N. *Race for sanctions: African Americans against apartheid, 1946-1994*. [S.l.]: Indiana University Press, 2004. 70

NEWMAN, M. E. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 103, n. 23, p. 8577–8582, 2006. 2

NVIDIA. *CUDA Toolkit*. 2015. http://www.nvidia.com/cuda. Accessed on 23.03.2015. Disponível em: <http://www.nvidia.com/cuda>. Acesso em: 2015-03-23. 51, 57

PENA, G. C. et al. An improved parallel algorithm using gpu for siting observers on terrain. In: *16th International Conference on Enterprise Information Systems (ICEIS-2014)*. [S.l.: s.n.], 2014. p. 367–375. 52

PÉREZ-STABLE, M. *The Cuban revolution: Origins, course, and legacy*. [S.l.]: Oxford University Press New York, 1993. 69

RESENDE, M. G.; RIBEIRO, C. C. Grasp with path-relinking: Recent advances and applications. In: *Metaheuristics: progress as real problem solvers*. [S.l.]: Springer, 2005. p. 29–63. 14

ROCKI, K.; SUDA, R. Accelerating 2-opt and 3-opt local search using gpu in the travelling salesman problem. In: IEEE. *High Performance Computing and Simulation (HPCS), 2012 International Conference on*. [S.l.], 2012. p. 489–495. 52

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, v. 177, n. 3, p. 2033–2049, 2007. 33

SLASHDOT. *Website*. 1997. URL http://slashdot.org. Accessed on March 2015. 3

SMITH, C. D. *Palestine and the Arab-Israeli conflict:[a history with documents]*. [S.l.]: Bedford/St. Martin's, 2010. 70

SRINIVASAN, A. Local balancing influences global structure in social networks. In: *Proceedings of the National Academy of Sciences of the United States of America*. [S.l.: s.n.], 2011. v. 108, p. 1751–1752. 2

STINNETT, D. M. et al. The Correlates of War (Cow) Project Direct Contiguity Data, Version 3.0. *Conflict Management and Peace Science*, v. 19, p. 59–67, 2002. 4, 69

SWAMY, C. Correlation clustering: maximizing agreements via semidefinite programming. In: SOCIETY FOR INDUSTRIAL AND APPLIED MATHEMATICS. *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. [S.l.], 2004. p. 526–527. 8

TRAAG, V.; BRUGGEMAN, J. Community detection in networks with positive and negative links. *Physical Review E*, v. 80, p. 036115, 2009. 1, 2, 4, 8, 69

YANG, B.; CHEUNG, W.; LIU, J. Community mining from signed social networks. *IEEE Transactions on Knowledge and Data Engineering*, v. 19, p. 1333–1348, 2007. 1, 2, 8, 20, 37

ZHANG, S.; WANG, R.-S.; ZHANG, X.-S. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, Elsevier, v. 374, n. 1, p. 483–490, 2007. 8

ZHANG, Z. et al. Correlation clustering based on genetic algorithm for documents clustering. In: *IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2008. p. 3193–3198. 3, 8, 20

A UNGA CC and SRCC results

Group	#	Countries
1	51	United States of America, Canada, Haiti, Dominican Republic, Mexico, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, United Kingdom, Ireland, Netherlands, Belgium, Luxembourg, France, Spain, Portugal, Austria, Italy, Greece, Cyprus, Finland, Sweden, Norway, Denmark, Iceland, South Africa, Turkey, Israel, Taiwan, Japan, India, Thailand, Malaysia, Philippines, Australia, New Zealand, Jamaica, Trinidad and Tobago,
2	59	Cuba, Poland, Hungary, Czechoslovakia, Albania, Yugoslavia, Bulgaria, RUM, Russia, Ukraine, Belarus, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Tanzania, Somalia, Ethiopia, Madagascar, Morocco, Tunisia, Libya, Sudan, Iran, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Afghanistan, Mongolia, Pakistan, Myanmar, Sri Lanka, Nepal, Cambodia, Laos, Indonesia, Uganda, Burundi, Rwanda, Algeria,

Table 21 – CC results for 1962 UNGA.

Group	#	Countries
1	6	United States of America, United Kingdom, France, Portugal, South Africa, Maldives
2	25	Canada, Dominican Republic, Nicaragua, Brazil, Bolivia, Paraguay, Uruguay, Ireland, Netherlands, Belgium, Luxembourg, Spain, German Federal Republic, Austria, Italy, Greece, Finland, Sweden, Norway, Denmark, Malawi, Israel, Japan, Australia, New Zealand
3	104	Bahamas, Cuba, Haiti, Jamaica, Trinidad and Tobago, Barbados, Mexico, Guatemala, Honduras, El Salvador, Costa Rica, Panama, Colombia, Venezuela, Guyana, Ecuador, Peru, Chile, Argentina, German Democratic Republic, Poland, Hungary, Czechoslovakia, Malta, Albania, Yugoslavia, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Iceland, Equatorial Guinea, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Ethiopia, Zambia, Lesotho, Botswana, Swaziland, Madagascar, Mauritius, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, Mongolia, Taiwan, India, Bhutan, Pakistan, Myanmar, Sri Lanka, Nepal, Thailand, Cambodia, Laos, Malaysia, Singapore, Philippines, Indonesia

Table 22 – SRCC results for 1973 UNGA.

Group	#	Countries
1	7	United States of America, United Kingdom, France, Malawi, South Africa, Israel, Maldives
2	40	Canada, Bahamas, Haiti, Dominican Republic, Barbados, Grenada, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Ecuador, Bolivia, Paraguay, Chile, Uruguay, Ireland, Netherlands, Belgium, Luxembourg, Spain, Portugal, German Federal Republic, Austria, Italy, Greece, Finland, Sweden, Norway, Denmark, Iceland, Ivory Coast, Turkey, Japan, Laos, Australia, New Zealand
3	91	Cuba, Jamaica, Trinidad and Tobago, Mexico, Venezuela, Guyana, Peru, Brazil, Argentina, German Democratic Republic, Poland, Hungary, Czechoslovakia, Malta, Albania, Yugoslavia, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Guinea-Bissau, Equatorial Guinea, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Ethiopia, Zambia, Lesotho, Botswana, Swaziland, Madagascar, Mauritius, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, China, Mongolia, India, Bhutan, Pakistan, Bangladesh, Myanmar, Sri Lanka, Nepal, Thailand, Cambodia, Malaysia, Singapore, Philippines, Indonesia

Table 23 – SRCC results for 1974 UNGA.

Group	#	Countries
1	1	Dominica
2	3	United States of America, United Kingdom, Israel
3	154	Canada, Bahamas, Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Grenada, St. Lucia, St. Vincent and the Grenadines, Antigua & Barbuda, St. Kitts and Nevis, Mexico, Belize, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Guyana, Suriname, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Netherlands, Belgium, Luxembourg, France, Spain, Portugal, German Federal Republic, German Democratic Republic, Poland, Austria, Hungary, Czechoslovakia, Italy, Malta, Albania, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Finland, Sweden, Norway, Denmark, Iceland, Cape Verde, Sao Tome and Principe, Guinea-Bissau, Equatorial Guinea, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Djibouti, Ethiopia, Angola, Mozambique, Zambia, Zimbabwe, Malawi, Lesotho, Botswana, Swaziland, Madagascar, Comoros, Mauritius, Seychelles, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, China, Mongolia, Japan, India, Bhutan, Pakistan, Bangladesh, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Cambodia, Laos, Vietnam, Malaysia, Singapore, Brunei, Philippines, Indonesia, Australia, Papua New Guinea, New Zealand, Vanuatu, Solomon Islands, Samoa

Table 24 – CC results for 1987 UNGA.

Group	#	**Countries**
1	5	United States of America, United Kingdom, France, German Federal Republic, Israel
2	153	Canada, Bahamas, Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Dominica, Grenada, St. Lucia, St. Vincent and the Grenadines, Antigua & Barbuda, St. Kitts and Nevis, Mexico, Belize, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Guyana, Suriname, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Netherlands, Belgium, Luxembourg, Spain, Portugal, German Democratic Republic, Poland, Austria, Hungary, Czechoslovakia, Italy, Malta, Albania, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Finland, Sweden, Norway, Denmark, Iceland, Cape Verde, Sao Tome and Principe, Guinea-Bissau, Equatorial Guinea, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Djibouti, Ethiopia, Angola, Mozambique, Zambia, Zimbabwe, Malawi, Lesotho, Botswana, Swaziland, Madagascar, Comoros, Mauritius, Seychelles, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, China, Mongolia, Japan, India, Bhutan, Pakistan, Bangladesh, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Cambodia, Laos, Vietnam, Malaysia, Singapore, Brunei, Philippines, Indonesia, Australia, Papua New Guinea, New Zealand, Vanuatu, Solomon Islands, Samoa

Table 25 – CC results for 1988 UNGA.

Group	#	Countries
1	3	United States of America, United Kingdom, Israel
2	155	Canada, Bahamas, Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Dominica, Grenada, St. Lucia, St. Vincent and the Grenadines, Antigua & Barbuda, St. Kitts and Nevis, Mexico, Belize, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Guyana, Suriname, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Netherlands, Belgium, Luxembourg, France, Spain, Portugal, German Federal Republic, German Democratic Republic, Poland, Austria, Hungary, Czechoslovakia, Italy, Malta, Albania, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Finland, Sweden, Norway, Denmark, Iceland, Cape Verde, Sao Tome and Principe, Guinea-Bissau, Equatorial Guinea, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Djibouti, Ethiopia, Angola, Mozambique, Zambia, Zimbabwe, Malawi, Lesotho, Botswana, Swaziland, Madagascar, Comoros, Mauritius, Seychelles, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, China, Mongolia, Japan, India, Bhutan, Pakistan, Bangladesh, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Cambodia, Laos, Vietnam, Malaysia, Singapore, Brunei, Philippines, Indonesia, Australia, Papua New Guinea, New Zealand, Vanuatu, Solomon Islands, Samoa

Table 26 – CC results for 1989 UNGA.

Group	#	**Countries**
1	4	United States of America, United Kingdom, France, Israel
2	5	Panama, Sao Tome and Principe, Equatorial Guinea, Liberia, Cambodia
3	149	Canada, Bahamas, Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Dominica, Grenada, St. Lucia, St. Vincent and the Grenadines, Antigua & Barbuda, St. Kitts and Nevis, Mexico, Belize, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Colombia, Venezuela, Guyana, Suriname, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Netherlands, Belgium, Luxembourg, Liechtenstein, Spain, Portugal, German Federal Republic, Poland, Austria, Hungary, Czechoslovakia, Italy, Malta, Albania, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Finland, Sweden, Norway, Denmark, Iceland, Cape Verde, Guinea-Bissau, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Djibouti, Ethiopia, Angola, Mozambique, Zambia, Zimbabwe, Malawi, Namibia, Lesotho, Botswana, Swaziland, Madagascar, Comoros, Mauritius, Seychelles, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, China, Mongolia, Japan, India, Bhutan, Pakistan, Bangladesh, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Laos, Vietnam, Malaysia, Singapore, Brunei, Philippines, Indonesia, Australia, Papua New Guinea, New Zealand, Vanuatu, Solomon Islands, Samoa

Table 27 – CC results for 1990 UNGA.

Group	#	Countries
1	2	United States of America, Israel
2	10	Equatorial Guinea, Croatia, Slovenia, Moldova, Georgia, Turkmenistan, Tajikistan, Kyrgyzstan, Uzbekistan, Kazakhstan
3	166	Canada, Bahamas, Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Dominica, Grenada, St. Lucia, St. Vincent and the Grenadines, Antigua & Barbuda, St. Kitts and Nevis, Mexico, Belize, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Guyana, Suriname, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, United Kingdom, Ireland, Netherlands, Belgium, Luxembourg, France, Liechtenstein, Spain, Portugal, German Federal Republic, Poland, Austria, Hungary, Czechoslovakia, Italy, Malta, Albania, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Estonia, Latvia, Lithuania, Ukraine, Belarus, Finland, Sweden, Norway, Denmark, Iceland, Cape Verde, Sao Tome and Principe, Guinea-Bissau, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Djibouti, Ethiopia, Angola, Mozambique, Zambia, Zimbabwe, Malawi, Namibia, Lesotho, Botswana, Swaziland, Madagascar, Comoros, Mauritius, Seychelles, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Kuwait, Bahrain, Qatar, United Arab Emirates, Oman, Afghanistan, China, Mongolia, North Korea, South Korea, Japan, India, Bhutan, Pakistan, Bangladesh, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Cambodia, Laos, Vietnam, Malaysia, Singapore, Brunei, Philippines, Indonesia, Australia, Papua New Guinea, New Zealand, Vanuatu, Solomon Islands, Marshall Islands, Federated States of Micronesia, Samoa, San Marino, Bosnia and Herzegovina, Armenia, Azerbaijan

Table 28 – CC results for 1991 UNGA.

Group	#	Countries
1	2	Yemen People's Republic, Mauritius
2	20	United States of America, Canada, United Kingdom, Netherlands, Belgium, Luxembourg, France, Portugal, Austria, Italy, Malta, Finland, Sweden, Norway, Denmark, Iceland, Malawi, South Africa, Australia, New Zealand
3	102	Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Mexico, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Guyana, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Spain, Poland, Hungary, Czechoslovakia, Albania, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Ethiopia, Zambia, Lesotho, Botswana, Madagascar, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Israel, Saudi Arabia, Yemen, Kuwait, Afghanistan, Mongolia, Taiwan, Japan, India, Pakistan, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Cambodia, Laos, Malaysia, Singapore, Philippines, Indonesia, Somalia

Table 29 – CC results for 1967 UNGA.

Group	#	Countries
1	30	United States of America, Canada, Barbados, Costa Rica, Brazil, United Kingdom, Ireland, Netherlands, Belgium, Luxembourg, France, Portugal, Austria, Italy, Malta, Finland, Sweden, Norway, Denmark, Iceland, Gambia, Malawi, South Africa, Lesotho, Botswana, Israel, Japan, Australia, New Zealand, Mauritius
2	37	Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Mexico, Guatemala, Honduras, El Salvador, Nicaragua, Panama, Colombia, Venezuela, Guyana, Ecuador, Peru, Bolivia, Paraguay, Chile, Argentina, Uruguay, Spain, Greece, Benin, Niger, Ivory Coast, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Rwanda, Madagascar, Turkey, Taiwan, Thailand, Laos, Philippines
3	57	Cuba, Poland, Hungary, Czechoslovakia, Albania, Yugoslavia, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Mali, Senegal, Mauritania, Guinea, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Ethiopia, Zambia, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Kuwait, Afghanistan, Mongolia, India, Pakistan, Myanmar, Sri Lanka, Maldives, Nepal, Cambodia, Malaysia, Singapore, Indonesia, Somalia, Yemen People's Republic

Table 30 – SRCC results for 1967 UNGA.

Group	#	Countries
1	1	Albania
2	13	United States of America, Canada, United Kingdom, Netherlands, Belgium, Luxembourg, France, Portugal, Malawi, South Africa, Botswana, Australia, New Zealand
3	112	Cuba, Haiti, Dominican Republic, Jamaica, Trinidad and Tobago, Barbados, Mexico, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Guyana, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Spain, Poland, Austria, Hungary, Czechoslovakia, Italy, Malta, Yugoslavia, Greece, Cyprus, Bulgaria, Russia, Ukraine, Belarus, Finland, Sweden, Norway, Denmark, Iceland, Equatorial Guinea, Gambia, Mali, Senegal, Benin, Mauritania, Niger, Ivory Coast, Guinea, Burkina Faso, Liberia, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Gabon, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Ethiopia, Zambia, Lesotho, Swaziland, Madagascar, Mauritius, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Turkey, Iraq, Egypt, Syria, Lebanon, Jordan, Israel, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Afghanistan, Mongolia, Taiwan, Japan, India, Pakistan, Myanmar, Sri Lanka, Maldives, Nepal, Thailand, Cambodia, Laos, Malaysia, Singapore, Philippines, Indonesia

Table 31 – CC results for 1969 UNGA.

Group	#	Countries
1	21	United States of America, Canada, United Kingdom, Netherlands, Belgium, Luxembourg, France, Portugal, Austria, Italy, Malta, Sweden, Norway, Denmark, Iceland, Gambia, Malawi, South Africa, Botswana, Australia, New Zealand
2	40	Dominican Republic, Jamaica, Trinidad and Tobago, Mexico, Guatemala, Honduras, El Salvador, Nicaragua, Costa Rica, Panama, Colombia, Venezuela, Ecuador, Peru, Brazil, Bolivia, Paraguay, Chile, Argentina, Uruguay, Ireland, Spain, Greece, Cyprus, Finland, Senegal, Ivory Coast, Liberia, Gabon, Lesotho, Swaziland, Madagascar, Turkey, Israel, Taiwan, Japan, Maldives, Thailand, Laos, Philippines
3	65	Cuba, Haiti, Barbados, Guyana, Poland, Hungary, Czechoslovakia, Albania, Yugoslavia, Bulgaria, Russia, Ukraine, Belarus, Equatorial Guinea, Mali, Benin, Mauritania, Niger, Guinea, Burkina Faso, Sierra Leone, Ghana, Togo, Cameroon, Nigeria, Central African Republic, Chad, Congo, Democratic Republic of the Congo, Uganda, Kenya, Tanzania, Burundi, Rwanda, Somalia, Ethiopia, Zambia, Mauritius, Morocco, Algeria, Tunisia, Libya, Sudan, Iran, Iraq, Egypt, Syria, Lebanon, Jordan, Saudi Arabia, Yemen, Yemen People's Republic, Kuwait, Afghanistan, Mongolia, India, Pakistan, Myanmar, Sri Lanka, Nepal, Cambodia, Malaysia, Singapore, Indonesia

Table 32 – SRCC results for 1969 UNGA.