Optimization Algorithms for Clique Problems

Yi Zhou

To cite this version:

Yi Zhou. Optimization Algorithms for Clique Problems. Optimization and Control [math.OC]. Université d'Angers, 2017. English. NNT : 2017ANGE0013 . tel-01707043

HAL Id: tel-01707043

https://tel.archives-ouvertes.fr/tel-01707043

Submitted on 12 Feb 2018

Thèse de Doctorat

Yi ZHOU

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
Label européen
sous le sceau de l'Université Bretagne Loire

École doctorale : 503 (STIM)

Discipline : Informatique, section CNU 27
Unité de recherche : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)

Soutenue le 29 Juin 2017
Thèse n° : 1

Optimization Algorithms for Clique Problems

JURY

Rapporteurs :	**M. Christian Blum**, Directeur de recherche, Spanish National Research Council (CSIC)
	M. Sébastien Vérel, Maître de conférences HDR, Université du Littoral Côte d'Opale
Examinateurs :	**M. André Rossi**, Professeur, Université d'Angers
	M. Marc Sevaux, Professeur, Université de Bretagne-Sud
	M^{me} Yang Wang, Professeur, Northwestern Polytechnic University
Directeur de thèse :	**M. Jin-Kao Hao**, Professeur, Université d'Angers
Co-directeur de thèse :	**M. Adrien Goëffon**, Maître de conférences HDR, Université d'Angers

Acknowledgement

Four years ago, I started my journey in the joint area of computer science, combinatorial optimization and artificial intelligence in LERIA, University of Angers. From a foreign beginner who hardly knows anything around to a final year PhD student who enjoys very much the research and daily life in France, I feel that a few lines are not enough to thank everyone who has helped me adapting the work and life.

I would like firstly thank my supervisor Prof. Jin-Kao Hao, who introduced me the interesting clique problems. In order to achieve an efficient algorithm, one may do many unfruitful experiments and try a lot unnecessary techniques. Thanks to him, I got many valuable advises and avoid some detours. Prof. Hao is also very kind and always be patient with my questions. I also appreciate my associate supervisor, Prof. Adrien Goëffon, who I cooperated a lot during the four years. He not only proposed some interesting ideas, but also proofread some of my manuscripts.

My thanks also go to another professor, Prof. André Rossi who had a lot of interesting talk with me. His rich experience in integer programming gave me another key to solve the combinatorial problems. I also thank my office mates Arthur Chambon, Marc Legeay who spent a great deal of time solving my daily-life problems since I spoke french poorly. We also have a lot of interesting non-work related discussions. Additionally, I want to thank our technicians Eric Girardeau and Jean-Mathieu Chantrein, our nice secretaries Catherine Pawlonski and Christine Bardarine, my new office mate Hugo Traverson and all the other lab members. Thanks to their supports, the research work becomes much more easier and enjoyable in LERIA. Also, I would like to thank the Center of French Language for Foreigners (CeLFE) in University of Angers. After following their free courses, I am able to speak french in daily life and also make some foreign friends.

I am particularly grateful to my parents, my elder sister Zhifang Zhou as well as her husband Min Zhu. Thanks for all their love and selfless support! Without them, this work would not be possible. I also thank my Chinese friends Yuning Chen, Zhanghua Fu, Yan Jin, Fuda Ma, Xiangjin Lai, Le Li, Wen Sun, Yangming Zhou, Zhi Lu, Jintong Ren in Angers for their accompanies. I also appreciate my best friends in Chengdu – thanks for being there for me.

I also thank all the jury members, for their efforts in reviewing and improving this thesis.

This research has been financially supported by China Scholarship Council (CSC).

Contents

General Introduction

Context

A clique in a graph is a subset of vertices which are pairwise adjacent. Generally, clique problems involve finding the maximum clique in the graph or grouping the vertices into disjoint cliques. In this thesis, we are interested in four representative clique problems. The first one is the maximum vertex weight clique problem (MVWCP) which is the generalized version of the classic maximum clique problem (MCP). The second one, the maximum s-plex problem (MsPlex), aims at locating the maximum relaxed clique (i.e., s-plex) in a graph. The maximum balanced biclique problem (MBBP) is the third clique problem, which limits MCP to bipartite graph and enforces the same number of vertices in both partitions. The last one, the clique partitioning problem (CPP), requires to partition the vertices of an edge-weighted complete graph into different groups such that the sum of edge weights within all groups is maximized. These clique problems are relevant models in practice since they can be used to formulate numerous applications in popular social network analysis, computer vision, manufacturing, economics and so on. These problems are also theoretically significant as they can represent other classical combinatorial optimization problems (e.g., winner determination problem), or used as sub-problems of other more complicated applications (e.g., community detection). Due to these reasons, research on these problems has become more and more intense in recent years.

However, all these problems are known to be NP-Hard. As a result and unless $P = NP$, it is hopeless to solve them exactly in the general case in polynomial time. Moreover, real-life graphs are normally massive (with millions even billions of vertices and edges), which increases the intractability of these combinatorial problems. For these reasons, we are interested mainly in tackling them by heuristic and metaheuristic algorithms, even if we also investigate some exact approaches. To assess the proposed algorithms, we perform extensive computational experiments on well-known benchmarks and show comparisons with state-of-the-art approaches whenever this is possible. We also investigate the key components of each proposed algorithm to shed light on their impact on the performance of the algorithm.

Objectives

The main objectives of this thesis are summarized as follows:

- detect problem-specific features and design corresponding heuristic rules for the search algorithms.
- develop perturbation strategies which are able to help local search algorithm escape from local optima.
- design graph reduction techniques in order to deal with massive graphs which emerge in real-life applications.
- design effective exact algorithms, and mainly focus on new bounding techniques and tightened mathematical formulations.
- devise specific techniques and data structures to ensure a high computational efficiency of the proposed algorithms.
- evaluate the proposed algorithms on a wide range of benchmark instances, and perform a comprehensive comparison with the state-of-the-art.

Contributions

The main contributions of this thesis are summarized below:

– For MVWCP, we proposed a powerful move operator $PUSH$, which generalizes the existing move operators. Based on this operator, we designed two restart tabu search algorithms (ReTS-I and ReTS-II) which explore different candidate push sets. Computational results indicates that our new algorithms compete favorably with the leading MVWCP algorithms of the literature. The work has been published in *European Journal of Operational Research 257(1): 41-54, 2017* [Zhou *et al.*, 2017a].

– For MsPlex, we proposed a frequency-driven multi-neighborhoods tabu search algorithm (FD-TS). The algorithm relies on two traditional move operators (ADD and $SWAP$) for search space exploration and a frequency-driven move operator ($PRESS$) for perturbation. We tested the algorithm on all 80 DIMACS instances and 47 representative real-life instances. The results indicated that FD-TS outperforms the existing algorithms and sets new records for numerous instances with different values s. A paper describing this work is accepted to *Computer & Operations Research* [Zhou and Hao, 2017b].

– For MBBP, we propose a new constraint-based tabu search algorithm called CBTS which is based on the aforementioned $PUSH$ operator for MWCP as well as two graph reduction techniques. This algorithm is particularly efficient on random dense instances and very large real-life sparse instances. Experiments indicate that this algorithm outperforms state-of-the-art algorithms. This study was presented in a paper submitted to *Expert Systems with Applications* [Zhou and Hao, 2017a].
  We also investigate exact solution approaches for MBBP. We propose an upper bound propagation procedure (UBP) to tighten the upper bound of all vertices, which are then used by specifically designed exact algorithms. We assessed the effectiveness of these new algorithms and showed that they are quite suitable to deal with large sparse real-life graphs. The work is currently under revision for *European Journal of Operational Research* [Zhou *et al.*, 2017b].

– For CPP, we proposed a three-phase local search heuristic CPP-P^3. The three phases correspond to a descent search, an exploration tabu search and a directed perturbation. One key element of CPP-P^3 is its restricted neighborhood which is based on the notion of *TOP MOVE* of a vertex. Experiments and additional analyses showed the high performance of the algorithm and the effectiveness of its components. This work has been published in *Journal of Combinatorial Optimization 32(2): 469-491, 2016* [Zhou *et al.*, 2016].

Organization

The manuscript is organized in the following way:

– In the first chapter, we first introduce the four clique problems considered in this thesis. Then we present a number of applications related to these problems and a brief overview of existing exact and heuristic clique algorithms. We also introduce the benchmark and methodologies for algorithm assessment.

– In the second chapter, we study the maximum vertex weight problem (MVWCP) and review existing algorithms for its resolution. Then, we introduce the generalized $PUSH$ operator, followed by two push-based algorithms, ReTS-I and ReTS-II. We provide an experimental study of the new algorithms, as well as comparisons with state-of-the-art algorithms.

– In the third chapter, we study the maximum s-plex problem (MsPlex). This chapter begins with a short introduction and a short review of existing approaches for this problem. Then, we present the proposed FD-TS algorithm and some important implementation issues concerning this new algorithm. Afterward, we proceed to a thorough computational study of FD-TS on three well-known benchmark sets.

– In the fourth chapter, we first introduce the maximum bipartite biclique problem (MBBP) and review the existing algorithms for MBBP. Then we propose a new heuristic algorithms called TSGR-MBBP which combines tabu search and graph reduction techniques, and report computational results and analysis of this algorithm. We also show our improvements for exact MBBP solutions. This involves introducing an upper bound propagation procedure, a new mathematical formulation and a novel exact search algorithm. Then, we present computational results in order to verify the effectiveness of these improvements.

- In the last chapter, we consider the clique partitioning problem (CPP). After introducing CPP and existing resolution strategies, we present our three-phase local search algorithm CPP-P^3. Computational comparisons between CPP-P^3 and other algorithms are presented. We provide an analysis of different components of CPP-P^3, as well as the landscape on some representative instances.

1

Introduction

Contents

1.1 Clique problems

A *clique*, defined as a subset of vertices which are pairwise adjacent, is an important concept in graph theory. There are many significant clique problems in combinatorial optimization and integer programming. The maximum clique problem (MCP), which is to find the clique of maximum cardinality from the given graph, is one of the most classic NP-hard problems. Many important combinatorial problems like graph coloring [Wu and Hao, 2012], graph clustering [Cramton *et al.*, 2006] and vertex cover [Cai, 2015] can be directed formulated as a maximum clique problem or have a sub-problem which requires to solve MCP efficiently. Practical applications of MCP in real-life are also very common in fields like bioinformatics [Malod-Dognin *et al.*, 2010], coding theory [Etzion and Ostergard, 1998], chemoinformatics [Ravetti and Moscato, 2008], etc. Due to the significance of MCP, there has been a rich number of theoretical and practical studies in the past decades, especially after the Second DIMACS Implementation Challenge dedicated to maximum clique, graph coloring, and satisfiability organized during 1992–1993.

However, in the recent years, relaxed or generalized clique models have become popular since these models are more convenient to formulate the real-life applications. In this work, we mainly concentrate on the following four variants or generalizations of the difficult clique problems. The four problems studied in this thesis extend the applications of clique model to more fields (see Section 1.2).

– **The Maximum Vertex Weight Clique Problem (MVWCP).**
Given an undirected graph $G = (V, E, W)$ with vertex set V and edge set E, let $W : V \to \mathbb{R}^+$ be a weighting function that assigns to each vertex $v \in V$ a positive value w_v. A clique $C \subseteq V$ of G is a subset of vertices such that its induced subgraph is complete, i.e., every two vertices in C are pairwise adjacent in G ($\forall u, v \in C, \{u, v\} \in E$). For a clique C of G, its weight is given by $W(C) = \sum_{v \in C} w_v$. The maximum vertex weight clique problem (MVWCP) is to determine a clique C^* of maximum vertex weight.
MVWCP can be formulated as a mixed integer linear program (MILP) as follows.

$$\max \quad W(G) = \sum_{i \in V} w_i x_i \tag{1.1}$$

subject to:

$$x_i + x_j \leq 1, \forall \{i, j\} \in \bar{E} \tag{1.2}$$

$$x_i \in \{0, 1\}, \forall i \in V \tag{1.3}$$

In this formulation, x_i is a binary variable associated with vertex i, $\bar{E}$ is the edge set of the complement graph of G. Objective (1.1) maximizes the clique weight. Constraint (1.2) indicates that only one of any two non-adjacent vertices can be selected in the clique.
Obviously, MCP can be considered as a special case of MVWCP where the weight of each vertex is equal to one. For this reason, MVWCP is also NP-Hard since it has at least the same computational complexity as its unweighted counterpart.

– **The Maximum s-Plex problem (MsPlex).**
MsPlex is similar to MCP except that the clique constraint is relaxed. Formally, given an undirected graph $G = (V, E)$ with vertex set V and edge set E, let $N(v)$ denote the set of vertices adjacent to v in G. Then, an s-plex for a given integer $s \geq 1$ ($s \in \mathbb{N}^+$) is a subset of vertices $C \subseteq V$ that satisfies the following condition: $\forall v \in C, |N(v) \cap C| \geq |C| - s$. Thus, each vertex of an s-plex C must be adjacent to at least $|C| - s$ vertices in the subgraph $G[C] = (C, E \cap (C \times C))$ induced by C. Therefore, MsPlex involves finding, for a fixed value of s, an s-plex of maximum cardinality among all possible s-plexes of a given graph.
MsPlex can be also formulated as a MILP as follows [Balasundaram *et al.*, 2011]:

$$\max \quad \omega_s(G) = \sum_{i \in V} x_i \tag{1.4}$$

subject to:

$$\sum_{j \in V \setminus (N(i) \cup \{i\})} x_j \leq (s-1)x_i + \bar{d}_i(1-x_i), \forall i \in V \tag{1.5}$$

$$x_i \in \{0,1\}, \forall i \in V \tag{1.6}$$

where x_i is the binary variable associated to vertex i ($x_i = 1$ if vertex i is in the s-splex, $x_i = 0$ otherwise). Also, $\bar{d}_i = |V \setminus N(i)| - 1$ denotes the degree of vertex i in the complement graph $\bar{G} = (V, \bar{E})$. Note that $i \notin N(i)$ by definition.

MsPlex was first proposed in [Seidman and Foster, 1978] in 1978. Clearly, when s equals one, MsPlex is equivalent to MCP, in this sense, MsPlex is a relaxed problem of MCP. The decision version of MsPlex was proven to be NP-Complete in [Balasundaram *et al.*, 2011].

– **The Maximum Bipartite Biclique Problem (MBBP).**

Given a bipartite graph $G = (U, V, E)$ with two disjoint vertex sets U, V and an edge set $E \subseteq U \times V$, a biclique $(X, Y) = X \cup Y$ is the union of two subsets of vertices $X \subseteq U$, $Y \subseteq V$ such that $u \in X, v \in Y$ implies that $\{u, v\} \in E$. In other words, the subgraph induced by the vertex set $X \cup Y$ is a complete bipartite graph. If $|X| = |Y|$, then (X, Y) is called a balanced biclique of G. The Maximum Balanced Biclique Problem (MBBP) is to find a balanced biclique (X^*, Y^*) of maximum cardinality of G, (X^*, Y^*) being the maximum balanced biclique of size $|X^*|$ (or $|Y^*|$) [Garey and Johnson, 1979].

$$\max \ \ \omega(G) = \sum_{i \in U} x_i \tag{1.7}$$

subject to:

$$x_i + x_j \leq 1, \forall \{i, j\} \in \bar{E} \tag{1.8}$$

$$\sum_{i \in U} x_i - \sum_{i \in V} x_i = 0 \tag{1.9}$$

$$x_i \in \{0,1\}, \forall i \in U \cup V \tag{1.10}$$

where each vertex of $U \cup V$ is associated to a binary variable x_i indicating whether the vertex is part of the biclique, $\bar{E}$ is the set of edges in the bipartite complement of G. Objective (1.7) maximizes the size of the biclique. Constraint (1.8) ensures that each pair of non-adjacent vertices cannot be selected at the same time (i.e., the solution must be a clique). Equation (1.9) enforces that the returned biclique is balanced.

Obviously, MBBP is another clique problem with regards to the bipartite graph. As shown in [Garey and Johnson, 1979; Alon *et al.*, 1994], the decision version of MBBP is NP-complete in the general case, even though the maximum biclique problem without the balance constraint (Eq. (1.9)) is polynomially solvable by the maximum matching algorithm [Cheng and Church, 2000].

– **The Clique Partitioning Problem (CPP).**

Let $G = (V, E, W)$ be a complete edge-weighted undirected graph with a vertex set V and edge set $E = V \times V$, $W : E \rightarrow \mathbb{R}$ be an edge weighting function that assigns to each edge $\{u, v\}$ a weight $w_{uv} \in \mathbb{R}$. A subset $A \subseteq E$ is called a clique partition if there is a partition of V into nonempty disjoint sets $V_1, ..., V_k$ ($k \in \mathbb{Z}^+$ is unfixed) such that

$$A = \bigcup_{p=1}^{k} \{\{i, j\} | i, j \in V_p, i \neq j\} \tag{1.11}$$

The weight of such a clique partition is defined as $\sum_{\{i,j\} \in A} w_{ij} x_{ij}$. In other words, the Clique Partitioning Problem (CPP) consists in clustering all the vertices into k mutually disjoint subsets (or groups), such that the sum of the edge weights of all groups is as large as possible [Grötschel and Wakabayashi, 1989; Grötschel and Wakabayashi, 1990; Wakabayashi, 1986].

We can reformulate this problem as a MILP by associating each edge $\{i, j\} \in E$ to a binary variable x_{ij} as follows [Dorndorf and Pesch, 1994].

$$\max \quad f(G) = \sum_{\{i,j\} \in E} w_{ij} x_{ij} \tag{1.12}$$

subject to:

$$x_{ij} + x_{jk} - x_{ik} \leq 1, \forall i, j, k \in V \wedge i < j < k \tag{1.13}$$

$$x_{ij} - x_{jk} + x_{ik} \leq 1, \forall i, j, k \in V \wedge i < j < k \tag{1.14}$$

$$- x_{ij} + x_{jk} + x_{ik} \leq 1, \forall i, j, k \in V \wedge i < j < k \tag{1.15}$$

$$x_{ij} \in \{0, 1\}, \forall \{i, j\} \in E \tag{1.16}$$

where variable x_{ij} is associate to edge $\{i, j\}$. x_{ij} equals 1 if the two end vertices of edge $\{i, j\}$ are inside the same group, 0 otherwise. The constraints (1.13), (1.14) and (1.15) assure that, for any triangle (a clique of three vertices) in the graph, if two of the vertices in the triangle are part of the same group, then the third vertex is also a part of this group.

The set of edges which have both ends in different groups is called a cut of G. In order to find groups as homogeneous as possible, positive edges should appear within groups and negative edges in the cut. Hence, a optimal partition has a minimal cut weight. The decision problem of CPP is also NP-complete [Wakabayashi, 1986].

1.2 Applications

As a generalization of MCP, MVWCP algorithms can be used to solve the MCP when the vertices are associated with different weights. A classical application is solving the Winner Determination Problem (WDP) in combinatorial auctions. Given a collection of bids $B = \{B_1, B_2, ..., B_n\}$, each bid B_j being specified by its set of items S_j and its associated price P_j, WDP is to find an allocation of items to bidders in order to maximize the auctioneer's revenue under the constraint that each item is allocated to at most one bid. One can transform an instance of WDP to a weighted graph $G = (V, E, W)$ where each vertex is associated to a bid and its weight is equal to the corresponding price. Two vertices are adjacent only if the corresponding bids do not share any common item. Then the WDP is to find the maximum weight clique from this graph [Wu and Hao, 2015b; Wu and Hao, 2016].

Another applications of MVWCP can be found in computer vision and robotics. In order to match a new image against a model, one approach is to find the maximum weight clique from an auxiliary graph where each vertex represents putative association between features of the graphs being matched, and vertex weight represents the similarity. An edge exists between two associations if they are compatible. Then the matching problem is equivalent to MVWCP [Ballard and Brown, 1982].

In social network analysis, clique is also a popular model used to describe cohesive subgroups whose members are closely related. Recent studies point out that the clique model idealizes the structural properties as the individuals in a cohesive group may not be pairwise connected. In this sense, the clique model biases the real situation especially when the graph is built on inexact, empirical data. Consequently, *clique relaxation* models like s-plex are often referred [Pattillo *et al.*, 2012; Pattillo *et al.*, 2013b]. Other clique relaxation models include s-defective clique [Yu *et al.*, 2006], quasi-clique [Pajouh *et al.*, 2014; Brunato *et al.*, 2007; Pattillo *et al.*, 2013a], and k-club [Bourjolly *et al.*, 2000], which are defined by relaxing the edge number, the edge density, and the pairwise distance of vertices in an induced subgraph, respectively. In terms of MsPlex, an application is to find profitable diversified portfolios on the stock market [Boginski *et al.*, 2014]. In this application, a market graph is firstly built by assigning each vertex (stock) a weight which corresponds to its return over the considered time period, and connecting each pair of vertices by an edge if the correlation between the corresponding pair of stocks does not exceed a certain threshold θ. The maximum weight s-plex of this market graph corresponds to the recommended set of profitable diversified portfolios.

The above cases mainly concerns the clique or relaxed clique models in a general graph (i.e., non-bipartite). When the graph is bipartite, MCP problem becomes trivial as it can be solved polynomially [Cheng and Church, 2000]. Now, a more challenging problem is to find the maximum balanced biclique. This problem denoted as MBBP is relevant in real-life applications. In nanoelectronic system design, MBBP is used to identify the maximum defect-free crossbar from a partially fabricated defective crossbar represented by a bipartite graph [Tahoori, 2006; Al-Yamani *et al.*, 2007; Tahoori, 2009]. In computational biology, MBBP is applied to simultaneously group genes and their expressions under different conditions (called biclustering) [Cheng and Church, 2000]. Another application can be found in the field of VLSI for PLA-folding [Ravi and Lloyd, 1988]. The clique and biclique models are also popular tools for mining patterns for clustering on numerical datasets [Gutierrez-Rodríguez *et al.*, 2015], semantic interpretation of clusters contents [Role and Nadif, 2014], and community detection in complex networks [Zhi-Xiao *et al.*, 2016].

CPP is a clique grouping problem, whereas the above MVWCP, MsPlex and MBBP problems are subset selection problems. CPP has also a number of practical applications such as biology, flexible manufacturing systems, airport logistics, and social sciences. For instance, in qualitative data analysis, CPP can be used to uncover natural groupings, or types of objects, each one being characterized by several attributes. One can bijectively associate these objects with vertices of an edge-weighted graph G; each positive or negative edge weight represents some measure of similarity or dissimilarity of two objects linked by the edge. The associated CPP problem consists in determining an appropriate partition of the vertices. In biology, the classification of animals and plants is based on qualitative and/or quantitative descriptions. As the number of classes is unknown a priori, CPP is well suited for determining such classifications [Grötschel and Wakabayashi, 1989]. In transportation, an application of CPP to a flight-gate scheduling problem is presented in [Dorndorf *et al.*, 2008]. In manufacturing, CPP can be used to determine different groups of products as shown in [Oosten *et al.*, 2001; Wang *et al.*, 2006].

1.3 Evolution of maximum clique algorithms

To introduce existing clique algorithms, it is inevitable to first briefly present MCP algorithms since many clique algorithms find their origin in the strategies developed for MCP. Existing approaches for MCP mainly fall into two categories: exact algorithms and heuristic algorithms.

An exact algorithm aims at finding provable optimal solutions. However, it is well known that MCP is a fundamental NP-Hard problem and its decision version is among the first 21 Karp's NP-complete problems [Karp, 1972]. Therefore, it is unrealistic to find a polynomial algorithm for MCP unless P=NP. One alternative is to implicitly enumerate the solutions under a branch and bound (B&B) scheme. Numerous exact solvers based on the original well-known B&B algorithm [Carraghan and Pardalos, 1990] for MCP have been proposed. The two key issues of improving the efficiency of B&B algorithms are bounding techniques and branch heuristics. From the literature, we mainly found the following techniques: 1) obtain improved upper bounds on the size of maximum clique with the help of some previously computed smaller graphs [Östergård, 2002]; 2) bound the maximum clique size using vertex coloring [Tomita and Seki, 2003; Tomita and Kameda, 2007; San Segundo *et al.*, 2011]; 3) remove unpromising vertices [Fahle, 2002; Verma *et al.*, 2015]; 4) combine vertex coloring with MaxSAT reasoning techniques [Li and Quan, 2010]. These algorithms have been demonstrated to be very efficient in solving a wide range of MCP instances. Consequently, these techniques have also been adapted for other clique problems like aforementioned MVWCP, MsPlex and MBBP. For example, the first bounding techniques mentioned above were also employed in [Östergård, 1999] to solve MVWCP, in [Trukhanov *et al.*, 2013] to solve MsPlex. The vertex coloring technique was used in [Kumlander, 2004; Warren and Hicks, 2006; Wu and Hao, 2016] to bounding and branching for solving MVWCP. In [McClosky and Hicks, 2012], an algorithm for MsPlex was proposed using the co-s-plex bounding technique, which is an adaption of the vertex coloring technique for MsPlex. In [Fang *et al.*, 2016], the MaxSAT Reasoning is used as a bounding technique for MVWCP as well.

Except the B&B based exact algorithms, mixed integer programming (MIP) solvers like CPLEX constitutes another approach to solve exactly clique problems. The efficiency of MIP solvers highly depends on the mathematical formulation of the target problem. Numerous studies on the formulations of MCP can be found for the last decades. The simplest formulation for MCP may be the one described in Equation (1.7) – (1.10) with vertex weights set to one. This formulation can be tightened by odd-cycle inequalities [Nemhauser and Trotter, 1974], clique inequalities [Padberg, 1973], rank inequalities [Mannino and Sassano, 1996], etc. Meanwhile, many polynomial time separation algorithms have been proposed to separate these inequalities. Interested readers are referred to [Rebennack *et al.*, 2012] for more details. These inequalities for MCP can be directly employed to describe MVWCP since they share the same polyhedral structure. With some adaptions, they can also be used for describing MsPlex. The co-k-plex inequalities and hole inequalities in [Balasundaram *et al.*, 2011] are adapted from the clique inequalities and odd-cycle inequalities. However, in terms of CPP, the polyhedral structure is quite different from that of clique. Some interesting studies and a cutting plane algorithm of CPP can be found in [Grötschel and Wakabayashi, 1989].

On the other hand, heuristic algorithms constitute another way to search high quality solutions in a reasonable time, but without proving optimality. As far as we know, local search is the most successful framework for designing effective MCP heuristics. The scheme of existing local search algorithms generally can be divided into two types: search a legal k-clique by increasing k values, namely k-fixed penalty strategy; or search all the feasible cliques to locate the ones with large cardinality, i.e., legal strategy. The two types of strategies can be further classified by the move operator used to explore the search space. For the k-fixed penalty strategy, a swap operator is often employed to find a clique of size k. While for the legal strategy, different move operators have been used, most of the time, adding a vertex to the incumbent clique, swapping a vertex from the clique against another one outside the clique or drop one vertex from the clique.

Meanwhile, local search has been successfully used for MVWCP in [Benlic and Hao, 2013a; Pullan, 2008; Wang *et al.*, 2016; Wu *et al.*, 2012; Wu and Hao, 2015b]. These algorithms are also based on add, swap, drop operators which were applied in local search algorithms for MCP for search intensification. As for MsPlex, only two GRASP based [Feo and Resende, 1995a] heuristic algorithms are found in [Miao and Balasundaram, 2012; Gujjula *et al.*, 2014]. For CPP, the reallocation of one vertex was also a common way of constructing new neighbor solutions in many local seach algorithms, for example tabu search and simulated annealing in [De Amorim *et al.*, 1992; Kirkpatrick *et al.*, 1983; Brusco and Köhn, 2009]. The reallocation algorithm was sometimes combined with an ejection chain heuristic in [Dorndorf and Pesch, 1994], with noisy heuristic in [Charon and Hudry, 2001; Charon and Hudry, 2006]. Recently, we noticed that two heuristic algorithms [Palubeckis *et al.*, 2014; Brimberg *et al.*, 2015] also employed reallocation and swap operators.

In the following chapters, we will review, for each considered problem, additional and specific state-of-the-art elements.

1.4 Algorithm assessment

For each clique problem considered in this thesis, one can find in the literature several heuristic or exact algorithms, which have to be carefully evaluated. Due to the lack of theory for metaheuristic algorithms, most evaluations are based on computational experiments. We give an introduction of the important aspects of algorithm assessment for the clique problems in this thesis.

1.4.1 Benchmarks

It is unrealistic to study the performance of existing algorithms to every possible instances. Hence, in practice, we often chose a set of diversified instances built under different settings. The benchmark sets used in this thesis include those commonly used in the literature and some new instances proposed in this work. We summarize these benchmarks according to their associate problems.

The instances for MVWCP can be divided into three sets.

- **The 2nd DIMACS Implementation Challenge Benchmark (2nd DIMACS)**. This set of 80 instances originated from the second DIMACS implementation challenge for the maximum clique problem [1]. These instances cover both real world problems (coding theory, fault diagnosis, Steiner Triple Problem...) and random graphs. They include small graphs (50 vertices and 1,000 edges) to large graphs (4,000 vertices and 5,000,000 edges). Though DIMACS graphs were originally collected for benchmarking MCP algorithms, these graphs are still very popular and widely used as a testbed for evaluating MVWCP algorithms [Benlic and Hao, 2013a; Fang *et al.*, 2016; Mannino and Stefanutti, 1999; Pullan, 2008; Wang *et al.*, 2016; Wu *et al.*, 2012], and MsPlex algorithms [Balasundaram *et al.*, 2011; McClosky and Hicks, 2012; Trukhanov *et al.*, 2013; Moser *et al.*, 2012].
- **BHOSLIB benchmarks**. The BHOSLIB (Benchmarks with Hidden Optimum Solutions) instances were generated randomly in the SAT phase transition area according to the model RB [2]. The 40 instances included in this set were widely used to test MCP and MVWCP algorithms. The sizes of these instances range from 450 vertices and 17,794 edges, to 1,534 vertices and 127,011 edges. When testing MWCP algorithms, the weight of each vertex i is equal to $i \mod 200 + 1$.
- **Winner Determination Problem (WDP) benchmarks**. The Winner Determination Problem can be reformulated as a MVWCP and thus solved by MVWCP solvers [Wu and Hao, 2015b; Wu and Hao, 2016]. Therefore, benchmark instances for WDP can also be used to test the performance of MVWCP solvers.

For MsPlex, except the classical 2nd DIMACS instances (with unit vertex weight), we additionally consider two benchmark sets which include very large real-life instances.

- **The 10th DIMACS Implementation Challenge Benchmark (10th DIMACS)** [3]. This testbed contains many large networks, including artificial and real-world graphs from different applications. This benchmark set is popular for testing graph clustering and partitioning algorithms. More information about the graphs can be obtained from [Bader *et al.*, 2012].
- **Stanford Large Network Dataset Collection (SNAP)** [4]. SNAP provides a large range of large-scale social and information networks [Leskovec and Sosič, 2016], including graphs retrieved from social networks, communication networks, citation networks, web graphs, product co-purchasing networks, Internet peer-to-peer networks, and Wikipedia networks. Some of these networks are directed graphs, so we simply ignored the direction of each edge and eliminate loops and multiple edges.

MBBP requires the input graph to be bipartite, thus the above popular instances are not directly applicable. In this work, we adopt a set of randomly generated benchmark instances as well as instances from popular Koblenz Network Collection [Kunegis, 2013].

- **Random Graphs**. This set of benchmark instances includes 30 randomly generated dense graphs. In each graph, the two vertex sets U and V have an equal cardinality (i.e., $|U| = |V|$) and an edge between a pair of vertices $\{u, v\}$ ($u \in U, v \in V$) exists with an uniform probability p ($0 < p < 1$) which defines the edge density of the graph. For our study, we used random graphs generated by the same rule of [Yuan *et al.*, 2015] so that the performances of different algorithms can be compared. A theoretical analysis in [Dawande *et al.*, 2001] showed that the maximum balanced size in such random graphs locates in range $[\frac{\ln n}{\ln(1/p)}, \frac{2 * \ln n}{\ln(1/p)}]$ with high probability (when n is sufficiently large).

[1] http://www.cs.hbg.psu.edu/txn131/clique.html

[2] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm

[3] http://www.cc.gatech.edu/dimacs10/downloads.shtml

[4] http://snap.stanford.edu/data/

- **The Koblenz Network Collection (KONECT)** [5]. The entire collection contains hundreds of networks derived from different real-life applications, including social networks, hyperlink networks, authorship networks, physical networks, interaction networks and communication networks. These networks are commonly used in the social network analyzing [Hardiman and Katzir, 2013].

For CPP, the input graphs are always complete ones. Thus the instances vary in terms of vertex number and edge weight distribution. We employ the following four sets [6].

- **Group I**: a set of 7 instances which constitutes the benchmark reported in the literature in 2006 [Charon and Hudry, 2006]. Instances named "rand100-100","rand300-100","rand 500-100" are generated by choosing a random integer for the edge weights in range [-100, 100], while the weights of "rand300-5" and "zahn300" respectively take value in range [-5, 5] and set $\{-1, 1\}$. To generate "sym300-50", 50 symmetric relations among 300 vertices were established, and the differences between related and unrelated components were used to compute the edge weights. To create the last instance named "regnier300-50", 50 bipartitions of 300 vertices were established and the difference between the number of bipartitions for which each pair of vertices is or is not in the same cluster was used to obtain the edge weights.
- **Group II**: a set of 6 instances originally proposed in 2009 [Brusco and Köhn, 2009] ("rand200-100", "rand400-100","rand100-5", "rand200-5", "rand400-5", "rand500-5"). For these graphs, (integer) edge weights are uniformly generated in range [-100,100] or [-5, 5].
- **Group III**: a set of 35 instances reported in 2014 [Palubeckis *et al.*, 2014]. These instances are grouped into 4 categories by the number of vertices. Edge weights are also uniformly distributed in range [-5,5] or [-100, 100].
- **Group IV**: a set of 15 additional instances specifically generated for this study. We provide a first set of 5 instances with 500 vertices, where edge weights are generated using gaussian distribution $\mathcal{N}(0, 5^2)$ (the prefix of the instances name is "gauss"). Moreover, we provide another set of 10 large instances involving graphs of 700 and 800 vertices, while the edge weights are uniformly distributed in range [-5, 5] (the prefix of the instances name is "unif").

1.4.2 Metaheuristic algorithms evaluation

As far as we know, there is no agreement upon the evaluation and analysis of metaheuristic algorithms. At the moment, a general approach to compare the performance of different algorithms is to run each algorithm multiple times and then give an overall evaluation. Specifically, the algorithms are given the same benchmark instances and tested under the same environment (ideally, the same test machine and time limitation). Important indicators like solution quality, running time or memory consumption for each independent run are recorded. Then we collect these records and compare some statistic features. In this study, we frequently use features like average and standard deviation of solution quality, the average time consumption, the average speed, etc. Note that for some problems and instances, the general purpose MILP solver CPLEX is also used as a benchmark algorithm.

Modern metaheuristics generally integrate many components which involve dedicated heuristics (solution initialization, solution perturbation, crossover, etc). To analyze the contribution of a specific component to the the overall performance, the control experiment is frequently used. We normally set up a variant of the tested algorithm in which only the discussed component is replaced by other implementations or a trivial one. Then the variant and original algorithms are compared experimentally on the benchmark. For the four clique problems studied in the following chapters, such control experiments are frequently used.

[5] http://konect.uni-koblenz.de/

[6] All these instances can be downloaded from https://drive.google.com/open?id=0Bxq63AJFrhOjd2h4YVRxOWNXNnc

1.4.3 Exact algorithms Evaluation

As opposed to stochastic metaheuristics, exact algorithms are normally deterministic. The exact algorithms involved in this work are mainly based on the branch and bound (B&B) framework. To evaluate them, we measure the computational time as well as the size of the B&B tree, (i.e., the number of B&B nodes enumerated during the search). Under certain settings, an exact algorithm is unable to complete in a reasonable time, we then evaluate the gap between the current best solution and the best-known lower/upper bound.

2

A Generalized Operator “PUSH” for the Maximum Vertex Weight Clique Problem

In this chapter, we introduce a generalized move operator called *PUSH*, which generalizes the conventional *ADD* and *SWAP* operators commonly used in the literature and can be integrated in a local search algorithm for MVWCP. The *PUSH* operator also offers opportunities to define new search operators by considering dedicated candidate push sets. To demonstrate the usefulness of the proposed operator, we implement two simple tabu search algorithms which use *PUSH* to explore different candidate push sets. The computational results on a total of 142 benchmark instances from different sources (2nd DIMACS, BHOSLIB, and Winner Determination Problem) indicate that the proposed approach competes favorably with the leading MVWCP algorithms. The generality of the *PUSH* operator could lead to new applications in other contexts bypassing the problem and search procedures studied in this work. The content of this chapter is based on an article published in *European Journal of Operational Research* [Zhou *et al.*, 2017a].

Contents

2.1 Introduction

Given an undirected graph $G = (V, E, W)$ with vertex set V and edge set E, let $W : V \to \mathbb{R}^+$ be a weighting function that assigns to each vertex $v \in V$ a positive value w_v. A clique $C \subseteq V$ of G is a subset of vertices such that its induced subgraph is complete, i.e., every two vertices in C are pairwise adjacent in G ($\forall u, v \in C, \{u, v\} \in E$). For a clique C of G, its weight is given by $W(C) = \sum_{v \in C} w_v$. The Maximum Vertex Weight Clique Problem (MVWCP) is to determine a clique C^* of maximum weight.

MVWCP is an important generalization of the classic Maximum Clique Problem (MCP) [Wu and Hao, 2015a]. Indeed, when the vertices of V are assigned the unit weight of 1, MVWCP is equivalent to MCP which is to find a clique C^* of maximum cardinality. Given that the decision version of MCP is NP-complete [Karp, 1972], the generalized MVWCP problem is at least as difficult as MCP. Consequently solving MVWCP represents an imposing computational challenge in the general case. Note that MVWCP is different from another MCP variant – the Maximum Edge Weight Clique Problem [Alidaee *et al.*, 2007; Dijkhuizen and Faigle, 1993] where a clique C^* of maximum *edge* weight is sought.

Like MCP which has many practical applications, MVWCP can be used to formulate and solve some relevant problems in diverse domains. For example, in computer vision, MVWCP can be used to solve image matching problems [Ballard and Brown, 1982]. In combinatorial auctions, the Winner Determination Problem can be recast as MVWCP and solved by MVWCP algorithms [Wu and Hao, 2015b; Wu and Hao, 2016].

Given the significance of MVWCP, much effort has been devoted to design various algorithms for solving the problem over the past decades. On the one hand, there are a variety of exact algorithms which aim to find optimal solutions. For instance, in 2001, Östergård [Östergård, 1999] presented a branch-and-bound (B&B) algorithm where the vertices are processed according to the order provided by a vertex coloring of the given graph. This MVWCP algorithm is in fact an adaptation of an existing B&B algorithm designed for MCP [Östergård, 2002]. In 2004, Kumlander [Kumlander, 2004] introduced a backtrack tree search algorithm which also relies on a heuristic coloring-based vertex order. In 2006, Warren and Hicks [Warren and Hicks, 2006] exposed three B&B algorithms which use weighted clique covers to generate upper bounds and branching rules. In 2016, Wu and Hao [Wu and Hao, 2016] developed an algorithm which introduces new bounding and branching techniques using specific vertex coloring and sorting. In 2016, Fang et al. [Fang *et al.*, 2016] presented an algorithm which uses Maximum Satisfiability Reasoning as a bounding technique. On the other hand, local search heuristics constitute another popular approach to find high-quality sub-optimal or optimal solutions in a reasonable computing time. In 1999, Mannino and Stefanutti [Mannino and Stefanutti, 1999] proposed a tabu search method based on edge projection and augmenting sequence. In 2000, Bomze et al. [Bomze *et al.*, 2000] formulated MVWCP as a continuous problem which is solved by a parallel algorithm using a distributed computational network model. In 2006, Busygin [Busygin, 2006] exposed a trust region algorithm. The same year, Singh and Gupta [Singh and Gupta, 2006] introduced a hybrid method combining genetic algorithm, a greedy search and the exact algorithm of [Carraghan and Pardalos, 1990]. In 2008, Pullan [Pullan, 2008] adapted the Phase Local Search for the classical MCP to MVWCP. In 2012, Wu et al. [Wu *et al.*, 2012] introduced a tabu search algorithm integrating multiple neighborhoods. In 2013, Benlic and Hao [Benlic and Hao, 2013a] presented the Breakout Local Search algorithm which also explores multiple neighborhoods and applies both directed and random perturbations. Recently in 2016, Wang et al. [Wang *et al.*, 2016] reformulated MVWCP as a Binary Quadratic Program (BQP) which was solved by a probabilistic tabu search algorithm designed for BQP.

As shown in the literature, local search represents the most popular and the dominating approach for solving MVWCP heuristically. Typically, local search heuristics explore the search space by iteratively transforming the incumbent solution into another solution by means of some move (or transformation) operators. Existing heuristic algorithms for MVWCP [Benlic and Hao, 2013a; Pullan, 2008; Wang *et al.*, 2016; Wu *et al.*, 2012; Wu and Hao, 2015b] are usually based on two popular move operators for search intensification: (1) *ADD* which inserts a vertex to the incumbent solution (a feasible clique), and (2) *SWAP* which exchanges a vertex in the clique against a vertex out of the clique. In studies like [Benlic and Hao, 2013a; Wu *et al.*, 2012], another operator called *DROP* was also used, which simply removes a vertex from the current clique. The algorithms using these operators have reported remarkable results on a large range of benchmark problems. Still as we show in this work, the performance of local search algorithms could be further improved by employing more powerful search operators.

This work introduces a generalized move operator called *PUSH*, which inserts one vertex into the clique and removes $k \geq 0$ vertices from the clique to maintain the feasibility of the transformed clique. The proposed *PUSH* operator shares similarities with some restart and perturbation operators used in MCP and MVWCP algorithms like [Benlic and Hao, 2013a; Grosso *et al.*, 2008] and finds its origin in these previous studies. Meanwhile, as we show in this work, the *PUSH* operator not only generalizes the existing *ADD* and *SWAP* operators, but also offers the possibility of defining additional clique transformation operators. Indeed, dedicated local search operators can be obtained by customizing the set of candidate vertices considered by *PUSH*. Such alternative operators can then be employed in a search algorithm as a means of intensification or diversification.

To assess the usefulness of the *PUSH* operator, we experiment two restart tabu search algorithms (ReTS-I and ReTS-II), which explore different candidate push sets for push operations. ReTS-I operates on the largest possible candidate push set while ReTS-II works with three customized candidate push sets. Both algorithms share a probabilistic restart mechanism. The proposed approach is assessed on three sets of well-known benchmarks (2nd DIMACS, BHOSLIB, and Winner Determination Problem) of a total of 142 instances. The computational results indicate that both ReTS-I and ReTS-II compete favorably with the leading MVWCP algorithms of the literature. Moreover, the generality of the *PUSH* operator could allow it to be integrated within any local search algorithm to obtain enhanced performances.

The chapter is organized as follows. Section 2.2 formally introduces the *PUSH* operator. Section 2.3 presents the two push-based tabu search algorithms. Section 4.3.7 reports our experimental results and comparisons with respect to state-of-the-art algorithms. Section 2.5 is dedicated to an experimental analysis of the restart strategy while conclusions and perspectives are given in Section 5.5.

2.2 PUSH: a generalized operator for MVWCP

2.2.1 Preliminary definitions

Let $G = (V, E, W)$ be an input graph as defined in the introduction, $C \subseteq V$ a feasible solution (i.e., a clique) such that any two vertices in C are linked by an edge in E (throughout the chapter, C is used to designate a clique), and $v \in V$ an arbitrary vertex. We introduce the following notations:

- $N(v)$ and $\bar{N}(v)$ denote respectively the set of adjacent and non-adjacent vertices of a vertex v in V, i.e., $N(v) = \{u : \{v, u\} \in E\}$ and $\bar{N}(v) = \{u : \{v, u\} \notin E\}$.
- $N_C(v)$ and $\bar{N}_C(v)$ denote respectively the set of adjacent and non-adjacent vertices of a vertex v in C, i.e., $N_C(v) = C \cap N(v)$ and $\bar{N}_C(v) = C \cap \bar{N}(v)$.
- Ω is the search space including all the cliques of G, i.e., $\Omega = \{C \subseteq V : \forall v, u \in C, v \neq u, \{u, v\} \in E\}$.
- m is a move operator which transforms a clique to another one. We use $C \oplus m$ to designate the clique $C' = C \oplus m$ obtained by applying the move operator m to C. C' is called a neighbor solution (or neighbor clique) of C.
- N_m is the set of neighbor solutions that can be obtained by applying m to an incumbent solution C.

The weight $W(C) = \sum_{v \in C} w_v$ of a solution (clique) $C \in \Omega$ is used to measure its quality (fitness). For two solutions C and C' in Ω, C' is said to be better than C if $W(C') > W(C)$. The weight $W(C^*)$ of the best solution ever found by a search procedure is abbreviated as W^*.

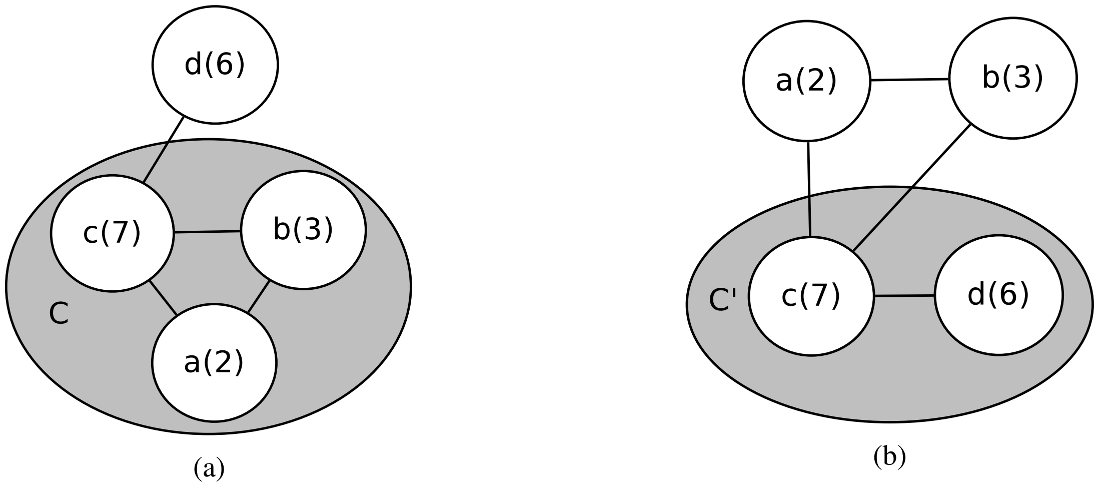

Figure 2.1: An example which shows that a better solution can be reached by the *PUSH* operator, but cannot be attained by the traditional ADD and $SWAP$ operators.

2.2.2 Motivations for the PUSH operator

As shown in the literature, local search is the dominating approach for tackling MVWCP (see for example, [Benlic and Hao, 2013a; Pullan, 2008; Wu *et al.*, 2012; Wu and Hao, 2015b]). Local search typically explores the search space by iteratively transforming an incumbent solution C to a neighbor solution (often of better quality) by means of the following move operators:

– *ADD* extends C with a vertex $v \in V \setminus C$ which is necessarily adjacent to all the vertices in C. Each application of *ADD* always increases the weight of C and leads to a better solution.

– *SWAP* exchanges a vertex $v \in V \setminus C$ with another vertex $v' \in C$, v being necessarily adjacent to all vertices in C except v'. Each application of *SWAP* can increase the weight of C (if $w_v > w_{v'}$), keep its weight unchanged (if $w_v = w_{v'}$) or decrease the quality of C (if $w_v < w_{v'}$).

In some cases like [Benlic and Hao, 2013a; Wu *et al.*, 2012], a third move operator (*DROP*) was also employed which simply removes a vertex from C (thus always leading to a worse neighbor solution).

Generally, local search for MVWCP aims to reach solutions of increasing quality by iteratively moving from the incumbent solution to a neighbor solution. This is typically achieved by applying *ADD* whenever it is possible to increase the weight of the clique, applying *SWAP* when no vertex can be added to the clique and occasionally calling for *DROP* to escape local optima.

However, as both *ADD* and *SWAP* have a prerequisite on the operating vertex in $V \setminus C$, these operators may miss improving solutions in some cases. To illustrate this point, we consider the example of Fig. 2.1 where vertex weights are indicated in brackets next to the vertex labels. As shown in Fig. 2.1(a), $C = \{a, b, c\}$ is a clique with a total weight of 12. Since vertex d is neither adjacent to a nor b, d cannot join clique C by means of the *ADD* and *SWAP* operators. Meanwhile, one observes that if we insert vertex d into the clique and remove both vertices a and b, we obtain a new clique C' (Fig. 2.1(b)) of weight of 13, which is better than C.

Inspired by this observation, the *PUSH* operator proposed in this work basically transforms C by pushing a vertex v taken from a *dedicated* subset of $V \setminus C$ into C and removing, if needed, one or more vertices from C to re-establish solution feasibility. Indeed, when the added vertex v is not adjacent to all the vertices in C, the vertices of C which are not adjacent to v (i.e., the vertices in the set $\bar{N}_C(v)$ as defined in Section 2.2.1) need to be removed from C to maintain the feasibility of the new solution. In the above example, C can be transformed to a better solution by pushing vertex d into the clique and then expelling both a and b.

2.2.3 Definition of the PUSH Operator

Let C be a clique, and v an arbitrary vertex which does not belong to C ($v \in V \setminus C$). $PUSH(v, C)$ (or $PUSH(v)$ for short if the current clique does not need to be explicitly emphasized) generates a new clique by first inserting v into C and then removing any vertex $u \in C$ such that $\{u, v\} \notin E$ (i.e., $u \in \bar{N}_C(v)$, see Section 2.2.1).

Formally, the neighbor clique C' after applying $PUSH(v)$ ($v \in V \setminus C$) to C is given by:

$$C' = C \oplus PUSH(v) = C \setminus \bar{N}_C(v) \cup \{v\}$$

Consequently, the set of neighbor cliques induced by $PUSH(v)$ in the general case, denoted by N_{PUSH} is given by:

$$N_{PUSH} = \bigcup_{v \in V \setminus C} \{C \oplus PUSH(v)\} \tag{2.1}$$

For each neighbor solution $C' = C \oplus PUSH(v)$ generated by a $PUSH(v, C)$ move, we define the move gain (denoted by δ_v) as the variation in the objective function value between C' and C:

$$\delta_v = W(C') - W(C) = w_v - \sum_{u \in \bar{N}_C(v)} w_u \tag{2.2}$$

Thus, a positive (negative) move gain indicates a better (worse) neighbor solution C' compared to C while the zero move gain corresponds to a neighbor solution of equal quality.

Typically, a local search algorithm makes its decision of moving from the incumbent solution to a neighbor solution based on the move gain information at each iteration. In order to be able to efficiently compute the move gains of neighbor solutions, we present in Section 2.3.5 fast streamlining evaluation techniques with the help of dedicated data structures.

One notices that *PUSH* shares similarities with some customized restart or perturbation operators used in [Benlic and Hao, 2013a; Grosso *et al.*, 2008] and finds its origin in these previous studies. In the iterated local search algorithm designed for MCP by Grosso et al. [Grosso *et al.*, 2008], the clique delivered at the end of each local optimization stage is perturbed by insertion of a random vertex and serves then as a new starting point for the next stage of local optimization. In the BLS algorithm for MCP and MVWCP presented by Benlic and Hao [Benlic and Hao, 2013a], each random perturbation adds a vertex such that the resulting clique must satisfy a quality threshold. In these two previous studies, clique feasibility is established by a repair process which removes some vertices after each vertex insertion.

2.2.4 Special cases of PUSH

From the general definition of *PUSH* given in the last section, we can customize the move operator by identifying a dedicated vertex subset of $V \setminus C$ called *candidate push set* (CPS) that provides the candidate vertices for *PUSH*. We first discuss two special cases by considering non-adjacency information convoyed by $\bar{N}_C(v)$ (see the notations introduced in Section 2.2.1).

- If CPS is given by $A = \{v : |\bar{N}_C(v)| = 0,\ v \in V \setminus C\}$, then *PUSH* is equivalent to *ADD*.
- If CPS is given by $B = \{v : |\bar{N}_C(v)| = 1,\ v \in V \setminus C\}$, then *PUSH* is equivalent to *SWAP*.

We can also use other information like move gain to constrain the candidate push set, as illustrated by the following examples.

1. If CPS is given by $M_1 = \{v : \delta_v > 0, v \in V \setminus C\}$, then *PUSH* always leads to a neighbor solution better than C.
2. If CPS is given by $M_2 = \{v : \delta_v \leq 0, |\bar{N}_C(v)| = 1, v \in V \setminus C\}$, then *PUSH* exchanges one vertex in $V \setminus C$ with one vertex in C, leading to a solution of equal or worse quality relative to C.

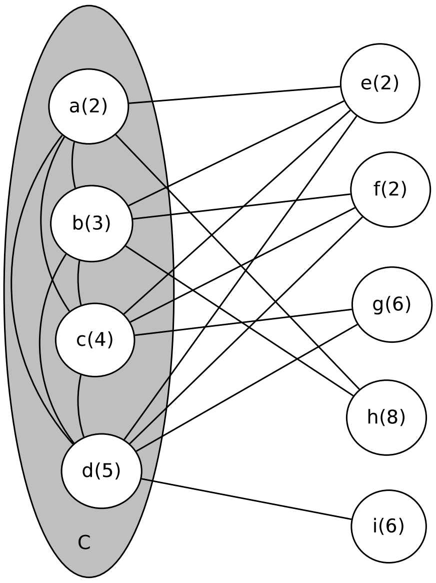

Figure 2.2: A simple graph labeled with vertex weights in brackets. The current clique is $C = \{a, b, c, d\}$, $W(C) = 2{+}3{+}4{+}5 = 14$, $\bar{N}_C(e) = \emptyset$, $\bar{N}_C(f) = \{a\}$, $\bar{N}_C(g) = \{a, b\}$, $\bar{N}_C(h) = \{c, d\}$, $\bar{N}_C(i) = \{a, b, c\}$, thus, $\delta_e = 2$, $\delta_f = 0$, $\delta_g = 1$, $\delta_h = -1$, $\delta_i = -3$. According to the definitions, $A = \{e\}$, $B = \{f\}$, $M_1 = \{e, g\}$, $M_2 = \{f\}$, $M_3 = \{h, i\}$.

3. If CPS is given by $M_3 = \{v : \delta_v \leq 0, |\bar{N}_C(v)| > 1, v \in V \setminus C\}$, then *PUSH* inserts one vertex into C and removes at least two vertices from C, leading to a solution of equal or worse quality relative to C.
4. If CPS is given by $V \setminus C$, then the candidate push set is not constrained. One notices that $V \setminus C = M_1 \cup M_2 \cup M_3$.

An example of these special cases is provided in Fig. 2.2.

In addition to the *ADD* and *SWAP* operators, several restart rules of local search algorithms for MCP can also be recast with the *PUSH* operator. In particular, the restart Rule 1 in [Grosso *et al.*, 2008] (previously used in [Pullan and Hoos, 2006]) states that $C := [C \cap N(v)] \cup \{v\}$, v picked at random in $V \setminus C$ (i.e., add a random vertex v in the clique while keeping in the clique the adjacent vertices of v). This rule is equivalent to push a vertex from candidate push set $V \setminus C$ into the current solution. As to the restart Rule 2 in [Grosso *et al.*, 2008], let us define the candidate push set $S_q = \{v : \delta_v \leq 1 - q, v \in V \setminus C\}$ ($q > 0$ is a fixed parameter). Then the Rule 2 is to push a random vertex from S_q into C if S_q is not empty; otherwise, push a random vertex from $V \setminus C$. Moreover, in the BLS algorithm for MCP and MVWCP [Benlic and Hao, 2013a], the so-called random perturbation modifies the incumbent clique by adding vertices such that the quality of the resulting clique is not deteriorated more than a quality threshold. This random perturbation strategy can simply be considered as applying the *PUSH* operator to vertices from the candidate push set $M_4 = \{v : \delta_v > (\alpha - 1) * W(C), v \in V \setminus C\}$ (where $0 < \alpha < 1$ is a predefined parameter).

Finally, by considering other candidate push sets subject to specific conditions, it is possible to obtain multiple customized search operators that can be employed by any local optimization procedure to effectively explore the search space. In the next section, we present two local search algorithms based on the above M_1, M_2, M_3 and $V \setminus C$ candidate push sets.

2.3 PUSH-based tabu search

In this section, we introduce two simple Restart Tabu Search [Glover and Laguna, 2013] algorithms (denoted by ReTS-I and ReTS-II). Both algorithms rely on the *PUSH* operator, but explore different candidate push sets. In ReTS-I, the candidate push set considered includes all the vertices out of the clique (i.e., CPS $= V \setminus C$) while in ReTS-II, the algorithm jointly considers the candidate push sets M_1, M_2 and M_3 introduced in Section 2.2.3.

Both ReTS-I and ReTS-II share the same restart local search framework as shown in Algorithm 4.1, but implement different local optimization procedures with different CPS (line 5, see Sections 2.3.3 and 2.3.4). The general framework starts from an initial solution C (or initial clique) generated by means of $Random\_Solution$ (Section 2.3.1). The solution is then improved by one of the dedicated tabu search procedures described respectively in Sections 2.3.3 and 2.3.4. When the search stagnates in a deep local optimum, the search restarts from a new solution, which is constructed either by $Reconstruct\_Solution$ (Section 2.3.2) with probability $\rho \in [0.0, 1.0]$ (a parameter), or by $Random\_Solution$ (Section 2.3.1) with probability $1 - \rho$. It is noted that $Reconstruct\_Solution$ reconstructs a new solution from C, while $Random\_Solution$ randomly generates a new solution from scratch. The whole search process repeats the above procedure until a prefixed stopping condition is met. The details of the tabu search optimization procedures and restart procedures are described in the following sections.

Algorithm 2.1: Framework of the Restart Tabu Search algorithms for MVWCP

```
Input: $G = (V, E, w)$ - MVWCP instance, $\rho$ - restart probability parameter, $L$ - maximum number
       of consecutive non-improving iterations.
Output: $C^*$ - maximum vertex weight clique.
begin
    $C^* \leftarrow \emptyset$ ;                                /* $C^*$ maintains the best solution found so far */
    $C \leftarrow Random\_Solution(G)$;                         /* Section 2.3.1.  $C$ is the current solution */
    while stopping condition is not met do
        $(C, C^*) \leftarrow Tabu\_Search(G, C, C^*, L)$;       /* Sections 2.3.3 and 2.3.4 */
        if random number $\in [0, 1] < \rho$ then
            $C \leftarrow Reconstruct\_Solution(G, C)$ ;        /* Section 2.3.2 */
        else
            $C \leftarrow Random\_Solution(G)$;                 /* Section 2.3.1 */
end
return $C^*$
```

2.3.1 Random initial solution

The $Random\_Solution(G)$ procedure (Alg. 4.1, lines 3 and 9) starts from an initial clique C composed by an unique random vertex. Then iteratively, a vertex v in candidate push set $A = \{v : |\bar{N}_C(v)| = 0, v \in V \setminus C\}$ (Sections 2.2.3 and 2.2.4) is randomly selected and added into C. A is then updated by $A \leftarrow A \setminus (\{v\} \cup \bar{N}_A(v))$. The procedure continues until the candidate push set A becomes empty. A maximal clique (i.e., $\forall v \in V \setminus C, |\bar{N}_C(v)| > 0$) is then reached and returned as the initial solution of the search procedure. This initialization procedure ignores the solution quality (the clique weight), but ensures a good randomness of the initial solutions generated. Such a feature represents a simple and useful diversification technique which helps the search algorithm to start the search in a different region of each repeated run. The initialization procedure can be efficiently implemented with a time complexity of $O(|V||E|)$. This process is similar to the initial constructive phase preceding the first *SWAP* move in the MCP algorithm of [Grosso *et al.*, 2008] and also applied in the MVWCP algorithms of [Benlic and Hao, 2013a; Wu *et al.*, 2012].

2.3.2 Solution reconstruction

The reconstruction procedure (Alg. 4.1, line 7) generates a new solution by iteratively replacing vertices of a given solution. At the beginning, considering a clique C, all the vertices of $V \setminus C$ are marked available to join C by means of the *PUSH* operator. Then, at each iteration, the available vertex belonging to candidate push set M_1 (see Section 2.2.4) with the maximum δ value (ties are broken randomly) is selected and pushed into C. Vertices which are removed from C during the *PUSH* operation are then marked unavailable. As a consequence, they cannot rejoin the solution during the remaining iterations. The reconstruction procedure stops after $|C|$ iterations or when no available vertex may be found from M_1. The current clique C is then returned as the reconstructed solution. Such a reconstruction procedure perturbs the given solution C but, in most cases, does not decrease the quality heavily. The time complexity of each iteration is bounded by $O(|V| + (\max_{v \in V}\{|\bar{N}(v)|\})^2)$ as it scans the M_1 set and calls the *PUSH* operator (The time complexity of the push operator is discussed in Section 2.3.5). This reconstruction procedure can also be viewed as an objective-guided strong perturbation procedure since the vertices in the original solution are totally replaced and vertex insertions are subject to the stipulation of the maximum δ value.

2.3.3 ReTS-I: Tabu search with the largest candidate push set

The first tabu search procedure denoted by ReTS-I uses a greedy rule which gives preference, at each step of the search, to neighbor solutions having the best objective value. ReTS-I implements this heuristic with the largest possible candidate push set $V \setminus C$. To prevent the search from falling into cycles, a tabu mechanism [Glover and Laguna, 2013] is incorporated.

The general process of ReTS-I is shown in Algorithm 2.2, where each element $tabu_v$ of vector $tabu$ (called the tabu list) records the earliest iteration number that vertex v is allowed to move inside C. At each iteration, one vertex is allowed to join the current solution only when it is not forbidden by the tabu list. Nevertheless, a move leading to a solution better than the best solution found so far is always accepted (this is the so-called aspiration criterion, line 7, Alg. 2.2). If v is the vertex to be pushed into C, then all the vertices moving out of C (i.e., those of $\bar{N}_C(v)$) are forbidden to rejoin the solution for the next $tt(v)$ (tabu tenure) iterations (lines 9-10, Alg. 2.2). The TS procedure ends when the best solution cannot be improved for L (a parameter) consecutive iterations. Note that similar strategies which temporarily forbid the removed vertices to rejoin the solution have been used in [Benlic and Hao, 2013a; Grosso *et al.*, 2008; Pullan, 2008; Wu *et al.*, 2012]. Also note that the added vertex is free to leave the clique. This rule is based on the fact that due to the objective of maximizing the clique weight, an added vertex has little chance to be removed anyway.

The tabu tenure $tt(v)$ for a vertex v is empirically fixed as follows:

$$tt(v) = 7 + random(0, \eta(v)) \tag{2.3}$$

where $\eta(v) = |\{u \in V \setminus C : \bar{N}_C(u) = \bar{N}_C(v)\}|$ is the number of vertices which have as many non-adjacent vertices in C as v, and $random(0, n)$ returns a random integer in range $[0, n)$.

Since ReTS-I needs to scan $V \setminus C$ at each iteration (line 6, Alg. 2.2), the time complexity of each iteration of ReST-I is bounded by $O(|V|)$. The ReTS-I algorithm is quite simple, but performs well as shown by the experimental outcomes presented in Section 4.3.7.

2.3.4 ReTS-II: Tabu search with three decomposed candidate push sets

Contrary to ReTS-I which explores the whole and unique candidate push set $V \setminus C$, ReTS-II, as shown in Algorithm 2.3, considers more features of the candidate vertices for *PUSH*. For this algorithm, we decompose $V \setminus C$ into three candidate push sets M_1, M_2 and M_3 as defined in Section 2.2.4. At each iteration, the three CPS are evaluated in a fixed order: $M_1 \rightarrow M_2 \rightarrow M_3$ (lines 7-15) and a vertex with the largest δ value is chosen by *PUSH* to perform the move. Note that M_1 contains preferable vertices as they necessarily increase the weight of the incumbent clique. If no candidate vertex is available in M_1, selecting a vertex from M_2 or M_3 will degrade the solution (or keep the solution unchanged). Pushing these vertices may be useful to help the search to leave the current local optimum. M_2 is evaluated before M_3 since pushing a vertex from M_2 will generally lead to less vertices to be removed from C than pushing a vertex from M_3. The

Algorithm 2.2: ReTS-I: Tabu search with the largest candidate push set

```
Input: $C$ - current solution, $C^*$ - best solution ever found, $L$ - maximum number of consecutive non-improving iterations.
Output: $C$ - renewed current solution, $C^*$ - maximum vertex weight clique.
begin
    $Iter \leftarrow 0$ ;                                    /* Counter of iterations */
    for each $v \in V$ do
        $tabu_v \leftarrow 0$ ;          /* $tabu_v$ is the earliest iteration vertex $v$ is allowed to join $C$ */
    $l \leftarrow 0$ ;                   /* Counter of consecutive iterations where $C^*$ is not improved */
    while $l < L$ do
        $M \leftarrow \{v \in V \setminus C, tabu_v \leq Iter$ or $W(C) + \delta_v > W(C^*)\}$ ; /* $M$ is the set of eligible vertices for PUSH */
        $v \leftarrow \mathrm{argmax}_{v \in M}\, \delta_v$;
        for $u \in C \setminus N_C(v)$ do
            $tabu_u \leftarrow Iter + tt(v)$
        $C \leftarrow C \oplus PUSH(v)$;
        if $W(C) > W(C^*)$ then
            $C^* \leftarrow C$;
            $l \leftarrow 0$;
        else
            $l \leftarrow l + 1$;
        $Iter \leftarrow Iter + 1$;
end
return $C$, $C^*$
```

motivation of using these three sets with a preference order is thus to keep the improvement possibilities as much as possible and proceed to more important perturbations only when no other alternative is possible.

Moreover, when M_3 is used, only a random sample (of a predetermined size r) of vertices in M_3 are evaluated for each $PUSH$ operation if no appropriate vertex is found in M_1 and M_2. Also, let us precise that *PUSH* selects the vertex with the best δ value in the sample set. This sampling strategy and its variants were previously used in several studies (ID-Walk [Neveu *et al.*, 2004], Candidate List [Glover and Laguna, 2013], Best from Multiple Choices [Cai, 2015]). This strategy is obviously more cost effective than evaluating an entire candidate set.

ReTS-II uses the same tabu mechanism as ReTS-I. Note that the aspiration criterion does not need to be considered for pushing a vertex from M_2 and M_3 as better solutions cannot be reached in these cases. Vertices dropped from C by applying *PUSH* to M_2 or M_3 are forbidden to rejoin C for consecutive $tt(v)$ iterations (lines 18-19, Alg. 2.3). The tabu tenure $tt(v)$ is tuned in the same way as in ReTS-I (Section 2.3.3).

Finally, it is interesting to contrast ReTS-I and ReTS-II. In fact, like ReTS-I, ReTS-II also gives priority to candidate vertices leading to a solution of better quality. Meanwhile, when no such kind of vertex exists, ReTS-II may choose a different vertex for the *PUSH* operation. For example, suppose that the same candidate push set $M = \{a, b\}$ is applied in Algorithms 2.2 and 2.3 with $\delta_a = -1$, $\delta_b = -3$, $|\bar{N}_C(a)| = 2$, $|\bar{N}_C(b)| = 1$. Then ReTS-I chooses vertex a while ReTS-II selects vertex b for the *PUSH* operation. Therefore, by using different candidate push sets, ReTS-I and ReTS-II visit different search trajectories to explore the search space. The computational experiments shown in Section 4.3.7 will allow us to observe the relative performances of both algorithms.

2.3.5 Fast evaluation of move gains

As presented in Section 2.2.3, each neighbor solution relative to a current clique leads to a move gain δ, which can be positive, null or negative. Since move gain evaluations are frequent in the TS procedures, we elaborate a fast streamlining technique which enables a direct access to all possible δ values (i.e., corresponding to the insertion in the current clique of each candidate vertex), as well as a fast update of the impacted move gains at each iteration. In this section, we present this incremental evaluation mechanism.

Let us consider a vector $\Delta = (\delta_v)_{v \in V}$ such that δ_v represents the move gain $W(C \oplus Push(v)) - W(C)$. According to the definition in Section 2.2.3, a *PUSH* operation is composed of two basic operations: adding a vertex to the current clique C (*ADD*), and possibly removing one or several vertices from C (*DROP*).

Algorithm 2.3: ReTS-II: Tabu search with three candidate push sets

```
Input: $C$ - current solution, $C^*$ - best solution ever found, $L$ - maximum number of consecutive non-improving iterations, $r$ - maximum sample size of $M_3$.
Output: $C$ - renewed current solution, $C^*$ - maximum vertex weight clique found.
begin
    $Iter \leftarrow 0$;
    for each $v \in V$ do
        $tabu_v \leftarrow 0$;
    $l \leftarrow 0$;
    while $l < L$ do
        $M \leftarrow \{v \in V \setminus C, W(C) + \delta_v > W(C^*)\}$;
        if $M \neq \emptyset$ then
            $l \leftarrow 0$ ;                                                   /* New best solution */
        else
            $M \leftarrow \{v \in V \setminus C,\ \delta_v > 0$ and $tabu_v \leq Iter\}$ ;              /* Restricted $M_1$ */
            if $M = \emptyset$ then
                $M \leftarrow \{v \in V \setminus C,\ \delta_v \leq 0$ and $|\bar{N}_C(v)| = 1$ and $tabu_v \leq Iter\}$ ;   /* Restricted $M_2$ */
            if $M = \emptyset$ then
                $M \leftarrow$ Randomly sample $r$ vertices from $\{v \in V \setminus C,\ tabu_v \leq Iter\}$ ;  /* Restricted $M_3$ */
            $l \leftarrow l + 1$;
        Randomly select $v \in \text{argmax}_{v \in M}\, \delta_v$;
        for each $u \in C \setminus N_C(v)$ do
            $tabu_u \leftarrow tt(u)$
        $C \leftarrow C \oplus PUSH(v)$;
        if $l = 0$ then
            $C^* \leftarrow C$;
        $Iter \leftarrow Iter + 1$;
end
return $C$, $C^*$
```

Pushing a vertex v into a clique C can be viewed as adding v in C before removing from C every vertex which is not adjacent to v. Nevertheless this decomposition implies to consider infeasible solutions between vertex insertion and removals. We propose then to update incrementally Δ after each basic operation (*ADD*, *DROP*) by first removing from C the vertices which are not adjacent to the pushed vertex v, and finally adding v to C.

If a vertex v' is removed (dropped) from the current solution C, then the move gain δ_u of any vertex $u \in V$ is updated as follows:

$$\forall u \in V,\ \delta_u \leftarrow \begin{cases} \delta_u + w_{v'} & \text{, if } u \in \bar{N}(v') \\ w_u & \text{, if } u = v' \\ \delta_u & \text{, otherwise} \end{cases} \tag{2.4}$$

To speed up the update process, we use the complementary graph $\bar{G}$ of the input graph G, so that $\bar{N}(v)$ sets can be explicitly defined. Since only the move gains associated to vertices of $\bar{N}(v') \cup \{v'\}$ need to be updated, the time complexity of updating Δ after a *DROP* operation is bounded by $O(\max_{v \in V}\{|\bar{N}(v)|\})$.

Similarly, when a vertex v is added into the current solution C, then Δ is updated as follows:

$$\delta_u \leftarrow \begin{cases} \delta_u - w_v & \text{, if } u \in \bar{N}(v) \\ 0 & \text{, if } u = v \\ \delta_u & \text{, otherwise} \end{cases} \tag{2.5}$$

This operation is obviously also bounded in time by $O(\max_{v \in V}\{|\bar{N}(v)|\})$. Thus updating the move gains after a *PUSH* operation can be performed in $O((\max_{v \in V}\{|\bar{N}(v)|\})^2)$, since operation $C \oplus Push(v)$ involves one *ADD* operation and $|\bar{N}_C(v)|$ *DROP* operations, with $|\bar{N}_C(v)|$ being bounded by $\max_{v \in V}\{|\bar{N}(v)|\}$.

2.4 Computational experiments

This section is dedicated to an experimental assessment of the two tabu search algorithms using the generalized *PUSH* operator. The assessment was based on three sets of 142 well-known benchmark instances and comparisons with state-of-the-art MVWCP algorithms.

2.4.1 Benchmarks

The three benchmark sets include the following instances: 80 2nd-DIMACS instances, 40 BHOSLIB instances, and 22 instances from the Winner Determination Problem in combinatorial auctions. These instances are introduced in Section 1.4.1 of Chapter 1. Since 2nd DIMACS and BHOSLIB benchmark instances are unweighted, we assign to each vertex i (i is an index number) the weight $i \mod 200 + 1$, following the rule in [Pullan, 2008]. The WDP instances are weighted, however, contrary to the 2nd DIMACS and BHOSLIB instances which are defined using integer weights, WDP weights are fractional.

2.4.2 Experimental protocol

As shown in Algorithms 4.1 to 2.3, ReTS-I and ReTS-II share two common parameters: the probability parameter ρ which controls the two types of restart, and the maximum number L of consecutive non-improving iterations before a restart. Besides, ReTS-II has one additional parameter which is the sample size r. For our experiments, we used the following default values: $L = 4,000$, $r = 50$, and $\rho = 0.7$. We provide an analysis of ρ in Section 2.5. In general, we observed that varying the parameter values around the default values did not alter much the computational outcomes for most of the tested instances even if some results can be further improved by fine-tuning the parameters.

ReTS-I and ReTS-II were coded in C++ [1] and compiled with g++ 4.4.7 with optimization flag -o3. Our experiments were performed on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Linux 2.6.32. When solving the 2nd DIMACS machine benchmarks without compilation optimization flag, the run time on our machine is 0.40, 2.50 and 9.55 seconds respectively for instances r300.5, r400.5 and r500.5.

Following the literature [Benlic and Hao, 2013a; Wang *et al.*, 2016; Wu *et al.*, 2012], both algorithms were run 100 times to solve each benchmark instance. For the 2nd DIMACS and BHOSLIB instances, a maximum of 10^8 iterations were allowed per run while for the WDP instances, the stopping condition was set to be a cutoff time limit of 10 minutes per run. As discussed in Section 2.4.4, these settings correspond to the computational effort used by the state-of-the-art MVWCP algorithms in the literature.

2.4.3 Computational results

Tables 2.1 to 2.3 report the computational results obtained by ReTS-I and ReTS-II on the 2nd DIMACS, BHOSLIB and WDP instances respectively. In these tables, column BKV reports the best-known objective values (BKV) ever found by the previous algorithms [Benlic and Hao, 2013a; Fang *et al.*, 2016; Pullan, 2008; Wang *et al.*, 2016; Wu *et al.*, 2012] (proven optima are indicated with the star symbol '*' with the BKV values). For each algorithm, column $best(hit)$ indicates the best objective value W^* found by ReTS-I and ReTS-II among 100 trials as well as the number of trails hitting the best value (success rate); column $ave(std)$ denotes the average value and the standard deviation of the 100 W^* values; column $time$ gives the average seconds of the trails hitting the W^* value. The value of 0.00 in columns $time$ indicates that the corresponding average time in seconds is inferior to 0.005.

Table 2.1 discloses that ReTS-I and ReTS-II reach all the best-known results of 2nd DIMACS instances except MANN_a45 and MANN_a81 (indicated in italic), which are believed to be quite challenging for heuristic algorithms [Fang *et al.*, 2016]. Moreover, for 73 out of 80 instances ($> 91\%$), both algorithms attain the best-known results in every single trial. For the remaining 7 instances except MANN_a45 and

[1] Our source code will be available at:
www.info.univ-angers.fr/pub/hao/ReTS.html.

MANN_a81, each algorithm still hits the best-known results in more than 30 trials. In terms of computational time, most of these instances are solved in less than 1 second. For the 3 hard MANN instances (MANN_a27, MANN_a45, MANN_a81), results are attained in 1 to 17 minutes.

From Table 2.2 on the BHOSLIB instances, one finds that ReTS-I and ReTS-II improve the best-known result of the literature on frb53-24-3 (from 5,640 to 5,655). Although both algorithms attain the best-known solutions on 38 out of 40 instances, each algorithm fails to do so in 2 cases (frb50-23-4 and frb56-25-5 for ReTS-I, frb56-23-3 and frb 56-23-4 for ReTS-II, indicated in italic). Interestingly, both algorithms achieve a success rate of at least 93% on the first 20 instances while the success rate drops to less than 50% on most of the last 20 instances. Concerning the computational time, both algorithms require more time to find the best-known solutions when the sizes of the graphs increase, but each average time is inferior to 12 minutes. In general, BHOSLIB instances are more difficult than 2nd DIMACS ones for ReTS-I and ReTS-II, but both algorithms still perform quite well by attaining together all the best-known results and finding even an improved best-known result (new best lower bound).

Table 2.3 (WDP instances) shows that ReTS-I and ReTS-II attain the best-known solutions on all WDP instances considered except the 4 *Decay* instances and Paths2000_100 (in italic); for Paths2000_100, the algorithms have a success rate of 88% and 76% respectively, while this rate drops to 10% or less for the 4 Decay instances, confirming that these Decay instances are particularly hard [Sandholm, 2002]. Actually, as shown in Section 2.4.4, these instances also represent a challenge for one of the best performing reference heuristics MN/TS [Wu *et al.*, 2012]. Finally, one observes that ReTS-II attains the best-known results every run on the *in* instances in less than 4 seconds, while ReTS-I fails to consistently hit the best results and requires longer computing times for these instances. Nevertheless we cannot claim that ReTS-II always dominates ReTS-I since the latter performs better on Uniform2000_400_10 in terms of successful trials and computing time.

2.4.4 Comparisons with state-of-the-art algorithms

As indicated in the introduction, a large number of heuristic algorithms for the Maximum Vertex Weight Clique Problem have been reported in the literature, including particularly AugSearch [Mannino and Stefanutti, 1999], HSSGA [Singh and Gupta, 2006], PLS [Pullan, 2008], MN/TS [Wu *et al.*, 2012], BLS [Benlic and Hao, 2013a], and BQP-PTS[Wang *et al.*, 2016]. To further assess the performance of the proposed approach, we compared ReTS-I and ReTS-II with three state-of-the-art algorithms (MN/TS, BLS and BQP-TS). Besides, these reference algorithms have been run on computing platforms which are the same as or very similar to our computer (2.8GHz and 2GB RAM running Linux 2.6.32). For the WDP instances, we used CPLEX as an additional reference as it achieves more competitive results than heuristic algorithms on some specific instances [Wu and Hao, 2015b]. Since CPLEX did not perform well on 2nd DIMACS and BHOSLIB instances [Fang *et al.*, 2016], it was not used for our comparisons for these two benchmarks.

- **MN/TS** is a tabu search algorithm with multiple move operators, designed for solving both MCP and MVWCP [Wu *et al.*, 2012]. The results reported for MN/TS on the 2nd DIMACS and BHOSLIB instances have been obtained using a maximum of 10^8 iterations per run (on a computer cadenced at 2.83GHz and 8GB RAM). Besides, the results of MN/TS on the WDP instances within 300 seconds per run were reported in [Wu and Hao, 2015b]. Each instance was solved for 100 independent trials in all these experiments. Thus, both the stopping condition and the computing platform are almost the same as those used in our experiments.
- **BLS** (Breakout Local Search) incorporates an adaptive perturbation strategy for the resolution of MCP and MVWCP [Benlic and Hao, 2013a]. BLS reported computational results on the sets of 2nd DIMACS and BHOSLIB benchmarks, by running the algorithm 100 times on each instance on the same computing platform as our algorithms (2.83GHZ Xeon E5440 CPU and 2GB RAM). The stopping condition for each of the 100 runs was set to 1.6×10^8 iterations, which was superior to the computational limit used by MN/TS and our algorithms.
- **BQP-PTS** is a probabilistic tabu search algorithm designed for solving unconstrained Binary Quadratic

Table 2.1: Computational results of ReTS-I and ReTS-II on 80 2nd DIMACS instances.

instance	BKV	ReTS-I			ReTS-II		
		$best(hit)$	$ave(std)$	$time$	$best(hit)$	$ave(std)$	$time$
C1000.9	9254	9254(100)	9254.00(0.00)	2.50	9254(100)	9254.00(0.00)	1.73
C125.9	2529	2529(100)	2529.00(0.00)	0.00	2529(100)	2529.00(0.00)	0.00
C2000.5	2466	2466(100)	2466.00(0.00)	2.34	2466(100)	2466.00(0.00)	7.39
C2000.9	10999	10999(92)	10996.44(8.72)	417.56	10999(82)	10993.08(12.76)	474.23
C250.9	5092*	5092(100)	5092.00(0.00)	0.01	5092(100)	5092.00(0.00)	0.01
C4000.5	2792	2792(100)	2792.00(0.00)	116.05	2792(100)	2792.00(0.00)	298.05
C500.9	6955	6955(100)	6955.00(0.00)	0.06	6955(100)	6955.00(0.00)	0.08
DSJC1000_5	2186*	2186(100)	2186.00(0.00)	0.38	2186(100)	2186.00(0.00)	0.37
DSJC500_5	1725*	1725(100)	1725.00(0.00)	0.13	1725(100)	1725.00(0.00)	0.10
MANN_a27	12283*	12283(78)	12282.78(0.41)	82.77	12283(99)	12282.99(0.10)	60.03
MANN_a45	34265*	*34259*(1)	34253.60(1.11)	157.98	*34254*(58)	34253.43(0.74)	357.19
MANN_a81	111386	*111370*(1)	111351.19(6.63)	990.02	*111277*(1)	111233.47(26.42)	477.75
MANN_a9	372	372(100)	372.00(0.00)	0.00	372(100)	372.00(0.00)	0.00
brock200_1	2821	2821(100)	2821.00(0.00)	0.00	2821(100)	2821.00(0.00)	0.00
brock200_2	1428	1428(100)	1428.00(0.00)	0.00	1428(100)	1428.00(0.00)	0.00
brock200_3	2062	2062(100)	2062.00(0.00)	0.00	2062(100)	2062.00(0.00)	0.00
brock200_4	2107	2107(100)	2107.00(0.00)	0.00	2107(100)	2107.00(0.00)	0.00
brock400_1	3422*	3422(100)	3422.00(0.00)	0.04	3422(100)	3422.00(0.00)	0.05
brock400_2	3350*	3350(100)	3350.00(0.00)	0.04	3350(100)	3350.00(0.00)	0.07
brock400_3	3471*	3471(100)	3471.00(0.00)	0.07	3471(100)	3471.00(0.00)	0.06
brock400_4	3626*	3626(100)	3626.00(0.00)	2.04	3626(100)	3626.00(0.00)	1.43
brock800_1	3121*	3121(100)	3121.00(0.00)	0.14	3121(100)	3121.00(0.00)	0.20
brock800_2	3043*	3043(100)	3043.00(0.00)	0.39	3043(100)	3043.00(0.00)	0.61
brock800_3	3076*	3076(100)	3076.00(0.00)	0.39	3076(100)	3076.00(0.00)	0.51
brock800_4	2971*	2971(31)	2970.31(0.46)	835.03	2971(93)	2970.93(0.26)	506.41
c-fat200-1	1284	1284(100)	1284.00(0.00)	0.00	1284(100)	1284.00(0.00)	0.00
c-fat200-2	2411	2411(100)	2411.00(0.00)	0.00	2411(100)	2411.00(0.00)	0.00
c-fat200-5	5887	5887(100)	5887.00(0.00)	0.00	5887(100)	5887.00(0.00)	0.00
c-fat500-1	1354	1354(100)	1354.00(0.00)	0.01	1354(100)	1354.00(0.00)	0.01
c-fat500-10	11586	11586(100)	11586.00(0.00)	0.11	11586(100)	11586.00(0.00)	0.03
c-fat500-2	2628	2628(100)	2628.00(0.00)	0.02	2628(100)	2628.00(0.00)	0.01
c-fat500-5	5841	5841(100)	5841.00(0.00)	0.09	5841(100)	5841.00(0.00)	0.03
gen200_p0.9_44	5043*	5043(100)	5043.00(0.00)	0.00	5043(100)	5043.00(0.00)	0.00
gen200_p0.9_55	5416*	5416(100)	5416.00(0.00)	0.12	5416(100)	5416.00(0.00)	0.00
gen400_p0.9_55	6718	6718(100)	6718.00(0.00)	0.18	6718(100)	6718.00(0.00)	0.12
gen400_p0.9_65	6940	6940(100)	6940.00(0.00)	0.05	6940(100)	6940.00(0.00)	0.04
gen400_p0.9_75	8006*	8006(100)	8006.00(0.00)	0.03	8006(100)	8006.00(0.00)	0.02
hamming10-2	50512*	50512(100)	50512.00(0.00)	0.20	50512(100)	50512.00(0.00)	0.20
hamming10-4	5129	5129(100)	5129.00(0.00)	26.25	5129(100)	5129.00(0.00)	15.74
hamming6-2	1072	1072(100)	1072.00(0.00)	0.00	1072(100)	1072.00(0.00)	0.00
hamming6-4	134	134(100)	134.00(0.00)	0.00	134(100)	134.00(0.00)	0.00
hamming8-2	10976	10976(100)	10976.00(0.00)	0.01	10976(100)	10976.00(0.00)	0.01
hamming8-4	1472	1472(100)	1472.00(0.00)	0.00	1472(100)	1472.00(0.00)	0.00
johnson16-2-4	548	548(100)	548.00(0.00)	0.00	548(100)	548.00(0.00)	0.00
johnson32-2-4	2033*	2033(100)	2033.00(0.00)	0.04	2033(100)	2033.00(0.00)	0.04
johnson8-2-4	66	66(100)	66.00(0.00)	0.00	66(100)	66.00(0.00)	0.00
johnson8-4-4	511	511(100)	511.00(0.00)	0.00	511(100)	511.00(0.00)	0.00
keller4	1153	1153(100)	1153.00(0.00)	0.00	1153(100)	1153.00(0.00)	0.00
keller5	3317	3317(100)	3317.00(0.00)	1.12	3317(100)	3317.00(0.00)	0.33
keller6	8062	8062(100)	8062.00(0.00)	532.74	8062(96)	8059.91(10.78)	929.74
p_hat1000-1	1514*	1514(100)	1514.00(0.00)	0.14	1514(100)	1514.00(0.00)	0.28
p_hat1000-2	5777*	5777(100)	5777.00(0.00)	0.11	5777(100)	5777.00(0.00)	0.11
p_hat1000-3	8111	8111(100)	8111.00(0.00)	0.19	8111(100)	8111.00(0.00)	0.21
p_hat1500-1	1619*	1619(100)	1619.00(0.00)	0.32	1619(100)	1619.00(0.00)	0.39
p_hat1500-2	7360	7360(100)	7360.00(0.00)	0.35	7360(100)	7360.00(0.00)	0.44
p_hat1500-3	10321	10321(100)	10321.00(0.00)	2.06	10321(100)	10321.00(0.00)	0.50
p_hat300-1	1057	1057(100)	1057.00(0.00)	0.00	1057(100)	1057.00(0.00)	0.00
p_hat300-2	2487	2487(100)	2487.00(0.00)	0.01	2487(100)	2487.00(0.00)	0.01
p_hat300-3	3774	3774(100)	3774.00(0.00)	0.01	3774(100)	3774.00(0.00)	0.01
p_hat500-1	1231*	1231(100)	1231.00(0.00)	0.03	1231(100)	1231.00(0.00)	0.04
p_hat500-2	3920*	3920(100)	3920.00(0.00)	0.02	3920(100)	3920.00(0.00)	0.03
p_hat500-3	5375*	5375(100)	5375.00(0.00)	0.04	5375(100)	5375.00(0.00)	0.05
p_hat700-1	1441*	1441(100)	1441.00(0.00)	0.04	1441(100)	1441.00(0.00)	0.05
p_hat700-2	5290*	5290(100)	5290.00(0.00)	0.06	5290(100)	5290.00(0.00)	0.05
p_hat700-3	7565	7565(100)	7565.00(0.00)	0.10	7565(100)	7565.00(0.00)	0.08
san1000	1716*	1716(100)	1716.00(0.00)	71.07	1716(100)	1716.00(0.00)	12.08
san200_0.7_1	3370	3370(100)	3370.00(0.00)	0.21	3370(100)	3370.00(0.00)	0.13
san200_0.7_2	2422*	2422(100)	2422.00(0.00)	0.04	2422(100)	2422.00(0.00)	0.01
san200_0.9_1	6825*	6825(100)	6825.00(0.00)	0.04	6825(100)	6825.00(0.00)	0.01
san200_0.9_2	6082*	6082(100)	6082.00(0.00)	0.00	6082(100)	6082.00(0.00)	0.00
san200_0.9_3	4748*	4748(100)	4748.00(0.00)	0.01	4748(100)	4748.00(0.00)	0.01
san400_0.5_1	1455	1455(100)	1455.00(0.00)	0.19	1455(100)	1455.00(0.00)	0.11
san400_0.7_1	3941*	3941(97)	3932.00(51.18)	172.04	3941(100)	3941.00(0.00)	74.66
san400_0.7_2	3110*	3110(97)	3105.26(26.95)	234.69	3110(100)	3110.00(0.00)	40.11
san400_0.7_3	2771*	2771(100)	2771.00(0.00)	0.41	2771(100)	2771.00(0.00)	0.07
san400_0.9_1	9776*	9776(100)	9776.00(0.00)	2.38	9776(100)	9776.00(0.00)	0.44
sanr200_0.7	2325*	2325(100)	2325.00(0.00)	0.00	2325(100)	2325.00(0.00)	0.00
sanr200_0.9	5126*	5126(100)	5126.00(0.00)	0.00	5126(100)	5126.00(0.00)	0.00
sanr400_0.5	1835*	1835(100)	1835.00(0.00)	0.02	1835(100)	1835.00(0.00)	0.02
sanr400_0.7	2992*	2992(100)	2992.00(0.00)	0.03	2990(100)	2990.00(0.00)	1.54

Table 2.2: Computational results of ReTS-I and ReTS-II on 40 BHOSLIB instances.

instance	BKV	ReTS-I			ReTS-II		
		$best(hit)$	$ave(std)$	$time$	$best(hit)$	$ave(std)$	$time$
frb30-15-1	2990*	2990(100)	2990.00(0.00)	1.43	2990(100)	2990.00(0.00)	1.54
frb30-15-2	3006*	3006(100)	3006.00(0.00)	2.09	3006(100)	3006.00(0.00)	0.28
frb30-15-3	2995*	2995(100)	2995.00(0.00)	1.84	2995(100)	2995.00(0.00)	1.31
frb30-15-4	3032*	3032(100)	3032.00(0.00)	0.31	3032(100)	3032.00(0.00)	0.17
frb30-15-5	3011*	3011(100)	3011.00(0.00)	0.80	3011(100)	3011.00(0.00)	1.16
frb35-17-1	3650	3650(100)	3650.00(0.00)	5.10	3650(100)	3650.00(0.00)	3.19
frb35-17-2	3738	3738(100)	3738.00(0.00)	87.05	3738(100)	3738.00(0.00)	65.51
frb35-17-3	3716	3716(100)	3716.00(0.00)	22.26	3716(100)	3716.00(0.00)	10.17
frb35-17-4	3683	3683(100)	3683.00(0.00)	14.80	3683(100)	3683.00(0.00)	1.89
frb35-17-5	3686	3686(100)	3686.00(0.00)	2.70	3686(100)	3686.00(0.00)	6.40
frb40-19-1	4063	4063(100)	4063.00(0.00)	51.68	4063(100)	4063.00(0.00)	61.32
frb40-19-2	4112	4112(100)	4112.00(0.00)	71.72	4112(100)	4112.00(0.00)	73.87
frb40-19-3	4115	4115(99)	4114.94(0.60)	127.00	4115(100)	4115.00(0.00)	79.66
frb40-19-4	4136	4136(98)	4135.92(0.56)	160.48	4136(100)	4136.00(0.00)	44.45
frb40-19-5	4118	4118(100)	4118.00(0.00)	34.72	4118(98)	4117.96(0.28)	208.18
frb45-21-1	4760	4760(98)	4759.76(1.68)	161.39	4760(93)	4759.10(3.29)	231.63
frb45-21-2	4784	4784(100)	4784.00(0.00)	68.11	4784(100)	4784.00(0.00)	35.50
frb45-21-3	4765	4765(90)	4764.80(0.60)	253.27	4765(100)	4765.00(0.00)	53.80
frb45-21-4	4799	4799(100)	4799.00(0.00)	105.52	4799(100)	4799.00(0.00)	31.81
frb45-21-5	4779	4779(100)	4779.00(0.00)	11.23	4779(100)	4779.00(0.00)	10.56
frb50-23-1	5494	5494(4)	5485.18(4.01)	154.05	5494(4)	5482.58(5.64)	590.72
frb50-23-2	5462	5462(9)	5451.94(3.20)	393.53	5462(44)	5455.27(6.08)	458.97
frb50-23-3	5486	5486(57)	5485.24(1.59)	358.92	5486(87)	5485.82(0.50)	292.35
frb50-23-4	5454	*5453*(91)	5452.54(1.48)	243.84	5454(6)	5450.70(4.36)	548.32
frb50-23-5	5498	5498(100)	5498.00(0.00)	118.21	5498(94)	5497.31(2.80)	277.51
frb53-24-1	5670	5670(33)	5661.37(8.67)	349.95	5670(58)	5664.29(8.05)	325.66
frb53-24-2	5707	5707(1)	5685.28(8.73)	880.86	5707(5)	5689.22(10.82)	415.40
frb53-24-3	5640	**5655**(3)	5636.51(6.54)	417.69	**5655**(3)	5632.08(8.50)	457.64
frb53-24-4	5714	5714(4)	5696.85(17.10)	421.52	5714(7)	5693.46(17.38)	402.50
frb53-24-5	5659	5659(1)	5651.39(2.96)	777.93	5659(5)	5649.69(6.01)	381.24
frb56-25-1	5916	5916(59)	5906.48(14.25)	344.18	5916(51)	5900.01(18.76)	428.24
frb56-25-2	5886	5886(9)	5873.03(8.70)	516.06	5886(37)	5878.44(8.03)	470.10
frb56-25-3	5859	5859(1)	5832.31(13.27)	450.99	*5854*(1)	5821.85(14.57)	30.19
frb56-25-4	5892	5892(2)	5866.11(13.48)	477.79	*5885*(2)	5856.77(13.99)	449.13
frb56-25-5	5853	*5841*(1)	5812.23(9.32)	354.28	5853(2)	5816.87(13.86)	514.91
frb59-26-1	6591	6591(20)	6578.65(7.24)	521.08	6591(32)	6576.59(11.57)	432.31
frb59-26-2	6645	6645(13)	6589.14(25.69)	505.28	6645(25)	6599.84(30.81)	660.05
frb59-26-3	6608	6608(1)	6579.05(13.05)	973.94	6608(25)	6593.63(12.32)	455.84
frb59-26-4	6592	6592(71)	6585.08(12.12)	377.92	6592(18)	6565.23(20.67)	541.34
frb59-26-5	6584	6584(3)	6558.48(10.04)	320.54	6584(9)	6563.39(10.95)	399.46

Table 2.3: Computational results of ReTS-I and ReTS-II on 22 selected WDP instances.

instance	BKV	ReTS-I			ReTS-II		
		$best(hit)$	$ave(std)$	$time$	$best(hit)$	$ave(std)$	$time$
in101	72724.61	72724.62(63)	72243.68(731.24)	216.92	72724.62(100)	72724.62(0.00)	0.31
in108	75813.21	75813.21(100)	75813.21(0.00)	5.02	75813.21(100)	75813.21(0.00)	1.87
in115	70221.56	70221.56(76)	70149.21(128.75)	234.30	70221.56(100)	70221.56(0.00)	1.76
in201	81557.74	81557.74(100)	81557.74(0.00)	2.35	81557.74(100)	81557.74(0.00)	0.41
in207	93129.25	93129.25(100)	93129.25(0.00)	11.57	93129.25(100)	93129.25(0.00)	0.65
in209	87268.96	87268.96(11)	86812.25(160.57)	254.65	87268.96(100)	87268.96(0.00)	0.98
in401	77417.48*	77417.48(100)	77417.48(0.00)	0.69	77417.48(100)	77417.48(0.00)	0.04
in403	74843.96*	74843.96(100)	74843.96(0.00)	0.05	74843.96(100)	74843.96(0.00)	0.04
in404	78761.69*	78761.69(100)	78761.69(0.00)	0.48	78761.69(100)	78761.69(0.00)	0.14
in601	108800.45	108800.45(100)	108800.45(0.00)	76.26	108800.45(100)	108800.45(0.00)	3.34
in604	107733.80	107733.80(40)	107062.40(548.20)	257.76	107733.80(100)	107733.80(0.00)	2.90
in614	108364.58	108364.58(100)	108364.58(0.00)	24.66	108364.58(100)	108364.58(0.00)	0.84
Decay2000_200	159.67*	*156.97*(10)	154.63(0.91)	496.68	*157.34*(4)	155.99(0.52)	215.57
Decay2000_300	226.82*	*221.17*(1)	218.46(1.24)	116.72	*221.92*(9)	219.74(0.77)	201.19
Decay2000_400	277.01*	*270.90*(5)	267.79(1.36)	127.14	*271.16*(5)	269.85(0.65)	495.26
Decay2000_500	340.81*	*334.36*(1)	330.12(1.45)	114.02	*333.80*(5)	330.77(1.09)	552.07
Random2000_500	12.63*	12.63(100)	12.63(0.00)	0.19	12.63(100)	12.63(0.00)	0.18
Uniform2000_400_10	22.02	22.02(100)	22.02(0.00)	83.76	22.02(90)	22.00(0.06)	197.04
Uniform2000_500_10	26.56	26.56(100)	26.55(0.03)	244.54	26.56(100)	26.50(0.07)	294.15
Wrandom2000_500	37.69*	37.69(100)	37.69(0.00)	0.20	37.69(100)	37.69(0.00)	0.19
Paths2000_100	36.77*	*35.56*(88)	35.13(0.13)	45.90	*36.32*(72)	36.05(0.09)	54.53
Regions2000_40	4558.90*	4558.90(19)	4503.94(35.25)	223.79	4558.90(54)	4540.09(22.69)	220.75

Table 2.4: Experimental results of ReTS-I and ReTS-II in comparison with 3 reference algorithms on 27 selected 2nd DIMACS and BHOSLIB instances.

		ReTS-I			ReTS-II			MN/TS			BLS			BQP-PTS		
instances	BKV	*gap*	*hit*	time	*gap*	*hit*	*time*	*gap*	*hit*	*time*	*gap*	*hit*	*time*	*gap*	*hit*	time
C2000.9	10999	0	92	417.56	0	82	474.23	0	72	168.11	0	74	1152.78	0	72	2711.97
MANN_a27	12283*	**0**	78	82.77	**0**	99	60.03	-2	1	88.28	-2	16	396.58	-6	4	12264
MANN_a45	34265*	**-6**	1	157.98	**-13**	58	357.19	-73	1	390.58	-36	1	929.41	-71	2	17524.05
MANN_a81	111386	**-16**	1	990.02	**-109**	1	477.75	-258	1	832.24	-149	1	2942.54	-249	1	6167.28
p_hat1000-3	8111	0	100	0.19	0	100	0.21	0	96	188.38	0	100	1.78	0	100	0.65
brock800_4	2971*	0	31	835.03	0	93	506.41	0	100	49.70	0	100	339.07	0	8	105.35
keller6	8062	0	100	532.74	0	96	929.74	0	5	606.15	0	44	1980.16	0	2	3418.36
frb50-23-1	5494	0	4	154.05	0	4	590.72	0	6	186.62	0	11	1221.72	0	20	1911.49
frb50-23-2	5462	0	9	393.53	0	44	458.97	0	3	14966	0	5	2837.74	0	15	2338.40
frb50-23-3	5486	0	57	358.92	0	87	292.35	0	53	158.71	0	98	537.96	0	100	418.35
frb50-23-4	5454	-1	91	243.84	0	6	548.32	0	9	176.41	0	14	1190.43	0	28	1957.22
frb50-23-5	5498	0	100	118.21	0	94	277.51	0	89	110.85	0	100	388.18	0	100	751.84
frb53-24-1	5670	0	33	349.95	0	58	325.66	0	5	233.22	0	13	1056.82	0	43	981.33
frb53-24-2	5707	0	1	880.86	0	5	415.40	0	6	145.22	0	3	147.65	0	25	1265.70
frb53-24-3	5640	**15**	3	417.69	**15**	3	457.64	0	15	215.79	0	48	984.53	0	90	1486.24
frb53-24-4	5714	0	4	421.52	0	7	402.50	0	7	449.39	0	13	1604.50	0	25	1753.36
frb53-24-5	5659	0	1	777.93	0	5	381.24	0	5	294.00	0	4	278.91	0	6	2802.83
frb56-25-1	5916	0	59	344.18	0	51	428.24	0	3	308.90	0	5	1764.87	0	19	1035.00
frb56-25-2	5886	0	9	516.06	0	37	470.10	-14	1	73.25	0	1	1013.85	0	3	1428.18
frb56-25-3	5859	0	1	450.99	-5	1	30.19	0	1	191.93	0	1	101.48	0	5	1756.22
frb56-25-4	5892	0	2	477.79	-7	2	449.13	0	3	104.58	0	12	1256.9	0	5	1756.22
frb56-25-5	5853	-12	1	354.28	0	2	514.91	0	1	322.70	0	1	4386.6	0	1	3549.57
frb59-26-1	6591	0	20	521.08	0	32	432.31	0	3	166.20	0	17	1435.99	0	67	2228.21
frb59-26-2	6645	0	13	505.28	0	25	660.05	0	3	212.49	0	13	1834.93	0	40	1820.56
frb59-26-3	6608	0	1	973.94	0	25	455.84	0	1	232.77	0	1	507.93	0	1	2561.16
frb59-26-4	6592	0	71	377.92	0	18	541.34	0	1	318.39	0	6	952.34	0	5	3322.64
frb59-26-5	6584	0	3	320.54	0	9	399.46	0	1	161.47	0	5	1512.09	0	9	747.80

Programs (BQP) [Wang *et al.*, 2016]. To solve the MVWCP instances, each instance is first recast into a BQP which is then solved by the probabilistic tabu search algorithm. The 2nd DIMACS and BHOSLIB instances were tested by this method on a PC with a Pentium 2.83GHz CPU and 2GB RAM. Each benchmark instance was solved by 100 independent trails, each trail being limited to 3,600 seconds, but extended to 36,000 seconds for large instances C4000.5, MABNN_a27, MANN_a45, MANN_a81.

- **CPLEX**. For the WDP instances, we include the results reported in [Wu and Hao, 2015b], which were obtained by the exact solver, CPLEX 12.4, within a maximum of 3600 seconds on a PC cadenced at 2.83GHz with 8GB of RAM.

Considering that MN/TS, BLS and BQP-PTS have a 100% success rate on most of the 2nd DIMACS instances in less than 1 second, we selected 7 hard and representative instances from this set in order to summarize the performances of the 5 compared algorithms. Moreover, as indicated in Section 2.4.3, the last 20 instances of BHOSLIB are more challenging for ReTS-I and ReTS-II than the first 20 instances, we only highlight the comparative results on these last 20 instances. The results of this comparison are summarized in Table 2.4. Column gap represents the gap between the objective value of the best solution found by an algorithm and the best-known value in the literature (BKV). A positive (negative) gap value indicates a better (worse) result compared to the current best-known value.

Table 2.4 indicates that both ReTS-I and ReTS-II attain better results than the 3 reference algorithms on 4 instances (highlighted in bold font). Though the MANN_aXX instances were reported as challenging for heuristic algorithms, ReTS-I and ReTS-II reach the optimal solution of MANN_a27 and better solutions on MANN_a45 and MANN_a81. BLS and BQP-PTS reach the best-known solutions for all other instances while the other algorithms fail on 1 or 2 instances. However, they achieve such a performance by using a larger cutoff time, which is also confirmed by the fact that the average time of BLS and BQP-PTS is significantly longer than the 3 other algorithms. In an additional experiment, we used a maximum of 1.6×10^8 iterations per run (the same condition as that used by BLS) and re-ran ReTS-I to solve frb53-23-4 and frb56-2-5, ReTS-II to solve frb56-25-3 (setting $\rho = 0.3$ in this case) and frb56-25-4. The results, shown in Table 2.5, indicate that ReTS-I and ReTS-II, like BLS and BQP-PTS, are also able to hit all the BKVs

Table 2.5: Improved results of ReTS-I on frb50-23-4 and frb56-25-5 and improved results of ReTS-II on frb56-25-3 and frb56-25-4 with an extended cutoff time limit.

solver	instance	BKV	$best(hit)$	$ave(std)$	*time*
ReTS-I	frb50-23-4	5454	5454(3)	5452.99(0.44)	504.07
	frb56-25-5	5853	5853(5)	5820.14(14.23)	763.00
ReTS-II	frb56-25-3	5859	5859(2)	5831.98(14.07)	1386.24
	frb56-25-4	5885	5885(5)	5863.16(13.09)	1003.07

Table 2.6: Average hits on 18 selected instances.

ReTS-I	ReTS-II	MN/TS	BLS	BQP-PTS
38.8	46.4	25.5	34.0	36.5

with a similar computational effort. Finally, we observe that MN/TS is the most time effective heuristic among the compared algorithms.

To further compare the competing algorithms, we extract the rows from Table 2.4 where $gap = 0$ for all 5 algorithms (18 rows in total), and recalculate the average number of the best trials ($hits$) for these 18 rows. Results are shown in Table 2.6, and indicate that ReTS-II and ReTS-I are the most robust algorithms, followed by BQP-PTS, BLS and MN/TS. We conclude thus that ReTS-I and ReTS-II compete favorably compared to the reference algorithms in terms of solution quality, robustness and computational time on the 2nd DIMACS and BHOSLIB instances.

Finally, Table 2.7 summarizes the results of ReTS-I, ReTS-II, MN/TS and CPLEX on 10 representative WDP instances, including the most challenging ones (with respect to results taken from Table 2.3). If we look at the solution quality, we observe that the three heuristic algorithms (ReTS-I, ReTS-II, MN/TS) attain the best-known solutions for the tested *in* instances (in101, in108, in109) and the unique *Uniform2000_400_10* instance while CPLEX fails to solve these instances. However, CPLEX is able to find the optimum solutions of Decay2000_yyy and Paths2000_100, contrary to the three heuristic methods. Among the compared heuristic algorithms, ReTS-I finds the best solution on 1 instance, ReTS-II on 2 instances and MN/TS on 2 (marked by bold font). Therefore, no algorithm outperforms the other algorithms in terms of solution quality and computational time. So, on the WDP instances, ReTS-I, ReTS-II and MN/TS perform similarly. Finally, this experiment confirms that exact solvers like CPLEX and heuristics like ReTS-I, ReTS-II and MN/TS are complementary solution methods and together can enlarge the class of MVWCP instances that can be solved effectively.

2.5 Effectiveness of restart strategy

As shown in Section 2.3, the proposed approach uses a restart strategy to displace the search to new regions when a deep local optimum is attained by tabu search (Alg. 1, lines 6-9). The restart strategy initializes the new starting solution of the next round of TS with either the reconstruction procedure (Section 2.3.2) or the random procedure (Section 2.3.1). The choice between these two restarting procedures is determined with a probability ρ. Intuitively, the reconstruction procedure leads the search to a nearby region (since it is guided by means of the objective function), while the random procedure diversifies more strongly the search.

In this section, we investigate the impact of the joint use of these two restart procedures by testing various probabilistic values $\rho \in \{k/10\}$ ($k \in [\![1, 10]\!]$). The two extreme values $\rho = 0$ and $\rho = 1$ correspond to the cases where only the random or the reconstruction procedure is applied. This study was based on 7 representative instances selected from the 3 benchmark sets. Each instance was solved 20 times by ReTS-I and ReTS-II with a given ρ value, each run being limited to 120 seconds.

Table 2.9 reports the results of this experiment. Column $ave(std)$ indicates the average and standard

Table 2.7: Comparison of our ReTS-I and ReTS-II algorithms with MN/TS, CPLEX on the WDP instances

		ReTS-I			ReTS-II			MN/TS			CPLEX	
instances	BKV	*gap*	*hit*	*time*	*gap*	*hit*	*time*	*gap*	*hit*	*time*	*gap*	*time*
in101	72724.61	0	63	216.92	0	100	0.31	0	100	5.46	*-5622.67*	3600
in108	75813.21	0	100	5.02	0	100	1.87	0	73	113.53	*-1175.42*	3600
in209	87268.96	0	11	254.65	0	100	0.98	0	100	11.25	*-4102.57*	3600
Decay2000_200	159.67*	*-2.70*	10	496.68	*-2.33*	3	215.57	***-0.49***	3	220.01	0	0.39
Decay2000_300	226.82*	*-5.65*	1	116.72	***-4.9***	9	201.19	*-6.16*	1	226.23	0	0.70
Decay2000_400	277.01*	*-10.11*	5	127.14	***-6.74***	5	495.26	-11.14	1	256.66	0	0.72
Decay2000_500	340.81*	***-6.45***	1	114.02	*-7.01*	5	552.07	*-24.70*	1	189.36	0	1.23
Uniform2000_400_10	22.02	0	100	83.76	0	90	197.04	0	100	79.70	*-2.84*	3600
Paths2000_100	36.77*	*-1.21*	88	45.90	*-0.45*	72	54.53	***-0.36***	1	225.39	0	0.09
Regions2000_40	4558.90	0	19	223.79	0	54	220.75	0	100	4.63	0	0.22

Table 2.8: Value of ρ which allows each algorithm to reach its best performance.

	in604	Decay2000_500	C2000.9	MANN_a45	keller6	frb50-23-2	frb59-26-5
ReTS-I	0.0-1.0	0.0	0.4	0.9	1.0	0.9	0.8
ReTS-II	0.1	0.0	0.5	0.7	0.9	0.7	1.0

deviation of the best objectives for the 20 runs, and column $time$ shows the average time in seconds needed to reach the best objective values of the 20 runs. We additionally report in Table 2.8 the values of parameter ρ which lead to the maximum average objective values.

Table 2.9: Impact of the parameter ρ on the results of ReTS-I and ReTS-II

Algorithm	ρ	in604		Decay2000_500		C2000.9		MANN_a45		keller6		frb50-23-2		frb59-26-5	
		$ave(std)$	$time$	$ave(std)$	$time$	$ave(std)$	$time$	$ave(std)$	$time$	$ave(std)$	$time$	$ave(std)$	$time$	$ave(std)$	$time$
ReTS-I	0	107733.80(0.00)	3.07	330.59(2.21)	24.37	10947.45(33.72)	54.14	34175.80(5.82)	44.17	7818.50(105.20)	57.88	5446.20(7.37)	49.42	6539.65(15.13)	48.74
	0.1	107733.80(0.00)	3.60	329.09(1.26)	25.08	10951.20(37.20)	59.43	34235.95(4.93)	55.55	7841.05(103.14)	59.34	5445.20(8.33)	53.60	6530.80(12.52)	56.83
	0.2	107733.80(0.00)	2.45	329.32(1.36)	37.79	10946.60(29.54)	44.86	34241.75(3.08)	49.92	7856.15(86.90)	60.70	5446.80(6.38)	63.48	6535.80(12.34)	46.74
	0.3	107733.80(0.00)	3.38	329.75(1.45)	38.69	10937.20(25.06)	57.10	34244.55(3.01)	63.80	7861.85(81.05)	66.02	5445.10(7.39)	59.91	6538.90(16.73)	48.74
	0.4	107733.80(0.00)	3.12	329.30(1.12)	35.88	10958.35(34.90)	66.12	34247.15(3.95)	53.13	7809.75(69.26)	62.93	5443.75(9.09)	54.26	6539.05(15.55)	58.78
	0.5	107733.80(0.00)	2.07	329.97(1.40)	27.37	10935.85(29.43)	65.60	34249.30(3.74)	40.57	7866.25(100.82)	72.52	5446.35(7.53)	61.54	6541.45(15.72)	50.78
	0.6	107733.80(0.00)	2.75	329.69(1.05)	45.96	10950.05(27.00)	52.82	34250.70(2.43)	54.44	7845.90(96.06)	50.91	5448.65(3.64)	53.85	6537.50(10.57)	69.19
	0.7	107733.80(0.00)	2.76	330.03(1.71)	38.22	10956.50(26.82)	55.75	34251.85(1.93)	38.78	7899.10(102.42)	53.65	5439.60(8.21)	44.17	6543.95(13.65)	41.12
	0.8	107733.80(0.00)	3.34	329.69(1.26)	38.46	10934.05(29.88)	42.43	34252.65(0.85)	50.09	7883.55(88.75)	56.29	5445.30(8.59)	53.97	6544.60(13.71)	54.71
	0.9	107733.80(0.00)	2.54	330.16(1.11)	52.61	10943.70(36.12)	72.01	34252.75(0.94)	35.58	7933.75(98.15)	68.88	5449.30(6.49)	61.68	6543.15(15.37)	58.47
	1	107733.80(0.00)	3.76	329.33(0.87)	37.26	10945.25(24.52)	54.04	34248.10(15.94)	32.01	7935.70(103.42)	56.19	5447.90(8.39)	63.21	6543.95(13.61)	76.84
ReTS-II	0	106814.21(471.76)	55.32	329.92(1.73)	17.48	10963.80(8.62)	27.10	34174.50(2.25)	56.36	7935.70(67.82)	49.87	5430.60(9.25)	36.61	6529.30(11.33)	46.01
	0.1	106950.50(512.80)	54.46	329.37(1.53)	27.42	10967.00(13.82)	38.95	34245.40(4.10)	62.46	7976.70(57.01)	48.09	5439.90(12.62)	47.10	6529.45(15.90)	50.60
	0.2	106758.26(423.24)	55.67	329.73(1.85)	22.10	10964.15(9.03)	38.18	34247.40(3.50)	68.88	7964.20(51.37)	48.16	5435.40(10.98)	47.06	6530.65(11.72)	43.64
	0.3	106726.69(335.70)	72.29	329.10(1.45)	20.09	10971.30(16.23)	48.58	34249.15(3.00)	46.54	7995.70(49.62)	47.75	5441.65(9.89)	41.88	6531.35(16.94)	54.23
	0.4	106702.31(359.79)	56.44	329.51(1.79)	21.91	10966.75(13.74)	38.46	34251.90(2.05)	57.77	7977.65(56.80)	52.74	5437.85(9.47)	61.21	6542.45(17.16)	68.41
	0.5	106590.41(106.27)	62.78	329.84(2.34)	16.85	10971.95(16.13)	44.63	34251.45(1.66)	54.16	7990.00(60.01)	45.35	5440.20(10.47)	54.10	6540.95(16.87)	46.51
	0.6	106838.59(447.61)	55.59	329.87(2.02)	27.76	10965.10(11.83)	46.16	34252.20(1.83)	57.73	8010.65(54.62)	48.87	5445.00(9.14)	47.43	6541.75(14.87)	53.48
	0.7	106805.36(486.01)	58.48	328.95(1.83)	18.86	10964.80(11.78)	33.79	34252.30(1.14)	52.75	7998.70(50.97)	51.47	5445.85(8.64)	52.97	6544.35(11.89)	48.30
	0.8	106726.69(335.70)	43.21	329.06(2.13)	17.22	10964.75(11.76)	57.51	34251.90(1.30)	39.51	7974.90(64.72)	55.58	5441.80(9.82)	31.95	6545.85(12.59)	34.31
	0.9	106539.06(756.19)	73.47	329.36(2.42)	20.08	10967.85(13.51)	36.50	34251.60(1.62)	38.47	8018.30(37.98)	48.27	5444.55(8.96)	43.34	6544.50(9.46)	61.41
	1	106511.78(449.03)	39.02	327.97(2.48)	7.60	10968.50(15.40)	41.86	34251.60(1.43)	58.75	7944.55(64.39)	61.43	5444.95(8.95)	44.45	6551.55(11.07)	42.17

According to Table 2.9, setting ρ to small values close to 0 lead to better results on instances in604 and Decay2000_500, while high values of ρ are preferable on other instances, which indicates that the reconstruction procedure guided by the objective function is helpful to attain better solutions in most cases. This observation also emphasizes the use of $\rho = 0.7$ for the experiments of Section 4.3.7. Moreover, Table 2.8 shows that for each instance, ReTS-I and ReTS-II have close best ρ values, which attests the relevance of integrating both algorithms into the same search framework.

Finally, Table 2.9 discloses that the impact of ρ on the performance of ReTS-I and ReTS-II varies according to the instances. In particular, for MANN_a45, the result is gradually improved with increasing ρ values, reaching the best objective value when $\rho = 0.7$ for ReTS-I and 0.9 for ReTS-II respectively.

2.6 Conclusion

In this chapter, we considered the maximum vertex weight clique problem (MVWCP). We first pointed out the limits of the commonly used *ADD* and *SWAP* operators, and proposed the new generalized operator *PUSH*. *PUSH* inserts one vertex into the clique and removes $k \geq 0$ vertices from the clique to maintain its feasibility. *PUSH* not only generalizes the existing move operators, but also offers possibility of defining additional clique transformation operators. We proposed different local search operators by customizing the candidate set of *PUSH*. Based on the new operator, we design two tabu search algorithms, ReTS-I and ReTS-II, which share the same probabilistic restart mechanism but explore different candidate push sets.

Experiments on 143 2nd DIMACS, BHOSLIB and WDP benchmark instances demonstrated the effectiveness of the algorithms. Further experimental comparisons indicated that our algorithms compete favorably with state-of-the-art ones. Complementary analysis of restart frequency justified the effectiveness of sharing the same probabilistic restart mechanism.

In the next chapter, we will consider the maximum s-plex problem, which is a relaxation of the maximum clique problem. To solve this problem effectively, a frequency based local search algorithm will be introduced.

3

Frequency-driven tabu search for the maximum s-plex problem

In this chapter, we present an effective frequency-driven multi-neighborhood tabu search algorithm (FD-TS) to solve the maximum s-plex problem (MsPlex) on very large networks. The proposed FD-TS algorithm relies on two transformation operators (*ADD* and *SWAP*) to locate high-quality solutions, and a frequency-driven perturbation operator (*PRESS*) to search beyond local optimum traps. We report computational results for 47 massive real-life (sparse) graphs from the SNAP Collection and the 10th DIMACS Challenge, as well as 52 (dense) graphs from the 2nd DIMACS Challenge (results for 48 more graphs are also provided in Appendix 5.6). We demonstrate the effectiveness of our approach by presenting comparisons with the current best-performing algorithms. An article describing FD-TS is accepted to *Computer & Operations Research* [Zhou and Hao, 2017b].

Contents

3.1 Introduction

Given a simple undirected graph $G = (V, E)$ with a vertex set V and an edge set E, let $N(v)$ denote the set of vertices adjacent to v in G. Then, an s-plex for a given integer $s \geq 1$ ($s \in \mathbb{N}^+$) is a subset of vertices $C \subseteq V$ that satisfies the following condition: $\forall v \in C$, $|N(v) \cap C| \geq |C| - s$. Thus, each vertex of an s-plex C must be adjacent to at least $|C| - s$ vertices in the subgraph $G[C] = (C, E \cap (C \times C))$ induced by C. Therefore, the maximum s-plex problem (MsPlex) consists of finding, for a fixed value of s, an s-plex of maximum cardinality among all possible s-plexes of a given graph. The mathematical formulation of the maximum s-plex has been introduced in chapter one (Equation (1.4) – (1.6)) and is not repeated here.

The s-plex concept was first introduced for graph-theoretic social network studies [Seidman and Foster, 1978]. The decision version of MsPlex with any fixed positive integer s is known to be NP-complete [Balasundaram *et al.*, 2011]. When s equals 1, MsPlex reduces to the popular maximum clique problem, the decision version of which was among Karp's 21 NP-complete problems [Karp, 1972]. MsPlex is often referred to as a *clique relaxation* model [Pattillo *et al.*, 2012; Pattillo *et al.*, 2013b].

Similar to a clique, an s-plex C has the heredity property, which means that every subset of vertices $C' \subseteq C$ remains an s-plex, i.e., the subgraph induced by C' always has the property of an s-plex [Trukhanov *et al.*, 2013]. The most successful combinatorial algorithms for s-plex essentially rely on the heredity property and a polynomial feasibility verification procedure. For example, a powerful exact algorithmic framework was introduced in [Trukhanov *et al.*, 2013] for detecting optimal hereditary structures (s-plex and s-defective clique), which is based on the maximum clique algorithm proposed in [Östergård, 2002]. This algorithm performed well on MsPlex for graphs in the 2nd DIMACS Challenge and popular large-scale social networks. Among exact algorithms for MsPlex, [Balasundaram *et al.*, 2011] introduced a branch-and-cut algorithm based on polyhedral study of MsPlex. Two branch-and-bound algorithms were presented in [McClosky and Hicks, 2012], which are based on popular exact algorithms for the maximum clique problem [Carraghan and Pardalos, 1990; Östergård, 2002]. In [Moser *et al.*, 2012], exact combinatorial algorithms were investigated using methods from parameterized algorithmics. Additionally, a parallel algorithm for listing all the maximal s-plexes was introduced in [Wu and Pei, 2007].

However, given the computational complexity of MsPlex, any exact algorithm is expected to require an exponential computational time to determine the optimal solution in the general case. Thus, it is useful to investigate heuristic approaches, which aim to provide satisfactory solutions within an acceptable time frame, but without a provable optimal guarantee for the solutions obtained. However, our literature review found only two heuristics for MsPlex [Miao and Balasundaram, 2012; Gujjula *et al.*, 2014], which are based on the general GRASP method [Feo and Resende, 1995a]. This situation contrasts sharply with the huge body of heuristics for the conventional maximum clique problem [Wu and Hao, 2015a] and other clique relaxation problems [Pattillo *et al.*, 2013b]. We note that exact and heuristic approaches may be complementary, and together they can enlarge the classes of problem instances that can be solved effectively. Moreover, they can even be combined within a hybrid approach, as exemplified in [Miao and Balasundaram, 2012] where the GRASP heuristic was used to enhance the exact algorithm proposed in [Balasundaram *et al.*, 2011] to solve very large social network instances.

In this study, we aim to partially fill the gap in terms of heuristic methods for solving MsPlex by introducing an effective heuristic approach. The main contributions of this study can be summarized as follows.

– From an algorithmic perspective, this is the first study to employ the tabu search metaheuristic [Glover and Laguna, 2013] to solve MsPlex (Section 3.2). Thus, the proposed frequency-driven tabu search algorithm (FD-TS) integrates several original components. First, FD-TS jointly employs three dedicated move operators called ADD, $SWAP$, and $PRESS$, two of which ($SWAP$ and $PRESS$) are applied for the first time to MsPlex. Second, we introduce a frequency-based mechanism for perturbation and to construct initial solutions, which is proving to be more effective than a random mechanism. We also apply a peeling procedure to dynamically reduce the graph with the best identified lower bound. Finally, specific design decisions are made in order to handle very large networks with thousands and even millions of vertices.

- From a computational perspective, our experimental results indicate that the proposed algorithm performs very well with both sparse and dense graphs (Section 3.4). For 47 very large networks from the SNAP collection and the 10th DIMACS Challenge benchmark set, our algorithm successfully obtained or improved the best-known results from previous studies for $s = 2, 3, 4, 5$. Our algorithm even proved the optimality of many instances for the first time using the peeling procedure. For 52 dense graphs from the collection used in the 2nd DIMACS Challenge, our algorithm also obtained or improved the best-known results for $s = 2, 3, 4, 5$. To comprehensively assess the performance of our algorithm, we compared FD-TS with several cutting edge algorithms, including the commercial CPLEX solver (version 12.6.1). More results for 48 additional graphs are also presented in Appendix 5.6.

The rest of this chapter is organized as follows. Section 3.2 presents the FD-TS algorithm. Section 3.3 discusses the implementation and complexity issues related to FD-TS. Section 3.4 presents the computational results obtained on benchmark instances and provide comparisons with state-of-the-art algorithms. In the final section of this chapter, we give our conclusions and discuss future research.

3.2 FD-TS algorithm for the maximum s-plex problem

3.2.1 General procedure

The general scheme of the proposed FD-TS algorithm is shown in Algorithm 4.1. FD-TS starts from an initial feasible solution (s-plex) built using the *Init_Solution()* procedure (Section 3.2.4), before entering the main multi-neighborhood local search procedure, *Freq_Tabu_Search()*, to improve the initial solution (Section 3.2.5). A vector $freq$, which records the number of times each vertex is moved in the last round of the *Freq_Tabu_Search()* procedure, is initialized as a null vector (Algorithm 4.1, line 3). This vector is used by the *Init_Solution()* procedure as well as the perturbation method explained in Section 3.2.5. If the solution returned by tabu search is better than the current best solution C^*, C^* is updated (Algorithm 4.1, lines 7–8). The new lower bound $|C^*|$ is then given to the *Peel()* procedure (Section 3.2.6) to reduce the current graph (Algorithm 4.1, line 9). If *Peel()* returns a reduced subgraph with fewer vertices than $|C^*|$, then C^* must be an optimal solution and the overall algorithm stops. Otherwise, the algorithm enters a new round of search to build a new starting solution with *Init_Solution()*, before improving the new starting solution with *Freq_Tabu_Search()* and reducing the graph with *Peel()* if this is possible. The algorithm continues until a given stopping condition (e.g., a cut-off time limit) is met.

3.2.2 Preliminary definitions

Given $G = (V, E)$, $s \in \mathbb{Z}^+$, let $C \subseteq V$ be a subset of vertices and $N(v)$ the set of vertices adjacent to v. The following definitions are provided, which are useful for the description of our algorithm.

We say that C is a (feasible) solution or an s-plex if $\forall v \in C, |N(v) \cap C| \geq |C| - s$; otherwise, C is an infeasible solution (i.e., $\exists v \in C, |N(v) \cap C| < |C| - s$). For a vertex $v \in C$, we say that v is *saturated* (first introduced in [Trukhanov *et al.*, 2013]) if $|N(v) \cap C| = |C| - s$. If $|N(v) \cap C| < |C| - s$, v is *deficient*. Obviously, whenever a deficient vertex exists in C, C is an infeasible solution. The saturated set S of set C is defined as the set of all saturated vertices in C, i.e., $S = \{v \in C : |N(v) \cap C| = |C| - s\}$. We can see that if C is a 1-plex (i.e., a clique), then all the vertices in C are saturated. The search space Ω of G includes all s-plexes, $\Omega = \{C \subseteq V : \forall v \in C, |N(v) \cap C| \geq |C| - s\}$. For brevity, we also use $N(C)$ to denote the set of vertices in $V \setminus C$ with at least one adjacent vertex in C, $N(C) = \bigcup_{v \in C} N(v) \setminus C$.

In Figures 3.1 and 3.2, we provide examples of S and $N(C)$, as well as their uses in the definitions of the *ADD* and *SWAP* operators introduced in the next section.

Finally, the quality of any candidate solution (s-plex) $C \in \Omega$ is evaluated by its cardinality $|C|$. Thus, given two candidate solutions C' and C, C' is better than C if $|C'| > |C|$.

Algorithm 3.1: Main framework of Frequency Driven Tabu Search

```
Input: Problem instance $(G, s)$, predefined sample size $q$, maximum allowed iterations in tabu
       search $L$.
Output: The largest $s$-plex ever found
begin
    $C^* \leftarrow \emptyset$;                              /* the best solution found so far */
    $freq(v) \leftarrow 0$ for all $v \in V$;               /* frequency count of vertex moves */
    while the stopping condition is not met do
        $\{C, freq\} \leftarrow Init\_Solution(G, s, freq, q)$;          /* §3.2.4 */
        $\{C, freq\} \leftarrow Freq\_Tabu\_Search(G, s, C, freq, L)$;   /* §3.2.5 */
        if $|C| > |C^*|$ then
            $C^* \leftarrow C$;
            $G \leftarrow Peel(G, s, |C^*|)$;                            /* §3.2.6 */
            if $|V| \leq |C^*|$ then
                return $C^*$ ;                          /* return the best solution found */
end
return $C^*$
```

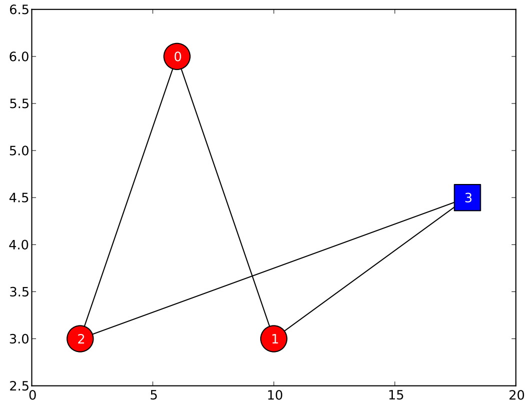

Figure 3.1: Suppose $s = 2$, $C = \{0, 1, 2\}$ is the incumbent solution, $S = \{1, 2\}$ is the saturated set of C, then $M_1 = \{3\}$ and $C \cup \{3\}$ is an extended s-plex.

3.2.3 Move operators

Our FD-TS algorithm explores the search space Ω by jointly applying three move (or transformation) operators, ADD, $SWAP$, and $PRESS$, to generate new solutions in Ω from the current solution (or s-plex). If we let C be the incumbent solution, then each move operator transfers one vertex $v \in N(C)$ inside C and eliminates zero, one, or more vertices from C to keep C feasible. If we let OP be a move operator, then we use $C' \leftarrow C \oplus OP(v, X)$ to denote the new (neighboring) solution obtained by applying OP to C (X represents the subset of vertices eliminated from C, which can possibly be empty). The details of these operators are described as follows. For simplicity, when the subset of eliminated vertices X is empty or a singleton, the set notation of X is ignored.

1. $ADD(v)$: This operator extends the incumbent solution C by including a new vertex from $N(C)$. Clearly, each application of this operator will increase the cardinality of the solution by one, which always leads to a better solution. However, we must take special care to ensure that the extended solution remains an s-plex. Thus, we identify the following vertex subset $M_1 \subseteq N(C)$, which has the required feasibility property (first introduced in [Trukhanov *et al.*, 2013]):

$$M_1 = \{v \in N(C) : |N(v) \cap C| \geq |C| - s + 1, S \setminus N(v) = \emptyset\} \tag{3.1}$$

By definition (3.1), if a vertex v in $N(C)$ is adjacent to at least $|C| - s + 1$ vertices in C and adjacent to all the saturated vertices of C, then adding v to C yields a new solution C', the size of which is increased by one (see Fig. 3.1 for an example).

The set of neighboring solutions of C induced by $ADD(v)$ is then given by:

$$\mathcal{N}_{ADD} = \{C' : C' \leftarrow C \oplus ADD(v), v \in M_1\} \tag{3.2}$$

The ADD operator is used by the search algorithm to improve the quality of the incumbent solution.

2. $SWAP(v, u)$: This operator exchanges a vertex $v \in N(C)$ with another vertex u in the incumbent solution C ($u \in C$), while keeping the quality of the solution unchanged. Similar to ADD, to ensure the feasibility of the transformed solutions, we need to identify the set of suitable candidate pairs $< v, u > \in N(C) \times C$. Considering the definition of s-plex, a pair of vertices $< v, u >$ is eligible for exchange only if it satisfies one of the following two conditions.
   - First, v is adjacent to at least $|C| - s$ vertices in C and u is the unique saturated vertex that is not adjacent to v (i.e., $S \setminus N(v) = \{u\}$).
   - Second, v is adjacent to exactly $|C| - s$ vertices in C and these $|C| - s$ vertices must include all the saturated vertices (i.e., $S \setminus N(v) = \emptyset$), and u is an arbitrary vertex from $C \setminus N(v)$.

   We use sets A and B to denote the candidate sets of v that satisfy the two conditions above, respectively, (see Fig. 3.2 for an example of these two types of vertices):

$$\begin{aligned} A &= \{v \in N(C) : |N(v) \cap C| \geq |C| - s, |S \setminus N(v)| = 1\} \\ B &= \{v \in N(C) : |N(v) \cap C| = |C| - s, S \setminus N(v) = \emptyset\} \end{aligned} \tag{3.3}$$

   The set of neighboring solutions induced by $SWAP(v, u)$ is then given by:

$$\mathcal{N}_{SWAP} = \{C' : C' \leftarrow C \oplus SWAP(v, u), (v \in A, u \in S \setminus N(v)) \vee (v \in B, u \in C \setminus N(v))\} \tag{3.4}$$

   If we let $M_2 = A \cup B$, a practical method for generating a suitable pair $< v, u >$ is to first build the set M_2, then pick a vertex v from M_2, and finally determine an appropriate vertex u from $S \setminus N(v)$ or $C \setminus N(v)$. The $SWAP$ operator is used by the search algorithm to visit neighboring solutions of equal quality, so a transition with $SWAP$ is also called a side-walk move.

3. $PRESS(v, X)$: If we let $M_3 = N(C) \setminus (M_1 \cup M_2)$, this operator adds one vertex $v \in M_3$ to C and then eliminates two or more vertices from $C \setminus N(v)$ so the s-plex structure of the new solution is maintained. Obviously, given a vertex $v \notin M_1$, the set $C' = C \cup \{v\}$ is an infeasible solution because v or the vertices in $S \setminus N(v)$ become deficient (see Section 3.2.2) in $G[C']$. Therefore, to restore the feasibility of the transformed solution, this operator iteratively eliminates vertices from $(C' \setminus \{v\}) \setminus N(v)$ (i.e., $C \setminus N(v)$) until the solution becomes an s-plex. Preference is given to the *deficient* vertices in $C \setminus N(v)$ and if no deficient vertex exists in $C \setminus N(v)$, the vertices to be eliminated are selected randomly from C. All of the eliminated vertices are collected in X. The set of neighboring solutions induced by the $PRESS(v, X)$ operator is given by:

$$\mathcal{N}_{PRESS} = \{C' : C' \leftarrow C \oplus PRESS(v, X), v \in V \setminus C, X \subseteq C \setminus N(v)\} \tag{3.5}$$

   By definition, $PRESS$ eliminates at least two vertices from the solution (i.e., $|X| > 1$). Thus, the application of $PRESS$ always degrades the quality of the current solution. Hence, this operator is only used by the perturbation procedure when the ADD and $SWAP$ operators are no longer applicable, or when the search stagnates in local optima.

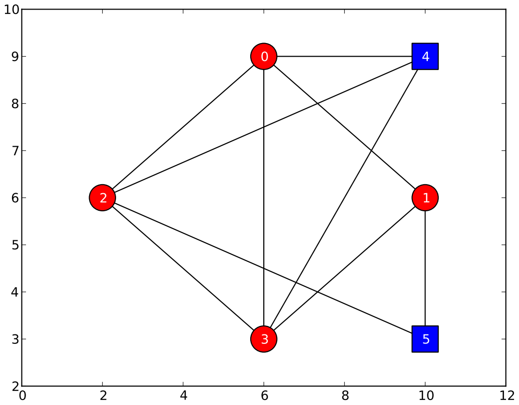

Figure 3.2: Suppose $s = 2$, $C = \{0, 1, 2, 3\}$ is the incumbent solution, $S = \{1, 2\}$ is the saturated set of C, $N(C) = \{4, 5\}$. Then for the $SWAP$ operator, we get $A = \{4\}$ and exchangeable pair $< 4, 1 >$; $B = \{5\}$ and exchangeable pairs $< 5, 0 >$ and $< 5, 3 >$.

Next, we provide some additional comments about these operators and discuss some implementation issues.

- ADD and $SWAP$ have been used in several algorithms for the maximum clique problem and the equivalent maximum independent set problem [Benlic and Hao, 2013a; Jin and Hao, 2015; Pullan and Hoos, 2006; Wu *et al.*, 2012]. However, given the generality of MsPlex, the definitions of these operators are different and more complex in the present study; in particular, the saturated set S must be involved. The $PRESS$ operator can be treated as a dedicated application to the s-plex problem of the $PUSH$ operator introduced in [Zhou *et al.*, 2017a] for the maximum weight clique problem.
- It should be noted that traditional maximum clique benchmark graphs, such as those from the 2nd DIMACS Challenge, include many *dense* graphs. Thus, most maximum clique algorithms typically operate on the complement graph $\bar{G}$ to search for the maximum independent sets in terms of run-time efficiency [Jin and Hao, 2015; Pullan and Hoos, 2006; Wu and Hao, 2013a]. The FD-TS algorithm proposed in the present study is designed to handle very large real-world networks, which are typically sparse graphs. Consequently, FD-TS operates directly on the original graph based on its adjacency-list representation. In addition, we restrict the candidate vertices considered by the three move operators to $N(C)$ instead of $V \setminus C$ because $N(C)$ is much smaller than $V \setminus C$ for very large networks. Indeed, given the size of the networks considered (up to millions of vertices), even a simple operation such as scanning all the vertices of $V \setminus C$ becomes too expensive, and it can slow down the algorithm considerably. Thus, special care is taken to avoid ineffective or unpromising examinations.
- When $SWAP$ and $PRESS$ are applied, each dropped vertex is forbidden from rejoining the solution during a number of subsequent iterations to avoid revisiting previously examined solutions. This is achieved by using a tabu list (see Section 3.2.5).

3.2.4 Constructing the initial solutions

Each round of the FD-TS algorithm requires a starting solution (see Algorithm 4.1, line 5). In general, the starting solutions can be generated by any method that ensures the s-plex property. In our method, we employ the following construction procedure, which applies the ADD operator while considering the frequency information for vertex moves.

From a random sample of q vertices ($q \in [50, 150]$), we use the vertex with the minimum frequency (ties are broken randomly) to create a singleton set C. From the singleton s-plex C, the procedure repeats the following three steps to extend the current solution: 1) generate the M_1 set from $N(C)$, 2) select one vertex with the minimum frequency from M_1 (ties are broken randomly), and 3) add the selected vertex to

C. We repeat this process until M_1 becomes empty. The final s-plex C is returned as the initial solution.

The intuitive assumption that less frequently moved vertices are preferred is intended to make the initial solution as diverse as possible. Moreover, using a sample of q vertices instead of the whole set V for seeding the solution helps to reduce the computational overheads in the initialization procedure. This is particularly true for massive graphs because scanning all the vertices of the graph can be time consuming in this case. In Section 3.4.2, we discuss the calibration of the parameter q.

3.2.5 FD-TS

Algorithm 3.2: Frequency driven tabu search

```
Input: Problem instance $(G, s)$, current solution $C$, max. allowed iterations $L$
Output: The largest $s$-plex found $C_{best}$
begin
    $C_{best} \leftarrow C$;
    $tabu\_list \leftarrow \emptyset$;
    $l \leftarrow 0$ ;                        /* Counter of cycles of TS$^2$ + PERTURB runs */
    $fix \leftarrow \emptyset$;                        /* The vertices that are forbidden to drop */
    $freq(v) \leftarrow 0$ for all $v \in V$ ;                        /* Reset frequency records */
    $\lambda \leftarrow 0$ ;                  /* Counter of consecutive non-improving iterations */
    // Run TS$^2$ + PERTURB a maximum of $L$ iterations
    while $l \leq L$ do
        // Start the two neighborhood tabu search procedure - TS$^2$
        Updating the saturated subset $S$ ;                                        /* Sect. 3.2.2 */
        Decompose set $N(C)$ into $M_1' = \{v \in M_1 : v \notin tabu\_list \vee |C| + 1 > |C_{best}|\}$, $M_2' = \{v \in M_2 : v \notin tabu\_list\}$,
        $M_3' = \{v \in M_3 : v \notin tabu\_list\}$ ;                                        /* Sect. 3.2.3 */
        if $M_1' \neq \emptyset$ then
            $v \leftarrow$ a random vertex from $M_1'$;
            $C \leftarrow C \oplus ADD(v)$;
            $freq(v) \leftarrow freq(v) + 1$;
        else if $M_2' \neq \emptyset$ then
            $(v, u) \leftarrow$ two random exchangeable vertices from $M_2' \times N(C)$;   /* See Sect. 3.2.5 for selection rule */
            $C \leftarrow C \oplus SWAP(v, u)$;
            Add $u$ to $tabu\_list$ with tabu tenure $T_u$;                                        /* Sect. 3.2.5 */
            $freq(v) \leftarrow freq(v) + 1$, $freq(u) \leftarrow freq(u) + 1$;
        if $|C| > |C_{best}|$ then
            $C_{best} \leftarrow C$;
            $\lambda \leftarrow 0$;
        else
            $\lambda \leftarrow \lambda + 1$;
        // Run the PERTURB procedure
        if (No feasible $ADD$ and $SWAP$ operation) $\vee$ $(\lambda > s * |C_{best}|) \wedge M_3' \neq \emptyset)$ then
            $v \leftarrow$ a vertex with maximum $freq(v)$ from $M_3'$, break ties randomly;
            $C \leftarrow C \oplus PRESS(v, X)$ ;        /* $X$ collects the removed vertices from $C$, Sect. 3.2.3 */
            $fix \leftarrow \{v\}, \lambda \leftarrow 0$;
            $freq(v) \leftarrow 0$, $freq(u) \leftarrow freq(u) + 1$ for all $u$ in $X$;
            Add each $u \in X$ to $tabu\_list$ with tabu tenure $T_u$;                                        /* Sect. 3.2.5 */
        $l \leftarrow l + 1$
end
Return $C_{best}, freq$;
```

General procedure

The key search procedure employed in the FD-TS algorithm (see Algorithm 3.2) combines a double-neighborhood search procedure (we refer to this procedure as TS^2; Algorithm 3.2, lines 11–24) to facilitate intensification (to obtain local optima) and a frequency-based perturbation procedure (we refer to this procedure as PERTURB; Algorithm 3.2, lines 25–30) for diversification (to escape from local optima).

Based on a given initial solution, TS^2 uses ADD and $SWAP$ to improve the current solution until search stagnation occurs. PERTURB then applies $PRESS$ to modify (perturb) the current local optimum and passes the modified solution to TS^2 for further improvement. FD-TS iterates this TS^2+PERTURB process a maximum of L times and then starts the next round of its search procedure.

In addition to the three move operators (ADD, $SWAP$, and $PRESS$), FD-TS employs a tabu mechanism (see Section 3.2.5) [Glover and Laguna, 2013] and a frequency technique to ensure the effective exploration of the search space. A tabu list ($tabu\_list$) is used to mark the vertices that are forbidden from joining the current solution during a specific number of iterations. Information related to the move frequency of each vertex v is collected, where $freq(v)$ records the number of times that v is operated upon by a move operator in the recent history. Initially, $tabu\_list$ is empty and the frequency of each vertex is set to 0. The fix set (a singleton set used by the PERTURB procedure) records an added vertex, which is forbidden from moving out of the current solution in TS^2 and the counter l counts the completed iterations, the upper limit of which is given by the parameter L. Moreover, in order to avoid being trapped by local optima, a counter λ records the consecutive iterations that have passed since the last improvement of the current solution. Each time that the counter λ reaches a threshold, the perturbation procedure is triggered to modify the current solution using the $PRESS$ operator.

At the beginning of each iteration, set C is checked to identify the saturated subset S. Next, $N(C)$ is decomposed into three disjoint subsets: M_1', M_2', and M_3'. These sets correspond to M_1, M_2, and M_3 (defined in Section 3.2.3), respectively, but they exclude the vertices in the $tabu\_list$. However, a vertex $v \in M_1$ is always retained in M_1' if adding v to C leads to a new solution that is better than the best solution found previously, i.e., $|C| + 1 > |C_{best}|$, regardless of the tabu status of the vertex.

Solution improvement with ADD and SWAP

The current solution is transformed successively by applying *ADD* and *SWAP*. Preference is given to the ADD operator. Thus, whenever M_1' is not empty, *ADD* is applied to improve the current solution by adding one vertex of M_1' to the solution (Algorithm 3.2, lines 11–14). If no vertex can be added to the solution ($M_1' = \emptyset$), but M_2' is not empty, then the search continues with the $SWAP(v, u)$ operator by visiting solutions of equal quality (Algorithm 3.2, lines 15–19).

To provide $SWAP(v, u)$ with an appropriate exchangeable pair $< v, u >$, v is first selected randomly from M_2', and u is then selected from the candidate set defined by the rules given in Section 3.2.3, while excluding the vertex recorded in fix. In particular, if v is a vertex of type (set) A, then u is selected from $S \setminus (N(v) \cup fix)$ (which is a trivial set with zero or one vertex); and if v is a vertex of type (set) B, u is selected randomly from $C \setminus (N(v) \cup fix)$. If there is no eligible candidate for u ($S \setminus N(v) = fix$ or $C \setminus N(v) = fix$), then we simply give up attempting to apply $SWAP(v, u)$ and move on to the PERTURB procedure (Algorithm 3.2, line 25).

Perturbation with PRESS

Inevitably, at a certain search stage, no candidate vertex v or candidate pair $< v, u >$ is available for the $ADD(v)$ or $SWAP(v, u)$ operator (i.e., both M_1' and M_2' are empty or no eligible vertex can be found for u), or the search stagnates on the $SWAP(v, u)$ operator. In the latter case, search stagnation occurs when the current solution has not been improved for $s * |C_{best}|$ consecutive iterations. The self-adaptive threshold, $s * |C_{best}|$, is identified based on the assumption that if the current s-plex cannot be improved even after the replacement of each of its vertices at least s times by the $SWAP$ operator, then no better solution can be found in the current search region. To continue the search, the algorithm triggers the PERTURB procedure in order to move the search to a distant new region. This procedure applies the $PRESS(v, X)$ operator, where v is the vertex from M_3' with the largest $freq$ value and X collects the dropped vertices from $C \setminus N(v)$ to recover a feasible solution (Algorithm 3.2, lines 26–27). It is important to reset the frequency of vertex v (Algorithm 3.2, line 29), or the accumulated frequency of this vertex could dominate other vertices in subsequent cycles. The dropped vertices in X are added to the tabu list (Algorithm 3.2, line 30) and they will not be considered during the forbidden period, as explained in the next section.

Tabu tenure and management

As mentioned above, to avoid revisiting recently examined solutions, we use a tabu list to record the vertices dropped from the current solution in order to exclude them from consideration during a number of consecutive iterations. According to the definitions in Section 3.2.3, each application of $SWAP(v, u)$ or $PRESS(v, X)$ removes one or more vertices from the current s-plex. Each dropped vertex u will be kept in the tabu list for the next T_u iterations (the tabu tenure), which is set according to the following two rules:

$$\begin{cases} T_u = 10 + random(0, |M_2|), \ u \text{ is dropped by } SWAP(v, u) \\ T_u = 7, \ u \in X \text{ is dropped by } PRESS(v, X) \end{cases} \tag{3.6}$$

where $random(0, I)$ is a random integer in $\{0, \ldots, I\}$. These rules are based on previous studies of the related maximum clique problem [Wu *et al.*, 2012; Jin and Hao, 2015]. The first rule estimates the forbidden period for vertices for side-walk moves with the $SWAP$ operator, which ensures that a dropped vertex will not be reconsidered for at least 10 iterations, and the second rule for the $PRESS$ operator prevents any dropped vertex from being reconsidered for a small number of iterations (seven in this case). Based on experiments, we observed that other values around these tabu tenures obtained similar performance. Thus, we selected the values used in [Wu *et al.*, 2012; Jin and Hao, 2015].

Finally, the tabu list is more useful when the number of candidate vertices for $SWAP$ or $PRESS$ is limited, because a dropped vertex u will have a high probability of being added again to the solution if it is not prohibited. However, if numerous candidate vertices exist (such as in massive graphs), there is little chance of a dropped vertex being re-selected immediately. Thus, the tabu mechanism is more useful for graphs of limited size than massive graphs.

3.2.6 Reducing large (sparse) graphs

Given a graph $G = (V, E)$ and a parameter s, suppose that after tabu search (Algorithm 4.1, line 6), the current best s-plex has cardinality $|C^*|$ (lower bound of the maximum s-plex of G). Clearly, to further improve C^*, considering any vertex in V with a degree smaller than or equal to $|C^*| - s$ would not be beneficial because such a vertex cannot extend C^*. Thus, these vertices can be safely removed from the graph [Abello *et al.*, 2002]. In FD-TS, we explore this strategy using the $Peel(G, s, |C^*|)$ procedure (Algorithm 4.1, line 9), which recursively deletes the vertices (and their incident edges) with a degree less than or equal to $|C^*| - s$ until no such vertex exists. Finally, if the subgraphs obtained after $Peel(G, s, |C^*|)$ have fewer vertices than $|C^*|$, then C^* must be an optimal solution because no better solution can exist.

For very dense graphs, the $Peel$ procedure may not reduce the graph size greatly because the degrees of most vertices will remain larger than $|C^*| - s$. However, this technique is highly effective when it is applied to large sparse graphs such as massive real-world complex networks. As shown in Section 3.4.3, by using the high-quality lower bound $|C^*|$ provided by our tabu search procedure, this pruning technique can effectively reduce large sparse graphs to very small graphs (even the null graph).

We note that the idea of removing unpromising vertices was used previously in a GRASP heuristic for detecting dense subgraphs (quasi-cliques) in massive sparse graphs [Abello *et al.*, 2002], as well as in several exact algorithms for the maximum clique and s-plex problems [Balasundaram *et al.*, 2011; Trukhanov *et al.*, 2013; Verma *et al.*, 2015].

3.3 Implementation and time complexity

To effectively implement FD-TS, we maintain two structures: the vector $deg_C[v]$ (i.e., $deg_C[v] = |N(v) \cup C|, v \in V$) and the set $N(C)$ (i.e., $N(C) = \bigcup_{v \in C} N(v) \setminus C$), which are updated whenever the current solution C changes. Thus, each time a vertex (say u) is added to or removed from C by a move operator, we increase or decrease $deg_C[v]$ by one for each $v \in N(u)$. Since the set $N(C)$ must only contain vertices with $deg_C[v] > 0$, it is also adjusted when $deg_C[v]$ changes.

Next, we discuss the time complexity of the main components of the proposed algorithm. First, we consider the procedure for constructing initial solutions (Section 3.2.4). In each iteration, we need to build the subset M_1 from $N(C)$ and update $deg_C[v]$ and $N(C)$ after adding a vertex, which can be achieved in $O(|N(C)| + \Delta)$ ($\Delta = max_{v \in V}\{|N(v)|\}$).

The efficiency of TS^2 (Section 3.2.5) and PERTURB (Section 3.2.5) is closely related to the method used for building sets M'_1, M'_2, and M'_3 from $N(C)$ from scratch in each iteration. In our implementation, we build the saturated set S (Algorithm 3.2, line 9) from C in a time of $O(|C|)$ at the very beginning of each iteration. Then, for each vertex in $N(C)$ (e.g., u), we count the number of saturated vertices in the set of vertices adjacent to u, i.e., $|S \cap N(u)|$ (the saturated connectivity of u). Obviously, if the saturated connectivity of u is 0, $S \setminus N(u) = \emptyset$; and if the saturated connectivity of u is 1, $|S \setminus N(u)| = 1$. According to the definitions of M_1, M_2, and M_3, once the saturated connectivity of u and $deg_C[u]$ is known, it is trivial to identify u as an element of M'_1, M'_2 or M'_3. Consequently, decomposing $N(C)$ (Algorithm 3.2, line 10) can be achieved in $O(|N(C)| * \Delta)$.

Next, we consider the time complexity when employing the move operators. First, for the *ADD* operator, we only need to update the vector $deg_C[v]$ and set $N(C)$ after reallocating a vertex v, which can be achieved in $O(\Delta)$. Second, to apply *SWAP* with a vertex $v \in N(C)$, we first need to identify the other vertex $u \in C$. According to the rule defined in Section 3.2.5, the sets $S \setminus N(v)$ and $C \setminus N(v)$ can be identified by traversing sets C and $N(v)$ respectively. Thus, the time required to identify u is bounded by $O(|C| + \Delta) = O(2 * \Delta + s) = O(\Delta + s)$ (because $|C| \leq \Delta + s$), while updating $deg_C[v]$ and $N(C)$ is bounded by $O(2 * \Delta) = O(\Delta)$ because two vertices are displaced during each application of *SWAP*. Finally, each application of the $PRESS$ operator can be achieved in $O(\Delta^2)$.

Overall, one operator (ADD, $SWAP$ or $PRESS$) is applied during one iteration, so the total time complexity of TS^2 and PERTURB for each iteration is bounded by $O(|C| + |N(C)| * \Delta + \Delta^2)$. We note that for sparse graphs, $|C|$, $N(C)$, and Δ are usually extremely small compared with the number of vertices in a graph.

3.4 Computational assessment

3.4.1 Benchmarks

In this section, we present computational evaluations of the proposed FD-TS algorithm for MsPlex based on 79 (19, 17, and 43, respectively) instances from benchmark sets SNAP, 10th DIMACS and 2nd DIMACS (see Section 1.4.1, Chapter 1).

3.4.2 Experimental protocol and parameter tuning

The proposed FD-TS algorithm was implemented in C++[1] and compiled by g++ with optimization option '-O3'. All experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8 GHz and 2 GB RAM) running CentOS 6.5. When we solved the DIMACS machine benchmarking program fdmax.c[2] without compilation optimization flag, the run time on our machine was 0.40, 2.50, and 9.55 seconds for graphs r300.5, r400.5, and r500.5, respectively.

An interesting feature of FD-TS is that it has very few parameters. In addition to the tabu tenure discussed in Section 3.2.5, the parameter q (the sample number of vertices in Section 3.2.4) was set to 100. As indicated in Section 3.2.4, q is not a sensitive parameter so we simply fixed it to the middle value in the range of [50,150]. However, according to our experiments, the best value of parameter L, which

[1] We will make our program available.

[2] ftp://dimacs.rutgers.edu/pub/dsj/clique/

specifies the maximum number of iterations in each round of tabu search, was highly dependent on the instance considered. We compared different values in $\{10,\ 100,\ 1000,\ 5000\}$ for L with $s = 2, 3, 4, 5$ for six selected instances from the 2nd DIMACS benchmark set (*MANN_a27, brock400_2, brock800_2, C1000.9, keller6, p_hat1500-3*). As a trade-off, we retained $L = 1000$ because the algorithm achieved the best average solution quality in most cases. We also observed that fine tuning L for each pair (instances, s) could improve the performance. However, to report our computational results, we used the fixed setting $L = 1000$, which allowed the algorithm to obtain highly competitive results. Given the stochastic nature of our algorithm, we ran FD-TS to solve each instance 20 times for each given s, where each run was limited to a maximum of 180 CPU seconds (3 minutes).

3.4.3 Computational results for very large networks from SNAP and the 10th DIMACS Challenge

Table 3.1 shows the performance of FD-TS on 47 instances taken from the SNAP and 10th DIMACS benchmarks, namely, all 37 instances tested in [Trukhanov *et al.*, 2013] for the s-plex problem, as well as 10 instances from the recent literature [Verma *et al.*, 2015]. Note that in [Verma *et al.*, 2015], only the best clique size was reported, which is just a lower bound of maximum s-plex for $s = 2, 3, 4, 5$. To be complete, we also report our results for the remaining 20 instances from [Verma *et al.*, 2015] in the Appendix (Table 5.6).

For each instance, the columns “Instance," “$|V|$," and “$|E|$" indicate basic information for the name, number of vertices, and number of edges, respectively. For each $s = 2, 3, 4, 5$, column “BKV" denotes the best-known objective values collected from [Trukhanov *et al.*, 2013] (for the first 37 instances) and [Verma *et al.*, 2015] (for the last 10 instances). For the items in this column, an additional symbol “*" indicates that this objective value was proved to be optimal in [Trukhanov *et al.*, 2013], and the symbol ω shows that this value is the maximum clique size mentioned in [Verma *et al.*, 2015] (which is a *lower bound* of the maximum s-plex). The “max" column shows the best objective value found by FD-TS among its 20 trials and the “time" column indicates the average time (in seconds) required for runs to obtain the best objective value (excluding the time spent reading the graph). The “$|V'|$" column shows the number of remaining vertices in the reduced subgraph after executing the $Peel(G, s, max)$ procedure (see Section 3.2.6). As mentioned earlier, if the number of vertices in the reduced subgraph (column “$|V'|$") is less than or equal to the lower bound (column “max"), then the latter is guaranteed to be the optimal solution. In these cases, we put a “*" beside the value.

Moreover, for instances where the optimum could not be determined in either [Trukhanov *et al.*, 2013] or FD-TS, we conducted an additional experiment with the CPLEX solver (version 12.6.1). First, we reduced the original graph by applying the $Peel(G, s, max)$ procedure. Then, for the reduced subgraph and a given $s \in \{2, 3, 4, 5\}$, a cutoff time of one CPU hour was used when running CPLEX with the mathematical model (Equation 1.4 – Equation 1.6) presented in Section 3.1. Experiments were conducted using the same machine employed for running FD-TS. The best objective values found by CPLEX are listed in the “cplex" column, where “*" indicates the optimal values. If CPLEX was unable to load the model, the entry is marked by “N/A."

Table 3.1 shows that FD-TS always obtained the same or better objective values compared with the current best-known results (BKV values) (better objective values are highlighted with a bold font). In particular, FD-TS improved the best-known results for more instances as s increased (FD-TS found better solutions for $7, 15, 19, 21$ instances with $s = 2, 3, 4, 5$, respectively). This observation indicates that the instances become more challenging for exact algorithms with a larger s. Using the $Peel$ procedure, FD-TS was also provably optimum for the first time for $6, 6, 6, 10$ instances and $s = 2, 3, 4, 5$, respectively. For the cases where the number of vertices in the reduced subgraph remained larger than the lower bound given by FD-TS, the majority of these cases were manageable when the subgraph had less than 10,000 vertices (exceptions include the instances 333SP, cage15, cit-Pattens, and wiki-Talk for $s = 2$). In terms of the computational time, FD-TS was able to obtain the best solutions in less than one second in most cases, including instances with millions of vertices. The average time required by FD-TS on cit-Patents for $s = 3$ was the longest but still less than one minute. Unfortunately, and similarly to CPLEX, it could not determine any solution better than that found by FD-TS in one hour when given the reduced subgraph. However, for the instances rgg_n_2_17_s0 with $s = 5$, and rgg_n_2_20_s0 with $s = 4$ and 5, optimal solutions were also obtained by CPLEX. Finally, we note that for the instances coPapersCiteseer, coPapersDBLP, and cond-mat-2005, the size of the maximum clique was the same as the size of the maximum s-plex for $s = 2, 3, 4, 5$.

Table 3.1: Computational results of FD-TS on 47 large networks from the SNAP Collection and the 10th DIMACS Implementation Challenge.

			s=2					s=3					s=4					s=5				
Instance	$\|V\|$	$\|E\|$	BKV	max	time	$\|V'\|$	cplex	BKV	max	time	$\|V'\|$	cplex	BKV	max	time	$\|V'\|$	cplex	BKV	max	time	$\|V'\|$	cplex
adjnoun	112	425	6*	6	0.00	102	-	8*	8	0.00	102	-	8*	8	0.00	89	-	10*	10	0.00	102	-
celegans_metabolic	453	2025	10*	10	0.00	313	-	11*	11	0.00	429	-	13*	13	0.00	240	-	14*	14	0.00	429	-
dolphins	62	159	6*	6	0.00	53	-	7*	7	0.00	36	-	7*	7	0.00	45	-	9*	9	0.00	45	-
email	1133	5451	12*	12	0.01	238	-	12*	12	0.02	349	-	12*	12	0.01	349	-	13*	13	0.01	848	-
football	115	613	10*	10	0.01	114	-	11*	11	0.01	115	-	12*	12	0.00	115	-	12*	12	0.00	115	-
jazz	198	2742	30*	30	0.00	130	-	30*	30	0.00	164	-	30*	30	0.00	127	-	30*	30	0.00	127	-
karate	34	78	6*	6	0.00	22	-	6*	6	0.00	10	-	8*	8	0.00	10	-	11^{a}	9*	0.00	0	-
netscience	1589	2742	20*	20	0.00	50	-	20*	20	0.00	137	-	20*	20	0.00	158	-	20	20*	0.00	20	-
polblogs	1490	16715	23*	23	0.02	541	-	27*	27	0.03	894	-	29*	29	0.04	489	-	32	32	0.03	293	32*
polbooks	105	441	7*	7	0.00	103	-	9*	9	0.00	103	-	10*	10	0.00	105	-	11*	11	0.00	105	-
power	4941	6594	6*	6	0.00	36	-	6*	6	0.00	231	-	6	8	0.01	12	8*	8	9	0.01	12	9*
PGPgiantcompo	10680	24316	29*	29	0.03	115	-	31*	31	0.02	43	-	33*	33	0.02	41	-	35*	35	0.02	41	-
as-22july06	22963	48436	19*	19	0.02	117	-	21*	21	0.02	110	-	22*	22	0.04	110	-	24*	24	0.09	104	-
astro-ph	16706	121251	57*	57	0.06	113	-	57*	57	0.05	113	-	57*	57	0.07	113	-	57*	57	0.07	165	-
caidaRouterLevel	192244	609066	20*	20	0.31	2039	-	22	**23**	0.50	1290	13	23	**24**	0.74	1290	12	23	**26**	0.82	1098	14
cnr-2000	325557	2738969	85*	85*	6.75	0	-	86*	86	10.51	0	-	86*	86*	7.66	86	-	86*	86*	8.09	86	-
coAuthorsCiteseer	227320	814134	87*	87*	0.21	87	-	87*	87*	0.21	87	-	87*	87*	0.22	87	-	87*	87*	0.22	87	-
coAuthorsDBLP	299067	977676	115*	115*	0.27	115	-	115*	115*	0.26	115	-	115*	115*	0.27	115	-	115*	115*	0.27	115	-
cond-mat-2005	40421	175691	30*	30*	0.08	30	-	30*	30*	0.07	30	-	30*	30*	0.08	30	-	30*	30	0.08	57	-
memplus	17758	54196	97*	97*	0.08	97	-	97*	97*	0.11	97	-	97*	97*	0.11	97	-	97*	97*	0.10	97	-
rgg_n_2_17_s0	131072	728753	16*	16*	0.15	0	-	17*	17*	0.15	0	-	18*	18*	0.14	0	-	18	18	0.14	34	18*
rgg_n_2_19_s0	524288	3269766	19*	19*	0.69	0	-	19*	19*	0.66	19	-	20*	20*	0.64	19	-	20	**21***	0.66	19	-
rgg_n_2_20_s0	1048576	6891620	18*	18	1.42	59	-	19*	19	1.37	59	-	19	20	1.36	59	20*	19	20	1.33	172	20*
cit-HepPh	34546	420877	24*	24	0.37	4768	-	25	**27**	0.27	3498	5	25	**30**	0.19	2344	11	25	**32**	0.30	1804	22
cit-HepTh	27770	352285	28*	28	1.20	4815	-	31	31	0.70	3999	N/A	31	**34**	0.81	2922	6	31	**37**	0.72	1726	23
email-EuAll	265214	364481	19*	19	0.15	1357	-	22	22	0.22	1198	20	22	**25**	0.22	1055	23	27	27	0.26	988	26
p2p-Gnutella04	10876	39994	5*	5	0.22	6089	-	5	**7**	0.09	5433	4	6	**9**	0.10	4857	5	7	**10**	0.01	4857	N/A
p2p-Gnutella24	26518	65369	5*	5	0.07	9271	-	5	**6**	0.02	9271	N/A	5	**8**	0.10	7480	N/A	5	**9**	0.08	7480	N/A
p2p-Gnutella25	22687	54705	5*	5	0.03	7892	-	5	**6**	0.01	7892	N/A	5	**8**	0.01	6091	5	5	**10***	0.01	0	-
soc-Epinions1	75879	405740	28*	28	0.09	4156	12	28	**32**	0.09	3791	12	28	**37**	0.09	3280	12	28	**39**	0.16	3178	10
soc-Slashdot0811	77360	469180	31*	31	0.05	3665	-	33	**34**	0.06	3188	6	36	**38**	0.12	2596	7	36	**40**	0.10	2416	7
soc-Slashdot0902	82168	504230	32*	32	0.05	3669	-	35	35	0.06	3185	6	36	**40**	0.10	2435	9	36	**42**	0.10	2287	7
web-BerkStan	685230	6649470	202*	202	4.19	392	-	202*	202	4.47	392	-	202*	202	4.21	392	-	202*	202	6.55	392	-
web-Google	875713	4322051	46*	46*	1.13	0	-	47*	47*	1.24	0	-	48*	48*	1.34	0	-	48*	48*	1.29	48	-
web-NotreDame	325729	1090108	155*	155	0.74	1367	-	155*	155	0.71	1367	-	155*	155	0.84	1367	-	155*	155	0.99	1367	-
web-Stanford	281903	1992636	64*	64	3.86	741	-	64*	64	5.61	985	-	64	65	4.77	985	65	64	**66**	6.80	985	65
wiki-Vote	7115	100762	21*	21	0.10	2098	-	24	24	0.11	2005	12	26	**27**	0.12	1925	16	27	**28**	0.13	1925	17
333SP	3712815	11108633	4(ω)	**5**	1.19	2261408	N/A	4(ω)	**6**	0.71	2261408	N/A	4(ω)	**7**	0.51	2261408	N/A	4(ω)	**8**	0.46	2261408	N/A
belgium.osm	1441295	1549970	3(ω)	**5***	0.66	0	-	3(ω)	**5***	0.35	5	-	3(ω)	**6***	0.37	5	-	3(ω)	**7***	0.34	5	-
cage15	5154859	47022346	6(ω)	6	2.48	5135355	N/A	6(ω)	**8**	5.68	5134115	N/A	6(ω)	**10**	6.02	5091619	N/A	6(ω)	**11**	26.71	5091619	N/A
coPapersCiteseer	434102	16036720	845(ω)	845*	7.92	845	-	845(ω)	845*	7.20	845	845	845(ω)	845*	7.01	845	-	845(ω)	845*	8.51	845	-
coPapersDBLP	540486	15245729	337(ω)	337*	2.83	337	-	337(ω)	337*	3.03	337	-	337(ω)	337*	3.18	337	-	337(ω)	337*	3.28	337	-
amazon0312	400727	2349869	11(ω)	**12***	0.54	0	-	11(ω)	**13***	0.58	0	-	11(ω)	**14***	0.61	0	-	11(ω)	**15***	0.60	0	-
amazon0505	410236	2439437	11(ω)	**12***	0.50	0	-	11(ω)	**13***	0.58	0	-	11(ω)	**14***	0.62	0	-	11(ω)	**15***	0.64	0	-
amazon0601	403394	2443408	11(ω)	**12***	0.75	0	-	11(ω)	**13***	0.82	0	-	11(ω)	**14***	0.79	0	-	11(ω)	**15***	0.74	0	-
cit-Patents	3774768	16518947	11(ω)	**17**	21.25	29585	N/A	11(ω)	**21**	51.02	14717	N/A	11(ω)	**26**	16.23	6293	N/A	11(ω)	**31**	11.09	3604	5
wiki-Talk	2394385	4659565	26(ω)	**32**	1.18	11327	N/A	26(ω)	**36**	3.60	9814	N/A	26(ω)	**41**	1.51	8376	N/A	26(ω)	**44**	3.72	7719	N/A

Note a: The value of 11 reported in [Trukhanov *et al.*, 2013] for 'karate' is wrongly claimed to be the optimal solution.

3.4.4 Computation results for graphs from the 2nd DIMACS Challenge

Table 3.2 shows the computational results obtained by FD-TS for 52 classical hard instances from the 2nd DIMACS set, with $s = 2, 3, 4, 5$. The first group contains *all* 36 instances that have been studied by four state-of-the-art s-plex algorithms [Balasundaram *et al.*, 2011; McClosky and Hicks, 2012; Moser *et al.*, 2012; Trukhanov *et al.*, 2013]. The second group includes 16 additional *large* 2nd DIMACS instances with at least 800 vertices, which have not been tested previously by any existing s-plex algorithm. For the sake of completeness, we also tested the remaining 28 instances of this benchmark set, for which no previous s-plex result is available. These results are reported in Table 5.7 of the Appendix.

For each instance, the following information is included. The "$|V|$" column shows the number of vertices in the original graph. The "ω" column indicates the best-known maximum clique size reported previously [Wu and Hao, 2015a] (lower bounds for the maximum s-plex). The "BKV" column indicates the best-known objective values obtained by algorithms in [Balasundaram *et al.*, 2011; McClosky and Hicks, 2012; Moser *et al.*, 2012; Trukhanov *et al.*, 2013] (proven optima are indicated by "*"). The letters between parentheses following each "BKV" value indicate the algorithm(s) that obtained the BKV value.

- "B" - A branch-and-cut algorithm [Balasundaram *et al.*, 2011] based on polyhedral analysis of the convex hull of MsPlex. This algorithm was evaluated based on instances from the 2nd DIMACS set with $s = 1, 2$. Each instance was solved within a maximum of 3 hours on a machine with a 2.66 GHz XEON® processor, 3 GB RAM, and 120 GB HDD.
- "M" - A branch-and-bound algorithm [McClosky and Hicks, 2012] adapted from the classical maximum clique algorithm [Östergård, 2002]. Results were obtained based on 2nd DIMACS instances with $s = 2, 3, 4$. The experiments were conducted on a machine with a 2.2 GHz Dual-Core AMD Opteron processor and 3 GB RAM. A time limit of one hour was allowed to solve each instance.
- "T" - A generalized algorithm framework used to detect optimal hereditary structures in graphs [Trukhanov *et al.*, 2013]. For MsPlex, this approach was tested based on instances from the 2nd DIMACS, 10th DIMACS and SNAP benchmark sets, for $s = 2, 3, 4, 5$. Experiments were conducted with a Dell Optiplex GX620 computer with an Intel Core™2 Quad 3 GHz processor and 4 GB RAM with a time limit of 3 hours for each instance.
- "H" - Exact combinatorial algorithms based on methods from parameterized algorithmics [Moser *et al.*, 2012]. Results were reported for a subset of the 2nd DIMACS instances ($s = 1, 2$) with a time limit of 3 hours on a machine with an AMD Athlon 64 3700+ 2.2 GHz CPU, 3 GB RAM, and 1M L2 cache.

For instances where the optimal solution has not been proven by any of the algorithms mentioned above (i.e., no "*" is indicated for "BKV"), we used the CPLEX solver to solve these instances (i.e., their reduced subgraphs after applying the $Peel$ procedure, see Section 3.2.6) with a time limit of one CPU hour on our computer. The "cplex" column shows the best feasible solutions attained by CPLEX. The "max(ave)" column reports the maximum value achieved by FD-TS in 20 runs and the average value (in parentheses) if the 20 best values were not the same. The "time" column shows the average time (in seconds) required by the runs that obtained the best value among the 20 runs. Obviously, the total time allowed to FD-TS in 20 runs (180*20 = 3600 s) was exactly one hour.

Table 3.2 shows that the FD-TS algorithm matched or improved (highlighted in bold font) the current best-known results with $s = 2, 3, 4, 5$. The average objective values obtained by our algorithm were even better than the best-known values based on these instances for different s (except for MANN_a27 and MANN_a45 with $s = 2$). In terms of the stability of the best solution, for most of the small instances ($|V| \leq 400$) in the first group, the best solution could be obtained in each of the 20 runs (except for MANN_a27, brock400_4 and san200_0.7_2 with $s = 2$, brock400_1 with $s = 5$). The larger graphs in the second group of the table (which were not reported previously), such as brock800_X, CXXX.X, hamming10-4, and keller6, represent the most challenging cases for FD-TS because the best solution could not be found in every run. Moreover, for instances such as brock400_1, brock800_2, brock800_3, keller6, and p_hat1500-2, FD-TS failed to achieve a 100% success rate as s increased. However, for instances such as MANN_a27 and brock800_4, there was no correlation between the success rate and the value of s. In terms of the computational time, FD-TS achieved its best values rather quickly, since it rarely exceeded one minute, whereas CPLEX failed to solve these instances (in fact, the reduced subgraphs) to optimality for any s within one hour. Nevertheless, for the cases where the optimal value is still unknown, the lower bounds obtained by CPLEX were competitive compared with the four other reference algorithms [Balasundaram *et al.*, 2011; McClosky and Hicks, 2012; Moser *et al.*, 2012; Trukhanov *et al.*, 2013].

Table 3.2: Computational results of FD-TS on 52 benchmark instances of the 2nd DIMACS Implementation Challenge

			s=2				s=3				s=4				s=5			
instance	$\|V\|$	ω	BKV	cplex	max(ave)	time	BKV	cplex	max(ave)	time	BKV	cplex	max(ave)	time	BKV	cplex	max(ave)	time
MANN_a9	45	16	26*(BMTH)	-	26	0.00	36*(MT)	-	36	0.00	36*(MT)	-	36	0.00	44(T)^{a}	**45**	**45**	0.00
MANN_a27	378	126	236*(H)	-	236(235.90)	12.64	351*(T)	-	351	0.41	351(M)	351	351	0.45	351(T)	351	351	0.45
MANN_a45	1035	345	662(BH)	662*	662(661.40)	5.46	990(M)	990*	990	7.40	990(M)	990*	990	7.48	990(M)	990*	990	7.44
brock200_1	200	21	25(B)	26	26	0.15	24(M)	29	**30**	0.05	27(M)	33	**35**	3.59	27(T)	38	**39**	2.36
brock200_2	200	12	13*(MTH)	-	13	0.01	16*(T)	-	16	0.27	17(TM)	17	**18**	0.06	17(TM)	20	20	0.04
brock200_3	200	15	17*(T)	-	17	0.02	19(T)	19	**20**	0.02	19(T)	22	**23**	0.03	19(T)	25	**26**	0.09
brock200_4	200	17	20*(TH)	-	20	0.06	20(T)	22	**23**	0.07	21(M)	26	26	0.03	21(M)	28	**30**	0.20
brock400_1	400	27	23(T)	29	**30**	0.42	23(T)	34	**36**	7.24	23(T)	37	**41**	41.37	23(T)	43	**46**(45.50)	36.57
brock400_2	400	29	27(B)	28	**30**	0.51	27(M)	33	**36**	15.37	29(M)	37	**41**	33.51	29(M)	41	**45**	2.14
brock400_4	400	33	27(B)	28	**33**(31.20)	64.18	27(B)	34	**36**	4.60	30(M)	36	**41**	1.76	30(M)	44	**46**	25.67
c-fat200-1	200	12	12*(BMTH)	-	12	0.00	12*(MT)	-	12	0.00	12*(MT)	-	12	0.00	14*(T)	-	14	0.00
c-fat200-2	200	24	24*(BMTH)	-	24	0.01	24*(MT)	-	24	0.01	24*(MT)	-	24	0.02	24*(T)	-	24	0.01
c-fat200-5	200	58	58*(BMTH)	-	58	0.01	58*(MT)	-	58	0.01	58*(MT)	-	58	0.00	58*(T)	-	58	0.00
c-fat500-1	500	14	14*(BMTH)	-	14	0.00	14*(MT)	-	14	0.00	14*(MT)	-	14	0.00	15*(T)	-	15	0.00
c-fat500-2	500	26	26*(BMTH)	-	26	0.00	26*(MT)	-	26	0.00	26*(MT)	-	26	0.00	26*(T)	-	26	0.00
c-fat500-5	500	64	64*(BMTH)	-	64	0.02	64*(MT)	-	64	0.02	64*(MT)	-	64	0.03	64*(T)	-	64	0.04
c-fat500-10	500	126	126*(BMTH)	-	126	0.08	126*(MT)	-	126	0.22	126*(MT)	-	126	0.23	126*(T)	-	126	0.16
hamming6-2	64	32	32*(BMTH)	-	32	0.00	32*(MT)	-	32	0.00	40*(MT)	-	40	0.00	48*(T)	-	48	0.00
hamming6-4	64	4	6*(BMTH)	-	6	0.00	8*(MT)	-	8	0.00	10*(MT)	-	10	0.00	12*(T)	-	12	0.00
hamming8-2	256	128	128*(BMT)	-	128	0.09	128*(MT)	-	128	0.09	128(MT)	128	**129**	22.41	128(MT)	152	152	0.97
hamming8-4	256	16	16*(BMT)	-	16	0.01	20(T)	20	20	0.01	20(T)	24	25	0.28	20(T)	32	32	0.02
hamming10-2	1024	512	512*(M)	-	512	8.97	512(M)	512	512	4.66	512(M)	512	512	8.62	512(M)	512	**513**(512.15)	16.82
johnson8-2-4	28	4	5*(BMTH)	-	5	0.00	8*(MT)	-	8	0.00	9*(MT)	-	9	0.00	12*(T)	-	12	0.00
johnson8-4-4	70	14	14*(BMTH)	-	14	0.00	18*(TH)	-	18	0.00	22*(T)	-	22	0.00	24(T)	24	**28**	0.00
johnson16-2-4	120	8	10*(T)	-	10	0.00	16(T)	16	16	0.00	19(T)	19	19	0.00	21(T)	24	**24**	0.00
keller4	171	11	15*(BMTH)	-	15	0.00	21*(T)	-	21	0.09	22(T)	23	23	0.06	22(T)	28	28	0.02
p_hat300-1	300	8	10*(MTH)	-	10	0.00	12*(MT)	-	12	0.00	14*(T)	-	14	0.01	14(T)	15	**16**	0.02
p_hat300-2	300	25	30*(T)	-	30	0.01	30(T)	36	36	0.02	33(M)	41	41	0.06	33(M)	46	46	0.03
p_hat300-3	300	36	43(B)	43	**44**	0.07	43(B)	52	52	0.06	43(B)	59	59	0.09	43(B)	65	65	0.11
p_hat500-1	500	9	12*(T)	-	12	0.06	14*(T)	-	14	0.20	14(T)	16	16	0.11	14(T)	17	**18**	0.15
p_hat700-1	700	11	13*(M)	-	13	0.06	13(M)	14	**15**	0.13	13(M)	15	**17**	0.45	13(M)	18	**19**	2.00
p_hat700-2	700	44	50(B)	51	**52**	0.06	50(B)	60	**62**	1.35	50(B)	68	**70**	0.26	50(B)	75	**79**	9.42
p_hat700-3	700	62	73(B)	75	**76**	0.54	73(B)	88	**89**	2.13	73(B)	97	**100**	1.46	73(B)	106	**109**	1.35
san200_0.7_2	200	18	24(M)	26	26(25.40)	9.74	36(M)	37	37	0.21	48(M)	49	49	1.90	48(M)	60	60	0.02
san200_0.9_1	200	70	90(M)	90*	90	0.01	125*(M)	-	125	0.02	125(M)	125*	125	0.03	125(M)	125*	125	0.03
hamming10-4	1024	40	41(BM)	44	**48**	1.53	46(M)	53	**64**	1.13	51(M)	64	**68**(67.20)	20.64	51(M)	73	**79**(78.05)	34.37
brock800_1	800	23	-	21	**25**	10.90	-	26	**29**	12.35	-	29	**34**(33.20)	24.43	-	33	**37**	27.12
brock800_2	800	24	-	22	**25**	11.36	-	27	**30**(29.30)	31.20	-	30	**34**(33.15)	26.40	-	33	**38**(37.15)	32.46
brock800_3	800	25	-	24	25	12.68	-	25	**30**(29.20)	11.71	-	30	**34**(33.15)	28.46	-	32	**38**(37.10)	36.78
brock800_4	800	26	-	22	26(25.55)	56.21	-	26	**29**	14.35	-	31	**33**	32.04	-	33	**37**	33.58
C1000.9	1000	68	-	71	**81**(80.55)	39.65	-	84	**95**(93.75)	58.59	-	97	**107**(106.00)	48.91	-	108	**119**(118.15)	60.39
C2000.5	2000	16	-	14	**19**(18.95)	27.96	-	18	**22**(21.90)	62.35	-	21	**25**(24.50)	22.79	-	23	**28**(27.15)	11.72
C2000.9	2000	80	-	76	**90**(88.90)	65.21	-	88	**105**(103.40)	69.41	-	94	**118**(116.80)	63.85	-	107	**132**(129.65)	76.66
C4000.5	4000	18	-	12	**20**	39.23	-	14	**23**	69.37	-	13	**26**(25.55)	53.64	-	16	**29**(28.20)	26.93
keller6	3361	59	-	52	**63**	3.60	-	65	**90**(87.80)	66.21	-	77	**107**(103.45)	67.88	-	90	**125**(123.20)	73.91
p_hat1000-1	1000	10	-	11	**13**	0.28	-	13	**15**	0.15	-	15	**18**	4.09	-	18	**20**	7.17
p_hat1000-2	1000	46	-	52	**56**	0.43	-	64	**67**	0.81	-	71	**76**	28.89	-	82	**84**	1.30
p_hat1000-3	1000	48	-	77	**82**	0.33	-	95	**98**	2.19	-	109	**111**	3.50	-	117	**122**	29.84
p_hat1500-1	1500	12	-	13	**14**	1.46	-	14	**17**	22.47	-	16	**19**	3.73	-	16	**21**	1.43
p_hat1500-2	1500	65	-	71	**80**	1.75	-	90	**93**	0.41	-	99	**107**(106.50)	29.46	-	113	**117**(116.55)	41.90
p_hat1500-3	1500	94	-	108	**114**	0.61	-	125	**133**	17.85	-	141	**150**	3.90	-	154	**164**	48.76
san1000	1000	15	-	17	17	9.39	-	25	25	6.73	-	33	33	1.70	-	41	41	6.39

Note α: The value of 44 reported in [Trukhanov *et al.*, 2013] for MANN_a9 is wrongly claimed to the optimal solution.

3.4.5 Impact of frequency information

As described in Sections 3.2.4 and 3.2.5, the construction procedure and perturbation procedure are guided by frequency information. In this section, we describe our evaluation of the effectiveness of this frequency strategy. We compared the original FD-TS algorithm with a variant, FD-TS-R, in which the frequency-based vertex selection rule was replaced by a random selection rule. In particular, to create a new solution, FD-TS-R randomly adds a vertex from M_1 to the current solution (Section 3.2.4) and randomly selects a vertex from M_3' for perturbation (Algorithm 3.2, line 31).

To better differentiate FD-TS and FD-TS-R, we selected 27 instances from the three benchmark sets, such that the selected instances cover different characteristics (random vs real-world, dense vs sparse) and are sufficiently challenging based on the search effort required to attain the best solutions according to the results of Tables 3.1 and 3.2. For this experiment, both FD-TS and FD-TS-R were run 20 times to solve each instance, each run being limited to 20 seconds for the 2nd DIMACS instances and 180 seconds for the other (larger) instances. We compared the average objective values reached by both algorithms (“ave" columns), the average time required to first obtain the best objective value (“time" columns), and the improvement in the average objective value achieved by FD-TS as a percentage (“ave_imp" column).

Table 3.3 shows the results achieved for $s = 2, 3, 4, 5$ respectively. A difference in the average solution quality obtained by the two algorithms was only observed with the 2nd DIMACS instances (the first 12 instances). For the large instances, both FD-TS and FD-TS-R converged so fast that the best solution was found quite early (the average time required to first obtain the best solution was less than one second in most cases). For the 2nd DIMACS instances, FD-TS achieved better solutions than FD-TS-R for $5, 6, 8, 8$ instances with $s = 2, 3, 4, 5$, respectively (marked in bold font). In addition, for $4, 4, 4, 2$ instances with $s = 2, 3, 4, 5$, respectively, the average objective values found by FD-TS were worse than those found by FD-TS-R (marked in italic font). In general, there was a slight advantage when using the frequency mechanism, and it increased with s. This experiment confirms that the frequency mechanism is helpful for solving hard dense graphs that require persistent search efforts.

Table 3.3: Impact of frequency information - comparison between FD-TS and FD-TS-R.

instance	s=2					s=3					s=4					s=5				
	FD-TS-R		FD-TS		ave_imp	FD-TS-R		FD-TS		ave_imp	FD-TS-R		FD-TS		ave_imp	FD-TS-R		FD-TS		ave_imp
	ave	time	ave	time		ave	time	ave	time		ave	time	ave	time		ave	time	ave	time	
C1000.9	79.40	8.59	79.70	7.45	**0.38%**	92.70	7.95	92.80	7.89	**0.11%**	104.95	5.18	105.10	6.68	**0.14%**	116.85	10.40	117.30	9.88	**0.38%**
C2000.5	18.45	3.25	18.50	5.05	**0.27%**	21.05	1.29	21.10	2.71	**0.24%**	24.15	3.35	24.10	4.06	*-0.21%*	26.85	6.27	26.95	9.35	**0.37%**
C2000.9	87.80	6.97	87.85	9.95	**0.06%**	102.20	9.50	101.70	7.97	*-0.49%*	115.05	8.65	114.85	9.61	*-0.17%*	128.25	9.76	127.70	8.95	*-0.43%*
C4000.5	19.60	4.80	19.25	2.75	*-1.82%*	22.35	3.96	22.25	3.86	*-0.45%*	25.05	3.99	25.20	4.18	**0.60%**	27.90	4.81	27.90	6.42	0.0%
brock800_1	24.85	4.98	24.70	4.30	*-0.61%*	28.90	7.04	28.95	5.38	**0.17%**	32.70	4.57	33.15	5.91	**1.36%**	36.40	3.93	36.50	4.00	**0.27%**
brock800_2	24.85	6.20	24.90	7.63	**0.20%**	29.00	7.62	28.95	5.08	*-0.17%*	32.80	5.74	32.75	5.48	*-0.15%*	36.45	5.56	36.65	6.22	**0.55%**
brock800_3	24.85	8.74	24.80	5.31	*-0.20%*	28.75	5.06	29.00	7.19	**0.86%**	32.80	7.74	32.95	6.84	**0.46%**	36.30	3.60	36.45	5.27	**0.41%**
brock800_4	24.70	6.84	24.85	7.28	**0.60%**	28.65	5.76	28.70	5.99	**0.17%**	32.40	5.43	32.55	5.49	**0.46%**	36.25	4.71	36.30	3.07	**0.14%**
hamming10-4	48.00	1.36	48.00	1.16	0.0%	64.00	0.98	64.00	1.12	0.0%	66.75	5.73	66.85	3.98	**0.15%**	77.50	3.92	77.45	3.05	*-0.06%*
keller6	63.00	2.98	63.00	3.33	0.0%	84.70	8.68	85.70	8.46	**1.17%**	99.50	9.00	98.95	11.69	*-0.56%*	121.00	9.89	121.40	14.53	**0.33%**
p_hat1500-2	80.00	1.66	80.00	2.02	0.0%	93.00	0.47	93.00	0.48	0.0%	106.00	1.40	106.05	1.50	**0.05%**	116.00	1.49	116.10	2.78	**0.09%**
san1000	16.80	4.73	16.75	2.50	*-0.30%*	25.00	3.30	24.95	2.46	*-0.20%*	32.95	3.35	33.00	1.29	**0.15%**	41.00	3.59	41.00	3.82	0.0%
333SP	5.00	1.21	5.00	1.19	0.0%	6.00	0.71	6.00	1.04	0.0%	7.00	1.03	7.00	0.51	0.0%	8.00	0.96	8.00	0.46	0.0%
cage15	6.00	2.41	6.00	2.48	0.0%	8.00	8.89	8.00	5.68	0.0%	10.00	5.84	10.00	6.02	0.0%	11.00	34.77	11.00	26.71	0.0%
caidaRouterLevel	20.00	0.55	20.00	0.31	0.0%	23.00	0.94	23.00	0.51	0.0%	24.00	0.81	24.00	0.74	0.0%	26.00	0.52	26.00	0.82	0.0%
cit-HepPh	24.00	0.32	24.00	0.37	0.0%	27.00	0.46	27.00	0.27	0.0%	30.00	0.37	30.00	0.19	0.0%	32.00	0.34	32.00	0.30	0.0%
cit-HepTh	28.00	1.31	28.00	1.20	0.0%	31.00	1.51	31.00	0.70	0.0%	34.00	1.02	34.00	0.81	0.0%	37.00	1.39	37.00	0.72	0.0%
cit-Patents	17.00	15.56	17.00	21.25	0.0%	21.00	18.90	21.00	51.02	0.0%	26.00	9.55	26.00	16.23	0.0%	31.00	10.13	31.00	11.09	0.0%
email-EuAll	19.00	0.13	19.00	0.15	0.0%	22.00	0.19	22.00	0.22	0.0%	25.00	0.17	25.00	0.22	0.0%	27.00	0.15	27.00	0.26	0.0%
p2p-Gnutella04	5.00	0.02	5.00	0.02	0.0%	7.00	0.08	7.00	0.09	0.0%	9.00	0.12	9.00	0.10	0.0%	10.00	0.00	10.00	0.01	0.0%
p2p-Gnutella24	5.00	0.04	5.00	0.07	0.0%	6.00	0.01	6.00	0.02	0.0%	8.00	0.12	8.00	0.10	0.0%	9.00	0.32	9.00	0.18	0.0%
p2p-Gnutella25	5.00	0.03	5.00	0.03	0.0%	6.00	0.01	6.00	0.01	0.0%	8.00	0.01	8.00	0.01	0.0%	10.00	0.08	10.00	0.10	0.0%
soc-Slashdot0811	31.00	0.06	31.00	0.05	0.0%	34.00	0.06	34.00	0.06	0.0%	38.00	0.13	38.00	0.12	0.0%	40.00	0.08	40.00	0.10	0.0%
soc-Slashdot0902	32.00	0.06	32.00	0.05	0.0%	35.00	0.07	35.00	0.06	0.0%	40.00	0.11	40.00	0.10	0.0%	42.00	0.12	42.00	0.10	0.0%
web-NotreDame	155.00	1.21	155.00	0.74	0.0%	155.00	1.30	155.00	0.71	0.0%	155.00	1.22	155.00	0.84	0.0%	155.00	1.18	155.00	0.99	0.0%
wiki-Talk	32.00	1.40	32.00	1.18	0.0%	36.00	3.41	36.00	3.60	0.0%	41.00	2.68	41.00	1.51	0.0%	44.00	3.82	44.00	3.72	0.0%
wiki-Vote	21.00	0.07	21.00	0.10	0.0%	24.00	0.13	24.00	0.11	0.0%	27.00	0.16	27.00	0.12	0.0%	28.00	0.08	28.00	0.13	0.0%

3.5 Conclusions

In this chapter, we proposed FD-TS, an effective local search algorithm for MsPlex. FD-TS combines a multi-neighborhood search procedure with vertex-moving frequency based perturbation. Three move operators ADD, $SWAP$ and $PRESS$ were integrated into the search process. FD-TS collects the vertex move frequency to guide the construction of the starting solutions and the perturbation process. A dynamic graph peeling technique is employed to reduce the large graphs.

To verify the effectiveness of FD-TS, we carried out experiments using SNAP, 10th DIMACS, and 2nd benchmark instances. FD-TS has set new records on numerous instances for these benchmark instances, or even found the optimal solutions due to the integration of graph peeling techniques. Comparisons with recent algorithms including CPLEX indicated that FD-TS was quite competitive for MsPlex. Additional results for 48 additional graphs from the above benchmark sets shown in Appendix 5.6 further demonstrated the performance of the proposed algorithm. Furthermore, we justified the frequency strategy by computational comparison between FD-TS and a variant without this strategy.

In the next chapter, we will consider the maximum balanced biclque problem, which is a special case of maximum clique problem in the bipartite graph. We propose both heuristic and exact algorithms for its solution.

4

Heuristic and Exact algorithms for the maximum balanced bipartite clique problem

The Maximum Balanced Biclique Problem is a well-known graph model with relevant applications in diverse domains. This chapter investigates both a new heuristic and some improvements of exact algorithms for MBBP. We design descent techniques combining graph reduction and upper bound propagation to tackle MBBP for very large sparse graphs. For the purpose of readability, we introduce the two kinds of algorithms separately.

In the first part, we introduce a novel heuristic algorithm, which combines an effective constraint-based tabu search procedure and two dedicated graph reduction techniques. We verify the effectiveness of the algorithm on 30 classical random benchmark graphs and 25 very large real-life sparse graphs from the popular Koblenz Network Collection (KONECT). The results show that the algorithm improves the best-known results (new lower bounds) for 10 classical benchmarks and obtains the optimal solutions for 14 KONECT instances.

In the second part, we propose novel ideas for designing effective exact algorithms for MBBP. Firstly, we present an Upper Bound Propagation (UBP) procedure to pre-compute an upper bound involving each vertex. Then we extend an existing branch-and-bound algorithm by integrating the pre-computed upper bounds. We also introduce a set of new valid inequalities induced from the upper bounds to tighten an existing mathematical formulation for MBBP. Lastly, we investigate another exact algorithm scheme which enumerates a subset of balanced bicliques based on our upper bounds. Experiments show that compared with existing approaches, the proposed algorithms and formulations are more efficient in solving hard random graphs and large real-life MBBP instances

The content of this chapter is based on two articles submitted to *Expert Systems with Applications* and *European Journal of Operational Research* respectively.

Contents

4.1 Introduction

Given a bipartite graph $G = (U, V, E)$ with two disjoint vertex sets U, V and an edge set $E \subseteq \{\{u, v\} : u \in U, v \in V\}$, a biclique $(X, Y) = X \cup Y$ is the union of two subsets of vertices $X \subseteq U$, $Y \subseteq V$ such that $u \in X, v \in Y$ implies that $\{u, v\} \in E$. In other words, the subgraph induced by the set of vertices $X \cup Y$ is a complete bipartite graph. If $|X| = |Y|$, then (X, Y) is called a balanced biclique. The Maximum Balanced Biclique Problem (MBBP) is to find a balanced biclique (X^*, Y^*) of maximum cardinality of G, (X^*, Y^*) being the maximum balanced biclique of size $|X^*|$ (or $|Y^*|$) [Garey and Johnson, 1979].

As shown in [Garey and Johnson, 1979; Alon *et al.*, 1994], the decision version of MBBP is NP-complete in the general case, even though the maximum biclique problem without the balance constraint (Eq. 4.5) is polynomially solvable by the maximum matching algorithm [Cheng and Church, 2000].

MBBP is a very important model with numerous applications in nanoelectronic system design, computational biology, VLSI design (see Chapter 1 for more details). Given the significance of MBBP as a NP-hard problem and its relevance in practice, a number of methods, including approximate, heuristic and exact algorithms have been proposed and investigated in the literature. For example, in [Feige and Kogan, 2004], the relations between the approximate hardness of MBBP and 3-SAT as well as the maximum clique problem were established. In [Mubayi and Turán, 2010], despite the NP-hardness of MBBP, a polynomial algorithm was given to find a balanced biclique with size $\lfloor \frac{\ln n}{\ln (2en^2/m)} \rfloor$ (the cardinality of $|X|$ or $|Y|$) for a graph with n vertices and m edges.

To cope with the computational challenge of MBBP, heuristic methods constitute a prominent approach. From an algorithmic point of view, rather than directly seeking the maximum balanced biclique in the given graph, the majority of existing heuristic algorithms solved the equivalent maximum balanced independent set problem for the bipartite complement. For example, in 2006, a greedy heuristic algorithm based on vertex-deletion was proposed in [Tahoori, 2006], which iteratively removes vertices with maximum degree from the bipartite complement until the set of remaining vertices forms an independent set (i.e., a set of vertices such that no edge exists between any pair of vertices in that set). In 2007, an improved greedy heuristic was presented in [Al-Yamani *et al.*, 2007], in which the vertex connecting the maximum number of vertices of minimum degree is removed. In 2011, another greedy heuristic algorithm was introduced in [Yuan and Li, 2011], which iteratively deletes vertices adjacent to the maximum number of vertices in a restricted set. Then in 2014, this algorithm was accelerated by removing multiple vertices at each iteration [Yuan and Li, 2014]. Recently in 2015, a powerful (and rather complex) evolutionary algorithm combining structure mutation and repair-assisted restart was proposed in [Yuan *et al.*, 2015]. The computational results showed that this algorithm performed very well on random dense graphs, which represent one type of the most challenging instances for MBBP. We will use this algorithm as one of the main references for our comparative studies.

Still, according to our literature review, there exists few exact algorithms which guarantee the optimality of solutions. In [Tahoori, 2006], a recursive exact algorithm for searching a maximum balanced independent set with a given half-size in the complement graph was proposed. However, the computational time of this algorithm becomes prohibitive when the number of vertices of the given graph exceeds 64 ($|U| = 32, |V| = 32$). In [McCreesh and Prosser, 2014], a branch-and-bound (B&B) algorithm for MBBP for general graphs (including non-bipartite graphs) was studied. The algorithm incorporates a clique cover technique for upper bound estimation (an equivalent technique of using graph coloring to estimate the upper bound for the maximum clique problem) and employs lex symmetry breaking techniques for general graphs. As far as we know, this algorithm is currently the best performing exact algorithm, even though the bounding and symmetry breaking techniques are only effective for non-bipartite graphs.

In addition to specifically designed exact algorithms, the general Mixed Integer Programming (MIP) constitutes an interesting alternative for addressing hard combinatorial problems such as MBBP. Commercial MIP solvers, like IBM CPLEX, can even solve some hard instances which cannot be handled by other approaches. Meanwhile, the success of a MIP solver highly depends on the tightness of the mathematical formulation of the problem. For MBBP, a MIP formulation based on the complement graph has been proposed in [Dawande *et al.*, 2001]. Another mathematical formulation which defines the constraints on the original graph was presented in [Yuan *et al.*, 2015]. However, this formulation was not applicable for MIP solvers as it contains non-linear constraints.

The remaining of this chapter is organized as follows. We first introduce the heuristic algorithm TSGR-MBBP in Section 4.2. Sections 4.2.1 and 4.2.2 provide some useful notations and rationales of TSGR-MBBP. Section 4.2.3 introduces the proposed algorithm TSGR-MBBP. Computational results on benchmark instances are presented in Section 4.2.4. Section 4.2.5 shows an analysis of the key components of the proposed algorithm.

In Section 4.3, we present the exact algorithms. Section 4.3.1 introduces the notations that will be used for elaborating the exact algorithms and Section 4.3.2 reviews BBClq introduced in [McCreesh and Prosser, 2014]. In Section 4.3.3, we present our Upper Bound Propagation procedure for upper bound estimation and explain how to use it to improve BBClq. Then, in Section 4.3.5, we show how the upper bounds can lead to new valid inequalities to tighten the MIP formulation of [Dawande *et al.*, 2001]. Furthermore, we introduce the novel ExtUniBBClq algorithm in Section 4.3.6. Computational results and experimental analyses are then presented in Section 4.3.7, followed by conclusions of this chapter.

4.2 Heurisitc algorithm with graph reduction

Graphs from real-life applications like social networks and biological networks are usually very large with millions even billions of vertices, rendering most existing approaches unpractical. In this study, we aim to fill the gap by developing improved methods for MBBP, which should be able to handle both random dense graphs and very large real-life networks. For this purpose, we introduce a new algorithm named tabu search with graph reduction for MBBP (TSGR-MBBP), which combines an effective Constraint-Balanced Tabu Search (CBTS) and two dedicated graph reduction techniques. We identify the main contributions of this study as follows.

1. From an algorithmic perspective, the proposed TSGR-MBBP algorithm seeks maximum balanced bicliques directly on the given graph. Compared to the existing approaches which search for balanced independent sets on the complement, operating on the given graph has an advantage of requiring less memory for large sparse graphs. More importantly, TSGR-MBBP employs the Constraint-Balanced Tabu Search (CBTS) algorithm to effectively explore the search space and two bound-based dedicated reduction techniques to shrink progressively the given graph. This is the first study combining local optimization and graph reduction within the iterated search framework for MBBP.
2. We demonstrate the effectiveness of the proposed algorithm on two sets of 55 MBBP benchmark instances. For the set of 30 random challenging instances, the algorithm dominates state-of-the-art algorithms including the current best-performing algorithm presented in [Yuan *et al.*, 2015] and the powerful mixed integer programming solver CPLEX. The algorithm also obtains 10 improved best solutions (i.e., new lower bounds) and matches the best-known results for the remaining 20 instances. For the 25 very large real-life instances from the well-known Koblenz Network Collection, the algorithm proves, for the first time, the optimal solutions for 14 instances (by obtaining the same upper and lower bounds) and obtains tight lower bounds (better than those of CPLEX) for the remaining instances. We also show an analysis of key components (CBTS and the reduction methods) to get insight of their usefulness.

4.2.1 Preliminary definitions

Let $G = (U, V, E)$ be a bipartite graph, we introduce the following notations and definitions which are needed for the description of the proposed approach.

- Given a vertex $v \in U \cup V$, $N(v)$ denotes the set of vertices adjacent to v, i.e., $N(v) = \{u : \{v, u\} \in E\}$. Clearly, if $v \in U$, then $N(v) \subseteq V$, otherwise, $N(v) \subseteq U$.

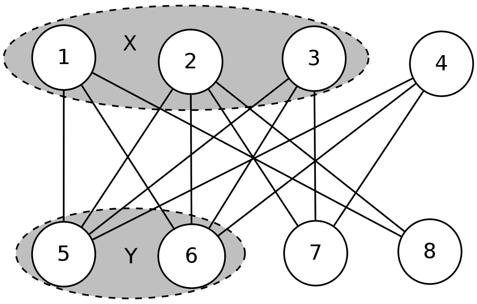

Figure 4.1: $X = \{1, 2, 3\}$, $Y = \{5, 6\}$, $N(X \cup Y) = \{4, 7, 8\}$, (X, Y) is a biclique of balanced size 2. The balance deviation of (X, Y) is 1.

- Given $S \subseteq U \cup V$, $N(S)$ denotes the subset of vertices from $(U \cup V) \setminus S$ that are adjacent to at least one vertex in S, i.e., $N(S) = (\bigcup_{i \in S} N(i)) \setminus S$.
- Given $X \subseteq U$, $Y \subseteq V$, $G[X \cup Y] = (X, Y, E(X \cup Y))$ denotes the subgraph induced by $X \cup Y$. If $G[X \cup Y]$ is a complete bipartite graph, i.e., $E(X, Y) = X \times Y$, then $X \cup Y$ is a biclique, which is also denoted by (X, Y).
- Given a biclique (X, Y), the balanced size of (X, Y) is $min(|X|, |Y|)$, and the balance deviation is $||X| - |Y||$. If the balance deviation is 0, (X, Y) is a balanced biclique of size $|X|$ (or $|Y|$).

Figure 4.1 illustrates the above definitions with a bipartie graph composed of 8 vertices and 13 edges.

Let $\Omega(G)$ denote the search space composed of all balanced bicliques in G, Ω^k be the relaxed search space including all bicliques with a balance deviation no more than k ($k \geq 0$), i.e., $\Omega^k = \{(X, Y) : X \subseteq U, Y \subseteq V, E(X, Y) = X \times Y, ||X| - |Y|| \leq k\}$, then, as explained in the next section, our algorithm explores bicliques in the (slightly) relaxed search space Ω^2 (i.e., with a balance deviation limited to 2) rather than the search space of strictly balanced bicliques $\Omega(G)$ (i.e., Ω^0).

Finally, the quality of a biclique (X, Y) in Ω^k is measured by its balanced size $min(|X|, |Y|)$. Given two bicliques (X_1, Y_1) and (X_2, Y_2), (X_1, Y_1) is better than (X_2, Y_2) if $min(|X_1|, |Y_1|) > min(|X_2|, |Y_2|)$.

4.2.2 Rationale of the proposed approach

Many real-life networks have millions or even billions of vertices with a very low edge density. Existing approaches for solving MBBP rely heavily on the complement and the adjacent matrix representation. Unfortunately, the complement of such a massive graph usually results in very high memory consumption, making most of existing MBBP approaches unpractical. To avoid this difficulty, the proposed algorithm operates directly on the given graph, implying that much less memory is required for processing very large real-life sparse networks. From an algorithmic perspective, our algorithm iteratively seeks improved solutions by local search combined with graph reduction strategies. Specifically, the algorithm starts from an initial solution (a slightly relaxed balanced biclique) and uses move operators to improve the solution iteratively. However, we still need to answer a crucial question: how to improve the solution effectively while maintaining the two main constraints of a solution (balanced and biclique)?

Intuitively, local search operators that are successful for the maximum clique problem [Wu and Hao, 2015a] can be applied to MBBP, such as “add" (adding a vertex to the solution), “swap" (exchanging a vertex in the solution with another vertex out of the solution) or still “drop" (dropping a vertex from the solution). However, given a balanced biclique, an application of any of these operators results in an unbalanced biclique. To cope with this difficulty, we propose to (slightly) relax the balance requirement of the solution and allow the algorithm to explore both balanced and slightly unbalanced bicliques. For this purpose, we adopt the generalized *PUSH* operator initially designed for the maximum vertex weight clique problem [Zhou *et al.*, 2017a] to explore solutions within the relaxed search space Ω^2 rather than Ω^0.

Another key idea we used is graph reduction. Given a bipartite graph G and a known best balanced size ω (a lower bound), it is clear that to further improve ω, it is useless to consider any vertex whose degree is smaller than or equal to ω since such a vertex can in no way extend the best solution found so far. Consequently, these vertices (with a degree smaller than or equal to ω) along with the incident edges can be safely removed from the graph. Our algorithm integrates this idea to dynamically prune the graph under consideration, which proves to be highly effective on massive sparse graphs.

Finally, applying the pruning techniques can disconnect the original graph into several connected subgraphs. This observation can be explored advantageously to further prune the graph in combination with an exact algorithm. Indeed, if a subgraph is small enough such that an exact algorithm can identify the *maximum* balanced biclique quickly, then the subgraph can be definitively removed since the subproblem (associated to the subgraph) is optimally solved. Moreover, the optimal solution of this subgraph can also be used to update the current best balanced biclique (and the lower bound bound), which can lead to additional reduction of the graph.

4.2.3 General procedure of TSGR-MBBP

Based on the rationale presented in Section 4.2.2, we introduce Tabu Search with Graph Reduction for MBBP (TSGR-MBBP) (Algorithm 4.1). TSGR-MBBP is an iterated two phase algorithm and includes two main components: the Constraint-Based Tabu Search (CBTS) procedure and the graph reducing procedure. The CBTS procedure is used to find high quality bicliques in the relaxed search space Ω^2, while the graph reducing procedure aims to shrink progressively the current graph without losing optimal solutions.

After setting the best biclique (X^b, Y^b) to $(\emptyset, \emptyset)$ and the best balanced size ω to 0 (lines 2 and 3), the algorithm repeats the main 'while' loop (lines 4-20) until a stopping condition is met. For each 'while' loop, an initial biclique, which is not necessarily balanced, is first generated by Random_Init_Solution() (line 5, see Section 4.2.3), and then further improved by the CBTS procedure (Constraint_Tabu_Improve(), line 6, see Section 4.2.3). If the resulting biclique has a balanced size larger than the current best balanced size ω, the best biclique (X^b, Y^b) and the best balanced size are updated (lines 7-9).

Now, if the current best balanced size is greater than or equal to the degree of any vertex in the current graph, the graph reduction procedure is activated (lines 10-18). This procedure includes two phases: first, reducing the current graph by the $Peel$ procedure to remove fruitless vertices and their incident edges (line 11, see Section 4.2.3); second, determining the maximum balanced size of each connected subgraphs with up to K (a predefined parameter) vertices by a branch-and-bound (B&B) exact algorithm (Exact_Search(), line 14, see Section 4.2.3) and then deleting these subgraphs from the current graph (line 18). The optimal solution found by exact search can also be used to update the current best solution found so far (lines 15-17). Finally, though TSGR-MBBP is a heuristic algorithm, thanks to the graph reduction procedure, ω is proven to be the optimal balanced size when the cardinality of any partition ($|U|$ or $|V|$, which is a upper bound of the maximum biclique) in the current graph is no more than ω (which is a lower bound) (lines 19-20).

As explained in Section 4.2.1, the proposed algorithm operates on the relaxed biclique space Ω^2. As a result, the current solution (X, Y) and the best biclique found so far (X^b, Y^b) are not necessarily balanced with nevertheless a balance deviation limited to 2. Actually, the three procedures: Random_Init_Solution(), Constraint_Tabu_Improve() and Exact_Search() generate or return a biclique with a balance deviation no more than 2. The procedure of retrieving a strict balanced biclique of size ω from an unbalanced biclique is accomplished by Make_Balance(). This procedure simply removes vertices from the larger set X^b or Y^b until a balanced biclique of size ω is obtained. Obviously, no more than 2 vertices will be removed by Make_Balance().

Construct random initial solutions

The Random_Init_Solution() procedure is invoked to initialize each restart of TSGR-MBBP with a new biclique. This procedure starts from a trivial solution formed by a random vertex from $U \cup V$, say $(X, Y) = (\{1\}, \emptyset)$ (without loss of generality). Then, it iteratively expands the current solution by alternatively adding one vertex v to the set X or Y, v being necessarily connected to all vertices of the other set. Specifically,

Algorithm 4.1: Main framework of TSGR-MBBP

```
Input: Graph instance $G = (U, V, E)$, tabu search depth $L$, cardinality threshold $K$ for graph reduction with
       exact algorithm, tabu tenure parameter $\alpha$.
Output: The maximum balanced biclique.
begin
    $(X^b, Y^b) \leftarrow (\emptyset, \emptyset)$;          /* The largest balanced biclique found so far */
    $\omega = 0$ ;                          /* The largest balanced size found so far */
    while stopping condition is not met do
        // Find an improved biclique from a new initial biclique
        $(X, Y) \leftarrow Random\_Init\_Solution(G)$ ;                          /* Section 4.2.3 */
        $(X, Y) \leftarrow Constraint\_Tabu\_Improve(G, (X, Y), L, \alpha)$ ;          /* Section 4.2.3 */
        if $min(|X|, |Y|) > \omega$ then
            $(X^b, Y^b) \leftarrow (X, Y)$;
            $\omega \leftarrow min(|X|, |Y|)$
        // Graph reduction procedure using improved balanced size $\omega$
        while $\omega \geq min_{v \in U \cup V}\{|N(v)|\}$ do
            // The first graph reduction
            $G \leftarrow Peel(G, \omega)$ ;                                   /* Section 4.2.3 */
            // The second graph reduction
            for each connected subgraph $G_i[U_i \cup V_i]$ in $G$ do
                if $|U_i| + |V_i| \leq K$ then
                    $(X, Y) \leftarrow Exact\_Search(G_i, \omega)$ ;                  /* Section 4.2.3 */
                    if $min(|X|, |Y|) > \omega$ then
                        $(X^b, Y^b) \leftarrow (X, Y)$;
                        $\omega \leftarrow min(|X|, |Y|)$
                    $G \leftarrow G[(U \setminus U_i) \cup (V \setminus V_i)]$
        if $|U| \leq \omega \vee |V| \leq \omega$ then
            return $Make\_Balance(X^b, Y^b)$ ;           /* $(X^b, Y^b)$ is an optimum solution */
end
return $Make\_Balance(X^b, Y^b)$
```

in the first iteration, a vertex is selected randomly from the candidate set $\cap_{i \in X} N(i) \setminus Y$. Then, in the next iteration, we switch to the candidate set $\cap_{i \in Y} N(i) \setminus X$. The procedure continues until the current candidate set becomes empty. The time complexity of this procedure is bounded by $O(|U \cup V| \times |E|)$.

Consider Figure 4.1 as an example and suppose that we start from solution $(X, Y) = (\{1\}, \emptyset)$, the algorithm expands the solution by selecting an arbitrary vertex from $N(1) \setminus \emptyset = \{5, 6, 8\}$ (say 5) in the first iteration. In the second iteration, the algorithm expands Y by adding a vertex from $N(5) \setminus \{1\} = \{2, 3, 4\}$. Suppose that the algorithm goes on likewise to achieve a solution $(X, Y) = (\{1, 2, 3\}, \{5, 6\})$ after four iterations. Then in the fifth iteration, we try to expand Y by adding a vertex from the candidate set $\cap_{i \in N(X)} \setminus Y$. However, since this candidate becomes empty, the Random_Init_Solution() procedure stops and returns $(X, Y) = (\{1, 2, 3\}, \{5, 6\})$ as its output.

The biclique (X, Y) returned by this procedure may not be strictly balanced, but the balance deviation can never exceed 1. This biclique is served as the starting solution for the tabu search procedure which is explained below.

Constraint-Based Tabu Search

The CBTS procedure (Algorithm 4.2) is the main search component of the proposed algorithm. CBTS iteratively transforms the current solution (biclique) to a neighbor solution while respecting the unbalance limit of 2. The parameter L (a positive integer) is called tabu search depth, which defines the total number of iterations of tabu search. The other parameter, $\alpha \in \mathbb{R}_+ \cup \{0\}$, is a coefficient of tabu tenure (see Section 4.2.3). In each iteration, CBTS applies the *PUSH* operator (lines 5-14, see Section 4.2.3), which either adds a vertex to the current solution or swaps a vertex of the biclique against a vertex outside of the biclique. Whenever the balance deviation exceeds 2 after an application of *PUSH*, a repairing procedure is followed to recover the balance of the current biclique (lines 15-23, see Section 4.2.3). The repairing procedure simply drops vertices from the larger partition of the biclique until the cardinality of both partitions becomes equal. CBTS terminates after L such “push" and “repair" iterations.

The push operator

The *PUSH* operator was first proposed for the maximum weight clique problem in [Zhou *et al.*, 2017a] where each application of *PUSH* adds a vertex (taken from a candidate set) in the clique and expels $p \geq 0$ vertices from the clique to maintain the feasibility of the transformed clique. In the context of MBBP, given a biclique (X, Y) with $X \subseteq U$ and $Y \subseteq V$, and without loss of generality, suppose that a vertex $v \in N(Y) \setminus X$ (i.e., $N(Y) \setminus X$ is the candidate set for *PUSH*) is chosen. The *PUSH* operator adds vertex v to X and expels from Y the vertices that are not adjacent to v. Let (X', Y') denotes the new biclique after the *PUSH* operation, then we represent this transformation by $(X', Y') \leftarrow (X, Y) \oplus push(v)$.

Algorithm 4.2: Constraint-Based Tabu Search

```
Input: Graph instance $G = (U, V, E)$, starting solution $(X, Y)$, tabu search depth $L$, tabu tenure parameter $\alpha$.
Output: The best biclique $(X^*, Y^*)$ found.
begin
    $I \leftarrow 0$, $(X^*, Y^*) \leftarrow (X, Y)$;  /* $I$ is the iteration counter, $(X^*, Y^*)$ keeps the best biclique found so far */
    $T[1...n] \leftarrow [0...0]_n$ ; /* initiate tabu list, each vertex $v$ being marked tabu for the next $T[v]$th iterations; $n = |U| + |V|$ */
    while $I \leq L$ do
        // Explore the neighbor solutions
        Build $C_{expand} \subseteq C$ and $C_{plateau} \subseteq C$ ;        /* Decompose candidate set, see Section 4.2.3 */
        $v \leftarrow null$;
        if $C_{expand} \neq \emptyset$ then
            $v \leftarrow random(C_{expand})$ ;
        else if $C_{plateau} \neq \emptyset$ then
            $v \leftarrow random(C_{plateau})$ ;
        if $v \neq null$ then
            $(X, Y) \leftarrow (X, Y) \oplus push(v)$ ;
            // Set tabu tenure for each vertex expelled by PUSH.
            for $u \leftarrow$ expelled vertex do
                $T[u] \leftarrow I + tt(\alpha, |S|)$ ;        /* $S = X$ if $u \in X$, otherwise $S = Y$ */
        // Recover balance when the balance deviation exceeds 2
        if $||X| - |Y|| > 2$ then
            while $|X| > |Y|$ do
                $u \leftarrow random(X)$ ;
                $(X, Y) \leftarrow (X, Y) \oplus drop(u)$ ;
                $T[u] \leftarrow I + tt(\alpha, |X|)$ ;        /* Set tabu tenure for the dropped vertex */
            while $|X| < |Y|$ do
                $u \leftarrow random(Y)$ ;
                $(X, Y) \leftarrow (X, Y) \oplus drop(u)$ ;
                $T[u] \leftarrow I + tt(\alpha, |Y|)$ ;
        // update the best solution
        if $min(|X|, |Y|) > min(|X^*|, |Y^*|)$ then
            $(X^*, Y^*) \leftarrow (X, Y)$
        $I \leftarrow I + 1$
end
return $(X^*, Y^*)$
```

Let $\delta_v = min(|X'|, |Y'|) - min(|X|, |Y|)$ be the change of the balanced sizes between (X', Y') and (X, Y), then δ_v can be calculated by the following rule.

$$\delta_v \leftarrow \begin{cases} -|Y \setminus N(v)| & \text{, if } |X| > |Y| \\ min(1, |Y| - |X| - |Y \setminus N(v)|) & \text{, otherwise} \end{cases} \tag{4.1}$$

Similarly, if $v \in N(X) \setminus Y$, $(X', Y') = (X \cap N(v), Y \cup \{v\})$, δ_v is updated by the same rule except that the roles of X and Y are exchanged.

The *PUSH* operator can be explained as a generalization of the conventional $(1, p)$-swap ($p \in Z^0$) operator. For example, if we restrict the candidate vertex v with property $N(v) \cap X = X$ or $N(v) \cap Y = Y$, $push(v)$ is equivalent to adding v without expelling any vertex (i.e., $(1, 0)$-swap); if we restrict v with property $|X \setminus N(v)| = 1$ or $|Y \setminus N(v)| = 1$, $push(v)$ exchanges v with another vertex in X or Y that is not adjacent to v (i.e., $(1, 1)$-swap). Actually, the two restrictions are employed in our CBTS algorithm to customize the *PUSH* operator, as explained in the next section.

Explore the neighbor solutions

The general *PUSH* operator applied to MBBP can add an arbitrary vertex from the candidate set $N(X \cup Y)$ into one set X or Y, and then expel $p \geq 0$ vertices from the other set. However, for the reason of computational efficiency, only a subset of $N(X \cup Y)$ is considered for each *PUSH* operation. Specifically, we add restrictions on the candidate vertices for the *PUSH* operation so that it adds one vertex to the current solution and at the same time, no more than one vertex from the current solution will be expelled. These restrictions lead exactly to the two cases that were introduced at the end of Section 4.2.3. In Algorithm 4.2, set C (line 5) includes the restricted candidate vertices for *PUSH*. Every vertex in C is adjacent to all the vertices of X (or Y), or all but one vertex of X (or Y).

Moreover, C_{expand} is a subset of C such that applying *PUSH* to any vertex (say v) of this subset always results in a solution of better quality (i.e., $\delta_v > 0$). Similarly, $C_{plateau} \subseteq C$ includes the vertices that can be exchanged by *PUSH* to obtain solutions of equal quality (i.e., $\delta_v = 0$).

To prevent CBTS from revisiting recently examined solutions, a tabu list [Glover and Laguna, 2013] is considered when we construct C_{expand} and $C_{plateau}$ from candidate set C: a vertex which is marked tabu in the current iteration will not be included in C_{expand} or $C_{plateau}$ unless pushing the vertex into the solution leads to a solution better than the best solution ever found (this is called aspiration rule in tabu search terminology). To sum up, let (X, Y) and (X^*, Y^*) be respectively the current solution and the best solution found so far during the current CBTS run, I the current iteration number, $T[v]$ the tabu tenure of vertex v (see below), then the restricted candidate set C, and sets C_{expand}, $C_{plateau}$ are defined as follows.

$$\begin{aligned} &C = \{v \in N(X \cup Y) : v \in U \wedge |N(v) \cap Y| \geq |Y| - 1, v \in V \wedge |N(v) \cap X| \geq |X| - 1\} \\ &C_{expand} = \{v \in C : \delta_v > 1, T[v] \leq I \vee min(|X|, |Y|) + 1 > min(|X^*|, |Y^*|)\} \\ &C_{plateau} = \{v \in C : \delta_v = 0, T[v] \leq I\} \end{aligned} \tag{4.2}$$

where $T[v] \leq I$ indicates that vertex v is no more forbidden by the tabu list for the current iteration and can take part in a future *PUSH* operation.

Given the subsets C_{expand}, $C_{plateau}$ as two alternative candidate sets for *PUSH*, CBTS gives priority to C_{expand} since pushing vertices of this set always improves the current biclique. Only when C_{expand} is empty, set $C_{plateau}$ is explored by the *PUSH* operator (lines 6-14). After each *PUSH* application with $C_{plateau}$, the vertex expelled by *PUSH* (u in line 13) is marked tabu for the following $tt(\alpha, |A|)$ ($A = X$ if $u \in X$, otherwise $A = Y$) consecutive iterations. According to [Zhou *et al.*, 2017a], $tt(\alpha, l)$ (called tabu tenure) is defined by the function: $tt(\alpha, l) = max(7, \alpha * random(l))$ where $\alpha \in \mathbb{R}_+ \cup \{0\}$ is a predefined parameter and $random(l)$ returns a random integer in $[0, l]$.

Recover biclique balance

Recall that with the restrictions on candidate vertices, the number of vertices expelled by *PUSH* in each iteration is either zero or one. As a result, if the balance deviation of the current biclique is greater than 2 (i.e., $||X| - |Y|| > 2$), it is impossible to make the biclique strictly balanced with one application of the *PUSH* operator. Consequently, each time the balance deviation of the solution exceeds 2, we restore the balance property by applying a repair procedure (lines 15-23). This repair procedure simply drops vertices from the larger set (X or Y) of the biclique until the solution becomes strictly balanced (denoted as $(X, Y) \leftarrow (X, Y) \oplus drop(u)$ at lines 18 and 22). Again, each dropped vertex u is forbidden to rejoin the solution during the period fixed by its tabu tenure ($tt(\alpha, |X|)$ if $u \in U$, $tt(\alpha, |Y|)$ if $u \in V$). In general, CBTS utilizes the *PUSH* operator to explore the space Ω^2 rather than Ω^0 by constraining the balance deviation of the visited solutions. In Section 4.2.5, we further investigate the effectiveness of this strategy.

Time complexity

CBTS operates directly on the input graph and uses the adjacent list representation to store the graph. Given a solution (X, Y), by our implementation, the time complexity of constructing C_{expand} and $C_{plateau}$ is bounded by $O(|N(X \cup Y)|)$. The time complexity of moving one vertex (outside the solution or into the solution) is bounded by $O(M)$ ($M = \max_{v \in U \cup V}\{|N(v)|\}$). Hence, the time complexity of one iteration in CBTS is bounded by $O(|N(X \cup Y)| + 2 \times M)$. Though $|N(X \cup Y)|$ is almost equal to $|U| + |V|$ in dense graphs, in very large real-life networks, both $|N(X \cup Y)|$ and M are very limited due to the sparsity of the graphs.

Reduction by the $Peel$ procedure

Our TSGR-MBBP algorithm employs the $Peel(G, w)$ procedure (Algorithm 4.1, line 11) to recursively delete all vertices whose degrees are smaller than or equal to ω until no such vertex exists. Obviously, if the cardinality of one vertex set of the reduced bipartite graph (which is a upper bound of the maximum biclique) is less than or equal to ω (which is a lower bound), then ω must be the optimal objective value because no better solution can exist in the reduced graph (Algorithm 4.1, lines 19-20).

The peeling procedure is triggered each time the balanced size of the largest biclique discovered so far (lower bound) is larger than or equal to the minimum degree of the current graph. This procedure is effective on large sparse graphs but may not reduce a dense graph much. The experiments reported in Section 4.3.7 confirm that, with a high quality lower bound, large real-life bipartite graphs can be significantly reduced.

We note that the idea of reducing a graph by removing unpromising vertices was previously used in a GRASP heuristic for detecting dense subgraphs (quasi-cliques) in massive sparse graphs [Abello *et al.*, 2002]. We adapted this technique for solving MBBP for the first time.

Reduction by exact search

Exact search algorithms guarantee the optimality of the solution found, but may require prohibitive computing time on large instances. However, since exact search algorithms are able to prove optimality on small graphs rapidly, they can still be used as a basis for graph reduction. In Algorithm 4.1 (lines 12-18), we show such an approach of using exact search for MBBP. If a solution has been confirmed to be optimal for a subgraph of the current graph, this subgraph can be safely eliminated from the current graph. Moreover, since the optimal value of the subgraph is a lower bound of the initial graph, we can use the optimal solution of the subgraph to update the current best balanced biclique, which in turn can further reduce the current graph. The exact algorithm used by TSGR-MBBP was adapted from a well-known B&B algorithm for the maximum clique problem [Carraghan and Pardalos, 1990] and described in Section 4.3.2. This exact algorithm is only applied to solve a subgraph with K vertices at most (K being the largest subgraph that is estimated to be solved in reasonable time by the algorithm). It is clear that K depends on the adopted exact algorithm and target subgraph. According to our experiments, we set K to 100 for random dense graphs and 500 for sparse real-life networks.

4.2.4 Computational experiments

To comprehensively evaluate the proposed TSGR-MBBP algorithm as well as its components, we tested our algorithm on two sets of benchmark instances including both (dense) random graphs and massive real-life networks.

Benchmark

We use the random graphs and KONECT benchmarks introduced in Section 1.4.1, Chapter 1. The random graphs for our study are generated by the same rule of [Yuan *et al.*, 2015] so that the performance of different algorithms can be compared. As for KONECT instances, we randomly select 25 bipartite instances 1.4.1.

Parameter tuning and experimental protocol

The TSGR-MBBP algorithm has three parameters: L - the tabu search depth; α - the coefficient for tabu tenure required by the Constraint_Tabu_Improve() procedure (Section 4.2.3); K - the threshold on the number of vertices of the subgraph for graph reduction with the exact algorithm (Section 4.2.3).

Since the first two parameters (L and α) are independent from the reduction procedure, we tuned them on a simplified version of TSGR-MBBP without the graph reduction procedure (i.e., by disabling lines 10-20 in Algorithm 4.1). We used the automatic parameter configuration package iRace [López-Ibánez *et al.*, 2011], which implements the Iterated F-Race (IFR) method. Given $L \in \{10, 100, 1000, 5000, 10000\}$, and $\alpha \in [0, 2]$, for each parameter configuration, we used a tuning budget of 500 hook-runs, each of which representing 10 independent calls of TSGR-MBBP. The training set for random graphs included 6 challenging instances, i.e., GraphU_500_XXX_1.clq and GraphU_500_XXX_2.clq (XXX can be replaced by 0.95, 0.90, 0.85). The experiments suggested that the combination ($L = 1000, \alpha = 0.30$) was a suitable configuration for random graphs. As for KONECT graphs, the training set included “actor-movie", “bookcrossing_full-rating", “dbpedia-genre", “dbpedia-team", “github", “stackexchange-stackoverflow". The final choice of parameters was $L = 100$ and $\alpha = 1.74$.

The use of two different settings for (L, α) is mainly due to the graph structures which vary much. According to our observations, for random dense graphs, a more intensified search is needed to find quality solutions. This is achieved with a large tabu search depth ($L = 1000$) and a short tabu tenure (with $\alpha = 0.30$). On the contrary, for large real-life sparse instances, the tabu search component is able to reach local optima very quickly. As a result, it is preferable to restart more frequently the tabu search component (with $L = 100$) and diversify more strongly the search during the optimization process (using a larger tabu tenure with $\alpha = 1.74$).

The third parameter K indicates the largest subgraph that can be solved in reasonable time by the exact algorithm described in Section 4.2.3. We set $K = 100$ for random graphs and 500 for KONECT graphs. In effect, since the random graphs we tested are very dense, they cannot be reduced by the reduction procedure, implying that no connected subgraph with less than 100 vertices exists in this set of benchmarks. A very large K is not acceptable, otherwise the computing time for exact search becomes prohibitive according to our observations for random graphs. Preliminary experiments also confirmed that the time consumption was normally insignificant (less than 2 seconds) for connected subgraphs with less than 500 vertices for sparse KONECT graphs. As the vertex number is just a rough estimation of the hardness of the subgraph for our exact algorithm, we terminate the exact algorithm if it does not finish during 10 seconds. This additional cutting-off condition prevents the algorithm from spending too much effort in searching optimal solutions for some potential hard subgraphs. If the exact search stops without giving an optimal solution, the corresponding subgraph will not be removed.

TSGR-MBBP was implemented in C++ and compiled with g++ v4.4.7 with optimization flag -o3. Our experiments were performed on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Linux 2.6.32. When solving the DIMACS machine benchmark procedure 'dfmax.c' [1] without compilation optimization flag, the run time on our machine is 0.40, 2.50 and 9.55 seconds for graphs r300.5, r400.5 and r500.5 respectively.

Considering the stochastic nature of TSGR-MBBP, we ran TSGR-MBBP 20 independent times to solve each instance. For the random graphs of 250 vertices, the time limit of each run was 30 seconds, while for the random graphs of 500 vertices, 60 seconds were allowed. As for the KONECT instances, we prolong this limitation to 360 seconds (6 minutes) since these instances are much larger than the random graphs.

Computational results of Random graphs

To evaluate the performance of TSGR-MBBP, we show computational results relative to three state-of-the-art MBBP approaches:

- EA/SM [Yuan *et al.*, 2015]: This is a hybrid algorithm mixing local search, structure mutation and repair-assisted restart. EA/SM is the most recent heuristic algorithm and outperforms the precedent algorithms like [Yuan and Li, 2011; Yuan and Li, 2014]. For our comparative experiment, we ran 20 times the source code of EA/SM (provided by its authors) to solve each instance, each run being limited to 200,000 fitness evaluations according to [Yuan *et al.*, 2015]. We observed that to attain its best solutions, EA/SM needed a run time ranging from 42 to 50 seconds for instances of 250 vertices and 75 to 94 seconds for instances of 500 vertices (see Table 4.1). Consequently, the stopping condition of EA/SM can be considered to be more favorable than that used to run our algorithm (a *cut off time* of 30 seconds for instances of 250 vertices and and 60 seconds for instances of 500 vertices).
- IBM CPLEX: CPLEX is one of the most popular commercial optimization software. We ran CPLEX (version 12.6.1) 2 hours (7200 seconds) on each instance with the binary linear formulation Equation 1.7 – Equation 1.10 in Chapter 1. Obviously, the total time given to TSGR-MBBP for 20 runs (60*20 = 1200 seconds for the random instances and 360*20 = 7200 seconds for the KONECT instances) is no more than 2 hours.
- AL_Greedy [Al-Yamani *et al.*, 2007]. This is a (fast) greedy algorithm which solves the equivalent maximum balanced independent set problem for the bipartite complement. According to [Yuan *et al.*, 2015], this algorithm performs better than its earlier version [Tahoori, 2006]. Thus, we re-implemented this algorithm and used it for our comparative study. Since AL_Greedy is a deterministic heuristic, only one run was needed to solve each instance. Moreover, AL_Greedy stops once its construction procedure reaches its end. Thus, no explicit stopping condition is required.

Table 4.1 reports the computational results of TSGR-MBBP together with the results of the reference approaches (EA/SM, CPLEX and AL_Greedy) on the 30 random dense graphs. Column "instance" shows the name of each instance. Column "BKV" presents the best known values reported in [Yuan *et al.*, 2015]. For TSGR-MBBP and EA/SM, column "best(ave)" indicates the maximum value of the 20 best balanced sizes found in 20 runs, the average size is given between parentheses if the 20 runs do not lead to an identical balanced size; column "time" reports the average time (in seconds) of first hitting the best balanced size in 20 runs; column "reduce" (only for TSGR-MBBP) reports the number of vertices removed by the two reduction methods in one of the runs where we find the best balanced size. For CPLEX, we report the best lower bounds and the time needed to complete the search. If CPLEX fails to report a feasible solution for an instance due to memory limitation, "-" is used in the corresponding entries of columns "best" and "time". For AL_Greedy, since its run time is negligible (shorter than 0.01 second for all instances), we only report the best biclique values.

[1] dfmax:ftp://dimacs.rutgers.edu/pub/dsj/clique/

Table 4.1: Computational results of TSGR-MBBP together with the results of EA/SM, CPLEX and AL_Greedy on the set of 30 random dense graphs.

instance	BKV [Yuan *et al.*, 2015]	TSGR-MBBP			EA/SM		CPLEX 12.6.1		AL_Greedy
		best(ave)	time	reduce	best(ave)	time	best	time	best
G_250_0.95_1	68	68	0.05	0	68(67.90)	50.28	66	$\geq$7200	64
G_250_0.95_2	66	66	0.21	0	66(65.05)	49.31	64	$\geq$7200	59
G_250_0.95_3	70	70	0.17	0	70(69.50)	48.87	-	-	67
G_250_0.95_4	68	68	0.42	0	68(67.10)	47.36	66	$\geq$7200	63
G_250_0.95_5	68	68	0.72	0	67(66.95)	47.41	67	$\geq$7200	62
G_250_0.90_1	44	44	0.06	0	44(43.70)	42.94	42	$\geq$7200	37
G_250_0.90_2	44	**45**	0.52	0	45(43.90)	43.28	42	$\geq$7200	39
G_250_0.90_3	44	44	0.13	0	44(43.45)	43.20	42	$\geq$ 7200	40
G_250_0.90_4	45	45	0.66	0	44(43.80)	43.13	42	$\geq$ 7200	40
G_250_0.90_5	45	45	0.23	0	45(44.10)	45.13	41	$\geq$7200	40
G_250_0.85_1	33	33	0.11	0	33(32.40)	47.92	-	-	30
G_250_0.85_2	33	33	0.04	0	33(32.75)	49.94	-	-	31
G_250_0.85_3	34	34	0.69	0	34(32.95)	44.66	-	-	31
G_250_0.85_4	33	33	0.07	0	33(32.90)	43.76	30	$\geq$7200	30
G_250_0.85_5	33	33	0.52	0	33(32.30)	44.16	30	$\geq$7200	30
G_500_0.95_1	91	**93**	14.37	0	91(90.20)	93.28	-	-	83
G_500_0.95_2	89	**91**	15.58	0	90(88.30)	92.02	-	-	81
G_500_0.95_3	89	**91**(90.05)	3.85	0	90(87.85)	92.62	85	$\geq$ 7200	81
G_500_0.95_4	88	**90**(89.40)	21.04	0	88(86.85)	93.28	83	$\geq$ 7200	78
G_500_0.95_5	90	**91**(90.90)	13.40	0	90(88.15)	94.30	81	$\geq$ 7200	83
G_500_0.90_1	56	56	12.21	0	55(53.75)	76.24	46	$\geq$ 7200	49
G_500_0.90_2	56	56	5.38	0	56(54.00)	79.34	47	$\geq$ 7200	48
G_500_0.90_3	54	**56**(55.60)	15.57	0	55(53.45)	79.52	46	$\geq$ 7200	48
G_500_0.90_4	55	**56**(55.55)	9.87	0	55(53.75)	79.59	47	$\geq$ 7200	48
G_500_0.90_5	55	**56**(55.50)	13.68	0	55(53.25)	82.23	44	$\geq$ 7200	48
G_500_0.85_1	40	40	4.59	0	40(38.45)	75.55	33	$\geq$ 7200	34
G_500_0.85_2	41	41	5.84	0	40(39.25)	75.56	32	$\geq$7200	33
G_500_0.85_3	40	**41**(40.50)	13.50	0	41(38.65)	81.48	35	$\geq$7200	35
G_500_0.85_4	40	40	1.84	0	39(38.30)	75.29	33	$\geq$7200	35
G_500_0.85_5	41	41	4.60	0	40(38.60)	75.06	31	$\geq$7200	34

From Table 4.1, we first observe that in terms of solution quality, TSGR-MBBP competes very favorably with the reference approaches. In particular, TSGR-MBBP improves the best-known results of [Yuan *et al.*, 2015] for 10 instances (marked in bold font). For the 20 remaining instances, the best objective values found by TSGR-MBBP are always as good as or better than those of the reference algorithms. The average objective values of the 20 runs of TSGR-MBBP are also better than that of EA/SM. Moreover, the performance of TSGR-MBBP is quite stable across the whole set of tested instances. In terms of computational efficiency, TSGR-MBBP is very competitive – it hits its best result within no more than one and 22 seconds for the instances of 250 and 500 vertices respectively, against up to 50 and 94 seconds for the best reference algorithm EA/SM. As for CPLEX, it cannot complete its search within a duration of 2 hours and thus fails to find the optimal solution for any instance (still CPLEX finds some solutions better than those of AL_Greedy). Unsurprisingly, the greedy algorithm AL_Greedy leads to solutions of very poor quality. Finally, as expected, neither reduction method is successful on these very dense graphs as the degree of any vertex is much larger than the best balanced size. For example, the vertex degree of “G_500_0.85_X” is closely around 425 while the optimal balanced size is estimated to be between 39 and 76 by the theorem of [Dawande *et al.*, 2001]. However, as we show in the next section, the reduction procedure becomes extremely effective when large sparse graphs are considered.

Computational results of KONECT networks

We report in Table 4.2 the computational results of TSGR-MBBP and CPLEX on the set of 25 KONECT instances. For this study, we ignore EA/SM and AL_Greedy since the EA/SM code cannot be run on these graphs (EA/SM imposes the input graph to be balanced, which is not the case for KONECT instances), while AL_Greedy performs very poorly (see Table 4.1). Columns “name”, “$(|U|, |V|)$”, “$|E|$” show the basic information of the original instances. For TSGR-MBBP, columns “best(ave)" and “time" report the same information as in Table 4.1. Columns “red_1” and “red_2” indicate the total number of vertices that are removed from the original graph by the two reduction methods (the $Peel$ procedure and the exact search procedure) in one of the runs where we find the best balanced size. To enable CPLEX to load large graphs, each original graph was pre-reduced by applying $Peel(G, best)$ before starting CPLEX. Column “$(|U'|, |V'|)$" reports the number of vertices after applying $Peel(G, best)$ while columns “best” and “time" report the best balanced size reached as well as the total consumed time. Symbol “*" indicates that the solution has been proven to be optimal by the corresponding algorithm, while symbol “-" means that the initial (and $Peel$ pre-reduced) graph cannot be loaded into CPLEX.

As explained in Section 4.2.3, when either of the two vertex sets of the current bipartite graph contains less than ω (the best balanced size found so far) vertices, ω is proven to be the optimal maximum balanced size. From Table 4.2, we observe that TSGR-MBBP proves optimality for 14 out of the 25 instances (indicated by “*"), even though these real-world instances are significantly larger than the random instances. Also, TSGR-MBBP achieves the same best balanced size in all 20 runs for all but 5 instances (whose average objective values are reported in the table). Observing the number of vertices that has been reduced, we find that the first reduction method (the $Peel$ method) prunes more than half or even all of the vertices during the search procedure. As for the second reduction method (which is based on exact search), though the vertices removed by this method are fewer than the first method, we cannot neglect its significance. For 5 instances “bibsonomy-2ui",“dpedia-genre",“dbpedia-starring", “moreno_crime" and “wiki-en-cat", the $Peel$ procedure fails to reduce these graphs to small enough subgraphs such that optimality can be proven (one vertex set of the subgraph includes fewer than ω vertices, see column “$(|U'|, |V'|)$"), TSGR-MBBP directly finds the optimal solution for the resulting subgraphs with less than K vertices. The CPLEX solver, unfortunately, is unable to load some of these massive graphs even after reducing these graphs significantly by applying $Peel(G, best)$ in the pre-processing step. For instances for which CPLEX finds a feasible solution, like “discogs_style", “edit-frwiktionary", “stackexchange-stackoverflow" and “youtube-groupmemberships", the results are still worse than those achieved by TSGR-MBBP. Besides, CPLEX always requires a longer time than TSGR-MBBP to attain the best solution.

Table 4.2: Computational results of TSGR-MBBP and CPLEX on the set of 25 large KONECT instances. The results of EA/SM and AL_Greedy are not available.

instance			TSGR-MBBP				CPLEX		
name	$(\|U\|, \|V\|)$	$\|E\|$	best(ave)	time	red_1	red_2	$(\|U'\|, \|V'\|)$	best	times
actor-movie	(127823, 383640)	1470418	8	8.91	474822	357	(100398, 88729)	N/A	N/A
bibsonomy-2ui	(5794, 767447)	2555080	8*	1.01	772062	1179	(137, 307)	8*	2209.76
bookcrossing_full-rating	(105278, 340523)	1149739	13(12.30)	122.25	433428	33	(26799, 76949)	N/A	N/A
dblp-author	(1425813, 4000150)	8649016	10*	8.92	5416361	9602	(0, 0)	-	-
dbpedia-genre	(258934, 7783)	463497	7*	1.22	265973	744	(385, 118)	7*	931.59
dbpedia-location	(172091, 53407)	293697	5*	0.22	224220	1278	(0, 0)	-	-
dbpedia-occupation	(127577, 101730)	250945	6*	0.88	228847	460	(0, 0)	-	-
dbpedia-producer	(48833, 138844)	207268	6*	0.17	183879	3798	(0, 0)	-	-
dbpedia-recordlabel	(168337, 18421)	233286	6*	0.23	186474	284	(0, 0)	-	-
dbpedia-starring	(76099, 81085)	281396	6*	0.29	156370	814	(44, 21)	6*	2.06
dbpedia-team	(901166, 34461)	1366466	6(5.50)	99.29	906083	341	(24858, 4345)	N/A	N/A
dbpedia-writer	(89356, 46213)	144340	6*	0.13	131338	4231	(0, 0)	-	-
discogs_affiliation	(1754823, 270771)	14414659	26	22.15	2008903	662	(11722, 4307)	N/A	N/A
discogs_lgenre	(270771, 15)	4147665	15*	10.16	270786	0	(0, 0)	-	-
discogs_style	(1617943, 383)	24085580	38(37.15)	131.85	1612732	0	(5289, 305)	36	$\geq$ 7200
edit-frwiki	(288275, 4022276)	46168355	41(27.50)	228.91	4250247	0	(6664, 56700)	N/A	N/A
edit-frwiktionary	(5017, 1907247)	7399298	19	31.88	1909273	0	(232, 2759)	16	$\geq$ 7200
flickr-groupmemberships	(395979, 103631)	8545307	67	94.74	458053	0	(213863, 61790)	N/A	N/A
github	(56519, 120867)	440237	12	4.74	169775	774	(4001, 2836)	N/A	N/A
moreno_crime	(829, 551)	1476	2*	0.00	1072	308	(4, 4)	2*	0.03
opsahl-collaboration	(16726, 22015)	58595	8*	0.05	37780	961	(0, 0)	-	-
opsahl-ucforum	(899, 522)	33720	5*	0.03	531	890	(0, 0)	-	-
stackexchange-stackoverflow	(545196, 96680)	1301942	9(8.95)	92.12	625399	52	(1432, 867)	8	$\geq$ 7200
wiki-en-cat	(1853493, 182947)	3795796	14*	17.58	2027887	8553	(87, 60)	14*	11.23
youtube-groupmemberships	(94238, 30087)	293360	12	0.81	121908	118	(1432, 867)	8	$\geq$ 7200

Table 4.3: Comparison between three different versions of the Constraint-Based Tabu Search procedure: CBTS$_{\Omega^\infty}$ can visit any biclique; CBTS$_{\Omega^1}$ visits only bicliques with a balance deviation no more than 1; CBTS$_{\Omega^2}$ (the original version of CBTS) visits bicliques with a balance deviation no more than 2.

instance	CBTS$_{\Omega^\infty}$			CBTS$_{\Omega^1}$			CBTS$_{\Omega^2}$		
	best(ave)	time	iter	best(ave)	time	iter	best(ave)	time	iter
GraphU_500_0.05_3	90(89.20)	8.48	2827917	70(68.25)	19.63	1387704	91(90.20)	7.76	2297846
GraphU_500_0.05_4	90(88.20)	19.48	2886301	69(67.35)	19.47	1386318	90(89.20)	10.99	2288732
GraphU_500_0.05_5	91(89.95)	20.37	2829679	71(68.40)	28.71	1389515	91(90.95)	16.41	2287010
GraphU_500_0.10_3	54(53.85)	18.77	5620994	42(40.70)	15.32	1387011	56(55.60)	13.49	2343998
GraphU_500_0.10_4	55(54.20)	17.11	5774901	42(40.90)	16.74	1385316	56(55.30)	7.17	2356738
GraphU_500_0.10_5	55(54.10)	10.04	5750204	42(41.45)	22.12	1388886	56(55.55)	13.27	2352404
GraphU_500_0.15_3	39(38.55)	15.57	6340722	31(29.75)	18.62	1441000	41(40.70)	16.92	2409855
dblp-author	8(5.40)	26.69	2674721	10(8.60)	27.25	1089614	10(9.50)	21.27	696636
dbpedia-genre	5(2.85)	18.07	546103	4(2.85)	19.69	121851	4(3.05)	9.35	147990
dbpedia-team	4(3.25)	16.25	5699436	4(3.30)	8.94	1232615	5(3.85)	24.11	1260575
discogs_style	9(3.30)	1.19	10651	7(3.80)	5.14	13617	25(7.20)	29.43	19900
edit-frwiktionary	9(2.65)	0.18	2476	9(2.85)	5.52	2204	9(3.05)	1.28	1946
wiki-en-cat	14(6.05)	29.28	3640199	13(7.50)	24.17	553078	14(8.75)	25.18	706691

4.2.5 Analysis

This section presents an empirical analysis of the restricted unbalance constraint related to the Constraint-Based Tabu Search procedure (Section 4.2.3) and the merit of the graph reduction procedure (Section 4.2.3).

Unbalance constraint of Constraint-Based Tabu Search

The Constraint-Based Tabu Search procedure (Algorithm 4.1, line 6) is one key component of the proposed TSGR-MBBP algorithm. One of the main features of CBTS is that while unbalanced bicliques are allowed, the balance deviation of the explored bicliques must be no more than 2 (see Sections 4.2.1 and 4.2.3) (this constraint is called unbalance constraint). To justify this specific unbalance constraint, we compare CBTS with two CBTS versions with different unbalance constraints. The first version (called "CBTS$_{\Omega^\infty}$") removes the unbalance constraint and allows the procedure to visit any bicliques (lines 15-23 are removed from Algorithm 4.2). The second version (named as "CBTS$_{\Omega^1}$") introduces a more restrictive unbalance constraint – the balance deviation is required to be no more than 1 after each iteration (i.e., change the repairing condition in line 15 to $|X| - |Y| > 1$). We also used "CBTS$_{\Omega^2}$" to denote the original CBTS procedure. As such, these three CBTS versions correspond to three restart algorithms searching within the solution spaces Ω^2, Ω^∞, and Ω^1 respectively. Note that the version with absolute balanced constraint is not considered. In effect, if we repair the solution whenever $|X| - |Y| \neq 0$, the current solution can never be improved because the *PUSH* operator only imports one vertex to one vertex set in each iteration.

For this study, we used 13 instances selected from the two benchmark sets. We ran each CBTS version 20 trials to solve each instance under the same configuration mentioned in Section 4.2.4. Each trial was given a time limit of 60 seconds. The comparative results of this study are summarized in Table 4.3. We denote one restart of CBTS as one iteration here (one 'while' loop, lines 10-20 in Algorithm 4.1). Column "best(ave)" indicates the best and average balanced biclique size found by each algorithm over 20 runs. Column "time" reports the average time to achieve the best balanced biclique size in all 20 runs. Column "iter" reports the average number of restarts for 20 runs.

As for the solution quality, the original Constraint-Based Tabu Search (CBTS$_{\Omega^2}$) procedure dominates the other variants both in terms of best and average values. CBTS$_{\Omega^2}$ also performs the best concerning the average time of attaining the best solution for random graphs. As for the total number of iterations (column "iter"), CBTS$_{\Omega^\infty}$ restarts more often than CBTS$_{\Omega^2}$ which on the other hand restarts more often

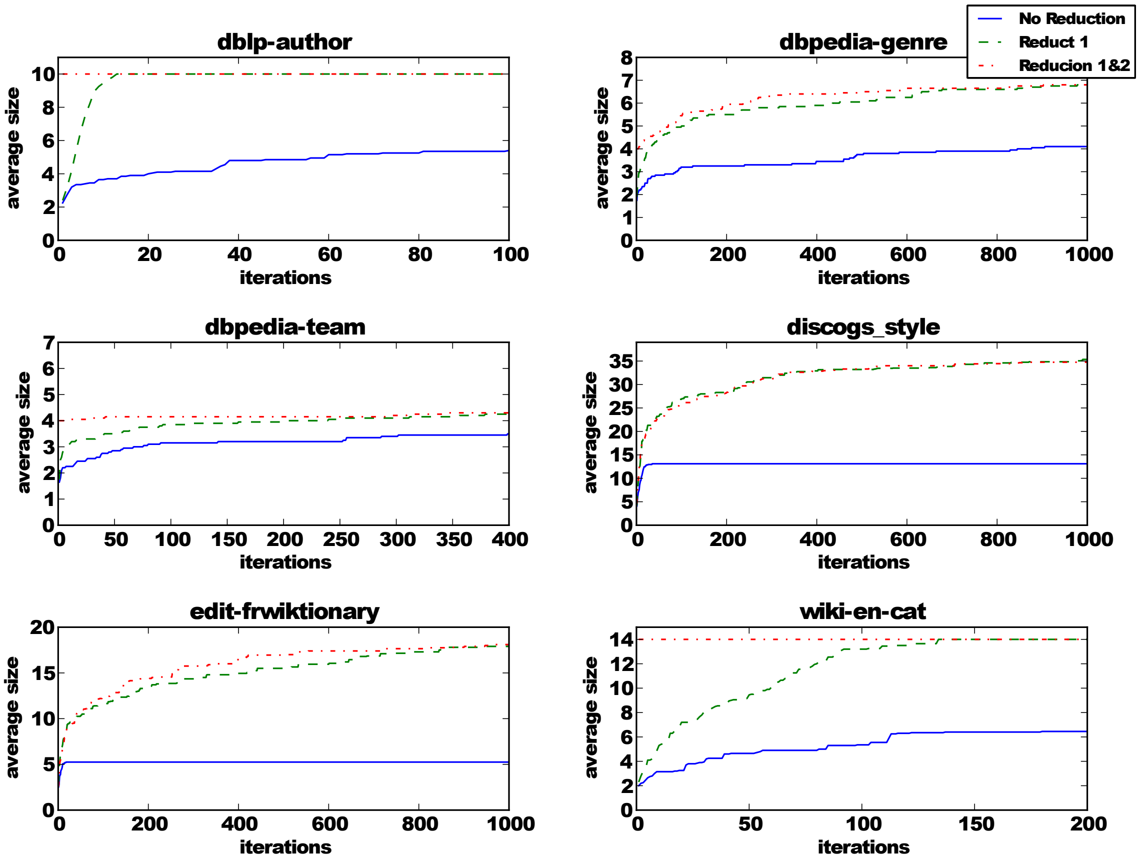

Figure 4.2: The relations between the number of iterations and the average best sizes of 20 runs on 6 selected instances from KONECT.

than CBTS$_{\Omega^1}$. Obviously, a tighter unbalance constraint leads to more frequent calls to the repair procedure, thus less iterations under the same time limitation. Meanwhile, the results suggest that the strategy of incorporating unbalance constraint is a good trade-off between solution quality and number of iterations.

Effectiveness of reduction methods

To gain a comprehensive understanding of the run-time behavior and efficiency of the two reduction methods, we show in this section an analysis of the convergence rate of three variants of the TSGR-MBBP algorithm:

- *No Reduction:* The reduction procedure is disabled, i.e., lines 10-18 are removed from Algorithm 4.1.
- *Reduction 1:* Only the first reduction method (the $Peel$ method) is used. i.e., lines 12-18 are removed from Algorithm 4.1.
- *Reduction 1&2:* Both reduction methods are used, i.e., the original TSGR-MBBP algorithm.

The variant with only the second reduction is not considered as the exact search will never be triggered without the $Peel$ procedure.

This study was based on 6 KONECT instances, “dblp-author", “dbpedia-genre", “dbpedia-team", “discog_style", “edit-frwikitionary" and “wiki-en-cat" which are large enough with different levels of difficulty for TSGR-MBBP (the difficulty is estimated by the time consumption of TSGR-MBBP in Table 4.2). We ran each algorithm variant 20 times to solve each instance with a time limit of 6 minutes per run. Again, we denote one restart of CBTS as one iteration. Figure 4.2 reports the relation between the number of iterations and the average best balanced size reached by each variant in 20 runs (abbreviated as ‘average size’). Considering the two variants with reduction can stop before reaching the time limit when the optimum is proven, we assume that the best size after termination is constantly the optimal size in this case.

According to Figure 4.2, in terms of the average result after the same number of iterations, the two variants using reduction always dominate the variant without reduction. Actually, “No Reduction" converges so slowly that it even has difficulties in reaching half of the best-known size in the given time limit. Comparing “Reduction 1" and “Reduction 1&2", for “dblp-author", “dbpedia-genre", “dbpedia-team", “edit_fiwikitionaryand" and “wiki-en-cat", “Reduction 1&2" always discovers solutions of high quality earlier. In particular, for two instances, “dblp-author" and “wiki-en-cat", “Reduction 1&2" reaches the optimal solution in the very first iteration. This is because for these graphs, the exact algorithm discovered the optimal solution in some of the connected subgraphs at the beginning of the search, which in turn enabled the *peel* procedure to prune the graph to trivial size and thus proves the global optimality. Nevertheless, for “discogs_style", “Reduction 1&2" and “Reduction 1" perform similarly. We also notice that the curves of “Reduction 1" and “Reduction 1&2" meet sooner or later for all the instances. In a nutshell, the convergence rate is highly related to the instance under consideration, but in any case, both reduction methods accelerate the search procedure.

4.3 Exact algorithms

In this section, we introduce new ideas for developing effective exact algorithms for MBBP, which can be applied to solve very large MBBP instances from applications like social networks. Our main contribution can be summarized as follows.

- We elaborate an upper bound propagation (UBP) procedure inspired from [Soto *et al.*, 2011], which produces an upper bound of the maximum balanced biclique involving each vertex in the bipartite graph. UBP propagates the initial upper bound involving each vertex and achieves an even tighter upper bound for each vertex. UBP is independent from the search procedure and is performed before the start of the algorithm. A new exact algorithm, denoted by (ExtBBClq), is devised by taking advantage of UBP to improve BBClq, the branch-and-bound algorithm introduced in [McCreesh and Prosser, 2014].
- Based on the upper bounds returned by UBP, we introduce new valid inequalities to tighten the MIP formulation of MBBP introduced in [Dawande *et al.*, 2001]. Our numerical experiments suggest that using the tightened model allows to achieve better results.
- We also present a new exact algorithm (ExtUniBBClq) to supplement the family of B&B based algorithms for MBBP. Unlike BBClq which goes through every possible balanced biclique, the new algorithm only enumerates the possible partial sets (half-sets) of the balanced bicliques in the graph. ExtUniBBClq also integrates UBP as a pre-processing procedure and performs generally well for the benchmark instances.

4.3.1 Preliminary definitions

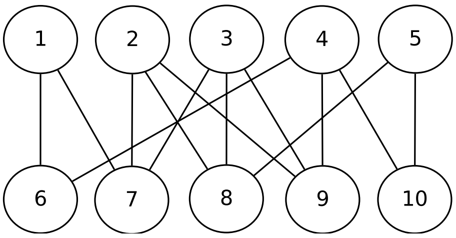

Figure 4.3: A bipartite graph $G = (U, V, E)$, $U = \{1, 2, 3, 4, 5\}$, $V = \{6, 7, 8, 9, 10\}$.

Given a bipartie graph $G = (U, V, E)$ ($|U| \leq |V|$ if not specifically stated), we still use $N(v) = \{u : \{v, u\} \in E\}$ to denote the set of vertices adjacent to v and $deg_G(v) = |N(v)|$ the degree of vertex v. However, to distinguish (X, Y) which represents a biclique in the heuristic algorithm, we use $(A, B) \subseteq$

(U, V) to represent a balanced biclique of G (i.e., $|A| = |B|$) in this part. The balanced size (half-size) of the balanced biclique (A, B) is the cardinality of $|A|$ (or $|B|$). Specifically, the upper bound involving vertex v is denoted by ub_v, which is an upper bound of the half-size of the maximum balanced biclique containing vertex v. For example, in Figure 4.3, a possible value for ub_1 could be 2, since $deg_G(1) = 2$.

4.3.2 Review of the BBClq algorithm

Algorithm 4.3 shows the BBClq algorithm, which is a recursive exact algorithm introduced in [McCreesh and Prosser, 2014] and used in TSGR-MBBP for graph reduction. BBClq is adapted from a well-known B&B algorithm for the maximum clique problem [Carraghan and Pardalos, 1990] and recursively builds up two sets A and B such that (A, B) forms a biclique. The algorithm maintains a candidate set C_A (C_B) that includes vertices which are eligible to move into A (B) while ensuring that (A, B) is a biclique (i.e., $C_A = \bigcap_{i \in B} N(i)$, $C_B = \bigcap_{i \in A} N(i)$). Initially, the algorithm sets lb, the global lower bound on the maximum biclique half-size to 0 and starts the search by calling BBClq$(G, \emptyset, \emptyset, U, V)$.

At each recursive call to BBClq, a vertex v (called branch vertex) is moved from C_A (lines 7-8). The algorithm then considers the branches (possibilities) of $v \in A$ in lines 9-12 and $v \notin A$ in the next `while` loop. The bounding procedure (line 9) prunes the branch of $v \in A$ if the upper bound after estimation in this context is not larger than the global lower bound. The upper bound estimating method, which is classically a key point concerning the performance of a B&B algorithm, will be introduced in the following section. If the current branch is not pruned, the search goes on by reconstructing A' with a new vertex v and C'_B by filtering from C_B those vertices not adjacent to v (every vertex in B must be adjacent to every vertex in A). After updating the two sets, the algorithm recursively calls BBClq in line 12, swapping the roles of A and B, as A and B are extended alternatively for the sake of satisfying the balance requirement. The above process is repeated in the next recursive call of BBClq.

When the algorithm loops back to line 4, as we just mentioned, it explores another branch implying $v \notin A$. The `while` loop stops when C_A becomes empty or when the remaining vertices in C_A do not allow to build a solution better than the global lower bound (lines 5-6). Besides, since $|A| + 1 = |B|$ or $|A| = |B|$ holds each time BBClq is called, we update the lower bound in lines 1-3 once $|A| > lb$ and store the incumbent solution (A, B) as the best solution found so far. As a result, the best solution (A^*, B^*) is an optimal biclique with $|A^*| = lb$ ($A^* \subseteq U$ or $A^* \subseteq V$), but it may not be totally balanced ($|A^*| - |B^*| \leq 1$). Thus, in line 13, the procedure of retrieving the maximum balanced biclique (of half-size lb) from a biclique is accomplished by *make_balance()*. This procedure simply removes vertices from the larger set A^* or B^* until a balanced biclique is obtained.

Figure 4.3 is now used to illustrate BBClq. Initially, $lb = 0$ and BBClq$(G, \emptyset, \emptyset, U, V)$ is called. According to the minimal degree heuristic in [McCreesh and Prosser, 2014], vertex 1 is chosen as the first branch vertex. Clearly, the current upper bound is greater than 0, the algorithm proceeds to BBClq$(G, \emptyset, \{1\}, \{6, 7\}, U \setminus \{1\})$ to explore the solutions containing vertex 1. As a result, the solution $(\{1\}, \{6\})$ is found and lb is updated to 1. Likewise, the algorithm selects 5 as the second branch vertex in the following loop, proceeds to BBClq$(G, \emptyset, \{5\}, \{8, 10\}, U \setminus \{1, 5\})$ if no upper bounding technique is applied. We can see that this recursive call to BBClq has to explore the case of expanding the given biclique by adding vertex 8 or 10. However, with the upper bounding estimating technique proposed in this section, the call of BBClq$(G, \emptyset, \{5\}, \{8, 10\}, U \setminus \{1, 5\})$ will not even start since the upper bound involving vertex 5 is 1 ($upper\_bound(\{5\}) = lb = 1$). The algorithm finds the optimal solution $(\{1, 2\}, \{4, 5\})$ after the third loop (which explores $A = \{2\}$ and calls BBClq$(G, \emptyset, \{2\}, \{7, 8, 9\}, U \setminus \{1, 5, 2\})$). There will be no additional iteration as $|A| + |C_A| \leq 2$ ($A = \emptyset, C_A = \{3, 4\}$).

4.3.3 Upper bound propagation and its use to improve BBClq

We introduce in this section our Upper Bound Propagation procedure (UBP) which is then used as a pre-processing technique to reinforce the BBClq algorithm presented in the last section.

Algorithm 4.3: BBClq(G, A, B, C_A, C_B), the B&B algorithm for MBBP taken from [McCreesh and Prosser, 2014].

```
Input: Graph instance $G = (U, V, E)$, $A$, $B$ - current sets that form a biclique, $C_A$, $C_B$ - the sets of
       eligible vertices that can be added to $A$ and $B$ respectively.
Output: A maximum balanced biclique of $G$.
if $|A| > lb$ then
    $lb \leftarrow |A|$ ;
    Record current best biclique in $(A^*, B^*)$;
while $C_A \neq \emptyset$ do
    if $|A| + |C_A| \leq lb$ then
        return ;
    $v \leftarrow branch\_vertex(C_A)$;
    $C_A \leftarrow C_A \setminus \{v\}$;
    if $upper\_bound(A \cup \{v\}) > lb$ then
        $A' \leftarrow A \cup \{v\}$;
        $C'_B \leftarrow C_B \cap N(v)$;
        BBClq($G$,$B$, $A'$, $C'_B$, $C_A$);
return $make\_balance(A^*, B^*)$;
```

4.3.4 The upper bound propagation procedure

The original BBClq algorithm calculates a biclique cover (based on addressing the graph coloring problem on the complement graph) to estimate the upper bound in a general graph relying on the fact that sets A and B are independent sets. However, when the given graph is bipartite, the upper bound found by this technique is trivial as two vertex sets are initially independent sets. Here, we introduce our Upper Bound Propagation to produce, for each vertex, an upper bound on the half-size of any maximum balanced biclique involving that vertex. UBP is based on the following propositions.

Proposition 1. *For each vertex $v \in U \cup V$, $deg_G(v)$ is an upper bound on the maximum half-size balanced biclique involving v.*

This proposition is obviously true since the half-size of a balanced biclique cannot exceed the degree of any vertex in the biclique.

Proposition 2. *Given a vertex $v \in U$, let $w_{vu} = |N(v) \cap N(u)|, \forall u \in U$. Let y_v be the maximum integer such that there exists at least y_v vertices in $\{w_{vu} : u \in U\}$ satisfying $w_{vu} \geq y_v$, then y_v is an upper bound on the maximum half-size balanced biclique involving v.*

Proof: Clearly, in the maximum balanced biclique (A, B) involving $v \in A$, for any vertex $u \in A$ (including v), we have $B \subseteq N(v) \cap N(u)$. Therefore, the maximum possible value y_v such that y_v vertices in U share at least y_v adjacent vertices with v is an upper bound involving v. Note that this proposition also holds given any vertex in V.

Proposition 3. *Given a vertex $v \in U \cup V$, let z_v be the largest integer such that there exists z_v vertices in $N(v)$ having upper bounds at least z_v. Then z_v is an upper bound on the maximum half-size balanced biclique involving v.*

Proof: We prove this proposition by contradiction. Suppose z_v is not an upper bound, then there exists a balanced biclique (A', B') involving $v \in A'$ of half-size z'_v such that $z'_v > z_v$, implying that all the z'_v vertices in B' ($B' \subseteq N(v)$) must have an upper bound of at least z'_v (i.e., $\forall u \in B, ub_v \geq z'_v$), which contradicts the condition that z_v is the maximum integer such that there exists in $N(v)$ at least z_v vertices having $z_v \geq ub_v$.

Consider the example of Figure 4.3, according to Proposition 1, we have $ub_1 = ub_5 = 2$, $ub_2 = ub_3 = ub_4 = 3$. Then, following Proposition 2, ub_1 can be improved (decreased) to 1 since $w_{12} = w_{13} = w_{14} = 1$, $w_{15} = 0$ ($y_1 = 1$). Similarly ub_2, ub_3, ub_4, ub_5 can also be improved to 2, 2, 1, 1 respectively. By Proposition 3, it can be deduced that $ub_6 = 1$ and $ub_7 = 2$ ($z_6 = 1$, $z_7 = 2$), which are better upper bounds than the degrees.

Based on these proposition, we devise the UBP procedure (see Algorithm 4.4) to calculate an upper bound involving each vertex. Initially ub_v is set to $deg_G(v)$, then the upper bound of each vertex in U is improved according to Proposition 2 (lines 2-9). From line 10 to the end of Algorithm 4.4, the procedure aims at propagating the upper bound based on Proposition 3 until the upper bounds cannot be improved any more. The propagation procedure is guaranteed to converge as the upper bounds cannot be smaller than 0. Experiments in Section 4.3.7 show that, for both random and real-life large instances, UBP converges very fast, only in a limited number of iterations.

In both lines 7 and 14, we use binary search to find, for a given set I of integers, the maximum element $x \in I$ such that there are at least x integers in I that are larger than or equal to x. The procedure works as follows: first, I is sorted by decreasing order, then, an iteration starts by comparing the middle element with its index in S (i.e., its position in the sorted list). If the middle element is greater (respectively lesser) than its index, the next iteration proceeds with the second half (respectively the first half) of I. This binary search procedure based on dichotomy performs at most $\log_2(|I|)$ operations.

Actually, we can also tighten the initial upper bound involving each vertex in V by repeating the process in lines 2-9 after replacing U with V before the propagating procedure (lines 10-17) starts. However, this procedure requires considerable memory and time especially for large graphs. For example, the matrix representing $w_{ij}, (i, j) \in V \times V$ requires a memory of $O(|V|^2)$, the computational time of computing y_v ($v \in V$) is bounded by $O(|U| \times (max_{v \in V} deg_G(v))^2) + |V| \times \log(|V|)$. Thus the overhead is not negligible. As a compromise, we set a threshold on the size of the vertex set. We apply the procedure of lines 2-9 to improve the upper bound involving each vertex only when the cardinality of the vertex set (U or V) is less than the threshold. In the following experiments, the threshold has empirically been set to 30000.

Algorithm 4.4: Upper bound propagation procedure

```
Input: Graph instance $G = (U, V, E)$
Output: An upper bound vector $ub$ for each vertex in $G$.
$\forall v \in U \cup V, ub_v \leftarrow deg_G(v)$;
$\forall (v, u) \in U \times U, w_{vu} \leftarrow 0$;
for $k \in V$ do
    for $(v, u) \in N(k) \times N(k)$ do
        $w_{vu} \leftarrow w_{vu} + 1$;
for $v \in U$ do
    Binary search for the largest integer $y_v$ such that $|\{u \in U : w_{vu} \geq y_v\}| \geq y_v$;
    if $y_v < ub_v$ then
        $ub_v \leftarrow y_v$;
$stable \leftarrow false$;
while $stable \neq true$ do
    $stable \leftarrow true$;
    for $v \in U \cup V$ do
        Binary search for the largest integer $z_v$ such that $|\{u \in N(v) : ub_u \geq z_v\}| \geq z_v$;
        if $z_v < ub_v$ then
            $ub_v \leftarrow z_v$;
            $stable \leftarrow false$;
return $ub$;
```

To see how tight the upper bounds provided by UBP are, consider the example of Figure 4.3, the final upper bound achieved by UBP is $ub_v = 1, 2, 2, 1, 1$ for $v \in U$ and $ub_v = 1, 2, 2, 2, 1$ for $v \in V$. These upper bounds are actually all tight.

Combining UBP with BBClq: ExtBBClq

As UBP is independent of the search algorithm, we use it as a pre-processing procedure for BBClq to obtain an extended version named ExtBBClq. In ExtBBClq, we use the same branching heuristic as in the original BBClq algorithm: the vertex of the minimum degree in C_A is given the highest priority for branching. To efficiently implement ExtBBClq, we sort the arrays $N(v)$ ($\forall v \in U \cup V$) in ascending order of index number before the beginning of BBClq, so that the intersection operation in line 11 (Algorithm 4.3) can be accomplished in $O(|C_B| * log(|N(v)|))$ asymptotic time by binary search. More importantly, to make use of the upper bound information calculated by UBP, in ExtBBClq, instead of calculating the upper bound by calling the upper bound estimation method (i.e., $upper\_bound(A \cup \{v\})$ in line 9, we use the pre-computed ub_v returned by UBP as the upper bound in the current branch.

4.3.5 A tighter mathematical formulation

In this section, we propose a tightened mathematical formulation for MBBP that takes advantage of the UBP procedure. Let us first recall the mathematical formulation of MBBP introduced in [Dawande *et al.*, 2001]:

$$\max \quad \omega(G) = \sum_{i=1}^{|U|} x_i \tag{4.3}$$

subject to:

$$x_i + x_j \leq 1, \forall \{i, j\} \in \bar{E} \tag{4.4}$$

$$\sum_{i=1}^{|U|} x_i - \sum_{i=|U|+1}^{|U|+|V|} x_i = 0 \tag{4.5}$$

$$x_i \in \{0, 1\}, \forall i \in U \cup V \tag{4.6}$$

where each vertex of $U \cup V$ is associated to a binary variable x_i indicating whether the vertex is part of the biclique, $\bar{E}$ is the set of edges in the complement bipartite graph of G. Constraint (4.4) requires that each pair of non-adjacent vertices cannot be selected at the same time (i.e., the solution must form a biclique). Constraint (4.5) enforces that the biclique is balanced.

To make use of the upper bounds returned by UBP, let $S^\ell \subseteq U$ (or $S^\ell \subseteq V$) be the set of all the vertices in U (respectively in V) such that $ub_i \leq \ell$ (ℓ is a positive integer) for all $i \in S^\ell$. Then the following inequality is valid:

$$\sum_{i \in S^\ell} x_i \leq \ell$$

Indeed, the vertices in S^ℓ can only be involved in balanced bicliques having half-size less than ℓ. We consider this inequality for $\ell = \max_{i \in U} ub_i$ as it dominates the inequalities associated with higher values of ℓ.

Before tightening this inequality, we observe that since $\ell = \max_{i \in U} ub_i$, we have $S^\ell = U$. Then for each $u \in U$ such that $ub_u < \ell$, or equivalently for all $u \in S^{\ell-1}$, we can lift the term associated with u:

$$(\ell - ub_u + 1)x_u + \sum_{i \in U \setminus \{u\}} x_i \leq \ell \qquad \forall u \in S^{\ell-1}$$

Let $T_u^{\ell-1}$ be any maximal subset of $S^{\ell-1}$ containing u such that for all i and $j \in T_u^{\ell-1}$, then $N(i) \cap N(j)$ is empty. The term 'maximal subset' means that no vertex $i \in S^{\ell-1}$ can be added to $T_u^{\ell-1}$.

We can deduce the following valid inequality:

$$\sum_{i \in T_u^{\ell-1}} (\ell - ub_i + 1)x_i + \sum_{i \in U \setminus T_u^{\ell-1}} x_i \leq \ell \quad \forall u \in S^{\ell-1}$$

It can be observed that the valid inequalities built from two vertices u and v of $S^{\ell-1}$ may possibly be identical, especially if $v \in T_u^{\ell-1}$. This is not an issue since modern solvers remove duplicate constraints automatically during presolving.

Naturally, the lower ub_i is, the tighter these inequalities are. Consider the example of Figure 4.3, since the upper bound involving each vertex is given by UBP, we can produce the following valid inequalities ($\ell = 2$):

– Vertex 1 (and also 4) leads to $2x_1 + x_2 + x_3 + 2x_4 + x_5 \leq 2$
– Vertex 5 leads to $2x_1 + x_2 + x_3 + x_4 + 2x_5 \leq 2$
– Vertex 6 leads to $2x_6 + x_7 + x_8 + x_9 + x_{10} \leq 2$
– Vertex 10 leads to $x_6 + x_7 + x_8 + x_9 + 2x_{10} \leq 2$

The LP relaxation of the original formulation (1)-(4) yields an objective of 2.5, and nearly all the variables are fractional. Adding these four inequalities yields an objective of 2 and an integer solution, which proves to be optimal.

4.3.6 A novel MBBP algorithm ExtUniBBClq

We observe that for any biclique (A, B) such that $A \subseteq U, B \subseteq V$ and $|A| \leq |B|$, the maximum balanced biclique in subgraph $G[A \cup B]$ is (A, B') with $B' \subseteq B$, and the maximum half-size of any (A, B') is still $|A|$. In other words, the half-size of the maximum balanced biclique in $G = (U, V, E)$ is the cardinality of maximum subset $A \subseteq U$ which satisfies $|\bigcap_{i \in A} N(i)| \geq |A|$. As a result, instead of building the two sets of balanced biclique alternatively, we can directly enumerate the eligible subset A from U (or B from V) such that $|\bigcap_{i \in A} N(i)| \geq |A|$. Based on this observation, we propose a new algorithm (Algorithm 4.5) which builds the maximum eligible subset from U (as $|U| \leq |V|$).

The framework of ExtUniBBClq is similar to Algorithm 4.3 except that ExtUniBBClq only builds the set A recursively such that (A, B) forms a biclique and $|A| \leq |B|$. Moreover, in ExtUniBBClq, we make use of the upper bound involving each vertex returned by UBP. Therefore, UBP has to be called before the start of ExtUniBBClq. For each call of ExtUniBBClq, a current set A, as well as the candidate set C_A that contains vertices that can be moved into A, and set B which is the common adjacent vertices of vertex in A (i.e., $B = \bigcap_{i \in A} N(i)$) are given. The algorithm initializes lb to 0 and begins from ExtUniBBClq$(G, \emptyset, U, V)$.

As in BBClq, in each call of ExtUniBBClq, a branch vertex v (with the maximum upper bound) is moved out from C_A (lines 9-10) and the algorithm goes to two branches: the branch where $v \in A$ before the end of current loop (lines 11-15) and another branch where $v \notin A$ in the next loop. In lines 11-12, when the upper bound associated with vertex v is not larger than the lower bound, ExtUniBBClq stops the current search immediately once ub_i is the largest upper bound of all vertices in C_A. In lines 13-15, the search goes on by expanding A' and rebuilding B'. Note that we filter out unpromising vertices from $N(v)$ which have an upper bound not larger than the lower bound. In the end of the loop, ExtUniBBClq is called to further enlarge set A. After it returns, the algorithm moves to the next loop, entering another branch with $v \notin A$. In lines 7-8, the search is stopped when C_A is not large enough to build a better solution. In lines 5 and 13, the *make_balance* procedure is called so that the final solution is a strictly balanced biclique of half-size lb.

At the start of each call to ExtUniBBClq, we update the lower bound if it is needed (lines 1-3) and terminate the current search if A is not eligible ($|A| \leq |B|$) or $|B|$ is not larger than the lower bound. For an efficient implementation, we pre-sort the array which represents U (the initial C_A) in ascending order of the upper bound involving each vertex. Consequently, the last element in the array will always be the vertex with the largest upper bound. We also sort the arrays representing V (the initial B) and $N(v)$ ($\forall v \in U$) in

Algorithm 4.5: ExtUniBBClq(G, A, C_A, B), a new B&B procedure for MBBP based on enumerating one vertex set.

```
Input: Graph instance $G = (U, V, E)$ , $A$ - the current subset of $U$, $C_A$ - the candidate subset of $U$, $B$
       - the common neighbors of vertices in $C$ ,i.e., $B = \bigcap_{i \in C} N(i)$
Output: A maximum balanced biclique of $G$
if $|A| \leq |B|$ AND $|A| > lb$ then
    $lb \leftarrow |A|$ ;
    Record current best biclique $(A^*, B^*)$;
if $|A| \geq |B|$ OR $|B| \leq lb$ then
    return $make\_balance(A^*, B^*)$
while $C_A \neq \emptyset$ do
    if $|A| + |C_A| \leq lb$ then
        return
    $v \leftarrow \text{argmax}_{i \in C_A} ub_i$;
    $C_A \leftarrow C_A \setminus \{v\}$;
    if $ub_v \leq lb$ then
        return
    $A' \leftarrow A \cup \{v\}$;
    $B' \leftarrow B \cap \{u \in N(v) : ub_u > lb\}$;
    ExtUniBBlq($G$, $A'$, $C_A$, $B'$)
return $make\_balance(A^*, B^*)$
```

ascending order of the index of each vertex so that the intersection operation in line 14 can be accomplished in linear time.

We illustrate the principle of this procedure by using the example of Figure 4.3 again. Firstly, the lower bound lb is initialized to 0 and ExtUniBBClq$(G, \emptyset, U, V)$ is then called. In the first `while` loop, vertex 2 is selected as the branch vertex as $ub_2 = 2$ is the largest upper bound of all vertices in U; then we get $A = \{2\}$, $C'_A = \{1, 4, 5, 3\}$, $B' = \{7, 8, 9\}$. The next call to ExtUniBBClq expands the incumbent set $A = \{2\}$, which leads to a solution $(\{2, 3\}, \{7, 8\})$ (and lb is updated to 2). The second `while` loop which branches on vertex 3 and builds candidate set $C'_A = \{1, 4, 5\}$, is stopped earlier as the largest upper bound ($ub_3 = 2$) is equal to lb. Thus, the whole search stops, returning $(\{2, 3\}, \{7, 8\})$ as the optimal solution.

4.3.7 Computational experiments

This section is dedicated to a computational evaluation of the proposed algorithms for MBBP, based on the Random graphs and KONECT networks (introduced in 1.4.1 in Chapter 1).

We compare the performance of 5 algorithms including both the existing approaches and the new approaches proposed in this work. The first 3 algorithms are B&B algorithms.

- **BBClq**: the algorithm introduced in [McCreesh and Prosser, 2014]. However, compared with the original algorithm, symmetry breaking and clique cover techniques are removed as they are irrelevant for bipartite graphs.
- **ExtBBClq**: the extended version of BBClq combining UBP with our new branching heuristic presented in Section 4.3.4
- **ExtUniBBClq**: the new algorithm introduced in Section 4.3.6.

To compare the original mathematical formulation and the tightened formulation presented in this work, we use IBM CPLEX 12.6.1 to solve the benchmark instances with both formulations.

- **Original**: the original mathematical formulation of MBBP from [Dawande *et al.*, 2001].
- **Tightened**: the formulation with the additional inequalities introduced in Section 4.3.5.

All the experiments are conducted on a computer with an Intel Xeon© E5-2670 processor (2.5GHz and

Table 4.4: Computational results of the 5 algorithms for the random graphs.

n	p	UBP		B&B Algorithms			MIP	
		time	iter	BBClq	ExtBBClq	ExtUniBBClq	Original	Tightened
50	0.1	0.00	3.1	0.00	0.00	0.00	4.02	**0.41**
50	0.3	0.00	2.8	0.02	**0.01**	**0.01**	4.62	**4.70**
50	0.5	0.00	3.3	0.45	**0.15**	0.3	**6.48**	11.52
50	0.7	0.00	3.0	24.19	**5.12**	11.60	8.49	**7.47**
50	0.9	0.00	3.2	10174.28(5)	**680.11**	4405.93(28)	0.39	**0.33**
100	0.1	0.00	3.8	0.01	**0.00**	**0.00**	30.26	**4.77**
100	0.3	0.00	3.2	0.96	**0.42**	0.78	**457.77**	510.60
100	0.5	0.01	3.3	118.97	**32.22**	52.75	7137.27	**4392.06**
100	0.7	0.01	3.1	[13.50-]	**9540.81**(17)	[13.73-]	[13.57-20.30]	[13.40-20.00]
100	0.9	0.00	2.8	[26.07-]	[25.37-]	[26.87-]	**9977.23**(6)	10358.21(4)
150	0.1	0.00	3.7	0.06	**0.04**	0.05	305.04	**225.67**
150	0.3	0.01	3.0	11.28	**3.44**	8.75	9953.88(15)	**9937.61**(14)
150	0.5	0.02	3.3	4716.49	**933.95**	2281.69	[8.86-28.38]	[8.89-28.46]
150	0.7	0.04	3.4	[14.73-]	[14.73-]	[15.00-]	[14.82-41.86]	[14.62-42.08]
150	0.9	0.05	3.3	[29.53-]	[27.93-]	[30.23-]	[35.04-55.58]	[34.71-55.54]
200	0.1	0.01	3.3	0.19	**0.07**	0.26	1725.24	**1239.80**
200	0.3	0.03	3.2	84.08	**21.95**	59.22	[6.00-38.44]	[6.00-22.52]
200	0.5	0.05	3.3	[9.97-]	**10761.34**(3)	[10.03-]	[8.95-55.68]	[9.05-50.76]
200	0.7	0.08	3.3	[15.30-]	[15.23-]	[15.93-]	[15.45-64.32]	[15.67-65.78]
200	0.9	0.11	3.3	[31.93-]	[30.10-]	[32.60-]	[38.70-79.19]	[38.21-79.52]

2GB RAM) running CentOS 6.5. The BBClq, ExtBBClq and ExtUniBBClq algorithms are implemented in C++ and compiled with g++ using optimization option `-O3` [2]. For each instance, a cut-off time limit of 3 hours (10800 seconds) is given to each trial. When solving the DIMACS machine benchmark procedure 'dfmax.c' [3] without compilation optimization flag, the run time on our machine is 0.46, 2.68 and 10.70 seconds for graphs r300.5, r400.5 and r500.5 respectively.

The experimental results for random graphs are summarized in Table 4.4. For each configuration which is a pairwise combination of $n \in \{50, 100, 150, 200\}$ (the cardinality of U or $|V|$) and $p \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ (the edge density), we generated 30 graphs independently. Column "time" reports the pre-processing time (the time of running UBP) and column "iter" shows the average number of `while` loops (i.e., lines 11-17, Algorithm 4.4) needed to stabilize the upper bound of all the vertices. For each algorithm, we report the average time consumed to solve the corresponding instance (the time for pre-processing is also included). Note that 0.00 means the instance can be solved in less than 0.01 second. If some of the 30 instances cannot be solved within 3 hours, we also append the number of solved instances in brackets. If none of the 30 instances can be solved, for the first three algorithms, we report the average best lower bound, for MIP, we report both the average best lower bound and average upper bound. The shortest times among the first three algorithms and between the two mathematical formulations are highlighted by bold font.

For the tested random graphs, the time consumption of UBP is insignificant with respect to the whole search time. Meanwhile, the number of iterations for propagating upper bounds is also trivial (closely around 3 for all the configurations). In terms of computational time of all the algorithms, ExtBBClq is generally the fastest algorithm to solve most of these instances while ExtUniBBClq also performs better than BBClq. The MIP formulation is found to be quite competitive compared with the other three algorithms when the density of the instance reaches 0.9. A possible explanation to this phenomenon is that the formulation of dense instances involves fewer constraints and may be solved more easily. In addition, when graphs

[2] The code of our algorithms will be available online.

[3] dfmax:ftp://dimacs.rutgers.edu/pub/dsj/clique/

Table 4.5: Computational results of the 5 algorithms for KONECT instances.

instance	BEST	UBP		B&B Algorithms			MIP	
		time	iter	BBClq	ExtBBClq	ExtUniBBClq	Original	Tightened
actor-movie	8*	5.54	27	6533.01	1671.29	807.25	-	-
bibsonomy-2ui	8*	1.56	7	491.36	13.84	**9.13**	-	-
bookcrossing_full-rating	13*	5.11	33	3102.66	**426.37**	[10-]	-	-
dblp-author	10*	19.86	21	[1-]	403.16	**30.06**	-	-
dbpedia-genre	7*	3.94	9	171.86	**5.83**	16.35	-	-
dbpedia-location	5*	0.18	8	633.98	0.52	**0.39**	-	-
dbpedia-occupation	6*	0.27	8	909.03	**1.29**	1.57	-	-
dbpedia-producer	6*	0.27	11	535.44	**0.62**	0.65	-	-
dbpedia-recordlabel	6*	24.33	7	214.45	24.67	**24.04**	-	-
dbpedia-starring	6*	1.07	31	530.61	4.67	**1.39**	-	-
dbpedia-team	6*	3.08	15	2982.24	**241.06**	1170.25	-	-
dbpedia-writer	6*	0.19	12	283.16	0.35	**0.23**	-	-
discogs_affiliation	26*	12.01	17	[1-]	**1688.95**	[18-]	-	-
discogs_lgenre	15*	0.06	1	37.08	1.01	**0.17**	-	-
discogs_style	38	17.42	22	[23-]	[38-]	[23-]	-	-
edit-dewiki	40	93.68	23	[1-]	[40-]	[14-]	-	-
edit-frwiktionary	19*	9.56	9	944.21	**152.5**	[19-]	-	-
escorts	6*	5.69	6	**7.68**	10.05	10.55	-	-
flickr-groupmemberships	36	47.37	36	[34-]	[36-]	[18-]	-	-
github	12*	1.01	16	677.72	**150.66**	[12-]	-	-
gottron-trec	83	549.21	35	[33-]	[38-]	[83-]	-	-
jester1	100*	1.86	1	1204.64	1123.66	**4.24**	248.87	-
moreno_crime	2*	0.05	3	**0.05**	0.06	0.06	3483.58	**55.22**
opsahl-ucforum	5*	0.09	10	0.18	**0.13**	0.26	[4-285]	[5-7]
pics_ut	27	21.84	7	[27-]	[23-]	[23-]	-	-
reuters	39	611.76	61	[35-]	[39-]	[12-]	-	-
stackexchange-stackoverflow	9*	4.62	29	4107.56	**265.8**	3690.8	-	-
unicodelang	4*	0.02	5	**0.01**	0.02	0.03	1218.46	**19.58**
wiki-en-cat	14*	7.67	20	[1-]	**28.72**	121.99	-	-
youtube-groupmemberships	12*	0.96	21	222.88	**11.49**	1784.76	-	-

density increases much, the maximum biclique and the maximum balanced biclique tend to be closer and closer, which means that the balancing constraint, which makes the problem NP-hard, tends to be less and less active. Since the maximum biclique problem is easy on bipartite graph, a balanced biclique is likely to be found easily by a MIP solver (as the quality of the linear programming relaxation improves with density) whereas high density is the worst situation for enumerative approaches. As expected, the tightened MIP is often solved faster than the original MIP, and the gap to optimality is generally less for those instances that could not be solved to optimality.

We report the results for a subset of 30 large KONECT instances in Table 4.5. Column “instance" indicates the name of graph. Column “best" shows the best half-size found by all algorithms. An extra “*" indicates the optimality of this best value. Column “UBP" also reports the time of pre-processing and number of iterations to propagate the upper bounds. For each algorithm, we report the computational time to solve the instances. As in Table 4.4, when optimality is not proven, the best lower bound is reported for the first three algorithm while for MIP, both lower bound and upper bound are presented.

For the large real-life instances (see Table 4.5), the time spent by UBP is still insignificant with respect to the total search time. The number of iterations is also limited in fewer than 40 for these very large instances. We note that the new algorithms with UBP (ExtBBClq and ExtUniBBClq) dominate the original algorithms in terms of computational time. The extended version ExtBBClq reduces the time of BBClq from hundreds of seconds to less than 30 seconds for 10 instances. It is also the only algorithm that solves discogs_affiliation. ExtUniBBClq is faster than ExtBBClq on 8 instances. It also achieves a substantial speed-up on jester-1 (where $|U| \ll |V|$). CPLEX is no longer able to give a lower bound for most instances (but we still observe that the tightened formulation leads to a better performance for the 2 solved instances).

4.3.8 Analysis

In this section, we compare the sizes of B&B trees generated by the algorithms to solve MBBP instances. Though in BBClq or ExtBBClq, there is no explicit declaration of B&B nodes, we can treat one call of BBClq() procedure as one enumeration of B&B node. In CPLEX, the number of nodes in the search tree is directly available. We exclude ExtUniBBClq as the search scheme and branching heuristic are different from BBClq and ExtBBClq.

Firstly, we generate 18 random instances with n fixing to 50 ($|U| = |V| = 50$) and p (density) ranging from 0.1 to 0.95, then we solve these instances by BBClq, ExtBBClq and CPLEX with the two formulations. Then we compare the number of B&B tree nodes between BBClq and ExtBBClq on the one hand, those generated by CPLEX with the original and tightened formulations on the other hand. The results are shown in Figure 4.4.

From Figure 4.4, we can observe that, the bounding technique and the extra inequalities significantly reduce the B&B tree size on sparse graphs (whose densities are below 0.2). When the density of random graph is lower than 0.9, ExtBBClq always enumerates fewer B&B nodes than BBClq. However, when the density increase to 0.9, the B&B tree of ExtBBClq has the same size as that of BBClq. Since we have improved the performance of intersection operation in the implementation of ExtBBClq (see Section 4.3.4), ExtBBClq still outperforms BBClq when the sizes of B&B trees are equal (see Table 4.4). As to the two mathematical formulations, CPLEX can solve the tightened formulation without expanding the B&B nodes when the graph is either quite sparse ($p = 0.1$) or very dense ($p = 0.95$).

Secondly, we compare the sizes of B&B trees for the real-life instances. Figure 4.5 shows the number of B&B tree nodes of BBClq and ExtBBClq for the 21 instances that can be solved by both algorithms in 3 hours (see Table 4.5). We no longer compare the two mathematical formulations as CPLEX fails to solve the majority of these large instances. Figure 4.5 indicates that ExtBBClq enumerates fewer tree nodes for all the real-life instances. This is especially true for dbpedia-producer, dbpedia-writer and moreno_crime, where ExtBBClq prunes more than half of the B&B nodes compared to BBClq.

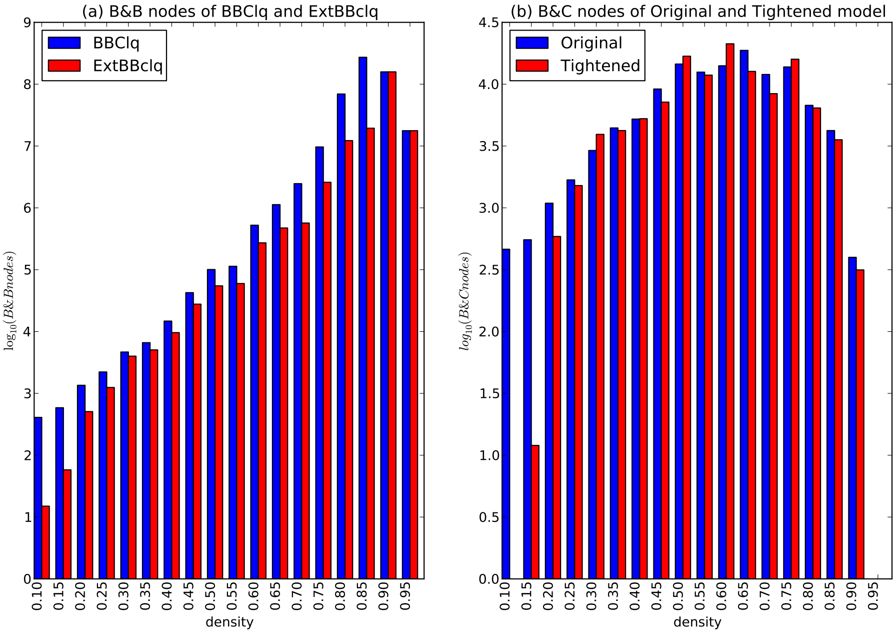

Figure 4.4: The base-10 log scale number of B&B tree nodes explored by BBClq, ExtBBClq, and CPLEX with the original and tightened formulations to solve the random graphs of different densities.

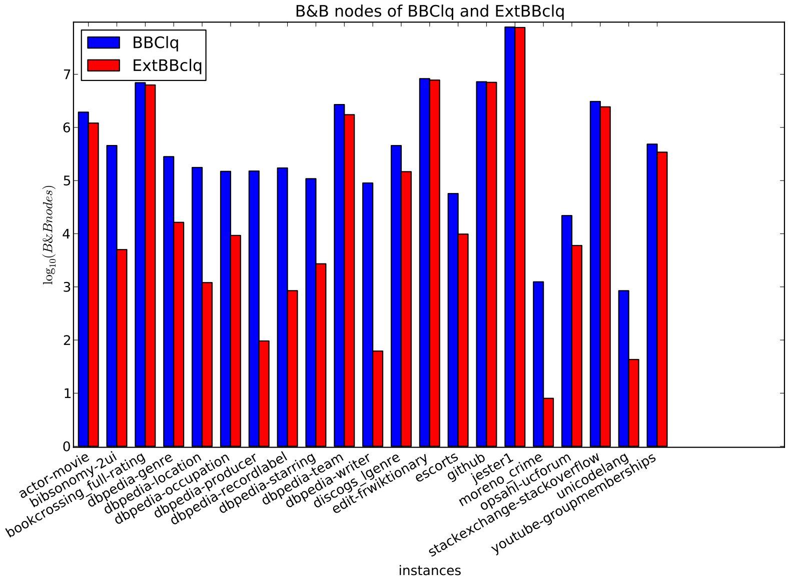

Figure 4.5: The base-10 log scaled number of B&B tree nodes explored by the BBClq and ExtBBClq algorithms for the 21 solvable instances.

4.4 Conclusion

The Maximum Balanced Biclique Problem is of great interest both theoretically and practically. We investigated a heuristic algorithm in the first part of this chapter and exact approaches in the second part. The heuristic algorithm (TSGR-MBBR) combines constraint-based tabu search (CBTS) with two graph reduction techniques. The *PUSH* operator, which was firstly introduced in Chapter 2, was also in employed and extended in the tabu search for MBBP. Meanwhile, distinguished from common local search routine, CBTS explores the relaxed search space including both balanced and unbalanced bicliques. An unbalanced constraint is employed to limit the search space in the following iterations. Besides CBTS, TSGR-MBBP uses $peel$ procedure and fast exact search algorithm [McCreesh and Prosser, 2014] to reduce the search space of very large graphs. We tested TSG-MBBP on 30 random instances and 25 KONECT instances from real-life applications. Comparison with existing algorithms (solvers) including EA/SM [Yuan *et al.*, 2015], GL_Greedy[Al-Yamani *et al.*, 2007], CPLEX indicates that the TSGR-MBBP is among the most competitive state-of-the-art algorithms. An interesting feature of TSGR-MBBP is that it can even prove optimality for large real-life instances thanks to the reduction techniques.

In the second part of this chapter, we investigated exact algorithms. A new pre-processing procedure called Upper Bound Propagation (UBP) is proposed for the first time. Since UBP provides tightened initial bound of each vertex, existing exact algorithm [McCreesh and Prosser, 2014] can be improved by using these tightened bounds. Based on UBP, we also discussed a new type of valid inequalities for MIP formulations in [Dawande *et al.*, 2001]. A new exact algorithm is proposed to enrich the exact algorithm family of MBBP. Experiments with random graphs and KONECT instances were carried out, which demonstrate the effectiveness of all these new proposed ideas.

5

A Three-Phased Local Search Approach for the Clique Partitioning Problem

This chapter presents a three-phased local search heuristic named CPP-P^3, for solving the Clique Partitioning Problem (CPP). CPP-P^3 iterates a descent search, an exploration search and a directed perturbation. We also define the *Top Move* of a vertex, in order to build a restricted and focused neighborhood. The exploration search is ensured by a tabu procedure, while the directed perturbation uses a GRASP-like method. To assess the performance of the proposed approach, we carry out extensive experiments on benchmark instances of the literature as well as newly generated instances. We demonstrate the effectiveness of our approach with respect to the current best performing algorithms both in terms of solution quality and computation efficiency. We present improved best solutions for a number of benchmark instances. Additional analyses are shown to highlight the critical role of the *Top Move*-based neighborhood for the performance of our algorithm and the relation between instance hardness and algorithm behavior. The content of this chapter is based on the work published in *Journal of Combinatorial Optimization*.

Contents

5.1 Introduction

Let $G = (V, E, W)$ be a complete edge-weighted undirected graph with a vertex set $V = \{v_1, v_2, ...v_n\}$, an edge set $E = \{\{u, v\} : u, v \in V, u \neq v\}$ and a set of edge weights $W = \{w_{uv} : w_{uv} \in \mathbb{R}, \{u, v\} \in E, w_{uv} = w_{vu}\}$. The Clique Partitioning Problem (CPP) consists in clustering all the vertices into k (unfixed) mutually disjoint subsets (or groups), such that the sum of the edge weights of all groups is as large as possible [Grötschel and Wakabayashi, 1989; Grötschel and Wakabayashi, 1990; Wakabayashi, 1986]. Formally, let $s = \{G_1, G_2, \ldots, G_k\}$ be such a partition of a given graph G such that $\bigcup_{i=1}^{k} G_i = V$ and $G_i \cap G_j = \emptyset$ $(\forall 1 \leq i < j \leq k)$. Let Ω denote the set of all partitions for G. CPP is then to find a partition $s^* \in \Omega$ that maximizes the following function:

$$f(s) = \sum_{p=1}^{k} \sum_{u,v \in G_p} w_{uv} \tag{5.1}$$

A mathematical formulation of CPP is presented in Chapter 1 and not repeated here. CPP has many of practical applications like biology, flexible manufacturing systems, airport logistics, and social sciences. Given the relevance of CPP, a number of solving procedures have been reported in the literature, including both exact and heuristic methods. Most of the exact methods follow the branch and bound framework [Dorndorf and Pesch, 1994; Jaehn and Pesch, 2013; Ji and Mitchell, 2007]) and cutting plane method [Grötschel and Wakabayashi, 1989]. However, exact methods may become prohibitively expensive when they are used to solve large instances. Consequently, various heuristic algorithms have been proposed to find high-quality solutions to large CPP instances with acceptable computation time. In [De Amorim *et al.*, 1992], tabu search [Glover and Laguna, 2013] and simulated annealing [Kirkpatrick *et al.*, 1983] were applied to this problem. Improved solutions were reported in [Brusco and Köhn, 2009] by using a reallocation heuristic and an embedded tabu search routine. In [Dorndorf *et al.*, 2008; Dorndorf and Pesch, 1994], an ejection chain heuristic was presented, borrowing the idea of classical Kernighan-Lin algorithm [Kernighan and Lin, 1970]. In [Charon and Hudry, 2001; Charon and Hudry, 2006], authors proposed a noising method which adds decreasing levels of noise to neighbor evaluations. More recently, two heuristic algorithms [Palubeckis *et al.*, 2014; Brimberg *et al.*, 2015] were presented. The work of [Palubeckis *et al.*, 2014] focuses on an iterated tabu search with tuned parameters. The approach of [Brimberg *et al.*, 2015] uses a generalized VNS algorithm designed for the *Maximally Diverse Grouping Problem* to solve CPP. These two last CPP algorithms integrate two neighborhood relations: (i) reallocating one vertex and (ii) swapping two vertices. Both of them performs quite well on the instances of [Brusco and Köhn, 2009] as well as large graphs with more than 1000 vertices. In the experimental section of this work, these two algorithms will be used as the main references for comparisons.

In this chapter, we introduce a novel heuristic algorithm, denoted by CPP-P^3, which uses a Three Phase Local Search framework combining a descent procedure, a tabu-based exploration search and GRASP-like perturbations. Different from previous local search algorithms, CPP-P^3 introduces a constrained *Top Move* neighborhood to make the search more focused and calls for a heap data structure to ensure a fast neighborhood examination. Computational experiments on four groups of graph instances (from 100 to 2000 vertices) show that CPP-P^3 competes favorably with the current best performing algorithms (including ITS [Palubeckis *et al.*, 2014] and SGVNS [Brimberg *et al.*, 2015]) both in terms of solution quality and computational efficiency. We also carry out additional analysis to gain insights about the behavior of the proposed algorithm.

The remaining part of this chapter is organized as follows. Section 5.2 describes the components of the CPP-P^3 algorithm and presents the concept of the *Top Move* neighborhood. Section 5.3 is dedicated to computational results and detailed comparisons with state-of-the-art algorithms. Section 5.4 investigates the effectiveness of the *Top Move* neighborhood and the hardness of problem instances by means of a landscape analysis. Conclusions are drawn in the last section of this chapter.

5.2 General procedure

The proposed CPP-P^3 method (see Algorithm 5.1) follows the general scheme of the Three-Phase Search (TPS) introduced in [Fu and Hao, 2015]. From a given starting solution, the first phase aims to reach a local optimal solution using a basic descent procedure. From this local optimum, the second phase is applied to discover nearby local optima of better quality within the current regional area of the search space. If no further improvement can be obtained, TPS switches to its third phase to perturb the incumbent solution to displace the search to a more distant area from which a new round of the three phased process is launched.

In this section, we first describe fundamental elements of the proposed procedure: the search space and the evaluation function (Section 5.2.1), the specific neighborhood induced by the *Top Move* concept (Section 5.2.2) and the associated heap structure (Section 5.2.3). Then, we present each subroutine of the Algorithm 5.1, i.e. the generation of initial solution and the three phases Descent_Search(), Explore_Local_Optima(), and Diversified_Perturbation(). We also discuss the main differences between CPP-P^3 and the existing heuristic algorithms in Section 5.2.7.

Algorithm 5.1: Pseudo-code of CPP-P^3

```
Input: A graph instance described by a weight matrix $G$.
Output: The best solution $s_{best}$ and its objective value $f_{best}$.
$s \leftarrow$ Build_Init_Solution() ;                    /* Section 5.2.4 */
$f_{best} \leftarrow 0$ ;
while stop condition is not met do
    $s \leftarrow$ Descent_Search($s$) ;                  /* Section 5.2.5 */
    $s \leftarrow$ Explore_Local_Optima($s$);             /* Section 5.2.6 */
    if $f(s) > f_{best}$ then
        $s_{best} \leftarrow s$;
        $f_{best} \leftarrow f(s)$;
$s \leftarrow$ Diversified_Perturbation($s$) ;            /* Section 5.2.7 */
return $s_{best}$
```

5.2.1 Search space and evaluation function

As previously mentioned, a solution s can be expressed as a partition of the set V of vertices into k mutually disjoint *groups* $\{G_1, G_2, ..., G_k\}$, such as $\bigcup_{i=1}^{k} G_i = V$, $G_i \cap G_j = \emptyset, \forall i \neq j$, and $\forall i, |G_i| > 0$. The number of groups $k \in \{1, \ldots, |V|\}$ is not fixed. The search space of a given CPP instance is composed of all such solutions and has a cardinality of $|\Omega| = \sum_{k=1}^{n} S(n, k)$, where $S(n, k)$ denotes the Stirling number of the second kind. Enumerating such a huge search space is obviously impractical, even on reasonably small graphs.

To evaluate the quality of a candidate solution $s \in \Omega$, we simply use the objective function f (see Equation (5.1)), which sums the weights of the edges whose end-points are inside the same group.

5.2.2 Top Move and restricted neighborhood

CPP-P^3 is a local search algorithm, which requires a neighborhood allowing the search to navigate through a given search space. The most intuitive way to transform one solution to a close solution for CPP is probably to reallocate one vertex from its current group to another group. Compared with other popular neighbor operators such as swapping two vertices or reallocating an edge, the size of the neighborhood of reallocating a vertex is relatively reasonable for complete graphs. Also, it is easy to observe that each solution can lead to any other solution by applying this reallocation operator. This property makes it possible for the algorithm to explore potentially the whole search space, depending on the selection criterion.

We now define the reallocation operator in a more formal way. Given an feasible solution (partition) $s = \{G_1, G_2, ..., G_k\}$, a neighbor of s is a solution s' which can be obtained after moving one vertex u from its original group G_u to another group G_t ($G_t \in s \cup \{\emptyset\} \setminus \{G_u\}$), including a potentially new group. When $G_t = \emptyset$, a new group $G'_t = G_t \cup \{u\} = \{u\}$ will be added to s' as the $(k+1)$-*th* group. On the other hand, if $G'_u = G_u \setminus \{u\} = \emptyset$, it will not be conserved in s' anymore.

The key concept related to our neighborhood is the *move gain*, which indicates the change of the objective value between solution s and a neighbor s'. In order to calculate the move gain as fast as possible, we firstly define the *potential* of vertex u relative to G_i, ($G_i \in s \cup \{\emptyset\}$):

$$p_{u,G_i} = \sum_{v \in G_i} w_{vu} \tag{5.2}$$

Given $u \in G_u$ in s, $move(u, G_t)$ the move which reallocates u from group G_u to G_t, the move gain Δ_{u,G_t} (the variation of objective f) of $move(u, G_t)$ can be incrementally computed by:

$$\Delta_{u,G_t} = f(s') - f(s) = p_{u,G_t} - p_{u,G_u} \tag{5.3}$$

In previous studies on CPP like [Brusco and Köhn, 2009; Charon and Hudry, 2006; De Amorim *et al.*, 1992; Palubeckis *et al.*, 2014], selecting an appropriate $move\ (u, G_t)$ requires the evaluation of all feasible neighboring solutions, i.e., by considering all reallocations of each vertex to each group including the empty one. Let $s \oplus move(u, G_t)$ designate the neighboring solution s' obtained by applying $move(u, G_t)$ to s, then this complete neighborhood $N(s)$ is defined by:

$$N(s) = \{s' \leftarrow s \oplus move(u, G_t) : \forall u \in V, \forall G_t \in s \cup \{\emptyset\} \setminus \{G_u\}\} \tag{5.4}$$

In order to make the search more focused and more efficient, our CPP-P^3 algorithm follows the idea of elite candidate list [Glover and Laguna, 2013] and builds a *restricted neighborhood* which is defined with the notion of *Top Move*.

Definition 5.2.1. A $move(u, G_\varsigma)$ is called *Top Move* of vertex u ($u \in G_i$) if

$$G_\varsigma = \underset{G_t \in s \cup \{\emptyset\} \setminus G_u}{\operatorname{argmax}} (\Delta_{u,G_t})$$

The target group G_ς is denoted as G_{TM_u}, while the move gain Δ_{u,G_ς} is denoted as Δ_{TM_u}.

So the *Top Move* of a vertex identifies the destination group with the largest move gain. Then the *restricted neighborhood* induced by the *Top Moves* of all vertices is given by:

$$N'(s) = \{s' \leftarrow s \oplus move(u, G_{TM_u}) : \forall\ u \in V\} \tag{5.5}$$

In the general case, $|N(s)| = k \times |V|$ for the complete neighborhood [1] while $N'(s) = |V|$ for the restricted neighborhood. Consequently, it is easier to evaluate the candidate solutions in $N'(s)$ than in $N(s)$ (we will show a comparison of using these two neighborhoods in Section 5.4). We present below a fast method to identify the *Top Move* of each vertex in each iteration of the algorithm.

Notice that in several recent studies [Chen and Hao, 2015; Wu and Hao, 2013a; Wu and Hao, 2013b], such a restricted neighborhood strategy has been shown to be particularly useful.

[1] If some groups contain exactly one vertex, then several moves lead to equivalent solutions. Thus, $|N(s)|$ can be slightly inferior to $k \times |V|$.

5.2.3 Heap structure

Let s be the incumbent solution. Suppose that $move(u, G_t)$ ($u \in G_u$) is chosen and executed to obtain the desired neighboring solution s', we update $G'_u = G_u \setminus \{u\}$, $G'_t = G_t \cup \{u\}$ as well as the potentials of each vertex $v \in V$ as follows:

$$\begin{cases} p'_{v,G'_u} = p_{v,G_u} - w_{uv}, \; G'_u = G_u \setminus \{u\} \\ p'_{v,G'_t} = p_{v,G_t} + w_{uv}, \; G'_t = G_t \cup \{u\} \end{cases} \tag{5.6}$$

To speed up the search and updating procedures, we store the potentials of each vertex in a heap structure. For a vertex u belonging to G_u in the current solution s, the potentials of u to the groups $s \cup \{\emptyset\} \setminus \{G_u\}$ are stored in the heap such that the top element $p_{u,G_{top}}$ of the heap is always the largest potential of vertex u. According to Equation (5.3), G_{top} will be the target group G_{TM_u}.

To maintain the heap, normally, when $move(u, G_t)$ ($u \in G_u$) is executed, p'_{v,G'_u} and p'_{v,G'_t} will replace p_{v,G_u} and p_{v,G_t} respectively and their positions in the heap of vertex v will be adjusted. However, to ensure potential $p_{v,\emptyset} = 0$ always be exclusively conserved in the heap, two exceptional situations should be considered. If $G'_u = \emptyset$ in s', p'_{v,G'_u} should be removed from the heap to avoid the repetition of $p_{v,\emptyset}$. If $G'_t = \{u\}$ (i.e., $G_t = \emptyset$), we should add a new potential $p'_{v,G'_t} = p_{v,\{u\}} = p_{v,\emptyset} + w_{uv}$, rather than replace $p_{v,\emptyset}$.

By this method, the *Top Move* of one vertex can be found in $O(1)$ time from the top of the heap. Moreover, updating the whole heap structure costs $O(|V| \cdot \log |V|)$.

5.2.4 Generation of initial solution

For the Clique Partitioning Problem, any partition of the vertices of G is a possible feasible solution. The question is then to fix the number of groups in the initial solution. In CPP-P^3, we simply assign each vertex to an exclusive group. Considering the initial solution will be immediately improved by the descent search, the quality of the initial solution is not essential in our case.

5.2.5 Descent search phase

The descent search phase aims at finding new solutions of better quality from a given initial solution. For this purpose, it builds a search trajectory by examining the restricted neighborhood defined in Section 5.2.2. More precisely, the descent iteratively improves the quality of the current solution by searching the given neighborhood and stops when a local optimum is reached, i.e., when no better solution can be found in the neighborhood. At each iteration, vertices are evaluated in a random order. Let u be the vertex under consideration and s be the current solution. If the *Top Move* gain Δ_{TM_u} of vertex u is a positive number (i.e., the Top Move with vertex u leads to an improved solution with respect to the incumbent solution), $move(u, G_{TM_u})$ is executed and the neighboring solution $s \oplus move(u, G_{TM_u})$ replaces s to become the new incumbent solution. Otherwise the evaluation continues with the next vertex (always randomly chosen). If no *Top Move* offers a positive gain during one iteration after examining all the vertices, then the descent search stops and the last solution (i.e., a local optimum) is returned and used as the starting solution of the second search phase (Exploration Search, see next section).

Note that within the restricted neighborhood N', the way the neighboring solutions are examined and selected corresponds to the *random-order first improvement* search strategy. Clearly, the selected neighbor using this strategy is not necessarily the best neighbor within the restricted neighborhood N' nor within the complete neighborhood N.

The computing time of each iteration is bounded by $O(|V| + |V| \cdot \log |V|)$.

5.2.6 Exploration search phase

From the local optimum solution returned by the descent search phase, the second search phase explores the nearby areas for better solutions. For this purpose, the Explore_Local_Optima procedure (Algorithm 5.2) adopts a tabu search method [Glover and Laguna, 2013] also based on the above restricted neighborhood.

At each iteration, a Top Move $move(u, G_{TM_u})$ is considered to be eligible only if two conditions are satisfied. First, the move must be a non-trivial one (i.e., reallocating a vertex from a group with only one vertex to an empty set is forbidden). Second, the move is not forbidden by the tabu list except when the move leads to a new solution better than any previous visited solution. Among these eligible moves, the one with the largest move gain is chosen and executed.

Algorithm 5.2: Tabu-based exploration phase (Explore_Local_Optima).

```
Input: current solution $s$, current best objective value $f_{best}$.
Output: best solution $s^*$ .
Initialize the tabu list;
$iter \leftarrow 0$ ;                                /* total number of iterations */
while $f_{best}$ has not been improved for $L$ iterations. do
    Find out vertex $w$ with the maxium $\Delta_{TM_w}$from the vertices which meets the following 2 conditions:
    1. ($|G_u| = 1$ AND $|G_{TM_u}| \neq \emptyset$) OR ($|G_u| > 1$)
    2. ($move(u, G_{TM_u})$ is not TABU) OR ($\Delta_{TM_u} + f(s) > f_{best}$)
    if $|G_w| = 1$ then
        Mark $move(w, \emptyset)$ tabu for the next $tt$ iterations;
    else
        Mark $move(w, G_w)$ tabu for the next $tt$ iterations;
    Execute $move(w, G_{TM_w})$ and update data structure ;
    if $f(s) > f_{best}$ then
        $s^* \leftarrow s$, $f_{best} \leftarrow f(s)$;
    $iter \leftarrow iter + 1$;
```

To avoid short-term cycling, the executed $move(w, G_{TM_w})$ as well as the index of the original group of vertex w are stored in the tabu list. As such, vertex w is prohibited to move back to its original group during the next tt iterations (tt is called the *tabu tenure*). However, if the vertex of the executed move is the only vertex in its original group, then the vertex will be forbidden to be moved into the empty group for the next tt iterations.

It is well known that the performance of a tabu search algorithm depends on the way the tabu tenure is determined [Glover and Laguna, 2013]. However, no optimal technique is universally available to tune the tabu tenure, which is often fixed empirically in practice. In our case, we adopt the technique proposed in [Galinier and Hao, 1999] and use a base length $tbase$ adjusted with a random value as follows:

$$tt = tbase + random(0, k) \tag{5.7}$$

where $random(0, k)$ is a random integer in $\{0, \dots, k\}$ (k being the number of groups of the current solution). After preliminary robustness tests (see Section 5.3.1), we set $tbase$ to 15.

For this exploration search phase, the algorithm is supposed to reach a local optimum if the best solution cannot be improved during L consecutive iterations. In this case, the Explore_Local_Optima phase stops and the best solution found is used as the input solution of the Diversified_Perturbation phase.

5.2.7 Directed perturbation phase

The perturbation phase (see Algorithm 5.3) aims to jump out of the current regional search area and displace the search into a distant area. Instead of relying on random perturbation (e.g., randomly move some vertices), CPP-P^3 uses a *directed perturbation* mechanism [Benlic and Hao, 2013b].

Algorithm 5.3: Perturbation phase (Diversified_Perturbation).

```
Input: Current solution $s$.
Output: perturbated solution $s^*$.
$PL \leftarrow random([\alpha \times |V|, \beta \times |V|])$;
$M \leftarrow \emptyset$ ;
while $|M| < PL$ do
    Construct $RCL = \{$The Maxrcl vertices in $V \setminus M$ with the greatest $\Delta_{TM_u}\}$ ;
    Randomly choose a vertex $w$ from $RCL$;
    Execute $move(w, G_{TM_w})$ and update data structure;
    $M \leftarrow M \cup \{w\}$;
return $s$
```

The perturbation concerns reallocating a number of specific vertices from a *Restricted Candidate List* (RCL). For each perturbation phase, the number of reallocated vertices, which defines the perturbation strength, is randomly selected from range $\{\alpha \times |V|, \ldots, \beta \times |V|\}$, where α and β ($0 \leq \alpha \leq \beta \leq 1$) are two parameters.

To determine the vertices that will be reallocated during the perturbation phase, we build an RCL to store the partial best candidates, like with the GRASP method [Feo and Resende, 1995b]. Here, the RCL contains the *Maxrcl* first vertices with the largest *Top Move* gain. For each perturbation step, a vertex w is randomly chosen from the RCL and the corresponding *Top Move* is executed. Once a vertex is reallocated, it is removed from RCL during the current perturbation phase.

To implement the RCL efficiently, we employ once again a heap structure to make sure that it always contains the best *Maxrcl* vertices and ignores the other ones. Thus, the construction of the RCL in each iteration takes $O(V \cdot \log Maxrcl)$ time, while updating potentials consumes $O(V \cdot \log V)$ after one move. The complexity of one iteration in each perturbation phase is bounded by $O(V \cdot \log V)$.

5.2.8 Singularity of CPP-P^3

We now discuss the major differences between CPP-P^3 and other local search algorithms for CPP. ITS [Palubeckis *et al.*, 2014] may be the closest approach for solving CPP as it also integrates a descent search, a tabu search and a perturbation procedure. However, there are three main differences between the two algorithms. First, ITS alternates two neighbor operators, i.e. reallocating a vertex and swapping two vertices, while only the first operator is considered in CPP-P^3. Second, ITS evaluates all the neighbor solution induced by the reallocating operators to determine the neighbor solution with the maximum objective gain while CPP-P^3 seeks the best neighbor solution from a restricted solution set associated with *Top Move* definition. Third, the two algorithms employ different strategies for managing the tabu tenure. The value in ITS is more deterministic while CPP-P^3 is more randomized. As we show in Section 5.3, these differences make the proposed CPP-P^3 more effective than ITS.

CPP-P^3 also shares similarities with Eject Chain algorithm [Dorndorf and Pesch, 1994] in the sense that both approaches use a heap structure to identify the move. In [Dorndorf and Pesch, 1994], the move with the largest objective gain is always preferred, while in the tabu phase of CPP-P^3, the vertex associates with the best *Top Move* is selected. Therefore, in CPP-P^3, the moves whose associated *Top Move* is marked tabu will not be considered, even though such a move may lead to the best gain of objective value. We investigate in Section 5.4.1 the effect of both strategies and demonstrate the interest of the strategy we adopted in CPP-P^3.

5.3 Computational experiments

This section is dedicated to an experimental assessment of CPP-P^3. For this purpose, we show extensive computational results obtained by CPP-P^3 on a large collection of benchmark instances. We also make a

Table 5.1: Parameter settings of testing CPP-P^3.

Parameters	§	Description	Values	Range
$tbase$	5.2.6	Tabu tenure base	15	{7, 10, 15, 20}
L	5.2.6	Maxium consecutive non-improving iterations	$\|V\|$	$\{0.5\|V\|, \|V\|, 1.5\|V\|, 2\|V\|\}$
α	5.2.7	Lower limit of perturbation strength	0.2	{0.1, 0.2, 0.3}
β	5.2.7	Upper limit of perturbation strength	0.5	{0.4, 0.5, 0.6}
$Maxrcl$	5.2.7	Maximum length of RCL	10	{7, 10, 15, 20}

comparison with the current best performing algorithms published in the literature.

5.3.1 Benchmark instances and parameter settings

As the instances considered in CPP are complete graphs, the differences between instances only concern the value range and distribution of the edge weights. We use the four groups of instances presented in Chapter 1. As stated, the first three groups of instances were originally proposed in [Brusco and Köhn, 2009; Charon and Hudry, 2006; Palubeckis *et al.*, 2014], while the fourth group was created by us with different edge weight distribution.

CPP-P^3 involves 5 parameters whose settings indicated in Table 5.1 are used for all our experiments. In order to estimate an appropriate value for these parameters, we first fixed a set of possible values for each parameter as indicated in column $Range$. Experiments with all these values have been made on a preliminary set of instances; values which achieved a good compromise in terms of best and average objective values as well as CPU time have been kept. Of course, fine-tuning these parameters would allow us to obtain better results. However, as we show in this section, the adopted parameter values of Table 5.1 lead already to highly competitive results.

5.3.2 Experiments and comparison

The CPP-P^3 algorithm was implemented in C++ [2], compiled by GNU g++ and run on 2.82GHz Xeon CPU with 2 GB RAM. The experiments were conducted on a computer with an AMD Opteron 4184 processor (2.8GHz and 2GB RAM) running Ubuntu 12.04. When solving the DIMACS machine benchmarks [3] without compilation optimization flag, the run time on our machine is 0.40, 2.50 and 9.55 seconds respectively for graphs r300.5, r400.5 and r500.5. We used the CPU clock as the stop condition of CPP-P^3. In the following experiments, in order to get relative stable results, we fixed cut-off times which refer to the order (i.e., the number of vertices) of the graphs. Each instance was solved 10 times independently using different random seeds.

As mentioned in the introduction, ITS [Palubeckis *et al.*, 2014] and SGVNS [Brimberg *et al.*, 2015] are the most recent and also the best performing algorithms among all previously developed heuristics. Consequently, we used ITS and SGVNS as the main references for our comparative study. To make a fair comparison between the three algorithms, we ran them under the same conditions, i.e. the same cut-off time for each instances and the same testing platform. The source code of ITS is publicly available from `http://www.proin.ktu.lt/~gintaras/`, and the code of SGVNS was kindly provided by the authors of [Brimberg *et al.*, 2015].

Computational results on instances of Groups I and II

[2] The source code will be available upon publication of the paper at: `http://www.info.univ-angers.fr/pub/hao/cpp.html`.

[3] dimacs: `ftp://dimacs.rutgers.edu/pub/dsj/clique/`.

Table 5.2: Computation results on instances of Groups I and II of our CPP-P^3, ITS [Palubeckis *et al.*, 2014] and SGVNS algorithms [Brimberg *et al.*, 2015]

Instance	f_{prev}	CPP-P^3				ITS				SGVNS			
		f_{best}	f_{avg}	hit	$time$	f_{best}	f_{avg}	hit	$time$	f_{best}	f_{avg}	hit	$time$
rand100-5	1407	1407	1407.00	10/10	0.05	1407	1407.00	10/10	0.10	1407	1407.00	10/10	0.54
rand100-100	24296	24296	24296.00	10/10	0.74	24296	24296.00	10/10	0.90	24296	24296.00	10/10	1.87
rand200-5	4079	4079	4079.00	10/10	6.57	4079	4079.00	10/10	17.80	4079	4078.80	9/10	31.30
rand200-100	74924	74924	74924.00	10/10	20.33	74924	74924.00	10/10	23.80	74924	74924.00	10/10	94.39
rand300-5	7732	7732	7732.00	10/10	36.01	7732	7730.50	5/10	169.50	7732	7731.40	8/10	231.48
rand300-100	152709	152709	152709.00	10/10	5.18	152709	152709.00	10/10	14.50	152709	152709.00	10/10	39.51
sym300-50	17592	17592	17592.00	10/10	69.53	17592	17590.00	9/10	101.00	17592	17591.40	9/10	438.21
regnier300-50	32164	32164	32164.00	10/10	0.90	32164	32164.00	10/10	1.20	32164	32164.00	10/10	1.56
zahn300	2504	2504	2504.00	10/10	6.21	2504	2504.00	10/10	15.90	2504	2504.00	10/10	32.69
rand400-5	12133	12133	12133.00	10/10	94.33	12133	12130.30	7/10	430.00	12133	12133.00	10/10	677.03
rand400-100	222757	222757	222757.00	10/10	170.32	222757	222690.80	5/10	356.90	222757	222725.20	8/10	522.16
rand500-5	17127	17127	17126.50	9/10	258.50	*17114*	17104.90	2/10	608.70	17127	17124.10	8/10	793.90
rand500-100	309125	309125	308985.50	4/10	205.39	*309007*	308860.90	1/10	406.70	*308984*	308860.30	1/10	1320.87

Table 5.2 shows the computational results of three algorithms on Group I and Group II instances. Column 1 and 2 provide instance name and best known objective value f_{prev} which are extracted from [Brusco and Köhn, 2009; Charon and Hudry, 2006]. The same time limits for each algorithm are fixed as follows: 200, 500, and 1000 seconds for graphs with vertices in range [1,200], [201,300], and [301,500] respectively.

For each instance and each algorithm (CPP-P^3, ITS and SGVNS), we report the best objective value attained f_{best} over 10 runs, the average objective value f_{avg}, the frequency hit of reaching f_{best} over 10 runs, as well as the average CPU $time$ (in seconds) for finding the best value.

In Table 5.2, one observes that in terms of solution quality, CPP-P^3 is able to hit all the previous best results while both ITS and SGVNS fail on the two largest instances rand500-5 and rand500-100. While comparing the performance robustness over 10 runs, we note that the average objective value of CPP-P^3 is better than the competing algorithms on all instances. Moreover, CPP-P^3 consistently reaches the previously best-known f_{prev} at each of 10 runs on 11 instances over 13, contrary to the ITS (7/13) and SGVNS (7/13). Finally, CPP-P^3 spends less average time than the other two algorithms to hit the best solutions. This experiment indicates that CPP-P^3 dominates the two references algorithms on instances from Groups I and II.

Computational results on instances of Group III

Table 5.3: Computation results on instances of Group III of our CPP-P^3, ITS [Palubeckis *et al.*, 2014] and SGVNS algorithms [Brimberg *et al.*, 2015]

Instance	f_{prev}	CPP-P^3				ITS				SGVNS			
		f_{best}	f_{avg}	hit	$time$	f_{best}	f_{avg}	hit	$time$	f_{best}	f_{avg}	hit	$time$
p500-5-1	17691	17691	17690.10	9/10	140.32	17691	17683.30	2/10	864.08	*17680*	17673.40	4/10	562.20
p500-5-2	17169	17169	17168.10	7/10	300.22	17169	17152.90	1/10	406.80	*17166*	17154.70	1/10	705.73
p500-5-3	16815	**16816**	16815.20	3/10	326.05	**16816**	16812.50	1/10	623.80	*16810*	16805.10	1/10	1845.85
p500-5-4	16808	16808	16808.00	10/10	141.59	16808	16790.90	2/10	473.00	16808	16800.60	6/10	598.65
p500-5-5	16957	16957	16957.00	10/10	99.00	16957	16948.50	6/10	595.00	16957	16943.80	2/10	446.62
p500-5-6	16615	16615	16615.00	10/10	185.04	16615	16606.80	3/10	482.20	16615	16609.00	3/10	481.48
p500-5-7	16649	16649	16648.80	9/10	285.55	16649	16632.70	1/10	547.80	*16647*	16632.20	1/10	314.10
p500-5-8	16756	16756	16756.00	10/10	156.56	16756	16752.30	4/10	376.90	16756	16748.30	9/10	590.98
p500-5-9	16629	16629	16629.00	10/10	161.42	16619	16607.60	2/10	526.50	16629	16623.80	5/10	549.73
p500-5-10	17360	17360	17360.00	10/10	51.52	17360	17356.80	8/10	547.40	17360	17359.80	8/10	430.33
p500-100-1	308896	308896	308896.00	10/10	317.40	308896	308810.90	2/10	481.90	308896	308890.60	4/10	741.65
p500-100-2	310163	**310241**	310225.40	8/10	367.47	**310241**	309843.40	1/10	530.50	**310241**	310170.80	1/10	449.71
p500-100-3	310477	310477	310471.10	9/10	267.23	310477	310028.60	2/10	555.10	*310418*	310259.20	3/10	690.50
p500-100-4	309567	309567	309528.60	9/10	376.93	*309494*	309193.80	1/10	412.00	309567	309411.60	1/10	359.16
p500-100-5	309135	309135	309116.90	9/10	286.63	309135	308998.60	2/10	639.20	309135	309116.90	9/10	897.59
p500-100-6	310280	310280	310280.00	10/10	251.93	310280	309973.70	6/10	482.60	310280	310218.50	8/10	353.39
p500-100-7	310063	310063	310063.00	10/10	153.16	310063	310029.40	5/10	479.90	310063	309992.70	3/10	342.03
p500-100-8	303148	303148	303148.00	10/10	284.58	303148	302830.10	5/10	580.00	303148	302774.30	5/10	737.08
p500-100-9	305305	305305	305305.00	10/10	81.30	305305	305236.40	8/10	497.50	305305	305299.80	9/10	623.31
p500-100-10	314864	314864	314864.00	10/10	43.99	314864	314815.30	5/10	392.90	314864	314818.20	5/10	492.15
p1000-1	883359	**884970**	884348.68	1/10	1221.80	*881855*	879382.10	1/10	1215.10	***885016**	884126.10	2/10	2125.72
p1000-2	879792	**881149**	880131.87	1/10	1212.22	*878854*	875849.90	1/10	1218.00	***881751**	880189.30	1/10	3424.15
p1000-3	862969	***866415**	864718.62	1/10	1058.54	**864309**	860987.90	1/10	1376.00	**866041**	864874.20	1/10	809.87
p1000-4	865754	**868573**	867295.37	1/10	885.94	**865866**	862984.70	1/10	1207.20	***869374**	867669.00	2/10	1347.60
p1000-5	887314	**888678**	887997.00	1/10	1257.44	*887066*	883768.50	1/10	876.50	***888720**	887783.00	1/10	3487.01
p1500-1	1614791	**1616999**	1614847.50	1/10	2711.04	*1608761*	1604753.00	1/10	2489.00	***1618281**	1615019.10	1/10	3300.02
p1500-2	1642442	***1648800**	1646916.12	1/10	2313.66	*1636856*	1631070.60	1/10	2643.40	**1647557**	1644372.50	1/10	4769.19
p1500-3	1600857	***1609854**	1607426.12	1/10	2991.33	*1600451*	1595268.00	1/10	2977.80	**1608877**	1605009.60	1/10	2328.56
p1500-4	1633081	**1639528**	1637375.12	1/10	1867.77	*1628349*	1626165.90	1/10	2552.20	***1640643**	1636222.90	1/10	1091.37
p1500-5	1585484	**1592732**	1590291.25	1/10	1791.04	*1580965*	1575690.70	1/10	2585.80	***1593518**	1591043.60	1/10	3567.00
p2000-1	2489880	***2505359**	2500050.00	1/10	5238.93	*2487048*	2480720.40	1/10	7710.90	**2501962**	2498417.40	1/10	3127.15
p2000-2	2479127	**2490104**	2488016.75	1/10	5997.60	*2473691*	2464960.90	1/10	6750.60	***2492662**	2485936.90	1/10	4925.60
p2000-3	2527119	***2540063**	2536091.50	1/10	6990.83	**2528777**	2517090.50	1/10	7639.30	**2538196**	2535414.30	1/10	1731.34
p2000-4	2523884	***2525903**	2521449.00	1/10	5992.97	*2509080*	2502059.40	1/10	6607.00	**2522156**	2516375.90	1/10	2730.68
p2000-5	2499690	***2508729**	2504678.00	1/10	5022.83	*2496036*	2487416.50	1/10	7239.00	**2506784**	2503396.90	1/10	1945.23

Table 5.3 shows comparative results among CPP-P^3, ITS and SGVNS on the instances of Group III. The same information as in Table 5.2 is provided. The f_{best} entries which are superior to f_{prev} and inferior to f_{prev} are marked respectively in bold and italic. Moreover, a star indicates a stricly f_{best} value among the three algorithms. Larger instances with more than 1000 vertices are included. The time limit was set to 1000, 2000, 4000 and 10000 seconds for instances with 500, 1000, 1500 and 2000 vertices respectively. Note that these time limits are shorter than those used in [Palubeckis *et al.*, 2014].

Table 5.3 discloses that CPP-P^3 outperforms ITS and SGVNS on instances of type 'p500' (i.e., with $|V| = 500$) both in terms of solution quality and computation time. We observe that CPP-P^3 reaches every f_{prev} while ITS and SGVNS fail respectively on 1 and 5 instances. Note that the best-known objective value of two instances (p500-5-3 and p500-100-2) is improved by CPP-P^3 and ITS. Another noticeable point is that CPP-P^3 achieves a 100% successful rate (hit) on 12 instances over 20, while considering ITS and SGVNS the corresponding robustness indicator is often low and never maximal on any instance. One also observes that the average time of CPP-P^3 is always shorter except slightly longer on p500-100-4 comparing to SGVNS.

Larger instances (with $|V| \geq 1000$) are unsurprisingly much difficult to tackle in a reasonable time limit, as confirmed by the less robust results: each algorithm hits the respective best solution only one or two times over 10 runs under the given time limit. The essential point is that all f_{prev} are improved by CPP-P^3 and SGVNS. for these two algorithms even their average scores f_{avg} are better than the previous best-known values. ITS also improves f_{prev} on three instances (p1000-3, p1000-4 and p2000-3), but systematically less than CPP-P^3 and SGVNS. Comparing the f_{best} (resp. f_{avg}) entries, CPP-P^3 reached better solutions on 7 (resp. 10) instances, and SGVNS on 8 (resp. 5). Contrary to the previous sets of instances where CPP-P^3 was the most powerful algorithm, it appears that on this particular set, CPP-P^3 and SGVNS perform similarly.

Computational results on instances of Group IV

As Group IV instances are newly created, no previous results are available. To obtain reasonable reference values f_{prev}, each instance is first solved by CPP-P^3 for 8 hours and the objective value of the best solution found is set as f_{prev}. Then, we run CPP-P^3, ITS and SGVNS on all the instances of this group under a cutoff limit of 1000 seconds. The computational results are given in Table 5.4.

We first examine the *uniform* graphs ('unif700' and 'unif800'). Recall that 1000 seconds are sufficient for CPP-P^3 to reach a stable objective value for random instances of size 500 (Group III, see table 5.3). Here, one can note that hit rates are decreasing when the number of vertices grows to 700 and 800. Nevertheless, CPP-P^3 and SGVNS are able to hit f_{prev} at least once for these 10 instances while ITS fails to match this performance.

The edge weights of the last five instances (of type "gauss500") are generated according to a Gaussian distribution. Comparing the results with the instances of Group III (p500-100-1 to p500-100-10), we observe that the hit and $time$ indicators, for each algorithm, are similar. This may imply that the distribution has no significant influence on the performance of these local search based algorithms.

Table 5.4: Computation results on new instances of our CPP-P^3, ITS [Palubeckis *et al.*, 2014] and SGVNS algorithms [Brimberg *et al.*, 2015]

Instance	f_{prev}	CPP-P^3				ITS				SGVNS			
		f_{best}	f_{avg}	*hit*	*time*	f_{best}	f_{avg}	*hit*	*time*	f_{best}	f_{avg}	*hit*	*time*
unif700-100-1	515016	515016	514768.40	6/10	768.90	*514689*	513368.50	1/10	965.40	515016	514291.50	2/10	322.68
unif700-100-2	519441	519441	518815.20	6/10	571.39	*519141*	517377.70	1/10	1055.60	519441	518265.40	3/10	2673.35
unif700-100-3	512351	512351	511457.70	3/10	749.91	512351	510002.50	1/10	1094.30	512351	511876.90	7/10	2033.99
unif700-100-4	513582	513582	513540.80	9/10	755.93	*511772*	510861.60	1/10	859.80	513582	513279.30	8/10	1328.62
unif700-100-5	510387	510387	510244.50	1/10	759.50	*510234*	509162.70	1/10	963.80	510367	509985.10	3/10	1026.23
unif800-100-1	639675	639675	639605.50	4/10	1033.06	*639307*	637927.40	1/10	879.90	639675	639373.70	3/10	1101.74
unif800-100-2	630704	630704	630488.00	2/10	1248.86	*630088*	628776.30	1/10	1203.50	630704	630034.00	2/10	882.15
unif800-100-3	629108	629108	628819.60	2/10	821.97	*628130*	627045.40	1/10	1267.70	629108	628740.50	1/10	3055.54
unif800-100-4	624728	624728	624153.20	1/10	1140.46	*623905*	622172.00	1/10	1130.60	624728	624165.50	2/10	2717.19
unif800-100-5	625905	625905	625378.40	1/10	1097.96	*625066*	623177.00	1/10	1336.60	625905	625355.90	2/10	639.25
gauss500-100-1	265070	265070	265049.10	9/10	410.31	265070	264741.50	1/10	554.10	265070	264882.40	2/10	964.81
gauss500-100-2	269076	269076	269039.20	7/10	469.05	269076	268815.20	2/10	512.60	269076	268953.10	4/10	967.99
gauss500-100-3	257700	257700	257467.40	3/10	343.48	257700	257205.30	1/10	485.20	257700	257444.50	2/10	871.46
gauss500-100-4	267683	267683	267633.70	9/10	251.65	267683	267266.90	4/10	424.10	267683	267589.50	4/10	458.60
gauss500-100-5	271567	271567	271567.00	10/10	94.08	271567	271485.40	5/10	576.80	271567	271539.60	7/10	638.76

Table 5.5: Comparison results of CPP-P^3-X and CPP-P^3

Instance	CPP-P^3-X					CPP-P^3				
	f_{best}	$f_{avg}(\sigma)$	hit	$time$	$iter_{avg}$	f_{best}	$f_{avg}(\sigma)$	hit	$time$	$iter_{avg}$
rand300-5	**7732**	**7732.0**(0.0)	10/10	64.29	26321	**7732**	**7732.0**(0.0)	10/10	33.82	33812
rand500-100	309007	308891.3(38.8)	1/10	131.60	15226	**309125**	**308935.6**(98.4)	2/10	259.65	21499
p500-5-3	**16815**	16814.2(0.4)	2/10	274.24	17011	**16815**	**16815.0**(0.0)	10/10	373.79	23100
gauss500-100-3	**257700**	257264.4(229.6)	1/10	500.59	13851	**257700**	**257558.6**(148.0)	3/10	245.04	21417
unif700-100-2	**519441**	518482.6(1026.3)	5/10	728.18	13919	**519441**	**519441.0**(0.0)	10/10	907.01	21289
unif800-100-4	624646	623951.8(383.2)	2/10	932.25	12099	**624728**	**624164.3**(372.0)	2/10	1001.97	15067
p1000-1	**884861**	883068.0(1133.7)	1/10	1111.76	6504	884709	**883565.0**(854.1)	1/10	921.53	9970
p1500-3	1607199	1604345.2(2026.0)	1/10	1595.41	4234	**1608549**	**1604854.2**(1568.4)	1/10	2549.95	4051
p2000-1	2504261	2489626.0(6067.3)	1/10	6593.32	1536	**2506312**	**2500859.4**(3420.5)	1/10	6256.35	9875
p2000-4	2517390	2510779.3(4151.5)	1/10	6405.45	1363	**2522665**	**2519964.9**(1835.7)	1/10	5761.96	4008

5.4 Analysis

5.4.1 The effectiveness of Top Move based neighborhood

One of the most crucial features of a local search algorithm is the definition of its neighborhood and the selection criterion used. Contrary to the previous algorithms like [Brusco and Köhn, 2009; Charon and Hudry, 2006; De Amorim *et al.*, 1992; Palubeckis *et al.*, 2014], CPP-P^3 is based on a different neighborhood definition. Indeed, while other methods consider all the possible moves for each vertex (see $N(s)$ in Section 5.2.2), CPP-P^3 only considers a restricted neighborhood defined by the *Top Move* of each vertex (see $N'(s)$ in Section 5.2.2).

To evaluate the most accurate strategy, we defined another algorithm, CPP-P^3-X which replaces $N'(s)$ in CPP-P^3 by the complete neighborhood $N(s)$ and keeps the other elements of CPP-P^3 unchanged. Then, we selected 10 instances of different sizes from the benchmarks (see Table 5.5) and ran CPP-P^3-X as well as CPP-P^3 under the same conditions as described in Section 5.3.2. To give a comprehensive comparison between the two methods, we report in Table 5.5, for each instance, the best objective value f_{best} obtained over 10 runs, the average objective value f_{avg} with the standard deviation σ, and the average $time$ needed to attain a best objective value. We define one pass of three phases of the algorithms as one iteration, and the average number of iterations over 10 runs is given in column $iter_{avg}$.

Experimental results show that CPP-P^3 globally outperforms CPP-P^3-X. One can observe that CPP-P^3 completes more iterations than CPP-P^3-X for the same time limit, which means that CPP-P^3 spends shorter times to find an appropriate move during one iteration. If we compare the results of CPP-P^3-X with the results of the ITS algorithm reported in the last section, one finds that even CPP-P^3-X is more efficient than ITS in reaching better solutions on large instances. This highlights the usefulness of our three phase approach.

To illustrate the convergence rate of both algorithms, we ran CPP-P^3 and CPP-P^3-X on two instances and record the best objective value f_{best} after every 10^4 reallocations. The comparative convergence can be visualized in Figure 5.1. One observes that using the same computation times, CPP-P^3-X finds a better solution than CPP-P^3 at first, but CPP-P^3 finally exceeds CPP-P^3-X and remains superior to CPP-P^3-X in the following steps. This indicates that CPP-P^3 has a stronger ability to reach better solutions after experiencing the same number of reallocations.

5.4.2 Landscape analysis

In order to obtain some information about the difficulty of the CPP instances, we carried out a landscape analysis based on the fitness distance correlation (FDC) [Jones and Forrest, 1995]. Such an analysis could shed lights on the behavior of the experimented algorithms. FDC estimates how closely the fitness and

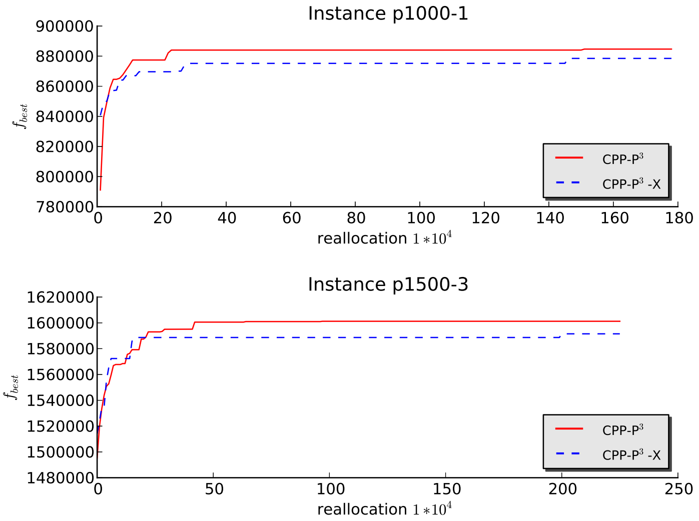

Figure 5.1: Running profiles of CPP-P^3 and CPP-P^3-X

distance to the nearest global optimum are related. For a maximization problem, if the fitness improves when the distance to the optimum decreases, then the search is expected to be effective as there is a "path" to the optimum via solutions with increasing fitness. The correlation coefficient $\rho_{pdf} \in [-1, 1]$ measures the correlation strength, and the perfect ρ_{pdf} value will be -1 for maximization problems, while for minimization problems, the ideal ρ_{pdf} will be 1.

For this study, we investigated several representative instances: sym300-50, regnier300-50, rand500-100, p500-5-3, p500-5-5, p500-100-6, gauss500-100-3, gauss500-100-4. For each graph, we ran CPP-P^3 and collect 5000 high quality local optimum solutions. The distances between these local optima to the global optimum (in our case, the best local optimum) are computed according to the following definition.

Definition 5.4.1. Let $s_1 = \{G_1, G_2, ..., G_k\}$, $s_2 = \{H_1, H_2, .., H_l\}$ be two solutions (partitions) of graph $G = \{V, E, W\}$. The Rand Index [Rand, 1971] computes a distance between s_1,s_2:

$$d(s_1, s_2) = \frac{\sum_{e \in E} d_e(s_1, s_2)}{|E|} \tag{5.8}$$

while $d_e(s_1, s_2)$ of edge e_{uv} is defined by:

$$d_e(s_1, s_2) = \begin{cases} 1, if\ \exists G_i \in s_1, \exists H_j \in s_2, and\ e \in G_i, e \in H_j \\ \quad or\ if\ \forall G_i \in s_1, \neg(e \in G_i)\ and\ \forall H_j \in s_2, \neg(e \in H_j) \\ 0, otherwise \end{cases} \tag{5.9}$$

The correlation between fitness (objective function value) and distance to the reference solution can be visualized in Figure 5.2. One observes that the instances on the left side of the figure have weak FD correlations as indicated by ρ_{pdf} values close to 0. On the other hand, the instances on the right side have stronger FD correlations. It is interesting to see that the correlation strength is proportional to the efforts of our algorithm to reach the best optimum, i.e., CPP-P^3 needs more time to solve weakly correlated instances contrary to strongly related instances. For example, Table 5.2 indicates that CPP-P^3 only needs 0.90s to reach the best local optimum on instance regnier300-50 (with a strong correlation $\rho_{pdf} = -0.89$) on average while 69.53s on sym300-50 (with a weak correlation$\rho_{pdf} = -0.12$). Although we did not include fitness-distance plots for all the instances, this observation suggests that ρ_{pdf} helps us estimate the performance of CPP-P^3 on a particular instance.

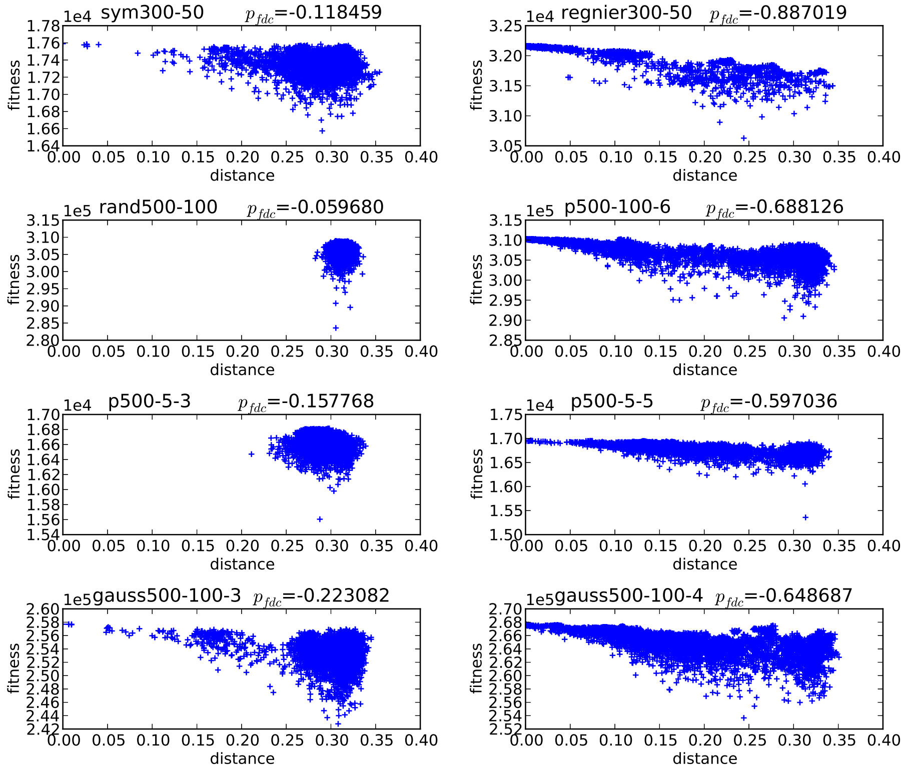

Figure 5.2: FD correlation plots with respect to the solution fitness and distance to the optimum for 8 graphs.

5.4.3 Impact of the descent search phase of CPP-P^3

As shown in Section 5.2, the first phase of CPP-P^3 employs a descent procedure to locate a first local optimum from a given starting solution (which is typically generated by the perturbation phase). Since the descent phase is followed by a tabu-based exploration phase, one may wonder if the descent phase is necessary. To clarify this, we investigated a CPP-P^3 variant where the descent search phase was disabled (i.e., line 6 of Algorithm 5.1 was removed) and the other components were kept unchanged. We then used the variant to solve all the benchmark instances under the same experimental condition as that used in Section 5.3.2. Without bothering to give a detailed tabulation of the detailed results, we mention that, between CPP-P^3 and its variant, no strict dominance is observed according to the main performance indicators (best and average objective values, hit and CPU time). In fact, CPP-P^3 performs better than its variant on a number of instances while the reverse is true for other instances. This experiment indicates that it would be more appropriate to consider the descent search phase as an option of the CPP-P^3 which can be switched on or off to solve a given problem instance.

5.5 Conclusion

The clique partitioning problem (CPP) is a significant NP-Hard problem with numerous applications. This chapter mainly studied an effective heuristic algorithm CPP-P^3 for CPP. The algorithm iterates three phases, i.e., the descent search, the exploration search and the directed perturbation before meeting the stop condition. The three phases play different roles during the search: descent search quickly converges solution to very high quality, tabu search explores nearby region of local optimum while the perturbation leads the search to explore new search areas for the purpose of diversification. The concept of *Top Move* was used to reduce the number of considered neighborhoods for all three phases. We conducted experiments on four groups of instances to verify the effectiveness of our algorithms. In terms of solution quality and computational time, CPP-P^3 generally outperforms two state-of-the-art algorithms ITS and SGVNS. Besides, CPP-P^3 also sets new records on some large instances. Analysis of *Top Move* demonstrates the effectiveness of our local move operator. Further landscape analysis using fitness distance correlation reveals the characteristics and hardness of some representative instances.

General Conclusion

Conclusions

This thesis concerns four NP-hard combinatorial problems in the clique problem family, namely, the maximum vertex weight clique problem (MVWCP), the maximum s-plex problem (MsPlex), the maximum balanced biclique problem (MBBP) and the clique partitioning problem (CPP). These problems are extensively studied in the literature not only for their theoretical intractability, but also for their real world applications in many domains like social network analysis, computer vision, investigation analysis, nano-electronic system design, etc. In this thesis, we mainly focused on developing heuristic algorithms to solve these problems. Some ideas for improving exact algorithms were also proposed. Besides, since emerging very large scale real-life networks in recent years have also introduced additional difficulty of solving these problems, we discussed some interesting reduction techniques to deal with very large sparse networks.

In Chapter 2, we presented the generalized *PUSH* operator for MVWCP. *PUSH*(v,C) adds to the current clique C a vertex v taken from a *candidate push set* of vertices, and removes from C any vertex which is not adjacent to v to keep the resulting clique feasible. By customizing the candidate push set, the *PUSH* operator can be used to define various dedicated neighborhoods which can be explored by any local optimization algorithm. In particular, we showed that the traditional *ADD* and *SWAP* operators as well as some restart and perturbation rules are also covered by the *PUSH* operator. To demonstrate the usefulness of the *PUSH* operator for solving MVWCP, we introduced two restart tabu search algorithms (ReTS-I and ReTS-II) which apply *PUSH* on different candidate push sets. In ReTS-I, *PUSH* operates with the single largest candidate push set $V \setminus C$, while ReTS-II explores three customized candidate push sets. Both ReTS-I and ReTS-II also share the same restart strategy which generates, according to a probability, new starting solutions either with an objective-guided reconstruction procedure or a randomized one. ReTS-I and ReTS-II were evaluated on 3 sets (2nd DIMACS, BHOSLIB and WDP) of 142 benchmark instances. Experimental results indicated that both algorithms compete very favorably with the state-of-the-art algorithms on the tested instances in terms of computational effort and solution quality. Both algorithms are even able to find an improved best-known result (new lower bound) for one instance (frb53-24-3). In addition to these interesting results, the generality of the *PUSH* operator enables a wider application surpassing the studied tabu search procedures.

In Chapter 3, we explored an effective tabu based local search algorithm for solving MsPlex heuristically. To ensure its efficiency, the proposed algorithm combines a multi-neighborhood search procedure with vertex-moving frequency, where the search process is driven by two intensification oriented operators (ADD and $SWAP$) and one diversification operator ($PRESS$). Dedicated rules are defined to explore the neighborhoods introduced by these operators. Information regarding vertex moves is collected and used to guide the construction of the starting solutions and the perturbation process. A graph peeling technique is also integrated to dynamically reduce large sparse graphs. We assessed the performance of the proposed algorithm using three popular benchmark sets: 47 instances from the Stanford Large Network Dataset Collection and the 10th DIMACS Implementation Challenge, and 52 dense graphs from the 2nd DIMACS Implementation Challenge. For the SNAP and 10th DIMACS benchmarks, FD-TS obtained improved solutions (new lower bounds) for $7, 15, 19, 20$ instances when $s = 2, 3, 4, 5$, respectively. Moreover, many of these solutions were proved to be optimal using the $Peel$ procedure. FD-TS also performed very well on the instances from the 2nd DIMACS benchmark set. The *Peel* procedure was no longer effective for these dense graphs, but FD-TS still obtained the current best-known results for all of the instances and discovered better solutions for most instances compared with four recent reference algorithms and the powerful CPLEX solver.

In Chapter 4, we discussed two types of algorithms for the maximum balanced biclique problem (MBBP): heuristic algorithm and exact algorithms. Firstly, we presented an original tabu search combined with two dedicated graph reduction techniques for solving MBBP approximately. The proposed TSGR-MBBP algorithm is driven by a Constraint-Based Tabu Search (CBTS) procedure to retrieve high quality solutions from the current graph. CBTS employs the “push" operator to explore relaxed search space including both balanced and unbalanced bicliques and imposes a specific unbalance constraint on explored solutions. Moreover, each time the lower bound is updated by CBTS, two reduction rules are used to prune the graph, which leads to a reduced search space for the following iterations. Specifically, the first reduction rule is based on removing unpromising vertices according to their degrees, while the second reduction rule removes small subgraphs using an exact search procedure. The TSGR-MBBP algorithm has been assessed on two benchmark sets: 30 random dense instances and 25 real-life large instances from the KONECT collection. For the random instances, TSGR-MBBP dominates existing state-of-the-art approaches EA/SM [Yuan *et al.*, 2015], GL_Greedy[Al-Yamani *et al.*, 2007] and CPLEX (version 12.6.1). Besides, new improved solutions (new lower bounds) were found by TSGR-MBBP for 10 out of the 30 instances. For the KONECT instances, TSGR-MBBP proved optimal solutions for 14 instances for the first time and found high quality solutions for the other instances. Experiments also indicated that TSGR-MBBP performs better than CPLEX both in terms of solution quality and computational time. Besides, we noticed that the two reduction methods are able to prune a significant number of vertices for large sparse graphs. Additional experiments demonstrated the effectiveness of the adopted unbalance constraint used by tabu search and confirmed that the combination of two reduction methods significantly accelerates the convergence of the search procedure.

In Chapter 5, we proposed an effective heuristic algorithm, CPP-P^3, to solve the clique partitioning problem. The algorithm is composed of three iterated search phases: a descent search, an exploration search and a directed perturbation. The descent search quickly converges from a starting solution to a local optimum. The exploration search uses a tabu procedure to explore nearby optimum solutions. The directed perturbation creates an effective diversification with a mechanism similar to a GRASP construction process. The originality of the neighborhood search comes from the concept of *Top Move*, which allows the algorithm to reduce drastically the number of considered neighbors. To verify the effectiveness of CPP-P^3, we evaluated it on a large number of CPP benchmark instances from the literature as well as large random instances specifically generated for this study. We also made a comprehensive comparison with the most recent and the best performing CPP algorithms available in the literature (ITS and SGVNS). Experimental results showed that CPP-P^3 dominates both ITS and SGVNS in terms of solution quality, computational time and robustness on a large set of graphs. On large instances (i.e., graphs with 1000 vertices and more), CPP-P^3 and SGVNS obtain comparable results (and dominate ITS) although improvements are clearly possible since we observe a loss of robustness for all algorithms on these difficult instances. Note that the best-known solutions of some large instances have been improved by our algorithm. We also provided an analysis of the *Top Move* neighborhood to assess its key role to the performance of CPP-P^3. Moreover, we presented a landscape analysis using the fitness distance correlation to shed light on the instance characteristics and hardness.

Perspectives

In this thesis, we mainly consider algorithms and experimental validations for the four aforementioned critical clique problems. The study can be extended in several directions.

Firstly, according to the no free lunch theorem [Wolpert and Macready, 1997], there is no single algorithm that can achieve the best performance for all possible instances. For an algorithm, in order to maximize the robustness across a large range of problem instances with very different characteristics, it would be useful to develop adaptive or learning techniques to help the algorithm to adjust its parameters or search strategy dynamically. For example, the ρ parameter impacts the performance of both algorithms for MVWCP (Chapter 2), it would be interesting to investigate ways of making this parameter self-adaptive during the search. Aside from adaptive techniques, off-line learning for algorithm selection and parameter tuning has also been successfully applied to classical NP-Hard combinatorial problem like MAX-SAT and TSP [Hutter *et al.*, 2014]. This technique also provides us with an approach to better tune parameters for the concerned problems.

Secondly, since the graph reduction techniques have been successfully employed for MsPlex, MBBP and MCP [Verma *et al.*, 2015], these techniques are believed to have great potential in solving other important graph problems with very large networks. In fact, though we did not reduce the graph in algorithms for MVWCP, we have observed that graph reduction has been successfully used in MVWCP in [Cai and Lin, 2016] shortly after our work and obtained very good results. Except the graph reduction techniques, it is worth testing other general ideas in this thesis on similar problems. One example is the *PUSH* operator we proposed in Chapter 2. As it has been successfully used in MVWCP and MBBP, this operator would work well on other similar problems like relaxed clique problems, such as s-defective clique [Yu *et al.*, 2006], quasi-clique [Abello *et al.*, 2002; Pajouh *et al.*, 2014; Pattillo *et al.*, 2013a; Veremyev *et al.*, 2016] and k-club [Bourjolly *et al.*, 2000; Moradi and Balasundaram, 2015; Shahinpour and Butenko, 2013; Veremyev and Boginski, 2012]. We could also investigate the frequency information in new selection rules for the transformation operators as well as other guided perturbation mechanisms such as those proposed in [Benlic and Hao, 2013a].

Lastly, as introduced in the beginning, the clique problems considered in this thesis have numerous real-life application. However, there is still a huge gap between application and theory. For example, in [Boginski *et al.*, 2014], MsPlex was applied to find the profitable portfolio, but the strategy of building the networks still obviously influences in the results. Same problems would be faced when using CPP algorithms to cluster different objects. Therefore, to make use of the algorithms proposed in this thesis, it is important to model the real application into a suitable graph. Besides, for some applications, the algorithms should be conducted on a distributed computing system, which raises another question of how to distribute or parallelize these algorithms.

List of Figures

List of Tables

Appendix

5.6 Computational results of MsPlex on additional instances

This section includes additional results of our FD-TS algorithm and CPLEX for 48 instances from the three benchmark sets. Note that these instances have not been tested previously by any s-plex algorithm. Only lower bounds (from the best-known maximum clique sizes [Verma *et al.*, 2015; Wu and Hao, 2015a]) are available. Table 5.6 contains the 20 large SNAP and 10th DIMACS Challenge instances while Table 5.7 includes the remaining 28 instances of 2nd DIMACS Challenge.

Table 5.6: Computational results of FD-TS on 20 large networks from the SNAP Collection and the 10th DIMACS Implementation Challenge

				s=2				s=3				s=4				s=5			
Instance	$\|V\|$	$\|E\|$	ω[Verma *et al.* 2015]	max	time	$\|V'\|$	cplex	max	time	$\|V'\|$	cplex	max	time	$\|V'\|$	cplex	max	time	$\|V'\|$	cplex
p2p-Gnutella30	36682	88328	3	**5**	0.01	12097	N/A	**7**	0.04	9779	4	8	0.01	9779	5	**10**	0.02	1458	10
p2p-Gnutella31	62586	147892	4	**5**	0.04	19765	N/A	**6**	0.02	19765	N/A	**8**	0.02	16174	N/A	10	0.04	1004	10*
Amazon0302	262111	899792	7	**8***	0.24	0	-	**9***	0.53	0	-	**10***	0.24	0	-	**11***	0.25	0	-
kron_g500-simple-logn16	65536	2456071	136	**140**	0.89	6885	N/A	**144**	0.92	6885	30	**148**	1.22	6884	N/A	**152**	1.38	6883	18
citationCiteseer	268495	1156647	10	**13**	2.49	6731	N/A	**14**	0.54	6731	N/A	**16**	0.72	2779	4	**18**	0.81	1150	17
eu-2005	862664	16138468	387	388	4.73	405	388*	390	5.17	405	390*	391	7.57	405	391*	**393***	9.08	0	-
in-2004	1382908	13591473	489	**490***	3.51	0	-	**491***	6.90	0	-	**491***	7.54	491	-	**491***	8.89	491	-
rgg_n_2_21_s0	2097152	14487995	19	19*	2.83	19	-	19	2.82	115	19*	20	2.62	115	20*	**22***	2.53	19	-
rgg_n_2_22_s0	4194304	30359198	20	20*	5.67	20	-	**21***	5.64	20	-	**22***	5.38	20	-	**23***	5.29	20	-
rgg_n_2_23_s0	8388608	63501393	21	**22***	12.55	0	-	**22***	11.17	22	-	**23***	12.16	22	-	**24***	11.50	22	-
rgg_n_2_24_s0	16777216	132557200	21	**22***	24.94	0	-	**23***	25.33	0	-	**24***	21.84	0	-	**25***	24.95	0	-
uk-2002	18520486	261787258	944	944*	45.00	944	-	944*	44.23	944	-	944*	47.05	944	-	944*	47.13	944	-
G_n_pin_pout	100000	501198	3	**5**	3.95	98961	N/A	**5**	0.03	99734	N/A	**7**	42.96	98961	N/A	**8**	26.29	98961	N/A
preferentialAt tach-ment	100000	499985	6	**7***	0.02	0	-	**8***	0.03	0	-	**9***	0.06	0	-	**10***	0.09	0	-
smallworld	100000	499998	5	**7**	0.00	99982	N/A	**8**	0.00	99982	N/A	**9**	0.00	99982	N/A	**10**	0.00	99982	N/A
luxembourg.osm	114599	119666	2	**4***	0.02	0	-	**5***	0.02	0	-	**6***	0.02	0	-	**7***	0.02	0	-
wave	156317	1059331	5	**9**	0.68	119747	N/A	**9**	0.85	155074	N/A	**11**	1.57	119747	N/A	**12**	0.81	119747	N/A
audikw1	943695	38354076	36	36	1.08	937779	N/A	**39**	34.83	936015	N/A	**45**	69.79	875349	N/A	**45**	75.77	929820	N/A
ldoor	952203	22785136	21	21	0.00	952203	N/A	21	0.01	952203	N/A	21	0.01	952203	N/A	**23**	0.10	952203	N/A
ecology1	1000000	1998000	2	**4***	0.00	0	-	**4**	0.00	1000000	N/A	**6***	0.01	0	-	**7***	0.00	0	-

Table 5.7: Computational results of FD-TS on 28 benchmark instances of the 2nd DIMACS Implementation Challenge

instance	$\|V\|$	ω[Wu and Hao, 2015a]	s=2			s=3			s=4			s=5		
			cplex	max(ave)	time	cplex	max(ave)	time	cplex	max(ave)	time	cplex	max(ave)	time
C125.9	125	34	43*	43	0.00	51*	51	1.89	58	58	0.07	65*	65	0.38
C250.9	250	44	53	**55**	8.43	63	**65**	22.29	75	75	4.81	84	84	4.72
C500.9	500	57	63	**69**	10.67	74	**81**(80.95)	57.72	84	**92**(91.75)	55.11	97	**103**(102.25)	36.65
DSJC1000_5	1000	15	15	**18**	26.61	18	**21**	25.35	21	**24**(23.05)	5.63	24	**27**(26.25)	32.14
DSJC500_5	500	13	15	**16**	0.29	17	**19**	2.81	20	**21**	0.12	23	**24**	1.07
MANN_a81	3321	1100	2162*	2162(2113.90)	139.35	3240*	3240(3125.35)	138.36	3240*	3240(2788.70)	147.51	3240*	*3135*(2660.75)	190.01
brock400_3	400	31	28	**30**	0.35	33	**36**	6.59	38	**41**	5.90	43	**46**(45.90)	55.95
gen200_p0.9_44	200	44	53	53	0.14	66	66	0.09	76	76	0.03	84	84	0.05
gen200_p0.9_55	200	55	57	57	0.02	64	64	0.14	73	73	0.12	80	80	0.19
gen400_p0.9_55	400	55	64	**68**(67.70)	65.76	85	**87**	26.62	108	**112**	0.19	124	124	0.16
gen400_p0.9_65	400	65	73	73(71.60)	28.01	100	**101**(100.45)	10.68	132	132	0.25	138	138	0.15
gen400_p0.9_75	400	75	78	**79**(78.05)	30.62	112	**114**	0.35	136	136	0.10	136	136	0.12
johnson32-2-4	496	16	21	21	0.02	32	32	0.05	38	38	0.38	48	48	0.09
keller5	776	27	31	31	0.09	41	**45**	8.19	46	**53**(52.75)	53.67	58	**61**	5.04
p_hat500-2	500	36	42	42	0.02	49	**50**	0.12	55	**57**	0.07	62	62	0.25
p_hat500-3	500	50	60	**62**	0.18	71	**72**	1.24	80	**81**	1.94	88	**89**	1.73
san200_0.7_1	200	30	31	31	0.59	46	46(45.70)	1.51	60	60	0.01	75*	75	0.01
san200_0.9_2	200	60	71	71	2.47	105*	105	0.02	105*	105	0.02	105*	105	0.03
san200_0.9_3	200	44	53	**54**(53.95)	72.31	73	73	7.48	96*	96	0.08	100*	100	0.03
san400_0.5_1	400	13	15	15	1.24	22	22	3.61	29	29	4.92	35	**36**(35.70)	55.40
san400_0.7_1	400	40	41	41	0.20	61	61	2.49	80	81(80.45)	17.54	100	100	0.06
san400_0.7_2	400	30	32	32	7.22	46	**47**(46.10)	0.46	61	61	0.57	76	76	10.34
san400_0.7_3	400	22	27	27(26.30)	12.27	38	38	11.17	50	50(49.45)	24.71	61	61	0.29
san400_0.9_1	400	100	102	102(101.30)	9.09	150	150	0.09	200*	200	0.12	200*	200	0.15
sanr200_0.7	200	18	22	22	0.01	25	26	0.03	30	30	0.14	33	33	0.05
sanr200_0.9	200	42	51	51	0.61	60	**61**	2.25	69	69	0.07	76	77	5.74
sanr400_0.5	400	13	14	**15**	0.02	18	18	0.09	20	**21**	0.56	23	**24**	1.63
sanr400_0.7	400	21	25	**26**	1.03	28	**30**	0.45	32	**35**	5.31	35	**39**	31.01

List of Publications

Published/accepted journal papers

- Yi Zhou, Jin-Kao Hao, Adrien Goëffon. A three-phased local search approach for the Clique Partitioning problem. Journal of Combinatorial Optimization 32(2): 469-491, 2016.
- Yi Zhou, Jin-Kao Hao, Adrien Goëffon. PUSH: a generalized operator for the maximum weight clique problem. European Journal of Operational Research 257(1): 41-54, 2017.
- Yi Zhou and Jin-Kao Hao. Frequency-driven tabu search for the maximum s-plex problem. Computer & Operations Research, 86: 65-78, 2017.

Submitted journal papers

- Yi Zhou, André Rossi and Jin-Kao Hao. Towards effective exact algorithms for the maximum balanced biclique problem. European Journal of Operational Research. Submitted Jan. 2017, revision May 2017.
- Yi Zhou and Jin-Kao Hao. Combining tabu search and graph reduction to solve the maximum balanced biclique problem. Expert Systems with Applications. Submitted April 2017, revised in July 2017.

References

[Abello *et al.*, 2002] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *Proceedings of the Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer, 2002. 43, 64, 105

[Al-Yamani *et al.*, 2007] Ahmad A. Al-Yamani, Sundarkumar Ramsundar, and Dhiraj K. Pradhan. A defect tolerance scheme for nanotechnology circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(11):2402–2409, 2007. 9, 57, 66, 83, 104

[Alidaee *et al.*, 2007] Bahram Alidaee, Fred Glover, Gary Kochenberger, and Haibo Wang. Solving the maximum edge weight clique problem via unconstrained quadratic programming. *European Journal of Operational Research*, 181(2):592–597, 2007. 17

[Alon *et al.*, 1994] Noga Alon, Richard A Duke, Hanno Lefmann, Vojtech Rodl, and Raphael Yuster. The algorithmic aspects of the regularity lemma. *Journal of Algorithms*, 16(1):80–109, 1994. 7, 57

[Bader *et al.*, 2012] David A Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner. Graph partitioning and graph clustering. In *Proceedings of the 10th Dimacs Implementation Challenge Workshop*, 2012. 11

[Balasundaram *et al.*, 2011] Balabhaskar Balasundaram, Sergiy Butenko, and Illya V Hicks. Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, 59(1):133–142, 2011. 6, 7, 10, 11, 36, 43, 48, 49

[Ballard and Brown, 1982] Dana H Ballard and Christopher M Brown. *Computer Vision*. Prentice-Hall Englewood Cliff., 1982. 8, 17

[Benlic and Hao, 2013a] Una Benlic and Jin-Kao Hao. Breakout local search for maximum clique problems. *Computers & Operations Research*, 40(1):192–206, 2013. 10, 11, 17, 18, 19, 20, 21, 22, 23, 26, 27, 40, 105

[Benlic and Hao, 2013b] Una Benlic and Jin-Kao Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013. 91

[Boginski *et al.*, 2014] Vladimir Boginski, Sergiy Butenko, Oleg Shirokikh, Svyatoslav Trukhanov, and Jaime Gil Lafuente. A network-based data mining approach to portfolio selection via weighted clique relaxations. *Annals of Operations Research*, 216(1):23–34, 2014. 8, 105

[Bomze *et al.*, 2000] Immanuel M Bomze, Marcello Pelillo, and Volker Stix. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 11(6):1228–1241, 2000. 17

[Bourjolly *et al.*, 2000] Jean-Marie Bourjolly, Gilbert Laporte, and Gilles Pesant. Heuristics for finding k-clubs in an undirected graph. *Computers & Operations Research*, 27(6):559–569, 2000. 8, 105

[Brimberg *et al.*, 2015] Jack Brimberg, Stefana Janićijević, Nenad Mladenović, and Dragan Urošević. Solving the clique partitioning problem as a maximally diverse grouping problem. *Optimization Letters*, pages 1–13, 2015. 10, 87, 93, 94, 96, 98

[Brunato *et al.*, 2007] Mauro Brunato, Holger H Hoos, and Roberto Battiti. On effectively finding maximal quasi-cliques in graphs. In *Proceedings of the International Conference on Learning and Intelligent Optimization*, pages 41–55. Springer, 2007. 8

[Brusco and Köhn, 2009] Michael J Brusco and Hans-Friedrich Köhn. Clustering qualitative data based on binary equivalence relations: neighborhood search heuristics for the clique partitioning problem. *Psychometrika*, 74(4):685, 2009. 10, 12, 87, 89, 93, 95, 99

[Busygin, 2006] Stanislav Busygin. A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics*, 154(15):2080–2096, 2006. 17

[Cai and Lin, 2016] Shaowei Cai and Jinkun Lin. Fast solving maximum weight clique problem in massive graphs. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 568–574. AAAI Press, 2016. 105

[Cai, 2015] Shaowei Cai. Balance between complexity and quality: Local search for minimum vertex cover in massive graphs. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 747–753. AAAI Press, 2015. 6, 24

[Carraghan and Pardalos, 1990] Randy Carraghan and Panos M Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9(6):375–382, 1990. 9, 17, 36, 64, 73

[Charon and Hudry, 2001] Irène Charon and Olivier Hudry. The noising methods: A generalization of some metaheuristics. *European Journal of Operational Research*, 135(1):86–101, 2001. 10, 87

[Charon and Hudry, 2006] Irène Charon and Olivier Hudry. Noising methods for a clique partitioning problem. *Discrete Applied Mathematics*, 154(5):754–769, 2006. 10, 12, 87, 89, 93, 95, 99

[Chen and Hao, 2015] Yuning Chen and Jin-Kao Hao. Iterated responsive threshold search for the quadratic multiple knapsack problem. *Annals of Operations Research*, 226(1):101–131, 2015. 89

[Cheng and Church, 2000] Yizong Cheng and George M. Church. Biclustering of expression data. In *Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology*, pages 93–103. AAAI Press, 2000. 7, 9, 57

[Cramton *et al.*, 2006] Peter Cramton, Yoav Shoham, and Richard Steinberg. *Combinatorial Auctions*. MIT Press, 2006. 6

[Dawande *et al.*, 2001] Milind Dawande, Pinar Keskinocak, Jayashankar M Swaminathan, and Sridhar Tayur. On bipartite and multipartite clique problems. *Journal of Algorithms*, 41(2):388–403, 2001. 11, 58, 68, 72, 76, 78, 83

[De Amorim *et al.*, 1992] Saul G De Amorim, Jean-Pierre Barthélemy, and Celso C Ribeiro. Clustering and clique partitioning: simulated annealing and tabu search approaches. *Journal of Classification*, 9(1):17–41, 1992. 10, 87, 89, 99

[Dijkhuizen and Faigle, 1993] G Dijkhuizen and U Faigle. A cutting-plane approach to the edge-weighted maximal clique problem. *European Journal of Operational Research*, 69(1):121–130, 1993. 17

[Dorndorf and Pesch, 1994] Ulrich Dorndorf and Erwin Pesch. Fast clustering algorithms. *ORSA Journal on Computing*, 6(2):141–153, 1994. 8, 10, 87, 92

[Dorndorf *et al.*, 2008] Ulrich Dorndorf, Florian Jaehn, and Erwin Pesch. Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science*, 42(3):292–301, 2008. 9, 87

[Etzion and Ostergard, 1998] Tuvi Etzion and Patric RJ Ostergard. Greedy and heuristic algorithms for codes and colorings. *IEEE Transactions on Information Theory*, 44(1):382–388, 1998. 6

[Fahle, 2002] Torsten Fahle. Simple and fast: Improving a branch-and-bound algorithm for maximum clique. In *Proceedings of the 10th European Symposium on Algorithms (Algorithms – ESA 2002)*, pages 485–498. Springer Berlin Heidelberg, 2002. 9

[Fang *et al.*, 2016] Zhiwen Fang, Chu-Min Li, and Ke Xu. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *Journal of Artificial Intelligence Research*, 55:799–833, 2016. 10, 11, 17, 26, 27

[Feige and Kogan, 2004] Uriel Feige and Shimon Kogan. Hardness of approximation of the balanced complete bipartite subgraph problem. Technical report, Weizmann Inst. Sci, 2004. 57

[Feo and Resende, 1995a] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. 10, 36

[Feo and Resende, 1995b] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. 92

[Fu and Hao, 2015] Zhang-Hua Fu and Jin-Kao Hao. A three-phase search approach for the quadratic minimum spanning tree problem. *Engineering Applications of Artificial Intelligence*, 46:113–130, 2015. 88

[Galinier and Hao, 1999] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999. 91

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. 7, 57

[Glover and Laguna, 2013] Fred Glover and Manuel Laguna. *Tabu Search*. Springer, 2013. 21, 23, 24, 36, 42, 63, 87, 89, 91

[Grosso *et al.*, 2008] Andrea Grosso, Marco Locatelli, and Wayne Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14(6):587–612, 2008. 18, 20, 21, 22, 23

[Grötschel and Wakabayashi, 1989] Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1):59–96, 1989. 7, 9, 10, 87

[Grötschel and Wakabayashi, 1990] Martin Grötschel and Yoshiko Wakabayashi. Facets of the clique partitioning polytope. *Mathematical Programming*, 47(1):367–387, 1990. 7, 87

[Gujjula *et al.*, 2014] Krishna Reddy Gujjula, Krishnan Ayalur Seshadrinathan, and Amirhossein Meisami. A hybrid metaheuristic for the maximum k-plex problem. In *Examining Robustness and Vulnerability of Networked Systems. NATO Science for Peace and Security Series - D: Information and Communication Security*, volume 37, pages 83–92, 2014. 10, 36

[Gutierrez-Rodríguez *et al.*, 2015] Andrés Eduardo Gutierrez-Rodríguez, J Fco Martínez-Trinidad, Milton García-Borroto, and Jesús Ariel Carrasco-Ochoa. Mining patterns for clustering on numerical datasets using unsupervised decision trees. *Knowledge-Based Systems*, 82:70–79, 2015. 9

[Hardiman and Katzir, 2013] Stephen J Hardiman and Liran Katzir. Estimating clustering coefficients and size of social networks via random walk. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 539–550. ACM, 2013. 12

[Hutter *et al.*, 2014] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014. 105

[Jaehn and Pesch, 2013] Florian Jaehn and Erwin Pesch. New bounds and constraint propagation techniques for the clique partitioning problem. *Discrete Applied Mathematics*, 161(13):2025–2037, 2013. 87

[Ji and Mitchell, 2007] Xiaoyun Ji and John E Mitchell. Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. *Discrete Optimization*, 4(1):87–102, 2007. 87

[Jin and Hao, 2015] Yan Jin and Jin-Kao Hao. General swap-based multiple neighborhood tabu search for the maximum independent set problem. *Engineering Applications of Artificial Intelligence*, 37:20–33, 2015. 40, 43

[Jones and Forrest, 1995] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann Publishers Inc., 1995. 99

[Karp, 1972] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*. Springer, 1972. 9, 17, 36

[Kernighan and Lin, 1970] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970. 87

[Kirkpatrick *et al.*, 1983] Scott Kirkpatrick, C Daniel Gelatt, Mario P Vecchi, et al. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 10, 87

[Kumlander, 2004] Deniss Kumlander. A new exact algorithm for the maximum-weight clique problem based on a heuristic vertex-coloring and a backtrack search. In *Proceedings of the 5th International Conference on Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 202–208. Citeseer, 2004. 9, 17

[Kunegis, 2013] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, New York, NY, USA, 2013. ACM. 11

[Leskovec and Sosič, 2016] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016. 11

[Li and Quan, 2010] Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI'10, pages 128–133. AAAI Press, 2010. 9

[López-Ibánez *et al.*, 2011] Manuel López-Ibánez, Jérémie Dubois-Lacoste, Thomas Stützle, and Mauro Birattari. The irace package, iterated race for automatic algorithm configuration. Technical report, Citeseer, 2011. 65

[Malod-Dognin *et al.*, 2010] Noël Malod-Dognin, Rumen Andonov, and Nicola Yanev. Maximum cliques in protein structure comparison. In *Proceedings of the International Symposium on Experimental Algorithms*, pages 106–117. Springer, 2010. 6

[Mannino and Sassano, 1996] Carlo Mannino and Antonio Sassano. Edge projection and the maximum cardinality stable set problem. *DIMACS series in discrete mathematics and theoretical computer science*, 26:205–219, 1996. 10

[Mannino and Stefanutti, 1999] Carlo Mannino and Egidio Stefanutti. An augmentation algorithm for the maximum weighted stable set problem. *Computational Optimization and Applications*, 14(3):367–381, 1999. 11, 17, 27

[McClosky and Hicks, 2012] Benjamin McClosky and Illya V Hicks. Combinatorial algorithms for the maximum k-plex problem. *Journal of combinatorial optimization*, 23(1):29–49, 2012. 9, 11, 36, 48, 49

[McCreesh and Prosser, 2014] Ciaran McCreesh and Patrick Prosser. An exact branch and bound algorithm with symmetry breaking for the maximum balanced induced biclique problem. In *Proceedings of the International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 226–234. Springer, 2014. 57, 58, 72, 73, 74, 78, 83

[Miao and Balasundaram, 2012] Zhuqi Miao and Balabhaskar Balasundaram. Cluster detection in large-scale social networks using k-plexes. In *Proceedings of the IIE Annual Conference*, page 1. Institute of Industrial and Systems Engineers (IISE), 2012. 10, 36

[Moradi and Balasundaram, 2015] Esmaeel Moradi and Balabhaskar Balasundaram. Finding a maximum k-club using the k-clique formulation and canonical hypercube cuts. *Optimization Letters*, pages 1–11, 2015. 105

[Moser *et al.*, 2012] Hannes Moser, Rolf Niedermeier, and Manuel Sorge. Exact combinatorial algorithms and experiments for finding maximum k-plexes. *Journal of Combinatorial Optimization*, 24(3):347–373, 2012. 11, 36, 48, 49

[Mubayi and Turán, 2010] Dhruv Mubayi and György Turán. Finding bipartite subgraphs efficiently. *Information Processing Letters*, 110(5):174–177, 2010. 57

[Nemhauser and Trotter, 1974] George L Nemhauser and Leslie E Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6(1):48–61, 1974. 10

[Neveu *et al.*, 2004] Bertrand Neveu, Gilles Trombettoni, and Fred Glover. ID Walk: A candidate list strategy with a simple diversification device. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 423–437. Springer, 2004. 24

[Oosten *et al.*, 2001] Maarten Oosten, Jeroen HGC Rutten, and Frits CR Spieksma. The clique partitioning problem: facets and patching facets. *Networks*, 38(4):209–226, 2001. 9

[Östergård, 1999] Patric RJ Östergård. A new algorithm for the maximum-weight clique problem. *Electronic Notes in Discrete Mathematics*, 3:153–156, 1999. 9, 17

[Östergård, 2002] Patric RJ Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1):197–207, 2002. 9, 17, 36, 48

[Padberg, 1973] Manfred W Padberg. On the facial structure of set packing polyhedra. *Mathematical programming*, 5(1):199–215, 1973. 10

[Pajouh *et al.*, 2014] Foad Mahdavi Pajouh, Zhuqi Miao, and Balabhaskar Balasundaram. A branch-and-bound approach for maximum quasi-cliques. *Annals of Operations Research*, 216(1):145–161, 2014. 8, 105

[Palubeckis *et al.*, 2014] Gintaras Palubeckis, Armantas Ostreika, and Arūnas Tomkevičius. An iterated tabu search approach for the clique partitioning problem. *The Scientific World Journal*, 2014, 2014. 10, 12, 87, 89, 92, 93, 94, 96, 97, 98, 99

[Pattillo *et al.*, 2012] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. Clique relaxation models in social network analysis. In *Handbook of Optimization in Complex Networks*, pages 143–162. Springer, 2012. 8, 36

[Pattillo *et al.*, 2013a] Jeffrey Pattillo, Alexander Veremyev, Sergiy Butenko, and Vladimir Boginski. On the maximum quasi-clique problem. *Discrete Applied Mathematics*, 161(1):244–257, 2013. 8, 105

[Pattillo *et al.*, 2013b] Jeffrey Pattillo, Nataly Youssef, and Sergiy Butenko. On clique relaxation models in network analysis. *European Journal of Operational Research*, 226(1):9–18, 2013. 8, 36

[Pullan and Hoos, 2006] Wayne Pullan and Holger H Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006. 21, 40

[Pullan, 2008] Wayne Pullan. Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14(2):117–134, 2008. 10, 11, 17, 19, 23, 26, 27

[Rand, 1971] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971. 100

[Ravetti and Moscato, 2008] Martín Gómez Ravetti and Pablo Moscato. Identification of a 5-protein biomarker molecular signature for predicting alzheimer's disease. *PloS One*, 3(9):e3111, 2008. 6

[Ravi and Lloyd, 1988] S.S. Ravi and Errol L. Lloyd. The complexity of near-optimal programmable logic array folding. *SIAM Journal on Computing*, 17(4):696–710, 1988. 9

[Rebennack *et al.*, 2012] Steffen Rebennack, Gerhard Reinelt, and Panos M Pardalos. A tutorial on branch and cut algorithms for the maximum stable set problem. *International Transactions in Operational Research*, 19(1-2):161–199, 2012. 10

[Role and Nadif, 2014] François Role and Mohamed Nadif. Beyond cluster labeling: Semantic interpretation of clusters' contents using a graph representation. *Knowledge-Based Systems*, 56:141–155, 2014. 9

[San Segundo *et al.*, 2011] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2):571–581, 2011. 9

[Sandholm, 2002] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002. 27

[Seidman and Foster, 1978] Stephen B Seidman and Brian L Foster. A graph-theoretic generalization of the clique concept. *Journal of Mathematical sociology*, 6(1):139–154, 1978. 7, 36

[Shahinpour and Butenko, 2013] Shahram Shahinpour and Sergiy Butenko. Algorithms for the maximum k-club problem in graphs. *Journal of Combinatorial Optimization*, 26(3):520–554, 2013. 105

[Singh and Gupta, 2006] Alok Singh and Ashok Kumar Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12(1-2):5–22, 2006. 17, 27

[Soto *et al.*, 2011] María Soto, André Rossi, and Marc Sevaux. Three new upper bounds on the chromatic number. *Discrete Applied Mathematics*, 159(18):2281–2289, 2011. 72

[Tahoori, 2006] Mehdi B. Tahoori. Application-independent defect tolerance of reconfigurable nanoarchitectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2(3):197–218, 2006. 9, 57, 66

[Tahoori, 2009] Mehdi B. Tahoori. Low-overhead defect tolerance in crossbar nanoarchitectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 5(2):11, 2009. 9

[Tomita and Kameda, 2007] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization*, 37(1):95–111, 2007. 9

[Tomita and Seki, 2003] Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete Mathematics and Theoretical Computer Science*, pages 278–289. Springer, 2003. 9

[Trukhanov *et al.*, 2013] Svyatoslav Trukhanov, Chitra Balasubramaniam, Balabhaskar Balasundaram, and Sergiy Butenko. Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, 56(1):113–130, 2013. 9, 11, 36, 37, 38, 43, 45, 47, 48, 49, 50

[Veremyev and Boginski, 2012] Alexander Veremyev and Vladimir Boginski. Identifying large robust network clusters via new compact formulations of maximum k-club problems. *European Journal of Operational Research*, 218(2):316–326, 2012. 105

[Veremyev *et al.*, 2016] Alexander Veremyev, Oleg A Prokopyev, Sergiy Butenko, and Eduardo L Pasiliao. Exact mip-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications*, 64(1):177–214, 2016. 105

[Verma *et al.*, 2015] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015. 9, 43, 45, 105, 111, 112

[Wakabayashi, 1986] Yoshiko Wakabayashi. *Aggregation of binary relations: algorithmic and polyhedral investigations*. PhD Thesis, Universität Augsburg, 1986. 7, 8, 87

[Wang *et al.*, 2006] Haibo Wang, Bahram Alidaee, Fred Glover, and Gary Kochenberger. Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*, 18(2):77–97, 2006. 9

[Wang *et al.*, 2016] Yang Wang, Jin-Kao Hao, Fred Glover, Zhipeng Lü, and Qinghua Wu. Solving the maximum vertex weight clique problem via binary quadratic programming. *Journal of Combinatorial Optimization*, 32(2):531–549, 2016. 10, 11, 17, 26, 27, 30

[Warren and Hicks, 2006] Jeffrey S Warren and Illya V Hicks. Combinatorial branch-and-bound for the maximum weight independent set problem. *Relatório Técnico, Texas A&M University, Citeseer*, 2006. 9, 17

[Wolpert and Macready, 1997] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997. 104

[Wu and Pei, 2007] Bin Wu and Xin Pei. A parallel algorithm for enumerating all the maximal k-plexes. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 476–483. Springer, 2007. 36

[Wu and Hao, 2012] Qinghua Wu and Jin-Kao Hao. Coloring large graphs based on independent set extraction. *Computers & Operations Research*, 39(2):283–290, 2012. 6

[Wu and Hao, 2013a] Qinghua Wu and Jin-Kao Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, 26(1):86–108, 2013. 40, 89

[Wu and Hao, 2013b] Qinghua Wu and Jin-Kao Hao. A hybrid metaheuristic method for the maximum diversity problem. *European Journal of Operational Research*, 231(2):452–464, 2013. 89

[Wu and Hao, 2015a] Qinghua Wu and Jin-Kao Hao. A review on algorithms for maximum clique problems. *European Journal of Operational Research*, 242(3):693–709, 2015. 17, 36, 48, 59, 111, 113

[Wu and Hao, 2015b] Qinghua Wu and Jin-Kao Hao. Solving the winner determination problem via a weighted maximum clique heuristic. *Expert Systems with Applications*, 42(1):355–365, 2015. 8, 10, 11, 17, 19, 27, 30

[Wu and Hao, 2016] Qinghua Wu and Jin-Kao Hao. A clique-based exact method for optimal winner determination in combinatorial auctions. *Information Sciences*, 334:103–121, 2016. 8, 9, 11, 17

[Wu *et al.*, 2012] Qinghua Wu, Jin-Kao Hao, and Fred Glover. Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, 196(1):611–634, 2012. 10, 11, 17, 18, 19, 22, 23, 26, 27, 40, 43

[Yu *et al.*, 2006] Haiyuan Yu, Alberto Paccanaro, Valery Trifonov, and Mark Gerstein. Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, 22(7):823–829, 2006. 8, 105

[Yuan and Li, 2011] Bo Yuan and Bin Li. A low time complexity defect-tolerance algorithm for nanoelectronic crossbar. In *Proceedings of the 2011 International Conference on Information Science and Technology (ICIST)*, pages 143–148. IEEE, 2011. 57, 66

[Yuan and Li, 2014] Bo Yuan and Bin Li. A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(3):25, 2014. 57, 66

[Yuan *et al.*, 2015] Bo Yuan, Bin Li, Huanhuan Chen, and Xin Yao. A new evolutionary algorithm with structure mutation for the maximum balanced biclique problem. *IEEE Transactions on Cybernetics*, 45(5):1040–1053, 2015. 11, 57, 58, 65, 66, 67, 68, 83, 104

[Zhi-Xiao *et al.*, 2016] Wang Zhi-Xiao, Li Ze-chao, Ding Xiao-fang, and Tang Jin-hui. Overlapping community detection based on node location analysis. *Knowledge-Based Systems*, 105:225–235, 2016. 9

[Zhou and Hao, 2017a] Yi Zhou and Jin-Kao Hao. Combining tabu search and graph reduction to solve the maximum balanced biclique problem. *Submitted to Expert Systems with Applications, revised in July*, 2017. 2

[Zhou and Hao, 2017b] Yi Zhou and Jin-Kao Hao. Frequency-driven tabu search for the maximum s-plex problem. *Computers & Operations Research*, 86:65–78, 2017. 2, 35

[Zhou *et al.*, 2016] Yi Zhou, Jin-Kao Hao, and Adrien Goëffon. A three-phased local search approach for the clique partitioning problem. *Journal of Combinatorial Optimization*, 32(2):469–491, 2016. 2

[Zhou *et al.*, 2017a] Yi Zhou, Jin-Kao Hao, and Adrien Goëffon. PUSH: A generalized operator for the maximum vertex weight clique problem. *European Journal of Operational Research*, 257(1):41–54, 2017. 2, 15, 40, 59, 62, 63

[Zhou *et al.*, 2017b] Yi Zhou, André Rossi, and Jin-Kao Hao. Towards effective exact algorithms for the maximum balanced biclique problem. *Submitted to European Journal of Operational Research, revised in March*, 2017. 2

Thèse de Doctorat

Yi ZHOU

Algorithmes d'Optimisation pour quelques Problèmes de Clique.

Optimization Algorithms for Clique Problems

Résumé

Cette thèse présente des algorithmes de résolution de quatre problèmes de clique : clique de poids maximum (MVWCP), s-plex maximum (MsPlex), clique maximum équilibrée dans un graphe biparti (MBBP) et clique partition (CPP). Les trois premiers problèmes sont des généralisations ou relaxations du problème de la clique maximum, tandis que le dernier est un problème de couverture. Ces problèmes, ayant de nombreuses applications pratiques, sont NP-difficiles, rendant leur résolution ardue dans le cas général. Nous présentons ici des algorithmes de recherche locale, principalement basés sur la recherche tabou, permettant de traiter efficacement ces problèmes ; chacun de ces algorithmes emploie des composants originaux et spécifiquement adaptés aux problèmes traités, comme de nouveaux opérateurs ou mécanismes perturbatifs. Nous y intégrons également des stratégies telles que la réduction de graphe ou la propagation afin de traiter des réseaux de plus grande taille. Des expérimentations basées sur des jeux d'instances nombreux et variés permettent de montrer la compétitivité de nos algorithmes en comparaison avec les autres stratégies existantes.

Abstract

This thesis considers four clique problems: the maximum vertex weight clique problem (MVWCP), the maximum s-plex problem (MsPlex), the maximum balanced biclique problem (MBBP) and the clique partitioning problem (CPP). The first three are generalization and relaxation of the classic maximum clique problem (MCP), while the last problem belongs to a clique grouping problem. These combinatorial problems have numerous practical applications. Given that they all belong to the NP-Hard family, it is computationally difficult to solve them in the general case. For this reason, this thesis is devoted to develop effective algorithms to tackle these challenging problems. Specifically, we propose two restart tabu search algorithms based on a generalized PUSH operator for MVWCP, a frequency driven local search algorithms for MsPlex, a graph reduction based tabu search as well as effective exact branch and bound algorithms for MBBP and lastly, a three phase local search algorithm for CPP. In addition to the design of efficient move operators for local search algorithms, we also integrate components like graph reduction or upper bound propagation in order to deal deal with very large real-life networks. The experimental tests on a wide range of instances show that our algorithms compete favorably with the main state-of-the-art algorithms.

Mots clés

Problèmes de clique, recherche locale, réseaux complexes réels, Métaheuristiques, algorithmes exacts, expérimentations.

Key Words

Clique problems, Local search, Large real-life networks, Metaheuristics, Exact algorithms, Computational experiments.