Multiple Operator Metaheuristics for Graph Partitioning Problems

Fuda Ma

UNIVERSITE BRETAGNE LOIRE

université angers

Thèse de Doctorat

Fuda MA

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'Études et de Recherches en Informatique d'Angers (LERIA)

Soutenue le June 2016
Thèse n° : 1

Multiple Operator Metaheuristics for Graph Partitioning Problems

JURY

Rapporteurs :	**M. Chu-Min LI**, Professeur, Université de Picardie Jules Verne
	M. Michel VASQUEZ, Professeur, Ecole des Mines d'Alès
Examinateurs :	**M. Matthieu BASSEUR**, Maître de Conférence HDR, Université d'Angers
	M. Nicolas DURAND, Professeur, Ecole Nationale d'Aviation Civile Toulouse
Directeur de thèse :	**M. Jin-Kao HAO**, Professeur, Université d'Angers

Thèse soutenue le 28/06/2016

UNIVERSITE BRETAGNE LOIRE

université angers

Thèse de Doctorat

Fuda MA

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'Études et de Recherches en Informatique d'Angers (LERIA)

Soutenue le June 2016
Thèse n° : 1

Multiple Operator Metaheuristics for Graph Partitioning Problems

JURY

Rapporteurs :	**M. Chu-Min LI**, Professeur, Université de Picardie Jules Verne
	M. Michel VASQUEZ, Professeur, Ecole des Mines d'Alès
Examinateurs :	**M. Matthieu BASSEUR**, Maître de Conférence HDR, Université d'Angers
	M. Nicolas DURAND, Professeur, Ecole Nationale d'Aviation Civile Toulouse
Directeur de thèse :	**M. Jin-Kao HAO**, Professeur, Université d'Angers

Contents

General Introduction

Context

Graph partitioning problems are a class of well-known NP-hard combinatorial optimization problems, which require to partition a graph into $k \geq 2$ disjoint subsets so as to optimize a given objective subject to certain constraints. Graph partitioning problems are extensively studied not only for its theoretical importance, but also for its applicability to many domains, such as VLSI layout design, statistical physics, sports team scheduling, data clustering, image segmentation, and protein conformation for instances.

Approaches for solving graph partitioning problems can be classified as approximation algorithms, exact algorithms and heuristic algorithms. Approximation algorithms can provide an approximate solution guaranteed to be within an approximation ratio to its optimal value, but the solution quality reached usually present a large gap to that of the optimal solution. Exact algorithms, building upon the theoretical knowledge of the investigated problem, can obtain optimal solutions in an acceptable computing time for either small graphs of limited size or larger graphs of special structures. For solving large and challenging problem instances, heuristic and metaheuristic algorithms are commonly used to find "good-enough" sub-optimal solutions.

Local search is an effective heuristic/metaheuristic approach that usually performs neighborhood exploration by a search operator that looks for a better solution in the neighborhood of the current solution. Different search operators have their advantages and drawbacks; no global optimal one exists. Hence, it would be beneficial to design a local search approach that collectively employs different search operators organized in an effective pattern. Motivated by this idea, the first key research work will design multiple operator local search for handling hard graph partitioning problems.

Furthermore, population based metaheuristic approaches are capable of attaining a good balance between intensification and diversification during the search, which often include a local search component for solution refinement to achieve search intensification. Therefore, the second work of this thesis will design a powerful population based path relinking approach, in which the multiple operator local search is used for search intensification while the other components of path relinking play the role of search diversification.

In short, this thesis is dedicated to developing effective multiple operator based heuristic and metaheuristic approaches for solving several representative graph partitioning problems, including the max-k-cut problem, the max-bisection problem and the vertex separator problem.

Objectives

This thesis aims to study effective multiple search operators based heuristic and metaheuristic approaches for solving three well-known graph partitioning problem, the max-k-cut problem, the max-bisection problem and the vertex separator problem. The main objectives of this thesis include:

– propose high performance heuristic and metaheuristic approaches for each of these problems to en-

hance the state of art in the literature.
– develop effective local search approaches based on combination of multiple search operators and demonstrate advantages of using multiple search operators over a single search operator to the performance of the developed approaches. Using graph partitioning problems as case study, we investigate different search operators with complementary properties and design effective patterns to combine them.
– design a powerful population based metaheuristic approach with multiple operator local search incorporated for solution refinement. For this purpose, we employ the path relinking search framework that has empirically demonstrated to attain a good balance between search intensification and diversification for many combinatorial optimization problems. In this respect, the multiple operator local search is used for search intensification while the other components in path relinking play the role of search diversification.

Contributions

The main contributions of this thesis are the following:

– We present a new and effective multiple operator heuristic (MOH) for the general max-k-cut problem. The main originality of the proposed algorithm is its multi-phased multi-strategy approach which relies on five distinct local search operators (O_1 - O_5) for solution transformations. These operators are organized into three different search phases (descent-based improvement, diversified improvement, perturbation) to ensure an effective examination of the search space. Specifically, the operator O_2 employs constrained double-transfer moves to greatly reduce the size of the transfer moves and prevents from expensive computational efforts. The decent improvement phase compares three different ways of combining the operators O_1 and O_2 and experimentally determines the best combination. The diversified improvement procedure collectively uses the operators O_3 and O_4, the selection of which is based on a probabilistic mechanism. The perturbation phase applies a random search operator O_5 to definitively lead the search to a distant region when the search is trapped in a deep local optimum. The use of the bucket sorting structure to accelerate the identification of the best move is another important ingredient of the MOH algorithm. Experiments on two sets of 91 well-known benchmark instances show that the proposed algorithm is highly effective on the max-k-cut problem and improves the current best known results (lower bounds) of most of the tested instances. For the popular special case $k = 2$ (i.e., the max-cut problem), MOH also performs remarkably well by discovering 4 improved best known results. We provide additional studies to shed light on the alternative combinations of the employed search operators.
– We presented an effective iterated tabu search (ITS) for the max-bisection problem based on the iterated local search (ILS) framework, which includes the following original features. First, ITS relies on a joint use of two complementary search operators to conduct an extensive exploitation of the search space. The *1-move* operator is used to quickly discover a local optimal solution from which improved solutions are sought by employing the more advanced *c-swap* operator. Second, in addition to an improvement phase and a perturbation phase used in conventional ILS algorithms, the proposed ITS algorithm additionally includes a fast descent procedure to quickly attain a promising search area which is deeply examined with the powerful tabu search procedure. This combination prevents the search procedure from running the more expensive tabu search procedure in an unpromising area and thus helps to increase the search efficiency of the algorithm. We assess the performance of the proposed algorithm on 71 well-known benchmark graphs in the literature which were commonly used to test new max-cut and max-bisection algorithms. Computational results show that ITS competes favorably with respect to the existing best performing max-bisection heuristics, by improving the

current best-known results (new lower bounds) on 10 instances.
– We proposed the first path relinking algorithm (PR-VSP) for solving the vertex separator problem (VSP) , which is composed of a reference set initialization and updating method, a solution improvement method, a path generation method and a solution selection method. The method to initialize and update a reference set is capable of maintaining a set of elite solutions with high quality and good diversity. The solution improvement method follows the framework of iterated tabu search, which alternates between a dedicated tabu search phase and a random perturbation phase. The tabu search procedure employs two complementary search operators (*1-move* and *swap-move*) to collectively perform neighborhood exploration, where the innovative *swap-move* operator is applied to solve VSP for the first time. The path generation method builds a solution path from an initiating solution to a guiding solution, on which a sequence of intermediate solutions are created by performing local moves based on a greedy selection mechanism. The solution selection method picks one or multiple solutions on the path which are submitted to the solution improvement method for quality optimization. Experimental assessment on four sets of benchmarks with a total of 355 instances discloses that our PR-VSP algorithm finds new best solutions (updated upper bounds) for 67 instances and matches previously best solutions for all except one instance.

Organization

The manuscript is organized in the following way:

– In the first chapter, we introduce the max-k-cut problem, the max-bisection problem and the vertex separator problem, and then provide an overview of three classes of approaches for solving them, including approximation algorithms, exact algorithms and heuristic/metaheuristic algorithms reported in the literature.
– In the second chapter, we first present the proposed multiple operator heuristic (MOH) for solving the max-k-cut problem, in which the five different search operators, the bucket sorting technique for fast move gain evaluation and updating, and the designed three search phases in MOH (descent-based improvement phase for intensified search, diversified improvement phase for discovering promising region, and perturbation phase for strong diversification) are described in detail. Then, we provide computational results and comparisons with state-of-the-art algorithms in the literature. Finally, we analyze the role of several important ingredients of the proposed algorithm.
– In the third chapter, we first present the general working scheme of the iterated tabu search (ITS) algorithm for the max-bisection problem. Then we detail the *1-move* and *c-swap* move operators and explain the three search phases to employ them, including the descent local search phase to locate local optima, the diversifying improvement phase to discover promising region, as well as the perturbation phase for strong diversification. In the following, we provide experimental results of our proposed algorithm and comparisons with other best performing algorithms in the literature. Meanwhile, we analyze the bucket-sorting based tie breaking strategies and the impact of the combined use of *1-move* and *c-swap* operators.
– In the fourth chapter, we present the proposed path relinking algorithm (PR-VSP) for solving the vertex separator problem. First, we expose the main scheme of the proposed algorithm and explain each of its internal components. Then we show our computational results and comparisons with state-of-the-art algorithms in the literature. Finally, we analyze the effectiveness of the designed new local search operator and the dedicated path-relinking procedure to the performance of the PR-VSP algorithm.
– In the last chapter, we give a general conclusion of this thesis and propose some perspectives.

1

Introduction

Graph partitioning problems are a class of well-known NP-hard combinatorial optimization problems with a wide range of applications. In this chapter, we introduce the three graph partitioning problems which are studied in the work: the max-k-cut, max-bisection and vertex separator problems and review state-of-the-art approaches for solving these problems in the literature.

Contents

1.1 Max-k-cut problem

1.1.1 Problem introduction

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subset V \times V$, each edge $(i, j) \in E$ being associated with a weight $w_{ij} \in Z$. Given $k \in [2, n]$, the max-k-cut problem is to partition the vertex set V into k (k is given) disjoint subsets $\{S_1, S_2, \ldots, S_k\}$, (i.e., $\bigcup_{i=1}^{k} S_i = V, S_i \neq \emptyset, S_i \cap S_j = \emptyset, \forall i \neq j$), such that the sum of weights of the edges from E whose endpoints belong to different subsets is maximized, i.e.,

$$\max \sum_{1 \leq p < q \leq k} \sum_{i \in S_p, j \in S_q} w_{ij}. \tag{1.1}$$

The max-k-cut is equivalent to the minimum k-partition (MkP) problem which aims to partition the vertex set of a graph into k disjoint subsets so as to minimize the total weight of the edges joining vertices in the same partition [Ghaddar *et al.*, 2011]. The k-way equipartition problem is a MkP with the restriction that the subsets in the partition are of equal size. The minimized version of max-k-cut is known as the k-way partitioning problem or graph k-partitioning problem [Karypis and Kumar, 1999]. If an additional constraint is added to the k-way partitioning problem requiring the difference of the cardinalities between the largest subset and the smallest subset is at most one, the problem is called balanced k-way partitioning.

The max-k-cut problem is a classical NP-hard problem in combinatorial optimization and can not be solved exactly in polynomial time unless $P = NP$ [Boros and Hammer, 1991; Kann *et al.*, 1997]. Moreover, when $k = 2$, the max-cut problem is one of the Karp's 21 NP-complete problems [Karp, 1972] which has been the subject of many studies in the literature. The max-k-cut and relevant graph partitioning problems have attracted increasing attention for its applicability to numerous important applications in the area of data mining [Ding *et al.*, 2001], VLSI layout design [Barahona *et al.*, 1988; Chang and Du, 1987; Chen *et al.*, 1983; Cho *et al.*, 1998; Pinter, 1984], frequency planning [Eisenblätter, 2002], sports team scheduling [Mitchell, 2003], and statistical physics [Liers *et al.*, 2004] among others.

Given the theoretical significance and large application potential, a number of solution procedures have been reported in the literature. In [Ghaddar *et al.*, 2011], the authors provide a review of several exact algorithms which are based on branch-and-cut and semidefinite programming approaches. But due to the high computational complexity of the problem, only instances of reduced size (i.e., $|V| < 100$) can be solved by these exact methods in a reasonable computing time. [Zhu *et al.*, 2013] proposed a discrete dynamic convexized (DC) method for solving the max-k-cut problem, which is characterized of the following two distinct features. Firstly, it formulates the max-k-cut problem into a nonlinear integer programming model for conveniently adapting the local search procedure proposed in [Fiduccia and Mattheyses, 1982]. Secondly, it employs an auxiliary function dependent on the similarity degree to help search escape from local optimum and direct search into promising search area. The drawback of the proposed DC method lies in expensive computational consumption as k increases.

After investigating the literature of graph partitioning problems, we find that very limited research directly aims at solving the max-k-cut problem ($k > 2$). However, there have been many researches for the related graph k-way partitioning problems. Hence, we review classic algorithms for the k-way partitioning problems in the following parts in hope of acquiring useful experiences for designing effective max-k-cut algorithms.

1.1.2 Exact approaches

[Ferreira *et al.*, 1996; Ferreira *et al.*, 1998] studied valid inequalities and designed a branch-and-cut algorithm for the problem where G is partitioned into at most k subsets and each subset has a capacity restriction on the sum of the nodes weights. [Mitchell, 2001; Mitchell, 2003] described an application of the k-way equipartition problem to the National Football League (NFL) and proposed a branch-and-cut algorithm.

1.1.3 Heuristic and metaheuristic approaches

[Kernighan and Lin, 1970] considered the graph partitioning problem where a given graph is partitioned into at most k subsets and each subset has at most p nodes. The basic idea of finding a near optimal k-way partition is to start from a feasible k-partition and continuously apply a 2-way partitioning procedure to pairs of subsets to make the partition near pairwise optimal. The designed 2-way partitioning heuristic repeats performing such pass that performs a varying length k of swap moves to produce a solution that maximally decreases the cut size of the given graph, until the best cut size can not be decreased. Experience shows that the proposed heuristic converges quickly.

[Fiduccia and Mattheyses, 1982] adapted the Kernighan-Lin heuristic for the 2-way partitioning problem in two aspects. The first is to move a vertex for each iteration instead of swapping a couple of vertices. The second is to use a bucket sorting technique to reduce the complexity of identifying the best move and of updating the move gains of vertices affected by each move. Experimental evaluation on several random-logic polycell designs indicates the effectiveness of the proposed algorithm.

[Fan and Pardalos, 2010] formulated the general graph partitioning problem as a zero-one quadratic programming model and studied equivalent zero-one linear integer programming formulations. Problem instances from various graphs and networks are represented by different formulations, which are then solved by the CPLEX optimization software. Computational comparisons reveal that all the formulations reach the same solution quality and the discrete quadratic formulation performs best in terms of computational time. In addition, bipartite graphs are also investigated and different quadratic and linear formulations are proposed. Experimental results show that the linear formulation with the fewest variables is most efficient.

[Rahimian *et al.*, 2015] proposed a distributed algorithm JA-BE-JA which employs local search for discovering high-quality solutions and simulated annealing for escaping from local optimum. The proposed algorithm is inherently parallel, which only needs to know local information of a graph instead of global knowledge of the entire graph as in most centralized algorithms. For a specific vertex, JA-BE-JA utilizes a hybrid sampling component to select its direct neighbors vertices or a randomized subset of vertices as candidates for swap moves. This is followed by a swapping component to choose another vertex for swapping that leads to the best utility function value. Extensive experiments disclose that JA-BE-JA outperforms well-known centralized algorithms METIS [Karypis and Kumar, 1998] on real-world graphs from social networks.

1.1.4 Multilevel graph partitioning approaches

When the size of graphs becomes very large (with up to millions of vertices), a research line advocates the use of multilevel algorithms for solving the graph k-partitioning problem, which approximates the initial problem by solving successively smaller (and easier) problems. The general multilevel scheme consists of a coarsening phase to produce a sequential level of smaller and coarser graphs, an initial partitioning phase to create an initial partition for the coarsest graph, as well as an uncoarsening phase to project the solution of the lower-level graph solved by a refinement procedure to its upper level graph. We introduce several representative multilevel graph partitioning algorithms and a detailed survey can be found in [Benlic and Hao, 2013c].

[Monien *et al.*, 2000] presented new theoretical based coarsening and local improvement methods for a multilevel graph partitioning paradigm. The proposed coarsening method utilizes a maximum weighted matching scheme in edge weighted graphs, which reaches a time complexity of $O(E)$ to calculate a matching with edge weight of at least 1/2 of that of the maximum weighted matching. The local search improvement uses a Helpful-Set strategy that reaches a theoretical upper bound of $((k-1)/2)|V|+1$ for partitioning graphs with maximum degree of $2k$ into two parts. Computational experience indicates that neither the proposed algorithm nor any state-of-the-art solver performs best in terms of all the measurements.

[Soper *et al.*, 2004] proposed a hybrid metaheuristic that combines an evolutionary search algorithm with a multilevel graph partitioner. The coarsening phase continuously contracts the series of graphs by heuristically constructing a maximal independent subset of edges until the number of vertices in the coarsest graph equals the number of the subsets k. At each graph level a multi-way variant of the Kernighan-Lin 2-way partitioning heuristic is used to find a refined partition, in which small non-integer biases are added to the edge weights to influence the partition. Crossover and mutation operators in the evolutionary algorithm utilize a multilevel graph partitioning heuristic to produce offspring solutions. The proposed algorithm is able to attain much better solution quality than state-of-the-art graph partitioning packages but needs significantly long running time.

[Benlic and Hao, 2011a] developed a multilevel tabu search algorithm for solving the balanced graph k-partitioning. The coarsening phase employs heavy-edge matching to produce a series of coarser graphs. The main originality of the proposed algorithm lies in a perturbation-based tabu search algorithm for partition refinement, which integrates a neighborhood combination to conduct neighborhood exploration, an adaptation of bucket sorting for quickly calculating objective gains of performing a move, a frequency memory to guide selection strategies for vertex migration, as well as a dynamic tabu tenure technique. Extensive testings on benchmark graphs from the graph partitioning archive, with the number of subsets k set as 2, 4, 8, 16, 32 and 64, indicate that the proposed algorithm outperforms the two state-of-the-art solvers METIS [Karypis and Kumar, 1998] and CHACO [Hendrickson and Leland, 1995] no matter in short or long running time.

[Benlic and Hao, 2011b] investigated a multilevel memetic algorithm that uses the same multilevel framework as the previous multilevel tabu search algorithm and differs in the partition refinement. The memetic algorithm combines a backbone guided multiparent crossover operator to enhance search diversification with a perturbation-based tabu search to ensure search intensification. Extensive experiments indicate that the proposed algorithm outperforms any existing algorithms in terms of solution quality. In addition, the roles of the backbone guided crossover operator and several other key issues are analyzed to show their merits to the performance of the proposed algorithm.

1.1.5 Summary

The max-k-cut problem has important theoretical significance and large application potential. Although heuristic and metaheuristic approaches for solving several other graph partitioning problems have been widely studied and demonstrated to be highly effective for finding near optimal solutions for large benchmark graphs, so far only one heuristic algorithm has been presented for the max-k-cut problem. Hence, the first work of this thesis is dedicated to developing an effective heuristic for handling the max-k-cut problem. For this purpose, we design a multiple operator heuristic (MOH), which employs five distinct search operators organized into three search phases to effectively explore the search space.

1.2 Max-cut problem

1.2.1 Problem introduction

Max-cut is a special case of the max-k-cut problem with $k = 2$. Given that max-cut has been among the widely studied NP-hard combinatorial optimization problems, we take the review of the max-cut references as an independent part. Theoretical results for the max-cut problem include the followings. [Orlova and Dorfman, 1972] established a polynomial time method for finding a maximum cut in a planar graph. [Bodlaender and Jansen, 2000] proved that max-cut is NP-hard for chordal, split, and 3-colorable graphs. [Scott and Sorkin, 2004] proved that a maximum cut of a sparse random graph can be solved in polynomial expected time.

1.2.2 Approximation approaches

There exist many approximation algorithms in the literature that provide an approximate solution guaranteed to be within an approximation ratio to its optimal value. [Sahni and Gonzalez, 1976] proposed an approximation algorithm with a 0.5 performance guarantee for the max-cut. [Goemans and Williamson, 1995] devised semidefinite relaxation to improve the approximation ratio to 0.878. [Homer and Peinado, 1997] developed parallel approximation algorithms for solving large graphs up to 13000 vertices. By coupling a projected gradient method for the max-cut semidefinite relaxation with a randomized method, [Burer and Monteiro, 2001] presented an effective approximation algorithm for the max-cut problem. Due to an order of magnitude increase of problem variables, the semidefinite relaxation method encounters difficulty in solving large scale problem instances. [Burer *et al.*, 2002] proposed a rank-two heuristic that considers tradeoff between computational efficiency and a theoretical guarantee. [Kahruman *et al.*, 2007] presented the first greedy worst-out construction heuristic to establish an approximation ratio of at least 1/3.

1.2.3 Exact approaches

[Mohar and Poljak, 1990] presented an upper bound for max-cut based on the maximum eigenvalue of an associated matrix and identified different classes of graphs where the obtained upper bound is satisfactory or poor. [Croce *et al.*, 2007] studied an exact algorithm, which enumerates cuts for a subgraph of the original graph G and then extends them to find optimal cuts in G, for computing a maximum cut in graphs with bounded maximum degree and in general graphs. [Rendl *et al.*, 2007] devised a branch-and-bound algorithm for max-cut based on semidefinite relaxation tightened by triangle inequalities, where the resulting

relaxation is solved by an interior-point method combined with a bundle method. [Rendl *et al.*, 2010] extended the previous branch-and-bound algorithm by using a dynamic version of the bundle method to solve the semidefinite relaxation for max-cut together with triangle inequalities. The proposed branch-and-bound algorithm is able to prove optimality for graphs with $|V| = 100$ nodes and for sparse graphs with $|V| = 300$ nodes. [Krislock *et al.*, 2014] presented an branch-and-bound algorithm for finding exact solutions, which improves standard semidefinite relaxation bounds by adding a quadratic regularization term to semidefinite relaxation and employing a quasi-Newton method to compute the bounds. Due to high computational complexity, graphs with $n \geq 500$ is beyond the reach of these exact methods. Hence, heuristic and metaheuristic methods are commonly used for approximating large graphs.

1.2.4 Heuristic and metaheuristic approaches

[Festa *et al.*, 2002] investigated pure and hybrid metaheuristics among greedy randomized adaptive search procedure (GRASP), variable neighborhood search (VNS) and path relinking (PR) for handling the max-cut problem. Computational experience discloses that the use of VNS in the local search phase of GRASP and path relinking in VNS for search intensification is capable of obtaining solution improvement with a little additional time. Among the proposed metaheuristic algorithms, GRASP with PR is the fastest to converge to a near optimal solution and the VNS with PR finds best quality solutions at the expense of longest running time.

[Palubeckis, 2004] designed two multi-start tabu search implementations, which differ in the strategies to produce initial solutions. The first one called MST produces initial solutions by using a variable fixing procedure and a steepest ascent procedure to the reformulated problem obtained by removing the fixed variables. The second one called RRT is a traditional random restart strategy that generates initial solutions in a random way. Computational comparisons indicate that MST performs better than RRT in particular to the best solution quality.

[Marti *et al.*, 2009] presented a scatter search algorithm with the following new elements 1) the solution of the maximum diversity problem is used to increase diversity in the reference set; 2) the length of the ejection chain for the compound moves is adjusted dynamically; 3) a probabilistic-based mechanism is incorporated to select a solution combination method. Experimental study is conducted to compare scatter search with state-of-the-art algorithms in the literature and discloses the effectiveness of the proposed new elements.

[Arráiz and Olivo, 2009] investigated tabu search and simulated annealing algorithms for solving the max-cut problem. Computational experience indicates that tabu search is suitable for finding high quality solutions in small computational time while simulated annealing is suggested when top quality is required within medium computational efforts.

[Shylo and Shylo, 2010; Shylo *et al.*, 2012] developed global equilibrium search algorithms for the max-cut problem, which borrows the idea of annealing curve to determine initial solutions and uses tabu search for solution improvement. Based on the linear temperature function $\mu_{k+1} = \alpha\mu_k$, the initial temperature is determined according to the rule that the probability vector obtained by the last temperature produces an initial solution which is approximately equal to the best found solution during the search process. The method to calculate the probability of assigning a vertex to each partition relies both on the current temperature value and a subset of the previously visited high-quality solutions. Experimental results show that the proposed algorithm performs quite well in terms of both solution quality and computational efficiency.

[Lin and Zhu, 2012] takes a simple modification of the Fiduccia-Mattheyses heuristic as local search

to maximize such an auxiliary function that has the same global maximizer as the max-cut problem. By increasing the value of the parameter in the auxiliary function, the algorithm enables search escape from local maximizers. Experimental results reveal the effectiveness of the proposed algorithm.

[Wang *et al.*, 2012; Kochenberger *et al.*, 2013] reformulated the max-cut problem into unconstrained binary quadratic programming (UBQP) and applied a general UBQP algorithm for solving the reformulated problem. Computational testing shows that the general UBQP approach is able to produce high quality solutions for large scale problem instances and outperform several specially tailored max-cut algorithms.

[Wu and Hao, 2012] devised a memetic algorithm that integrates a grouping based multi-parent crossover operator to maximally preserve the common solution components among parents and an iterated tabu search procedure based on a random perturbation mechanism to conduct neighborhood exploration. Evaluated on graphs with up to 10000 vertices, the proposed algorithm is demonstrated to be highly effective in discovering high quality solutions. Additional analysis shows the importance of the devised crossover operator to the success of the proposed algorithm.

[Benlic and Hao, 2013a] presented a breakout local search, which jointly uses local search to discover an attractor and an adaptive perturbation strategy to escape from the basin of attraction. The local search procedure uses steepest ascent to reach local optimum, where each iteration displaces such a vertex from its current subset into the other subset that produces the best objective gain. The perturbation strategy collectively utilizes directed perturbation and random perturbation operators to increase search intensification and diversification. Extensive testings show that the break local search outperforms other state-of-the-art algorithms in the literature.

1.2.5 Summary

The max-cut problem is the most widely studied case of the max-k-cut problem and various approaches have been reported in the literature. Exact algorithms can only solve small benchmark graphs with no more than 100 vertices in a reasonable computing time. For handling large benchmark graphs, many heuristic and metaheuristic algorithms are accordingly proposed. Given that the max-cut problem can be directly solved by the max-k-cut algorithm, an important part of the first work in this thesis is to verify the performance of our proposed max-k-cut algorithm by comparing with best performing max-cut algorithms.

1.3 Max-bisection problem

1.3.1 Problem introduction

Given an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$, edge set $E \subset V \times V$ and a set of edge weights $\{w_{ij} \in \mathbb{Z} : (i, j) \in E\}$ ($w_{ij} = 0$ if $(i, j) \notin E$). The maximum bisection problem (max-bisection for short) is to partition the vertex set V into two disjoint subsets S_1 and S_2 of equal cardinality (i.e., $S_1 \cup S_2 = V, S_1 \cap S_2 = \emptyset, |S_1| = |S_2|$), such that the weight sum of the edges whose endpoints belong to different subsets is maximized, i.e.,

$$\max \sum_{i \in S_1, j \in S_2} w_{ij}. \tag{1.2}$$

If this objective function is to be minimized, the problem is known as minimum equicut, graph bisection, min-bisection or *balanced* graph bipartitioning. Max-bisection is a cardinality constrained max-cut problem with the restriction that the two subsets in the partition are of equal size.

Max-bisection is a fundamental graph partitioning problem and cannot be solved exactly in polynomial time unless $P = NP$ [Murty and Kabadi, 1987]. It has attracted increasing attention in recent decades due to its relevance to numerous applications like VLSI layout design [Barahona *et al.*, 1988; Chang and Du, 1987; Cho *et al.*, 1998], data clustering [Ding *et al.*, 2001] and sports team scheduling [Elf *et al.*, 2003] among others.

1.3.2 Approximation approaches

[Frieze and Jerrum, 1995] extended the approach in [Goemans and Williamson, 1995] to max-bisection and obtained a randomized 0.651-approximation algorithm. By combining this method with rotation argument applied to the optimal solution of the semidefinite relaxation of max-bisection, [Ye, 2001] improved the performance ratio to 0.699. [Halperin and Zwick, 2002] further improved the approximation ratio to 0.7016 by adding triangle inequalities to the max-bisection formulation.

1.3.3 Exact approaches

[Conforti *et al.*, 1990a; Conforti *et al.*, 1990b] studied the facial structure of equicut and s-t equicut polytopes, and several classes of facet-inducing inequalities. Building upon these theoretical knowledge, an integer programming based branch-and-cut approach for the equicut problem was implemented in [Brunetta *et al.*, 1997]. A branch-and-cut algorithm based on semidefinite programming and polyhedral relaxation for the graph bisection problem with each subset of prespecified size was described and is shown to be particularly effective for special classes of graphs such as planar and grid graphs [Karisch *et al.*, 2000]. A computational comparison within a branch-and-cut framework to evaluate relative strength between integer programming and semidefinite programming formulations is presented in [Armbruster *et al.*, 2008; Armbruster *et al.*, 2012]. [Anjos *et al.*, 2013] compared basic linear and semidefinite relaxations for calculating bounds of the equicut problem and presented an improved version of the branch-and-cut algorithm proposed in [Brunetta *et al.*, 1997]. [Delling *et al.*, 2015] presented a novel exact algorithm within a branch-and-cut framework, which introduces packing trees based lower bounds and a new decomposition technique. This algorithm works particularly well on graphs with relatively small minimum bisections and it remarkably solves several large real-world instances with up to millions of vertices to optimality.

1.3.4 Heuristic and metaheuristic approaches

[Battiti and Bertossi, 1999] presented a reactive randomized tabu search algorithm for the min-bisection problem, which integrates a min-max greedy construction with an adaptive choice of tabu tenure. Each iteration in the min-max greedy construction adds such a vertex to a subset that minimizes the cut, and meantime maximizes the number of edges with the other vertices in the subset for breaking ties. Extensive testings indicate that the proposed algorithm performs better than previous state-of-the-art algorithms and obtains significantly better results than multilevel algorithms for large "real world" graphs at the cost of a much large computational time.

[Inayoshi and Manderick, 1994; Bui and Moon, 1996; Steenbeek *et al.*, 1998; Merz and Freisleben,

2000] presented hybrid evolutionary algorithms (memetic algorithms) for solving min-bisection, which differ in strategies of solution representation, mating selection, genetic and mutation crossovers, as well as local search. Computational comparisons indicate that the memetic algorithm proposed in [Merz and Freisleben, 2000] outperforms all the other algorithms.

[Chardaire *et al.*, 2007] presented a population reinforced optimization based exploration (PROBE) heuristic for min-bisection, which uses a population to determine search subspaces of optimal solutions. The method of generating initial solutions is to first construct a partial bisection by fixing vertices shared by two parent solutions and then use a differential-greedy heuristic, in which each iteration selects such a vertex to join the subset that maximizes the difference of the internal degree and the external degree proposed in [Battiti and Bertossi, 1997], to form a bisection of the full graph. The local search designed in [Bui and Moon, 1996] is utilized for bisection refinement. Experimental results indicate that PROBE compares favorably with other population based solution methods, randomized reactive tabu search, and more specialized multilevel partitioning techniques.

[Dang *et al.*, 2002] formulated the max-bisection problem into a linearly constrained continuous optimization problem and developed a deterministic annealing algorithm based on a square-root barrier function and a feasible descent direction. The convergence of the proposed algorithm is proved. Numerical results indicate that the proposed algorithm is faster than the .699 approximation algorithm, while attaining more or less the same solution quality.

[Kohmoto *et al.*, 2003] presented a genetic algorithm with a local search incorporated for solving the min-bisection problem. Experiments on well-known benchmark graphs disclose that the proposed algorithm performs better than multi-start local search and simulated annealing algorithms.

[Ling *et al.*, 2008] reformulated max-bisection into max-cut by combing the cardinality constraint with the objective function, which is solved by the VNS algorithm for the max-cut problem proposed in [Festa *et al.*, 2002]. Numerical comparisons with .699 approximation algorithm indicate that VNS performs better in terms of both solution quality and computational time for most test problems.

[Xu *et al.*, 2011] proposed a Lagrangian net algorithm, which uses a penalty function method to relax the constraint and a discrete Hopfield neural network to find near optimal bisections. During the search, the penalty factor is adjusted to help search escape from local attractors. The convergence analysis of the proposed algorithm is provided. Computational experience shows that the proposed algorithm performs much better than other relaxation methods.

[Wu and Hao, 2013] presented a memetic algorithm for the max-bisection problem, which is characterized of a diversification-guided grouping crossover operator, a tabu search optimization procedure and a quality-and-diversity based population updating strategy. Experimental comparisons disclose that the proposed memetic algorithm performs better than the Lagrangian net algorithm. Furthermore, although the max-bisection problem includes the balance constraint which is not required in the max-cut problem, computational comparisons with excellent max-cut algorithms show that the proposed max-bisection algorithm is able to obtain solution improvement for many problem instances. Finally, additional experiments on the structure similarity analysis between high quality solutions, the dynamic tabu tenure management and the pool updating strategy are performed to shed light on the merit of these key ingredients to the performance of the proposed memetic algorithm.

[Lin and Zhu, 2014] proposed a memetic algorithm, which integrates a Fiduccia-Mattheyses heuristic to refine solutions, a crossover operator to produce offspring solutions, and a distance-quality based population updating strategy. Evaluated on a number of benchmarks, the proposed memetic algorithm performs better than CirCut and Lagrangian net algorithm in terms of both solution quality and computational efforts.

1.3.5 Summary

Max-bisection is a cardinality constrained max-cut problem and is also a computationally challenging problem. It has attracted increasing attention in recent decades due to its relevance to numerous applications. To solve this problem, many solution procedures have been reported in the literature. Given that the proposed multiple operator based heuristic performs quite effectively for the max-k-cut problem, it is worthy of investigating its performance to the interesting max-bisection problem. Hence, the second work of this thesis is to design an iterated tabu search algorithm to solve max-bisection, which collectively employs two distinct search operators organized into three search phases to explore the search space in an efficient way.

1.4 Vertex separator problem

1.4.1 Problem introduction

Given an undirected graph G (which may be disconnected) with an vertex set $V = \{v_1, \ldots, v_n\}$ where each vertex v_i is associated with a non-negative weight w_i and an unweighted edge set E, the vertex separator problem (VSP) is to partition V into three non-empty disjoint subsets A, B and C such that the total weight of vertices in C is minimized subject to two constraints: (i) there is no edge between A and B and (ii) the cardinality of A and B does not exceed a given positive integer b. Set C is called the separator of G while A and B are called the shores of the separator. Formally, VSP is formulated as follows:

$$\min \quad \sum_{i \in C} w_i \tag{1.3}$$

$$\text{subject to} \quad C = V \setminus (A \cup B), (A \times B) \cap E = \emptyset, A \cap B = \emptyset \tag{1.4}$$

$$\max\{|A|, |B|\} \leq b \tag{1.5}$$

$$A, B, C \subset V \tag{1.6}$$

where constraint (2) ensures that no edge exists for any pair of vertices between shores A and B and constraint (3) requires both A and B contain no more than b vertices. A separator C is considered as balanced if $\max\{|A|, |B|\} \leq 2|V|/3$.

This VSP problem was first introduced in the domain of Very Large Scale Integration (VLSI) design [Leighton and Rao, 1999]. Additional applications of VSP include, for instance, detection of brittle nodes in telecommunication networks [Biha and Meurs, 2011], identification of the minimal separator in the divide-and-conquer based graph algorithms [Evrendilek, 2008; Lipton and Tarjan, 1979] as well as finding protein conformation in bioinformatics [Fu and Chen, 2006]. From the point view of computational complexity, VSP is known to be NP-hard for general graphs [Bui and Jones, 1992] and even for planar graphs [Fukuyama, 2006].

1.4.2 Approximation approaches

[Leighton, 1983] presented an approximation algorithm based on a linear relaxation technique and achieved an approximation ratio of $O(log\ n)$. [Feige *et al.*, 2005] improved this result to $O(\sqrt{log\ n})$ by

utilizing a semidefinite relaxation method.

1.4.3 Exact approaches

There are also several exact algorithms able to solve medium scale VSP instances. [de Souza and Balas, 2005] designed a branch-and-cut algorithm which explored valid polyhedral inequalities obtained in [Balas and de Souza, 2005] and conducted extensive computational experiments. [de Souza and Cavalcante, 2011] proposed a hybrid algorithm that combines Lagrangian relaxation with cutting plane techniques. Computational results showed that the hybrid algorithm outperforms the best exact algorithm available. [Biha and Meurs, 2011] presented an exact approach based on a new class of valid inequalities and provided experimental comparisons with the algorithm in [de Souza and Balas, 2005].

1.4.4 Heuristic and metaheuristic approaches

In addition to the above approximation and exact approaches, heuristic and metaheuristic algorithms have been devised to obtain good approximate solutions for large VSP instances in reasonable computing time. We provide in the following a description of state-of-the-art heuristic and metaheuristic algorithms from the literature.

[Benlic and Hao, 2013b] presented a breakout local search (BLS) algorithm for VSP which combines a local search procedure with an adaptive perturbation procedure. The local search procedure is based on a dedicated move operator which transforms the incumbent solution to a neighbor solution by displacing a vertex v from the separator C to the shore subset A or B, followed by displacing all the adjacent vertices of v from the opposite shore subset to the separator C. Each iteration performs such a move that leads to a neighbor solution with the largest objective improvement. The perturbation procedure employs an adaptive selection mechanism to apply either a directed perturbation or a random perturbation to escape locally optimal solutions and direct the search toward unexplored areas. Experimental results on benchmark instances with up to 3000 vertices demonstrate the efficacy of the BLS method.

[Sánchez-Oro *et al.*, 2014] introduced several variable neighborhood search (VNS) algorithms for solving the VSP, which alternates between a local search phase and a shaking phase. Two initial solution constructive procedures (random and greedy) are proposed to generate seeding solutions. The local search phase relys on three types of basic moves and two delicate combined moves to attain a local optimum. A variable neighborhood descent procedure is then used to further improve the quality of a locally optimal solution by an alternating use of two combined neighborhoods. The shaking phase carries out random perturbations to produce new solutions without violating the feasibility conditions. Extensive experiments on benchmark instances with up to 1000 vertices disclose the effectiveness of the proposed VNS algorithms.

[Hager and Hungerford, 2015] proposed a continuous optimization approach. The problem is formulated as a continuous bilinear quadratic program, which is solved by a multilevel algorithm. Following the general multilevel graph approach, the proposed algorithm is composed of three phases including 1) a coarsening phase that hierarchically coarsens a graph into a sequence of coarser (smaller) graphs; 2) a refinement phase that solves the graph in the coarsest level to obtain a separator; 3) an uncoarsening phase that propagates the separator back to the hierarchy to obtain the solution for the original graph. Both mountain climbing and Fiduccia–Mattheyses heuristics are investigated for solving each hierarchy of graphs. Experimental results show that the proposed continuous program based heuristics in a multilevel framework outperform METIS in terms of solution quality for a large test set of graphs with between 1000 and 5000

vertices. However, the proposed continuous optimization approach is outperformed by the state-of-the-art BLS metaheuristic.

1.4.5 Summary

The vertex separator problem is another intriguing graph partitioning problem and has received more attention in recent years. Population based approaches are capable of attaining a good balance between intensification and diversification during the search and have shown highly effective for solving large scale hard combinatorial optimization problems. The third work of this thesis is to devise an effective algorithm within a population based search framework for challenging vertex separator problem instances. For this purpose, we follow the path relinking framework to design the first path relinking algorithm, in which specific path relinking components targeted to the vertex separator problem are developed. In particular, the solution improvement component is designed along the previous successful research line of the multiple operator based heuristic for detecting high quality solutions.

2

A multiple search operator heuristic for the max-k-cut problem

In this chapter, we present a multiple operator heuristic (MOH) for the general max-k-cut problem. MOH employs five distinct search operators organized into three search phases to effectively explore the search space. Experiments on two sets of 91 well-known benchmark instances show that the proposed algorithm is highly effective on the max-k-cut problem and improves the current best known results (lower bounds) of most of the tested instances for $k \in [3, 5]$. For the popular special case $k = 2$ (i.e., the max-cut problem), MOH also performs remarkably well by discovering 4 improved best known results. We provide additional studies to shed light on the key ingredients of the algorithm. The content of this chapter is based on an article submitted to Annals of Operations Research which was revised in March 2016.

Contents

2.1 Introduction

This chapter is dedicated to the max-k-cut problem which was introduced in Chapter 1. Recall that max-k-cut is to partition the vertices of an edge-weighted graph $G = (V, E)$ into $k \geq 2$ disjoint subsets such that the weight sum of the edges crossing the different subsets is maximized. We propose a new and effective multiple operator heuristic for the general max-k-cut problem. The main originality of the proposed algorithm is its multi-phased multi-strategy approach which relies on five distinct local search operators for solution transformations. These operators are organized into three different search phases (descent-based improvement, diversified improvement, perturbation) to ensure an effective examination of the search space. The basic idea of our approach is as follows. The descent-based improvement procedure aims to locate a good local optimum from an initiating solution. This is achieved with two dedicated intensification operators. Then the diversified improvement phase discovers promising areas around the obtained local optimum by applying two additional operators. Once an improved solution is found, the search switches back to the descent-based improvement phase to make an intensive exploitation of the regional area. If the search is trapped in a deep local optimum, the perturbation phase applies a random search operator to definitively lead the search to a distant region from which a new round of the three-phased search procedure starts. This process is repeated until a stop condition is met.

We assess the performance of the proposed algorithm on two sets of well-known benchmarks with a total of 91 instances which are commonly used to test max-k-cut and max-cut algorithms in the literature. Computational results show that the proposed algorithm competes very favorably with respect to the existing max-k-cut heuristics, by improving the current best known results on most instances for $k \in [3, 5]$. Moreover, for the very popular max-cut problem ($k = 2$), the results yielded by our algorithm remain highly competitive compared with the most effective and dedicated max-cut algorithms. In particular, our algorithm manages to improve the current best known solutions for 4 (large) instances, which were previously reported by specific max-cut algorithms of the literature.

This chapter is organized as follows. Section 2.2 describes the general scheme and the components of our proposed multiple search operator heuristic for max-k-cut. Detailed computational results and comparisons with state-of-the-art algorithms are presented in Section 2.3. Before concluding, Section 2.4 is dedicated to an analysis of several essential parts of the proposed algorithm.

2.2 Multiple search operator heuristic for max-k-cut

2.2.1 General working scheme

The proposed multiple operator heuristic algorithm (MOH) for the general max-k-cut problem is described in Algorithm 1 whose components are explained in the following subsections. The algorithm ex-

Algorithm 1 General procedure for the max-k-cut problem

1: **Input**: Graph $G = (V, E)$, number of partitions k, max number ω of diversified moves, max number ξ of consecutive non-improvement rounds of the descent improvement and diversified improvement phases before the perturbation phase, probability ρ for applying operator O_3, γ the perturbation strength.
2: **Output**: the best solution I_{best} found so far
3: $I \leftarrow$ Generate_initial_solution(V, k) ▷ I is a partition of V into k subsets
4: $I_{best} \leftarrow I$ ▷ I_{best} Records the best solution found so far
5: $f_{lo} \leftarrow f(I)$ ▷ f_{lo} Records the objective value of the latest local optimum reached by $O_1 \cup O_2$
6: $f_{best} \leftarrow f(I)$ ▷ f_{best} Records the best objective value found so far
7: $c_{non\_impv} \leftarrow 0$ ▷ Counter of consecutive non-improvement rounds of descent and diversified search
8: **while** stopping condition not satisfied **do**
9: /* lines 10 to 19: Descent-based improvement phase by applying O_1 and O_2, see Section 2.2.4*/
10: **repeat**
11: **while** $f(I \oplus O_1) > f(I)$ **do** ▷ Descent Phase by applying operator O_1
12: $I \leftarrow I \oplus O_1$ ▷ Perform the move defined by O_1
13: Update Δ ▷ Δ is the bucket structure recording move gains for vertices, see Section 2.2.5
14: **end while**
15: **if** $f(I \oplus O_2) > f(I)$ **then** ▷ Descent Phase by applying operator O_2
16: $I \leftarrow I \oplus O_2$
17: Update Δ
18: **end if**
19: **until** I can not be improved by operator O_1 and O_2
20: $f_{lo} \leftarrow f(I)$
21: **if** $f(I) > f_{best}$ **then**
22: $f_{best} \leftarrow f(I)$; $I_{best} \leftarrow I$ ▷ Update the best solution found so far
23: $c_{non\_impv} \leftarrow 0$ ▷ Reset counter c_{non_impv}
24: **else**
25: $c_{non\_impv} \leftarrow c_{non\_impv} + 1$
26: **end if**
27: /* lines 28 to 38: Diversified improv. phase by applying O_3 and O_4 at most ω times, see Section 2.2.4 */
28: $c_{div} \leftarrow 0$ ▷ Counter c_{div} records number of diversified moves
29: **repeat**
30: **if** $Random(0, 1) < \rho$ **then** ▷ Random(0,1) returns a random real number between 0 to 1
31: $I \leftarrow I \oplus O_3$
32: **else**
33: $I \leftarrow I \oplus O_4$
34: **end if**
35: Update H (H, λ) ▷ Update tabu list H where λ is the tabu tenure, see Section 2.2.4
36: Update Δ
37: $c_{div} \leftarrow c_{div} + 1$
38: **until** $c_{div} > \omega$ or $f(I) > f_{lo}$
39: /* Perturbation phase by applying O_5 if f_{best} not improved for ξ rounds of phases 1-2, see Sect. 2.2.8 */
40: **if** $c_{non\_impv} > \xi$ **then**
41: $I \leftarrow I \oplus O_5$ ▷ Apply random perturbation γ times, see Section 2.2.8
42: $c_{non\_impv} \leftarrow 0$
43: **end if**
44: **end while**

plores the search space (Section 2.2.2) by alternately applying five distinct search operators (O_1 to O_5) to make transitions from the current solution to a neighbor solution (Section 2.2.4). Basically, from an initial solution, the descent-based improvement phase aims, with two operators (O_1 and O_2), to reach a local optimum I (Alg. 1, lines $10 - 19$, descent-based improvement phase, Section 2.2.6). Then the algorithm continues to the diversified improvement phase (Alg. 1, lines $28 - 38$, Section 2.2.7) which applies two other operators (O_3 and O_4) to locate new promising regions around the local optimum I. This second phase ends once a better solution than the current local optimum I is discovered or when a maximum number of diversified moves ω is reached. In both cases, the search returns to the descent-based improvement phase with the best solution found as its new starting point. If no improvement can be obtained after ξ descent-based improvement and diversified improvement phases, the search is judged to be trapped in a deep local optimum. To escape the trap and jump to an unexplored region, the search turns into a perturbation-based diversification phase (Alg. 1, lines $40 - 43$), which uses a random operator (O_5) to strongly transform the current solution (Section 2.2.8). The perturbed solution serves then as the new starting solution of the next round of the descent-based improvement phase. This process is iterated until the stopping criterion (typically a cutoff time limit) is met.

2.2.2 Search space and evaluation solution

Recall that the goal of max-k-cut is to partition the vertex set V into k subsets such that the sum of weights of the edges between the different subsets is maximized. As such, we define the search space Ω explored by our algorithm as the set of all possible partitions of V into k disjoint subsets, $\Omega = \{\{S_1, S_2, \ldots, S_k\} : \bigcup_{i=1}^{k} S_i = V, S_i \cap S_j = \emptyset, S_i \subset V, \forall i \neq j\}$, where each candidate solution is called a k-cut.

For a given partition or k-cut $I = \{S_1, S_2, \ldots, S_k\} \in \Omega$, its objective value $f(I)$ is the sum of weights of the edges connecting two different subsets:

$$f(I) = \sum_{1 \leq p < q \leq k} \sum_{i \in S_p, j \in S_q} w_{ij}. \tag{2.1}$$

Then, for two candidate solutions $I' \in \Omega$ and $I'' \in \Omega$, I' is better than I'' if and only if $f(I') > f(I'')$. The goal of our algorithm is to find a solution $I_{best} \in \Omega$ with $f(I_{best})$ as large as possible.

2.2.3 Initial solution

The MOH algorithm needs an initial solution to start its search. Generally, the initial solution can be provided by any eligible means. In our case, we adopt a randomized two step procedure. First, from k empty subsets $S_i = \emptyset, \forall i \in \{1, \ldots, k\}$, we assign each vertex $v \in V$ to a random subset $S_i \in \{S_1, S_2, \ldots, S_k\}$. Then if some subsets are still empty, we repetitively move a vertex from its current subset to an empty subset until no empty subset exists.

2.2.4 Move operations and search operators

Our MOH algorithm iteratively transforms the incumbent solution to a neighbor solution by applying some *move* operations. Typically, a move operation (or simply a move) changes slightly the solution, e.g., by transferring a vertex to a new subset. Formally, let I be the incumbent solution and let mv be a move, we use $I' \leftarrow I \oplus mv$ to denote the neighbor solution I' obtained by applying mv to I.

Associated to a move operation mv, we define the notion of *move gain* Δ_{mv}, which indicates the objective change between the incumbent solution I and the neighbor solution I' obtained after applying the move, i.e.,

$$\Delta_{mv} = f(I') - f(I) \tag{2.2}$$

where f is the optimization objective (see Formula (2.1)).

In order to efficiently evaluate the move gain of a move, we develop dedicated techniques which are described in Section 2.2.5. In this work, we employ two basic move operations: the *'single-transfer move'* and the *'double-transfer move'*. These two move operations form the basis of our five search operators.

– Single-transfer move (*st*): Given a k-cut $I = \{S_1, S_2, \ldots, S_k\}$, a vertex $v \in S_p$ and a target subset S_q with $p, q \in \{1, \ldots, k\}, p \neq q$, the 'single-transfer move' displaces vertex $v \in S_p$ from its current subset S_p to the target subset $S_q \neq S_p$. We denote this move by $st(v, S_p, S_q)$ or $v \rightarrow S_q$.

– Double-transfer move (*dt*): Given a k-cut $I = \{S_1, S_2, \ldots, S_k\}$, the 'double-transfer move' displaces vertex u from its subset S_{cu} to a target subset $S_{tu} \neq S_{cu}$, and displaces vertex v from its current subset S_{cv} to a target subset $S_{tv} \neq S_{cv}$. We denote this move by $dt(u, S_{cu}, S_{tu}; v, S_{cv}, S_{tv})$ or $dt(u, v)$, or still dt.

From these two basic move operations, we define five distinct *search operators* $O_1 - O_5$ which indicate precisely how these two basic move operations are applied to transform an incumbent solution to a new solution. After an application of any of these search operators, the move gains of the impacted moves are updated according to the dedicated techniques explained in Section 2.2.5.

– **The O_1 search operator** applies the single-transfer move operation. Precisely, O_1 selects among the $(k-1)n$ single-transfer moves a best move $v \rightarrow S_q$ such that the induced move gain $\Delta_{(v \rightarrow S_q)}$ is maximum. If there are more than one such moves, one of them is selected at random. Since there are $(k-1)n$ candidate single-transfer moves from a given solution, the time complexity of O_1 is bounded by $O(kn)$. The proposed MOH algorithm employs this search operator as its main intensification operator which is complemented by the O_2 search operator to locate good local optima (see Alg. 1, lines $10-19$ and Section 2.2.6).
– **The O_2 search operator** is based on the double-transfer move operation and selects a best dt move with the largest move gain Δ_{dt}. If there are more than one such moves, one of them is selected at random.
Let $dt(u, S_{cu}, S_{tu}; v, S_{cv}, S_{tv})$ ($S_{cu} \neq S_{tu}$, $S_{cv} \neq S_{tv}$) be a double-transfer move, then the move gain Δ_{dt} of this double transfer move can be calculated by a combination of the move gains of its two underlying single-transfer moves ($\Delta_{u \rightarrow S_{tu}}$ and $\Delta_{v \rightarrow S_{tv}}$) as follows:

$$\Delta_{dt(u,v)} = \Delta_{u \rightarrow S_{tu}} + \Delta_{v \rightarrow S_{tv}} + \psi \omega_{uv} \tag{2.3}$$

where ω_{uv} is the weight of edge $e(u, v) \in E$ and ψ is a coefficient which is determined as follows:

$$\psi = \begin{cases} -2, & \text{if } S_{cu} = S_{cv}, S_{tu} = S_{tv} \\ 2, & \text{if } S_{tu} = S_{cv}, S_{cu} = S_{tv} \\ -1, & \text{if } S_{cu} = S_{cv}, S_{tu} \neq S_{tv} \\ 1, & \text{if } S_{cu} = S_{tv}, S_{tu} \neq S_{cv} \\ -1, & \text{if } S_{cu} \neq S_{cv}, S_{tu} = S_{tv} \\ 1, & \text{if } S_{cu} \neq S_{tv}, S_{tu} = S_{cv} \\ 0, & \text{if } S_{cu} \neq S_{cv}, S_{tu} \neq S_{cv}, S_{cu} \neq S_{tv}, S_{tu} \neq S_{tv} \end{cases} \tag{2.4}$$

The operator O_2 is used when O_1 exhausts its improving moves and provides a first means to help the descent-based improvement phase to escape the current local optimum and discover solutions of increasing quality. Given an incumbent solution, there are a total number of $(k-1)^2 n(n-1)$ candidate double-transfer moves denoted as set DT. Seeking directly the best move with the maximum Δ_{dt} among all these possible moves would just be too computationally expensive. In order to mitigate this problem, we devise a strategy to accelerate the move evaluation process.

From Formula (2.3), one observes that among all the vertices in V, only the vertices verifying the condition $\omega_{uv} \neq 0$ and $\Delta_{dt(u,v)} > 0$ are of interest for the double-transfer moves. Note that without the condition $\omega_{uv} \neq 0$, performing a double-transfer move would actually equal to two consecutive single-transfer moves, which on the one hand makes the operator O_2 meaningless and on the other hand fails to get an increased objective gain. Thus, by examining only the endpoint vertices of edges in E, we shrink the move combinations by building a reduced subset: $DT^R = \{dt(u, v) : dt(u, v) \in DT, \omega_{uv} \neq 0, \Delta_{dt(u,v)} > 0\}$. Based on DT^R, the complexity of examining all possible double-transfer

moves drops to $O(|E|)$, which is not related to k. In practice, one can examine $\phi|E|$ endpoint vertices in case $|E|$ is too large. We empirically set $\phi = 0.1/d$, where d is the highest degree of the graph.

To summarize, the O_2 search operator selects two st moves $u \rightarrow S_{tu}$ and $v \rightarrow S_{tv}$ from the reduced set DT^R, such that the combined move gain $\Delta_{dt(u,v)}$ according to Formula (2.3) is maximum.

- **The O_3 search operator**, like O_1, selects a best single-transfer move (i.e., with the largest move gain) while considering a tabu list H [Glover and Laguna, 1999]. The tabu list is a memory which is used to keep track of the performed st moves to avoid revisiting previously encountered solutions. As such, each time a best st move is performed to displace a vertex v from its original subset to a target subset, v becomes tabu and is forbidden to move back to its original subset for the next λ iterations (called tabu tenure). In our case, the tabu tenure is dynamically determined as follows.

$$\lambda = rand(3, n/10) \tag{2.5}$$

where $rand(3, n/10)$ denotes a random integer between 3 and $n/10$.

Based on the tabu list, O_3 considers all possible single-transfer moves except those forbidden by the tabu list H and selects the best st move with the largest move gain Δ_{st}. Note that a forbidden move is always selected if the move leads to a solution better than the best solution found so far. This is called aspiration in tabu search terminology [Glover and Laguna, 1999].

Although both O_3 and O_1 use the single-transfer move, they are two different search operators and play different roles within the MOH algorithm. On the one hand, as a pure descent operator, O_1 is a faster operator compared to O_3 and is designed to be an intensification operator. Since O_1 alone has no any diversification capacity and always ends with the local optimum encountered, it is jointly used with O_2 to visit different local optima. On the other hand, due to the use of the tabu list, O_3 can accept moves with a negative move gain (leading to a worsening solution). As such, unlike O_1, O_3 has some diversification capacity, and when jointly used with O_4, helps the search to examine nearby regions around the input local optimum to find better solutions (see Alg. 1, lines $28-38$ and Section 2.2.7).

- **The O_4 search operator**, like O_2, is based on the double-transfer operation. However, O_4 strongly constraints the considered candidate dt moves with respect to two target subsets which are randomly selected. Specifically, O_4 operates as follows. Select two target subsets S_p and S_q at random, and then select two single-transfer moves $u \rightarrow S_p$ and $v \rightarrow S_q$ such that the combined move gain $\Delta_{dt(u,v)}$ according to Formula (2.3) is maximum.
  Operator O_4 is jointly used with operator O_3 to ensure the diversified improvement search phase.
- **The O_5 search operator** is based on a randomized single-transfer move operation. O_5 first selects a random vertex $v \in V$ and a random target subset S_p, where $v \notin S_p$ and then moves v from its current subset to S_p. This operator is used to change randomly the incumbent solution for the purpose of (strong) diversification when the search is considered to be trapped in a deep local optimum (see Section 2.2.8).

Among the five search operators, four of them ($O_1 - O_4$) need to find a single-transfer move with the maximum move gain. To ensure a high computational efficiency of these operators, we develop below a streamlining technique for fast move gain evaluation and move gain updates.

2.2.5 Bucket sorting for fast move gain evaluation and updating

The algorithm needs to rapidly evaluate a number of candidate moves at each iteration. Since all the search operators basically rely on the single-transfer move operation, we developed a fast incremental

evaluation technique based on a bucket data structure to keep and update the move gains after each move application [Cormen *et al.*, 2001]. Our streamlining technique can be described as follows: let $v \rightarrow S_x$ be the move of transferring vertex v from its current subset S_{cv} to any other subset S_x, $x \in \{1, \ldots, k\}, x \neq cv$. Then initially, each move gain is determined as follows:

$$\Delta_{v \rightarrow S_x} = \sum_{i \in S_{cv}, i \neq v} \omega_{vi} - \sum_{j \in S_x} \omega_{vj}, \; x \in \{1, \ldots, k\}, x \neq cv \tag{2.6}$$

where ω_{vi} and ω_{vj} are respectively the weights of edges $e(v, i)$ and $e(v, j)$.

Suppose the move $v \rightarrow S_{tv}$, i.e., displacing v from S_{cv} to S_{tv}, is performed, the move gains can be updated by performing the following calculations:

1. for each $S_x \neq S_{cv}$, $S_x \neq S_{tv}$,
   $\Delta_{v \rightarrow S_x} = \Delta_{v \rightarrow S_x} - \Delta_{v \rightarrow S_{tv}}$

2. $\Delta_{v \rightarrow S_{cv}} = -\Delta_{v \rightarrow S_{tv}}$

3. $\Delta_{v \rightarrow S_{tv}} = 0$

4. for each $u \in V - \{v\}$, moving $u \in S_{cu}$ to each other subset $S_y \in S - \{S_{cu}\}$,

$$\Delta_{u \rightarrow S_y} = \begin{cases} \Delta_{u \rightarrow S_y} - 2\omega_{uv}, & \text{if } S_{cu} = S_{cv}, S_y = S_{tv} \\ \Delta_{u \rightarrow S_y} + 2\omega_{uv}, & \text{if } S_{cu} = S_{tv}, S_y = S_{cv} \\ \Delta_{u \rightarrow S_y} - \omega_{uv}, & \text{if } S_{cu} = S_{cv}, S_y \neq S_{tv} \\ \Delta_{u \rightarrow S_y} + \omega_{uv}, & \text{if } S_{cu} = S_{tv}, S_y \neq S_{cv} \\ \Delta_{u \rightarrow S_y} - \omega_{uv}, & \text{if } S_{cu} \neq S_{cv}, S_y = S_{tv} \\ \Delta_{u \rightarrow S_y} + \omega_{uv}, & \text{if } S_{cu} \neq S_{tv}, S_y = S_{cv} \\ \Delta_{u \rightarrow S_y}, & \text{if } S_{cu} \neq S_{cv}, S_{cu} \neq S_{tv}, S_y \neq S_{cv}, S_y \neq S_{tv} \end{cases} \tag{2.7}$$

For low-density graphs, $\omega_{uv} = 0$ stands for most cases. Hence, we only update the move gains of vertices affected by this move (i.e., the displaced vertex and its adjacent vertices), which reduces the computation time significantly.

The move gains can be stored in an vector, with which the time for finding the best move grows linearly with the number of vertices and partitions ($O(kn)$). For large problem instances, the required time to search the best move can still be quite high, which is particular true when k is large. To further reduce the computing time, we adapted the bucket sorting technique of Fiduccia and Mattheyes [Fiduccia and Mattheyses, 1982] initially proposed for the two-way network partitioning problem to the max-k-cut problem. The idea is to keep the vertices ordered by the move gains in decreasing order in k arrays of buckets, one for each subset $S_i \in \{S_1, S_2, \ldots, S_k\}$. In each bucket array i, the j^{th} entry stores in a doubly linked list the vertices with the move gain $\Delta_{v \rightarrow S_i}$ currently equaling j. To ensure a direct access to each vertex in the doubly linked lists, we maintain another array for all vertices, where each element points to its corresponding vertex in the doubly linked lists.

Fig. 2.1 shows an example of the bucket structure for $k = 3$ and $n = 8$. The 8 vertices of the graph (Fig. 2.1, left) are divided to 3 subsets S_1, S_2 and S_3. The associated bucket structure (Fig. 2.1, right) shows that the move gains of moving vertices e, g, h to subset S_1 equal -1, then they are stored in the entry of B_1 with index of -1 and are managed as a doubly linked list. The array AI shown at the bottom of Fig. 2.1 manages position indexes of all vertices.

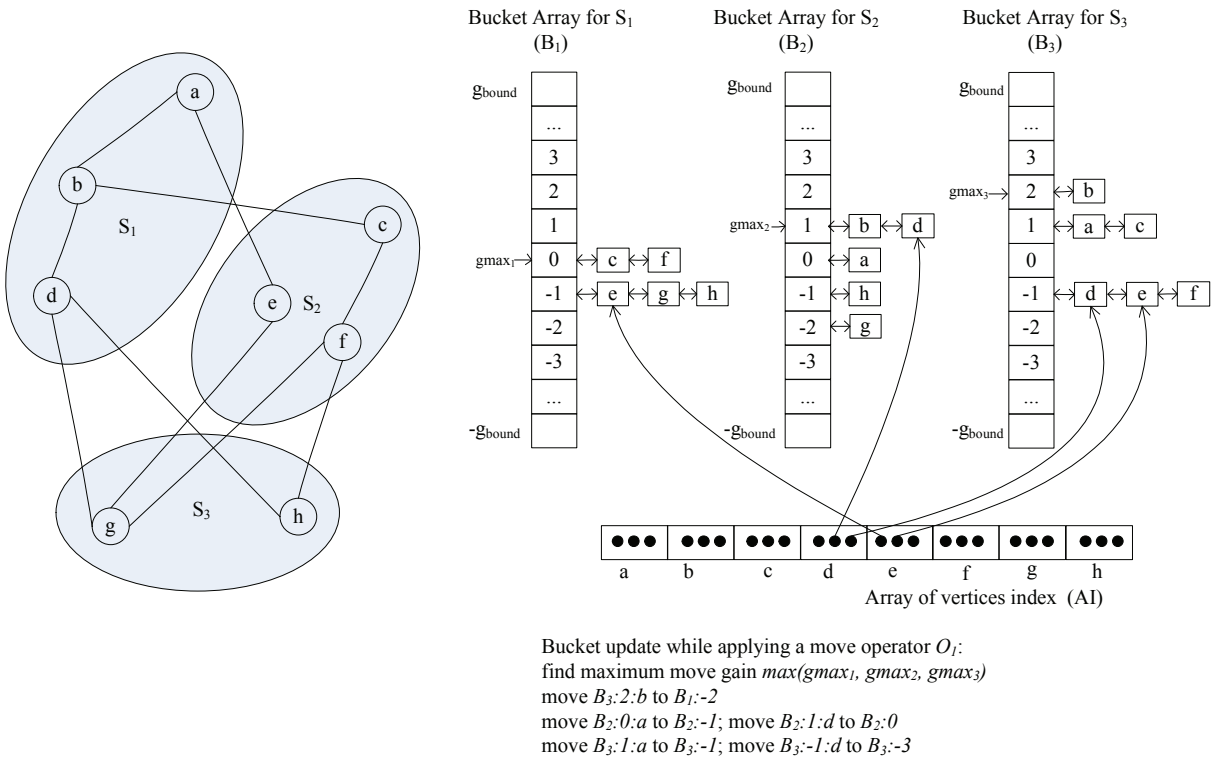


Figure 2.1: An example of bucket structure for max-3-cut

For each array of buckets, finding the best vertex with maximum move gain is equivalent to finding the first non-empty bucket from top of the array and then selecting a vertex in its doubly linked list. If there are more than one vertices in the doubly linked list, a random vertex in this list is selected. To further reduce the searching time, the algorithm memorizes the position of the first non-empty bucket (e.g., $gmax_1, gmax_2, gmax_3$ in Fig. 2.1). After each move, the bucket structure is updated by recomputing the move gains (see Formula (2.7)) of the affected vertices which include the moved vertex and its adjacent vertices, and shifting them to appropriate buckets. For instance, the steps of performing an O_1 move based on Fig. 2.1 are shown as follows: First, obtain the index of maximum move gain in the bucket arrays by calculating $max(gmax_1, gmax_2, gmax_3)$, which equals $gmax_3$ in this case. Second, select randomly a vertex indexed by $gmax_3$, vertex b in this case. At last, update the positions of the affected vertices a, b, d.

The complexity of each move consists in 1) searching for the vertex with maximum move gain in $O(l)$ (l being the current length of the doubly link list with the maximum gain, typically much smaller than n), 2) recomputing the move gains for the affected vertices in $O(kd_{max})$ (d_{max} being the maximum degree of the graph), and 3) updating the bucket structure in $O(kd_{max})$.

2.2.6 Descent-based improvement phase for intensified search

The descent-based local search is used to obtain a local optimum from a given starting solution. As described in Algorithm 1 (lines 10 - 19), we alternatively uses two search operators O_1 and O_2 defined in Section 2.2.4 to improve a solution until reaching a local optimum. Starting from the given initial solution, the procedure first applies O_1 to improve the incumbent solution. According to the definition of O_1 in Section 2.2.4, at each step, the procedure examines all possible single-transfer moves and selects a move $v \rightarrow S_q$ with the largest move gain $\Delta_{v \rightarrow S_q}$ subject to $\Delta_{v \rightarrow S_q} > 0$, and then performs that move. After the move, the algorithm updates the bucket structure of move gains according to the technique described in Section 2.2.5.

When the incumbent solution can not be improved by O_1 (i.e., $\forall v \in V, \forall S_q, \Delta_{v \to S_q} \leq 0$), the procedure turns to O_2 which makes one *best* double-transfer move. If an improved solution is discovered with respect to the local optimum reached by O_1, we are in a new promising area. We switch back to operator O_1 to resume an intensified search to attain a new local optimum. The descent-based improvement phase stops when no better solution can be found with O_1 and O_2. The last solution is a local optimum I_{lo} with respect to the single-transfer and double-transfer moves and serves as the input solution of the second search phase which is explained in the next section.

2.2.7 Diversified improvement phase for discovering promising region

The descent-based local phase described in Section 2.2.6 alone can not go beyond the best local optimum I_{lo} it encounters. The diversified improvement search phase is used 1) to jump out of this local optimum and 2) to intensify the search around this local optimum with the hope of discovering other improved solutions better than the input local optimum I_{lo}. The diversified improvement search procedure alternatively uses two search operators O_3 and O_4 defined in Section 2.2.4 to perform moves until a prescribed condition is met (see below and Alg. 1, line 38). The application of O_3 or O_4 is determined probabilistically: with probability ρ, O_3 is applied; with $1 - \rho$, O_4 is applied.

When O_3 is selected, the algorithm searches for a best single transfer move $v \to S_q$ with maximum move gain $\Delta_{v \to S_q}$ which is not forbidden by the tabu list or verifies the aspiration criterion. Each performed move is then recorded in the tabu list H and is classified tabu for the next λ (calculated by Formula (2.5)) iterations. The bucket structure is updated to actualize the impacted move gains accordingly. Note that the algorithm only keeps and updates the tabu list during the diversified improvement search phase. Once this second search phase terminates, the tabu list is cleared up.

Similarly, when O_4 is selected, two subsets are selected at random and a best double-transfer dt move with maximum move gain Δ_{dt} is determined from the bucket structure (break ties at random). After the move, the bucket structure is updated to actualize the impacted move gains.

The diversified improvement search procedure terminates once a solution better than the input local optimum I_{lo} is found, or a maximum number ω of diversified moves (O_3 or O_4) is reached. Then the algorithm returns to the descent-based search procedure and use the current solution I as a new starting point for the descent-based search. If the best solution founded so far (f_{best}) can not be improved over a maximum allowed number ξ of consecutive rounds of the descent-based improvement and diversified improvement phases, the search is probably trapped in a deep local optima. Consequently, the algorithm switches to the perturbation phase (Section 2.2.8) to displace the search to a distant region.

2.2.8 Perturbation phase for strong diversification

The diversified improvement phase makes it possible for the search to escape some local optima. However, the algorithm may still get deeply stuck in a non-promising regional search area. This is the case when the best-found solution f_{best} can not be improved after ξ consecutive rounds of descent and diversified improvement phases. Thus the random perturbation is applied to strongly change the incumbent solution.

The basic idea of the perturbation consists in applying the O_5 operator γ times. In other words, this perturbation phase moves γ randomly selected vertices from their original subset to a new and randomly selected subset. Here, γ is used to control the perturbation strength; a large (resp. small) γ value changes strongly (resp. weakly) the incumbent solution. In our case, we adopt $\gamma = 0.1|V|$, i.e., as a percent of

the number of vertices. After the perturbation phase, the search returns to the descent-based improvement phase with the perturbed solution as its new starting solution.

2.3 Experimental results and comparisons

2.3.1 Benchmark instances

To evaluate the performance of the proposed MOH approach, we carried out computational experiments on two sets of well-known benchmarks with a total of 91 large instances of the literature [1]. The first set (G-set) is composed of 71 graphs with 800 to 20000 vertices and an edge density from 0.02% to 6%. These instances were previously generated by a machine-independent graph generator including toroidal, planar and random weighted graphs. These instances are available from: http://www.stanford.edu/yyye/yyye/Gset. The second set comes form [Burer *et al.*, 2002], arising from 30 cubic lattices with randomly generated interaction magnitudes. Since the 10 small instances (with less than 1000 vertices) of the second set are very easy for our algorithm, only the results of the 20 larger instances with 1000 to 2744 vertices are reported. These well-known benchmarks were frequently used to evaluate the performance of max-bisection, max-cut and max-k-cut algorithms [Benlic and Hao, 2013a; Festa *et al.*, 2002; Shylo *et al.*, 2012; Shylo *et al.*, 2015; Wang *et al.*, 2013; Wu and Hao, 2012; Wu and Hao, 2013; Wu *et al.*, 2015; Zhu *et al.*, 2013].

2.3.2 Experimental protocol

The proposed MOH algorithm was programmed in C++ and compiled with GNU g++ (optimization flag "-O2"). Our computer is equipped with a Xeon E5440/2.83GHz CPU with 2GB RAM. When testing the DIMACS machine benchmark [2], our machine requires 0.43, 2.62 and 9.85 CPU time in seconds respectively for graphs r300.5, r400.5, and r500.5 compiled with g++ -O2.

2.3.3 Parameters

The MOH algorithm requires five parameters: tabu tenure λ, maximum number ω of diversified moves, maximum number ξ of consecutive non-improving rounds of the descent and diversified improvement phases before the perturbation phase, probability ρ for applying the operator O_3, and perturbation strength γ. For the tabu tenure λ, we adopted the recommended setting of the Breakout Local Search [Benlic and Hao, 2013a], which performs quite well for the benchmark graphs. For each of the other parameters, we first identified a collection of varying values and then determined the best setting by testing the candidate values of the parameter while fixing the other parameters to their default values. This parameter study was based on a selection of 10 representative and challenging G-set instances (G22, G23, G25, G29, G33, G35, G36, G37, G38 and G40). For each parameter setting, 10 independent runs of the algorithm were conducted for each instance and the average objective values over the 10 runs were recorded. If a large parameter value presents a better result, we gradually increase its value; otherwise, we gradually decrease its value. By repeating the above procedure, we determined the following parameter settings:

1. Our best results are available at: http://www.info.univ-angers.fr/pub/hao/maxkcut/MOHResults.zip.
2. dfmax:ftp://dimacs.rutgers.edu/pub/dsj/clique/

$\lambda = rand(3, |V|/10), \omega = 500, \xi = 1000, \rho = 0.5$, and $\gamma = 0.1|V|$, which were used in our experiments to report computational results.

Considering the stochastic nature of our MOH algorithm, each instance was independently solved 20 times. For the purpose of fair comparisons reported in Sections 2.3.4 and 2.3.5, we followed most reference algorithms and used a timeout limit as the stopping criterion of the MOH algorithm. The timeout limit was set to be 30 minutes for graphs with $|V| < 5000$, 120 minutes for graphs with $10000 \geq |V| \geq 5000$, 240 minutes for graphs with $|V| \geq 10000$.

To fully assess the performance of the MOH algorithm, we performed two comparisons with the state-of-the-art algorithms. First, we focused on the max-k-cut problem ($k = 2, 3, 4, 5$), where we thoroughly compared our algorithm with the recent discrete dynamic convexized algorithm [Zhu *et al.*, 2013] which provides the most competitive results for the general max-k-cut problem in the literature. Secondly, for the special max-cut case ($k = 2$), we compared our algorithm with seven most recent max-cut algorithms [Benlic and Hao, 2013a; Kochenberger *et al.*, 2013; Shylo *et al.*, 2012; Wang *et al.*, 2013; Wu and Hao, 2012; Wu and Hao, 2013]. It should be noted that those state-of-the-art max-cut algorithms were specifically designed for the particular max-cut problem while our algorithm was developed for the general max-k-cut problem. Naturally, the dedicated algorithms are advantaged since they can better explore the particular features of the max-cut problem.

2.3.4 Comparison with state-of-the-art max-k-cut algorithms

In this section, we present the results attained by the MOH algorithm for the max-k-cut problem. As mentioned above, we compare the proposed algorithm with the discrete dynamic convexized algorithm (DC) [Zhu *et al.*, 2013], which was published very recently. DC was tested on a computer with a 2.11 GHz AMD processor and 1 GB of RAM. According to the Standard Performance Evaluation Cooperation (SPEC) (www.spec.org), this computer is 1.4 times slower than the computer we used for our experiments. Note that DC is the only heuristic algorithm available in the literature, which published computational results for the general max-k-cut problem.

Tables 2.1 to 2.4 respectively show the computational results of the MOH algorithm ($k = 2, 3, 4, 5$) on the 2 sets of benchmarks in comparison with those of the DC algorithm. The first two columns of the tables indicate the name and the number of vertices of the graphs. Columns 3 to 6 present the results attained by our algorithm, where f_{best} and f_{avg} show the best objective value and the average objective value over 20 runs, std gives the standard deviation and $time(s)$ indicates the average CPU time in seconds required by our algorithm to reach the best objective value f_{best}. Columns 7 to 10 present the statistics of the DC algorithm, including the best objective value f_{best}, average objective value f_{avg}, the time required to terminate the run $tt(s)$ and the time $bt(s)$ to reach the f_{best} value. Considering the difference between our computer and the computer used by DC, we normalize the time of DC by dividing them by 1.4 according to the SPEC mentioned above. The entries marked as "-" in the tables indicate that the corresponding results are not available. The entries in bold indicate that those results are better than the results provided by the reference DC algorithm. The last column (gap) indicates the gap of the best objective value for each instance between our algorithm and DC. A positive gap implies an improved result.

From Table 2.1 on max-2-cut, one observes that our algorithm achieves a better f_{best} (best objective value) for 50 out of 74 instances reported by DC, while a better f_{avg} (average objective value) for 71 out of 74 instances. Our algorithm matches the results on other instances and there is no result worse than that obtained by DC. The average standard deviation for all 91 instances is only 2.82, which shows our algorithm is stable and robust.

From Table 2.2, 2.3, and 2.4, which respectively show the comparative results on max-3-cut, max-4-cut and max-5-cut. One observes that our algorithm achieves much higher solution quality on more than 90 percent of 44 instances reported by DC while getting 0 worse result. Moreover, even our *average* results (f_{avg}) are better than the *best* results reported by DC.

Note that the DC algorithm used a stopping condition of 500 generations (instead of a cutoff time limit) to report its computational results. Among the two timing statistics ($tt(s)$ and $bt(s)$), $bt(s)$ roughly corresponds to column $time$ of the MOH algorithm. Still given that the two algorithms attain solutions of quite different quality, it is meaningless to directly compare the corresponding time values listed in Tables 2.1–2.4. To fairly compare the computational efficiency of MOH and DC, we reran the MOH algorithm with the best objective value of the DC algorithm as our stopping condition and reported our timing statistics in Table 2.5. One observes that our algorithm needs at most 16 seconds (less than 1 second for most cases) to attain the best objective value reported by the DC algorithm, while the DC algorithm requires at least 44 seconds and up to more than 2000 seconds for several instances. More generally, as shown in Table 2.1–2.4, except the last 17 instances of the very competitive max-2-cut problem for which the results of DC are not available, the MOH algorithm requires rarely more than 1000 seconds to attain solutions of much better quality.

We conclude that the proposed algorithm for the general max-k-cut problem dominates the state-of-the-art reference DC algorithm both in terms of solution quality and computing time.

2.3.5 Comparison with state-of-the-art max-cut algorithms

Our algorithm was designed for the general max-k-cut problem for $k \geq 2$. The assessment of the last section focused on the general case. In this section, we further evaluate the performance of the proposed algorithm for the special max-cut problem ($k = 2$).

Recall that max-cut has been largely studied in the literature for a long time and there are many powerful heuristics which are specifically designed for the problem. These state-of-the-art max-cut algorithms constitute thus relevant references for our comparative study. In particular, we adopt the following 7 best performing sequential algorithms published since 2012.

1. Global equilibrium search (GES) (2012) [Shylo *et al.*, 2012] - an algorithm sharing ideas similar to simulated annealing and utilizing accumulated information of search space to generate new solutions for the subsequent stages. The reported results of GES were obtained on a PC with a 2.83GHz Intel Core QUAD Q9550 CPU and 8.0GB RAM.

2. Breakout local search (BLS) (2013) [Benlic and Hao, 2013a] - a heuristic algorithm integrating a local search and adaptive perturbation strategies. The reported results of BLS were obtained on a PC with 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.

3. Two memetic algorithms respective for the max-cut problem (MACUT) (2012) [Wu and Hao, 2012] and the max-bisection problem (MAMBP) (2013) [Wu and Hao, 2013] - integrating a grouping crossover operator and a tabu search procedure. The results reported in the two papers were obtained on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.

4. GRASP-Tabu search algorithm (2013) [Wang *et al.*, 2013] - a method converting the max-cut problem to the UBQP problem and solving it by integrating GRASP and tabu search. The reported results were obtained on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.

5. Tabu search (TS-UBQP) (2013) [Kochenberger *et al.*, 2013] - a tabu search algorithm designed for UBQP. The evaluation of TS-UBQP were performed on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2GB RAM.

6. Tabu search based hybrid evolutionary algorithm (TSHEA) (2016) [Wu *et al.*, 2015] - a very recent hybrid algorithm integrating a distance-and-quality guided solution combination operator and a tabu search procedure based on neighborhood combination of one-flip and constrained exchange moves. The results were obtained on a PC with 2.83GHz Intel Xeon E5440 CPU and 8GB RAM.

One notices that except GES, the other five reference algorithms were run on the same computing platform. Nevertheless, it is still difficult to make a fully fair comparison of the computing time, due to the differences on programming language, compiling options, and termination conditions, etc. Our comparison thus focuses on the best solution achieved by each algorithm. Recall that for our algorithm, the timeout limit was set to be 30 minutes for graphs with $|V| < 5000$, 120 minutes for graphs with $5000 \leq |V| < 10000$, 240 minutes for graphs with $|V| \geq 10000$. Our algorithm employed thus the same timeout limits as [Wu and Hao, 2012] on the graphs $|V| < 10000$, but for the graphs $|V| \geq 10000$, we used 240 minutes to compare with BLS [Benlic and Hao, 2013a].

Table 2.6 gives the comparative results on the 91 instances of the two benchmarks. Columns 1 and 2 respectively indicate the instance name and the number of vertices of the graphs. Columns 3 shows the current best known objective value f_{pre} reported by any existing max-cut algorithm in the literature including the latest *parallel* GES algorithm [Shylo *et al.*, 2015]. Columns 4 to 10 give the best objective value obtained by the reference algorithms: GES [Shylo *et al.*, 2012], BLS [Benlic and Hao, 2013a], MACUT [Wu and Hao, 2012], TS-UBQP [Kochenberger *et al.*, 2013], GRASP-TS/PM [Wang *et al.*, 2013], MAMBP [Wu and Hao, 2013] and TSHEA [Wu *et al.*, 2015]. Note that MAMBP is designed for the max-bisection problem (i.e., balanced max-cut), however it achieves some previous best known max-cut results. The last column 'MOH' recalls the best results of our algorithm from Table 2.1. The rows denoted by 'Better', 'Equal' and 'Worse' respectively indicate the number of instances for which our algorithm obtains a result of better, equal and worse quality relative to each reference algorithm. The entries are reported in the form of x/y/z, where z denotes the total number of the instances tested by our algorithm, y is the number of the instances tested by a reference algorithm and x indicates the number of instances where our algorithm achieved 'Better', 'Equal' or 'Worse' results. The results in bold mean that our algorithm has improved the best known results. The entries marked as "-" in the table indicate that the results are not available.

From Table 2.6, one observes that the MOH algorithm is able to improve the current best known results in the literature for 4 instances, and match the best known results for 74 instances. For 13 cases (in italic), even if our results are worse than the current best known results achieved by the latest *parallel* GES algorithm [Shylo *et al.*, 2015], they are still better than the results of other existing algorithms, except for 4 instances if we refer to the most recent TSHEA algorithm [Wu *et al.*, 2015]. Note that the results of the parallel GES algorithm were achieved on a more powerful computing platform (Intel CoreTM i7-3770 CPU @3.40GHz and 8GB RAM) and with longer time limits (4 parallel processes at the same time and 1 hour for each process).

Such a performance is remarkable given that we are comparing our more general algorithm designed for max-k-cut with the best performing specific max-cut algorithms. The experimental evaluations presented in this section and last section demonstrate that our algorithm not only performs well on the general max-k-cut problem, but also remains highly competitive for the special case of the popular max-cut problem.

Table 2.1: Comparative results for max-2-cut between the proposed MOH algorithm and DC

Instance	$\|V\|$	MOH				DC				gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	f_{avg}	$tt(s)$	$bt(s)$	
G1	800	11624	**11624.00**	0.00	1.46	11624	11617.20	131.73	90.98	0
G2	800	11620	**11620.00**	0.00	4.61	11620	11610.00	131.38	79.96	0
G3	800	11622	**11622.00**	0.00	1.25	11622	11612.20	130.78	64.22	0
G4	800	11646	**11646.00**	0.00	5.23	11646	11633.90	133.78	48.17	0
G5	800	11631	**11631.00**	0.00	0.99	11631	11623.20	131.71	36.46	0
G6	800	2178	**2178.00**	0.00	3.03	2178	2175.90	132.08	83.88	0
G7	800	2006	**2006.00**	0.00	2.98	2006	1997.70	137.61	59.61	0
G8	800	2005	**2005.00**	0.00	5.72	2005	2000.00	139.17	31.28	0
G9	800	**2054**	**2054.00**	0.00	3.21	2049	2043.50	134.94	40.03	5
G10	800	**2000**	**2000.00**	0.00	68.09	1999	1998.40	133.26	18.34	1
G11	800	564	**564.00**	0.00	0.22	564	563.80	58.84	7.78	0
G12	800	556	**556.00**	0.00	3.52	556	555.40	58.73	17.09	0
G13	800	582	**582.00**	0.00	0.85	582	580.00	60.95	43.21	0
G14	800	**3064**	**3064.00**	0.00	251.27	3057	3054.30	82.68	56.77	7
G15	800	**3050**	**3050.00**	0.00	52.19	3044	3038.00	82.43	27.69	6
G16	800	3052	**3052.00**	0.00	93.68	3052	3039.60	81.12	15.19	0
G17	800	**3047**	**3047.00**	0.00	129.53	3043	3037.80	81.61	15.05	4
G18	800	**992**	**992.00**	0.00	112.65	989	984.00	89.05	3.73	3
G19	800	906	**906.00**	0.00	266.92	906	899.90	84.43	24.96	0
G20	800	941	**941.00**	0.00	43.71	941	938.20	86.28	15.17	0
G21	800	931	**931.00**	0.00	155.34	931	926.00	86.24	12.44	0
G22	2000	**13359**	**13357.00**	1.91	352.37	13339	13315.90	683.67	108.56	20
G23	2000	**13344**	**13344.00**	0.00	433.79	13323	13298.90	705.23	433.48	21
G24	2000	**13337**	**13336.70**	0.46	777.86	13314	13286.00	692.07	237.38	23
G25	2000	**13340**	**13335.50**	2.40	442.45	13324	13293.70	694.73	667.19	16
G26	2000	**13328**	**13325.50**	2.31	535.14	13313	13282.20	689.61	251.36	15
G27	2000	**3341**	**3341.00**	0.00	42.25	3326	3285.40	677.86	464.32	15
G28	2000	**3298**	**3298.00**	0.00	707.18	3292	3272.00	680.47	594.81	6
G29	2000	**3405**	**3397.85**	5.31	555.23	3390	3357.20	693.45	375.90	15
G30	2000	**3413**	**3412.15**	0.36	330.46	3398	3369.50	676.54	587.80	15
G31	2000	**3310**	**3307.85**	0.91	592.56	3295	3273.90	696.42	212.48	15
G32	2000	**1410**	**1410.00**	0.00	65.75	1408	1402.70	514.87	115.58	2
G33	2000	**1382**	**1381.60**	0.80	504.10	1378	1373.70	508.85	271.75	4
G34	2000	**1384**	**1384.00**	0.00	84.23	1378	1376.70	531.51	97.37	6
G35	2000	**7686**	**7681.65**	1.59	796.70	7647	7632.20	614.51	391.36	39
G36	2000	**7680**	**7673.60**	1.62	664.48	7625	7618.50	613.15	594.82	55
G37	2000	**7691**	**7685.75**	2.26	652.78	7640	7627.70	623.72	609.25	51
G38	2000	**7688**	**7683.60**	2.27	779.69	7641	7614.40	632.95	587.98	47
G39	2000	**2408**	**2405.35**	1.85	787.69	2375	2352.50	659.34	281.45	33
G40	2000	**2400**	**2397.35**	2.43	472.50	2384	2371.70	656.75	425.90	16
G41	2000	**2405**	**2405.00**	0.00	377.35	2377	2357.40	666.79	244.21	28
G42	2000	**2481**	**2476.35**	2.01	777.42	2469	2441.30	657.13	374.11	12
G43	1000	**6660**	**6660.00**	0.00	1.15	6657	6648.90	156.66	29.04	3
G44	1000	6650	**6650.00**	0.00	5.28	6650	6643.70	155.84	24.82	0
G45	1000	**6654**	**6654.00**	0.00	6.87	6647	6640.70	155.28	95.98	7
G46	1000	**6649**	**6648.90**	0.30	67.27	6647	6637.90	157.02	61.02	2
G47	1000	6657	**6657.00**	0.00	43.25	6657	6648.50	157.81	144.33	0
G48	3000	6000	6000.00	0.00	0.02	6000	6000.00	420.15	0.26	0
G49	3000	6000	6000.00	0.00	0.03	6000	6000.00	440.26	0.36	0
G50	3000	5880	5879.70	0.71	532.13	5880	5880.00	552.51	0.59	0
G51	1000	**3848**	**3848.00**	0.00	189.20	3842	3831.50	137.56	122.03	6
G52	1000	**3851**	**3851.00**	0.00	209.69	3840	3830.50	132.69	119.09	11
G53	1000	**3850**	**3849.95**	0.22	299.28	3844	3835.00	136.25	62.86	6
G54	1000	**3852**	**3851.10**	0.30	190.38	3831	3824.40	136.04	60.29	21
G55	5000	10299	10283.40	7.13	1230.40	-	-	-	-	-
G56	5000	4016	4007.47	6.49	990.40	-	-	-	-	-
G57	5000	3494	3486.80	2.45	1528.34	-	-	-	-	-
G58	5000	19288	19275.40	4.58	1522.29	-	-	-	-	-
G59	5000	6087	6077.19	7.90	2498.80	-	-	-	-	-
G60	7000	14190	14173.00	6.98	2945.40	-	-	-	-	-
G61	7000	5798	5782.67	5.72	6603.34	-	-	-	-	-
G62	7000	4868	4851.73	7.10	5568.63	-	-	-	-	-
G63	7000	27033	27019.20	6.23	6492.11	-	-	-	-	-
G64	7000	8747	8700.87	19.28	4011.10	-	-	-	-	-
G65	8000	5560	5531.93	6.43	4709.53	-	-	-	-	-
G66	9000	6360	6323.53	6.34	6061.92	-	-	-	-	-
G67	10000	6942	6903.93	8.91	4214.28	-	-	-	-	-
G70	10000	9544	9527.80	9.32	8732.40	-	-	-	-	-
G72	10000	6998	6957.80	7.36	6586.64	-	-	-	-	-
G77	14000	9928	9920.00	3.08	9863.56	-	-	-	-	-
G81	20000	14036	14020.30	8.50	20422.00	-	-	-	-	-
3dl101000	1000	896	**896.00**	0.00	8.35	896	888.70	113.30	48.64	0
3dl102000	1000	900	**900.00**	0.00	9.50	900	898.50	111.50	2.56	0
3dl103000	1000	**892**	**892.00**	0.00	148.25	888	884.70	112.96	23.59	4
3dl104000	1000	898	**898.00**	0.00	4.20	898	895.00	112.19	30.17	0
3dl105000	1000	**886**	**886.00**	0.00	17.00	884	882.80	115.04	14.16	2
3dl106000	1000	888	**888.00**	0.00	5.55	888	883.70	114.72	32.87	0
3dl107000	1000	**900**	**899.60**	0.80	61.10	898	892.40	114.06	39.41	2

Table 2.1 – continued from previous page

Instance	$\|V\|$	MOH				DC				gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	f_{avg}	$tt(s)$	$bt(s)$	
3dl108000	1000	**882**	**882.00**	0.00	76.95	880	877.70	120.03	15.83	2
3dl109000	1000	**902**	**902.00**	0.00	21.55	902	894.40	113.64	9.72	0
3dl1010000	1000	**894**	**894.00**	0.00	12.15	894	893.40	110.87	21.37	0
3dl141000	2744	**2446**	**2445.00**	1.61	552.20	2434	2416.40	1039.73	694.21	12
3dl142000	2744	**2458**	**2457.70**	1.31	479.15	2444	2431.00	1016.16	496.31	14
3dl143000	2744	**2444**	**2439.60**	2.33	58.75	2426	2415.00	1012.31	121.79	18
3dl144000	2744	**2450**	**2448.10**	2.23	220.55	2440	2425.30	997.51	587.98	10
3dl145000	2744	**2446**	**2444.90**	2.23	372.35	2432	2422.40	999.31	277.75	14
3dl146000	2744	**2452**	**2449.60**	2.06	227.80	2438	2430.00	1035.41	930.23	14
3dl147000	2744	**2444**	**2442.70**	1.31	239.05	2428	2413.40	1022.70	556.16	16
3dl148000	2744	**2448**	**2446.40**	1.50	405.35	2432	2424.40	1030.67	954.38	16
3dl149000	2744	**2428**	**2424.70**	2.12	112.05	2418	2403.70	1020.11	832.95	10
3dl1410000	2744	**2458**	**2455.70**	2.63	286.35	2438	2429.30	1018.15	466.77	20
Better		50/74/91	71/74/91							
Equal		24/74/91	3/74/91							
Worse		0/74/91	0/74/91							

Table 2.2: Comparative results for max-3-cut between the proposed MOH algorithm and DC

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G1	800	**15165**	15164.90	0.36	557.25	15127	508.34	339.41	38
G2	800	**15172**	15171.20	0.99	333.25	15159	497.49	228.37	13
G3	800	**15173**	15173.00	0.00	269.60	15149	506.45	205.06	24
G4	800	15184	15181.40	2.46	300.55	-	-	-	-
G5	800	15193	15193.00	0.00	98.15	-	-	-	-
G6	800	2632	2631.95	0.22	307.30	-	-	-	-
G7	800	2409	2408.40	1.07	381.00	-	-	-	-
G8	800	2428	2427.55	0.67	456.50	-	-	-	-
G9	800	2478	2475.85	2.52	282.00	-	-	-	-
G10	800	2407	2406.40	0.86	569.30	-	-	-	-
G11	800	**669**	667.80	0.75	143.80	660	240.99	132.51	9
G12	800	**660**	658.95	0.50	100.70	655	212.56	59.09	5
G13	800	**686**	685.40	0.58	459.35	679	230.20	111.53	7
G14	800	**4012**	4009.45	1.88	88.20	3984	271.47	190.40	28
G15	800	**3984**	3982.40	0.58	80.30	3960	271.88	183.92	24
G16	800	**3991**	3986.30	1.87	1.30	3958	272.44	75.02	33
G17	800	3983	3981.00	1.05	7.80	-	-	-	-
G18	800	1207	1205.60	1.56	0.30	-	-	-	-
G19	800	1081	1078.05	2.38	0.20	-	-	-	-
G20	800	1122	1115.00	4.05	13.25	-	-	-	-
G21	800	1109	1106.75	2.30	55.75	-	-	-	-
G22	2000	**17167**	17157.80	7.62	28.45	17008	2121.42	986.19	159
G23	2000	**17168**	17156.70	6.40	45.05	17021	2190.36	1208.18	147
G24	2000	**17162**	17152.10	4.98	16.30	17037	2230.09	1385.32	125
G25	2000	17163	17155.20	3.44	64.75	-	-	-	-
G26	2000	17154	17146.30	4.61	44.80	-	-	-	-
G27	2000	4020	4013.80	3.33	53.15	-	-	-	-
G28	2000	3973	3966.45	5.10	38.85	-	-	-	-
G29	2000	4106	4097.30	5.40	68.15	-	-	-	-
G30	2000	4119	4109.90	5.34	150.40	-	-	-	-
G31	2000	4003	3999.20	6.69	124.70	-	-	-	-
G32	2000	**1653**	1651.85	0.73	160.05	1635	1274.91	905.73	18
G33	2000	**1625**	1622.30	0.95	62.55	1603	1215.13	664.57	22
G34	2000	**1607**	1604.00	1.00	88.85	1589	1303.88	827.79	18
G35	2000	**10046**	10039.90	2.59	66.15	9965	1793.30	1048.97	81
G36	2000	**10039**	10034.40	3.81	74.25	9945	1822.04	1196.02	94
G37	2000	**10052**	10047.80	1.96	3.35	9952	1845.20	1288.13	100
G38	2000	10040	10035.50	3.26	116.60	-	-	-	-
G39	2000	2903	2890.05	6.75	8.95	-	-	-	-
G40	2000	2870	2850.65	8.08	82.80	-	-	-	-
G41	2000	2887	2862.90	9.77	87.70	-	-	-	-
G42	2000	2980	2964.30	5.99	2.45	-	-	-	-
G43	1000	**8573**	8573.00	0.00	380.30	8510	512.48	112.20	63
G44	1000	**8571**	8569.60	2.35	616.80	8526	491.34	47.87	45
G45	1000	**8566**	8564.85	1.11	186.20	8515	504.19	44.00	51
G46	1000	8568	8564.60	2.01	215.30	-	-	-	-
G47	1000	8572	8568.70	2.72	239.35	-	-	-	-
G48	3000	**6000**	6000.00	0.00	0.40	5998	2591.27	293.30	2
G49	3000	6000	6000.00	0.00	0.90	6000	2653.42	1587.05	0
G50	3000	**6000**	6000.00	0.00	119.15	5998	2547.78	279.78	2
G51	1000	5037	5031.35	1.90	47.90	-	-	-	-
G52	1000	5040	5037.50	0.81	0.65	-	-	-	-
G53	1000	5039	5038.00	1.05	223.85	-	-	-	-
G54	1000	5036	5033.55	2.29	133.95	-	-	-	-
G55	5000	12429	12423.70	2.61	383.10	-	-	-	-

Table 2.2 – continued from previous page

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G56	5000	4752	4741.90	7.84	569.20	-	-	-	-
G57	5000	4083	4079.00	1.55	535.60	-	-	-	-
G58	5000	25195	25182.10	8.89	576.00	-	-	-	-
G59	5000	7262	7246.70	9.20	27.50	-	-	-	-
G60	7000	17076	17067.00	4.40	683.00	-	-	-	-
G61	7000	6853	6842.10	5.26	503.10	-	-	-	-
G62	7000	5685	5681.50	1.43	242.40	-	-	-	-
G63	7000	35322	35301.60	10.35	658.50	-	-	-	-
G64	7000	10443	10408.80	25.23	186.90	-	-	-	-
G65	8000	6490	6485.80	2.04	324.70	-	-	-	-
G66	9000	7416	7411.50	2.42	542.50	-	-	-	-
G67	10000	8086	8083.50	2.29	756.70	-	-	-	-
G70	10000	9999	9999.00	0.00	7.80	-	-	-	-
G72	10000	8192	8186.70	3.35	271.20	-	-	-	-
G77	14000	11578	11568.90	4.01	154.90	-	-	-	-
G81	20000	16321	16313.00	4.05	331.20	-	-	-	-
3dl101000	1000	**1067**	1066.10	0.54	150.40	1043	333.45	179.20	24
3dl102000	1000	**1072**	1071.95	0.22	669.50	1044	339.38	188.68	28
3dl103000	1000	**1065**	1063.60	0.66	142.85	1042	326.69	114.20	23
3dl104000	1000	**1071**	1070.30	0.46	160.20	1045	341.58	109.75	26
3dl105000	1000	**1064**	1061.90	0.77	4.40	1039	320.88	178.88	25
3dl106000	1000	**1063**	1061.80	0.60	120.00	1032	353.75	23.96	31
3dl107000	1000	**1075**	1074.40	0.58	414.05	1053	335.95	157.18	22
3dl108000	1000	**1071**	1069.95	0.38	78.55	1049	325.50	209.77	22
3dl109000	1000	**1079**	1078.20	0.81	208.85	1052	328.38	232.87	27
3dl1010000	1000	**1070**	1069.50	0.50	478.65	1044	346.13	184.91	26
3dl141000	2744	**2924**	2919.75	2.45	25.00	2845	2527.70	1496.07	79
3dl142000	2744	**2935**	2929.25	2.53	55.95	2856	2556.83	1408.24	79
3dl143000	2744	**2912**	2909.50	1.40	110.25	2829	2658.27	1659.44	83
3dl144000	2744	**2924**	2919.90	2.41	81.15	2861	2490.92	1759.67	63
3dl145000	2744	**2914**	2911.25	1.92	67.50	2839	2515.36	1764.88	75
3dl146000	2744	**2913**	2909.00	2.00	22.05	2834	2541.43	1529.38	79
3dl147000	2744	**2913**	2909.30	1.73	70.05	2834	2554.19	1748.39	79
3dl148000	2744	**2925**	2919.40	4.05	73.95	2845	2495.00	1440.25	80
3dl149000	2744	**2906**	2901.50	2.62	6.35	2823	2476.52	1699.97	83
3dl1410000	2744	**2933**	2927.65	2.22	29.90	2851	2519.16	1476.52	82
Better		43/44/91							
Equal		1/44/91							
Worse		0/44/91							

Table 2.3: Comparative results for max-4-cut between the proposed MOH algorithm and DC

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G1	800	**16803**	16801	0.86	26.45	16740	450.16	290.51	63
G2	800	**16809**	16808	1.12	268.55	16735	455.81	388.76	74
G3	800	**16806**	16804.7	0.78	138.25	16752	431.86	245.50	54
G4	800	16814	16811.2	1.49	146.65	-	-	-	-
G5	800	16816	16815.8	0.36	577.45	-	-	-	-
G6	800	2751	2748.45	1.07	89.95	-	-	-	-
G7	800	2515	2513.75	0.54	57.15	-	-	-	-
G8	800	2525	2523.35	0.65	78.6	-	-	-	-
G9	800	2585	2583.35	0.96	16.45	-	-	-	-
G10	800	2510	2507.6	1.24	79.85	-	-	-	-
G11	800	**677**	676	0.32	20.3	675	171.27	152.04	2
G12	800	**664**	662.25	0.54	41.25	660	179.99	117.52	4
G13	800	**690**	689.1	0.44	198.7	685	187.54	127.56	5
G14	800	**4440**	4435.35	1.93	55.95	4402	243.08	159.14	38
G15	800	**4406**	4403.4	0.8	89.55	4373	249.66	129.21	33
G16	800	**4415**	4414.05	1.02	392.45	4378	246.11	75.89	37
G17	800	4411	4406.45	2.27	0.2	-	-	-	-
G18	800	1261	1253.9	3.06	0.3	-	-	-	-
G19	800	1121	1115.35	3.69	1.2	-	-	-	-
G20	800	1168	1160.95	3.12	0.4	-	-	-	-
G21	800	1155	1148.25	3.74	54.7	-	-	-	-
G22	2000	**18776**	18765.7	5.67	107.25	18615	1988.31	1314.45	161
G23	2000	**18777**	18765.8	5.71	73.7	18612	1941.85	1775.80	165
G24	2000	**18769**	18763.6	3.75	26.4	18620	1822.82	407.66	149
G25	2000	18775	18767.6	4.36	75.65	-	-	-	-
G26	2000	18767	18761.2	4.49	96.55	-	-	-	-
G27	2000	4201	4188.5	4.6	45.35	-	-	-	-
G28	2000	4150	4138.85	5.91	24.95	-	-	-	-
G29	2000	4293	4281.65	5.68	87.4	-	-	-	-
G30	2000	4305	4296.4	4.12	33.5	-	-	-	-
G31	2000	4171	4164.4	6.46	107.8	-	-	-	-
G32	2000	**1669**	1667.85	1.01	120.9	1659	1140.66	736.15	10

Table 2.3 – continued from previous page

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G33	2000	**1638**	1634.65	1.15	0	1629	1052.38	870.96	9
G34	2000	**1616**	1611.7	1.65	0.05	1604	1105.02	1016.31	12
G35	2000	**11111**	11106.2	2.14	17.2	11007	1890.32	1764.52	104
G36	2000	**11108**	11101.4	2.9	17.25	10993	1738.64	1634.13	115
G37	2000	**11117**	11112.5	2.33	36.05	11023	1754.17	115.08	94
G38	2000	11108	11101.1	3.16	48.4	-	-	-	-
G39	2000	3006	2998.7	3.91	1.15	-	-	-	-
G40	2000	2976	2955.65	8.99	48.7	-	-	-	-
G41	2000	2983	2970.3	6.91	1.8	-	-	-	-
G42	2000	3092	3084.05	4.8	16.9	-	-	-	-
G43	1000	**9376**	9373.95	1.2	84.15	9306	422.97	62.38	70
G44	1000	**9379**	9373.55	2.52	67.9	9315	430.52	43.88	64
G45	1000	**9376**	9375.1	0.94	249.5	9312	463.45	319.58	64
G46	1000	9378	9375.35	1.96	139.75	-	-	-	-
G47	1000	9381	9377.05	2.04	60.5	-	-	-	-
G48	3000	6000	6000	0	0	6000	1673.79	0.48	0
G49	3000	6000	6000	0	0	6000	1675.56	0.49	0
G50	3000	6000	6000	0	0	6000	1678.91	0.50	0
G51	1000	5571	5567.65	1.93	14.6	-	-	-	-
G52	1000	5584	5581.15	1.74	20.9	-	-	-	-
G53	1000	5574	5571.85	1.19	6.85	-	-	-	-
G54	1000	5579	5576.25	1.58	0.7	-	-	-	-
G55	5000	12498	12498	0	0.9	-	-	-	-
G56	5000	4931	4917.1	6.49	424.6	-	-	-	-
G57	5000	4112	4110.5	1.12	298.1	-	-	-	-
G58	5000	27885	27870.9	8.68	435.4	-	-	-	-
G59	5000	7539	7515.1	15.09	969.3	-	-	-	-
G60	7000	17148	17148	0	2.3	-	-	-	-
G61	7000	7110	7104.6	5.08	1305.2	-	-	-	-
G62	7000	5743	5738.7	2.69	385.5	-	-	-	-
G63	7000	39083	39063.5	9.18	660.2	-	-	-	-
G64	7000	10814	10797.4	13.28	910.5	-	-	-	-
G65	8000	6534	6525.4	4.48	1.5	-	-	-	-
G66	9000	7474	7467.8	4.24	2.2	-	-	-	-
G67	10000	8155	8142.5	5.57	3	-	-	-	-
G70	10000	9999	9999	0	0.5	-	-	-	-
G72	10000	8264	8254.6	7.36	3.1	-	-	-	-
G77	14000	11674	11658.9	10.08	6.4	-	-	-	-
G81	20000	16470	16454.3	8.5	27.9	-	-	-	-
3dl101000	1000	**1103**	1100.6	0.86	64.5	1073	304.44	187.92	30
3dl102000	1000	**1102**	1100	0.95	1.5	1070	351.27	301.64	32
3dl103000	1000	**1108**	1106.4	0.86	22.8	1072	340.99	249.06	36
3dl104000	1000	**1103**	1101.65	0.65	87.7	1076	323.51	276.29	27
3dl105000	1000	**1098**	1096.3	0.78	58.6	1074	334.38	294.70	24
3dl106000	1000	**1097**	1095.15	0.91	94.05	1063	358.27	307.91	34
3dl107000	1000	**1114**	1112.2	1.08	108.3	1093	308.31	101.66	21
3dl108000	1000	**1105**	1103	0.77	28.9	1079	276.09	260.12	26
3dl109000	1000	**1115**	1113.45	0.8	108.35	1086	271.29	60.70	29
3dl1010000	1000	**1109**	1106.1	0.89	54.9	1088	277.18	257.21	21
3dl141000	2744	**3016**	3012.05	1.91	57.05	2893	1990.54	1511.84	123
3dl142000	2744	**3026**	3019.8	2.04	18.45	2893	2007.26	464.84	133
3dl143000	2744	**3006**	3001.7	2.88	37.2	2892	1956.09	1339.53	114
3dl144000	2744	**3012**	3007.85	1.85	47.8	2897	1980.32	1923.14	115
3dl145000	2744	**3006**	3001.2	2.16	58.1	2882	1972.18	1866.67	124
3dl146000	2744	**3005**	3001.35	1.46	14	2888	1948.91	1892.88	117
3dl147000	2744	**3007**	3001.95	2.31	30.5	2879	1995.73	1983.25	128
3dl148000	2744	**3018**	3014.5	1.96	165.45	2883	1982.66	1914.45	135
3dl149000	2744	**2999**	2993.95	2.62	20	2877	2024.45	1769.77	122
3dl1410000	2744	**3023**	3021.15	1.68	389.4	2904	2007.36	2003.40	119
Better		41/44/91							
Equal		3/44/91							
Worse		0/44/91							

Table 2.4: Comparative results for max-5-cut between the proposed MOH algorithm and DC

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G1	800	**17703**	17700.80	1.18	76.40	17627	532.14	376.14	76
G2	800	**17706**	17702.50	1.63	122.20	17636	537.26	288.13	70
G3	800	**17701**	17699.20	1.47	210.20	17623	525.92	357.24	78
G4	800	17709	17706.50	1.75	141.20	-	-	-	-
G5	800	17710	17708.60	1.66	269.70	-	-	-	-
G6	800	2781	2776.00	2.26	146.20	-	-	-	-
G7	800	2533	2530.75	2.00	56.50	-	-	-	-
G8	800	2535	2532.75	1.13	105.00	-	-	-	-
G9	800	2601	2598.65	1.28	6.55	-	-	-	-

Table 2.4 – continued from previous page

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
G10	800	2526	2520.00	4.18	143.70	-	-	-	-
G11	800	**677**	675.40	0.58	0.00	670	239.03	147.55	7
G12	800	**662**	661.40	0.49	153.10	660	240.87	191.89	2
G13	800	**689**	688.40	0.49	317.15	687	222.88	177.50	2
G14	800	**4639**	4634.60	1.83	37.65	4597	297.49	63.30	42
G15	800	**4606**	4599.90	1.79	80.05	4571	293.47	99.68	35
G16	800	**4613**	4610.30	1.31	94.60	4579	291.25	243.93	34
G17	800	4603	4600.85	1.01	96.50	-	-	-	-
G18	800	1268	1261.85	3.48	0.05	-	-	-	-
G19	800	1132	1122.45	7.08	0.10	-	-	-	-
G20	800	1172	1163.90	4.73	0.35	-	-	-	-
G21	800	1162	1153.50	5.34	0.05	-	-	-	-
G22	2000	**19553**	19547.00	3.64	42.40	19413	2429.87	1685.57	140
G23	2000	**19558**	19549.20	4.04	85.40	19413	2422.00	2248.13	145
G24	2000	**19555**	19547.20	2.93	88.55	19423	2255.39	1668.64	132
G25	2000	19554	19547.80	3.18	140.35	-	-	-	-
G26	2000	19552	19545.00	2.80	85.00	-	-	-	-
G27	2000	4236	4224.30	6.23	143.10	-	-	-	-
G28	2000	4182	4171.45	6.84	65.10	-	-	-	-
G29	2000	4327	4317.50	4.25	72.85	-	-	-	-
G30	2000	4340	4329.75	4.44	50.45	-	-	-	-
G31	2000	4211	4196.40	7.89	37.40	-	-	-	-
G32	2000	**1670**	1666.45	1.94	0.75	1647	1304.51	1272.00	23
G33	2000	**1638**	1635.05	1.20	0.20	1615	1194.92	678.48	23
G34	2000	**1615**	1610.20	2.84	0.40	1594	1232.62	629.56	21
G35	2000	**11605**	11595.20	4.15	68.80	11521	2030.16	961.14	84
G36	2000	**11601**	11593.80	3.03	12.25	11516	2074.70	510.45	85
G37	2000	**11603**	11599.40	2.46	70.15	11532	2026.00	1661.50	71
G38	2000	11601	11596.20	3.19	163.65	-	-	-	-
G39	2000	3022	3014.35	5.32	70.15	-	-	-	-
G40	2000	2986	2967.20	9.45	0.50	-	-	-	-
G41	2000	2986	2972.85	7.84	20.05	-	-	-	-
G42	2000	3109	3099.15	5.29	0.60	-	-	-	-
G43	1000	**9770**	9767.30	1.38	56.50	9700	583.20	76.61	70
G44	1000	**9772**	9768.05	1.60	16.85	9702	518.05	482.50	70
G45	1000	**9771**	9768.10	1.30	25.60	9708	502.37	470.51	63
G46	1000	9774	9769.55	1.66	47.80	-	-	-	-
G47	1000	9775	9770.05	1.86	60.70	-	-	-	-
G48	3000	6000	6000.00	0.00	0.00	6000	1871.21	0.50	0
G49	3000	6000	6000.00	0.00	0.00	6000	1864.70	0.48	0
G50	3000	6000	6000.00	0.00	0.00	6000	1887.36	0.50	0
G51	1000	5826	5822.30	2.05	0.75	-	-	-	-
G52	1000	5837	5832.35	1.68	4.90	-	-	-	-
G53	1000	5829	5825.90	1.09	55.75	-	-	-	-
G54	1000	5830	5826.70	1.42	28.40	-	-	-	-
G55	5000	12498	12498.00	0.00	0.00	-	-	-	-
G56	5000	4971	4957.90	8.75	243.70	-	-	-	-
G57	5000	4111	4108.70	1.19	293.50	-	-	-	-
G58	5000	29105	29090.70	9.28	272.10	-	-	-	-
G59	5000	7566	7541.20	19.22	120.40	-	-	-	-
G60	7000	17148	17148.00	0.00	0.00	-	-	-	-
G61	7000	7188	7174.50	7.74	437.60	-	-	-	-
G62	7000	5744	5736.90	2.88	4.20	-	-	-	-
G63	7000	40786	40767.50	10.50	420.80	-	-	-	-
G64	7000	10896	10851.50	23.04	48.60	-	-	-	-
G65	8000	6540	6528.90	4.93	8.50	-	-	-	-
G66	9000	7476	7470.60	4.74	10.90	-	-	-	-
G67	10000	8165	8151.60	7.32	8.20	-	-	-	-
G70	10000	9999	9999.00	0.00	0.10	-	-	-	-
G72	10000	8266	8256.00	6.74	8.60	-	-	-	-
G77	14000	11687	11672.10	11.41	21.10	-	-	-	-
G81	20000	16501	16480.20	10.06	271.50	-	-	-	-
3dl101000	1000	**1106**	1102.95	1.50	38.00	1073	321.44	79.97	33
3dl102000	1000	**1106**	1103.50	1.12	51.95	1067	358.55	78.05	39
3dl103000	1000	**1111**	1106.95	1.86	74.10	1072	343.13	106.00	39
3dl104000	1000	**1108**	1105.65	0.91	44.00	1076	330.08	223.84	32
3dl105000	1000	**1098**	1096.15	1.01	76.90	1074	327.13	197.17	24
3dl106000	1000	**1099**	1097.55	0.92	48.25	1071	329.38	304.61	28
3dl107000	1000	**1119**	1115.85	1.62	48.80	1084	321.82	230.50	35
3dl108000	1000	**1113**	1110.70	1.27	126.30	1077	333.74	147.03	36
3dl109000	1000	**1119**	1117.30	0.84	17.85	1089	327.09	186.92	30
3dl1010000	1000	**1115**	1114.10	0.83	336.95	1081	330.26	301.70	34
3dl141000	2744	**3029**	3022.00	3.51	4.15	2912	2416.83	1114.20	117
3dl142000	2744	**3033**	3025.75	3.73	58.40	2916	2665.55	1512.49	117
3dl143000	2744	**3015**	3007.75	5.23	100.10	2891	2568.33	706.35	124
3dl144000	2744	**3021**	3015.95	2.65	30.85	2914	2658.98	2066.46	107
3dl145000	2744	**3014**	3005.25	2.90	7.45	2897	2405.89	2252.09	117
3dl146000	2744	**3013**	3010.05	2.22	102.50	2906	2363.11	2227.79	107
3dl147000	2744	**3016**	3009.55	4.17	85.60	2900	2536.90	257.75	116
3dl148000	2744	**3027**	3022.70	2.12	12.85	2920	2376.40	2127.40	107

Table 2.4 – continued from previous page

Instance	$\|V\|$	MOH				DC			gap
		f_{best}	f_{avg}	std	$time(s)$	f_{best}	$tt(s)$	bt(s)	
3dl149000	2744	**3005**	2994.15	4.15	0.25	2901	2711.61	2687.12	104
3dl1410000	2744	**3033**	3023.25	3.78	17.75	2917	2432.17	1767.87	116
Better		41/44/91							
Equal		3/44/91							
Worse		0/44/91							

Table 2.5: Average computing time needed by the MOH algorithm (MOH(tavg)) to attain the best objective value of the DC algorithm. The time required by DC (DC(t)) to reach the same objective value is also included.

Instance	max-3-cut		max-4-cut		max-5-cut	
	DC(t)	MOH(tavg)	DC(t)	MOH(tavg)	DC(t)	MOH(tavg)
G1	339.41	0.16	290.51	0.18	376.14	0.01
G2	228.37	2.05	388.76	0.12	288.13	0.01
G3	205.06	0.35	245.50	0.24	357.24	0.01
G11	132.51	0.11	152.04	6.67	147.55	8.39
G12	59.09	2.11	117.52	6.65	191.89	16.02
G13	111.53	0.29	127.56	0.68	177.50	0.29
G14	190.40	0.09	159.14	0.13	63.30	0.01
G15	183.92	0.12	129.21	0.16	99.68	0.00
G16	75.02	0.08	75.89	0.09	243.93	0.01
G22	986.19	0.06	1314.45	0.09	1685.57	0.01
G23	1208.18	0.05	1775.80	0.08	2248.13	0.01
G24	1385.32	0.10	407.66	0.10	1668.64	0.01
G32	905.73	0.37	736.15	0.36	1272.00	2.00
G33	664.57	0.27	870.96	1.50	678.48	5.16
G34	827.79	0.31	1016.31	1.64	629.56	1.58
G35	1048.97	0.24	1764.52	0.10	961.14	0.00
G36	1196.02	0.13	1634.13	0.09	510.45	0.00
G37	1288.13	0.09	115.08	0.13	1661.50	0.00
G43	112.20	0.06	62.38	0.05	76.61	0.01
G44	47.87	0.09	43.88	0.08	482.50	0.01
G45	44.00	0.07	319.58	0.07	470.51	0.01
G48	293.30	0.52	0.48	0.01	0.50	0.00
G49	1587.05	0.53	0.49	0.01	0.48	0.00
G50	279.78	4.36	0.50	0.01	0.50	0.00
sg3dl101000	179.20	0.06	187.92	0.06	79.97	0.05
sg3dl102000	188.68	0.05	301.64	0.05	78.05	0.03
sg3dl103000	114.20	0.09	249.06	0.05	106.00	0.03
sg3dl104000	109.75	0.07	276.29	0.05	223.84	0.05
sg3dl105000	178.88	0.07	294.70	0.10	197.17	0.06
sg3dl106000	23.96	0.03	307.91	0.04	304.61	0.05
sg3dl107000	157.18	0.08	101.66	0.17	230.50	0.05
sg3dl108000	209.77	0.06	260.12	0.10	147.03	0.05
sg3dl109000	232.87	0.07	60.70	0.07	186.92	0.06
sg3dl1010000	184.91	0.05	257.21	0.14	301.70	0.04
sg3dl141000	1496.07	0.14	1511.84	0.05	1114.20	0.07
sg3dl142000	1408.24	0.14	464.84	0.04	1512.49	0.07
sg3dl143000	1659.44	0.11	1339.53	0.07	706.35	0.06
sg3dl144000	1759.67	0.25	1923.14	0.05	2066.46	0.09
sg3dl145000	1764.88	0.15	1866.67	0.05	2252.09	0.08
sg3dl146000	1529.38	0.12	1892.88	0.05	2227.79	0.07
sg3dl147000	1748.39	0.12	1983.25	0.05	257.75	0.07
sg3dl148000	1440.25	0.13	1914.45	0.05	2127.40	0.10
sg3dl149000	1699.97	0.14	1769.77	0.06	2687.12	0.11
sg3dl1410000	1476.52	0.11	2003.40	0.06	1767.87	0.07

Table 2.6: Comparative results of the proposed MOH algorithm with 7 state-of-the-art max-cut algorithms

Instance	$\|V\|$	f_{pre}	GES	BLS	MACUT	TS-UBQP	TS/PM	MAMBP	TSHEA	MOH
G1	800	11624	11624	11624	11624	11624	11624	11624	11624	11624
G2	800	11620	11620	11620	11620	11620	11620	11617	11620	11620
G3	800	11622	11622	11622	11622	11620	11620	11621	11622	11622
G4	800	11646	11646	11646	-	11646	11646	11646	11646	11646
G5	800	11631	11631	11631	-	11631	11631	11631	11631	11631
G6	800	2178	2178	2178	-	2178	2178	2177	2178	2178
G7	800	2006	2006	2006	-	2006	2006	2002	2006	2006
G8	800	2005	2005	2005	-	2005	2005	2004	2005	2005
G9	800	2054	2054	2054	-	2054	2054	2052	2054	2054
G10	800	2000	2000	2000	-	2000	2000	1998	2000	2000

Table 2.6 – continued from previous page

Instance	$\|V\|$	f_{pre}	GES	BLS	MACUT	TS-UBQP	TS/PM	MAMBP	TSHEA	MOH
G11	800	564	564	564	564	564	564	564	564	564
G12	800	556	556	556	556	556	556	556	556	556
G13	800	582	582	582	582	580	582	582	582	582
G14	800	3064	3064	3064	3064	3061	3063	3062	3064	3064
G15	800	3050	3050	3050	3050	3050	3050	3050	3050	3050
G16	800	3052	3052	3052	3052	3052	3052	3052	3052	3052
G17	800	3047	3047	3047	-	3046	3047	3047	3047	3047
G18	800	992	992	992	-	991	992	992	992	992
G19	800	906	906	906	-	904	906	905	906	906
G20	800	941	941	941	-	941	941	941	941	941
G21	800	931	931	931	-	930	931	930	931	931
G22	2000	13359	13359	13359	13359	13359	13349	13359	13359	13359
G23	2000	13344	13342	13344	13344	13342	13332	13344	13344	13344
G24	2000	13337	13337	13337	13337	13337	13324	13336	13337	13337
G25	2000	13340	13340	13340	-	13332	13326	13340	13340	13340
G26	2000	13328	13328	13328	-	13328	13313	13328	13328	13328
G27	2000	3341	3341	3341	-	3336	3325	3341	3341	3341
G28	2000	3298	3298	3298	-	3295	3287	3298	3298	3298
G29	2000	3405	3405	3405	-	3391	3394	3403	3405	3405
G30	2000	3413	3413	3412	-	3403	3402	3412	3413	3413
G31	2000	3310	3310	3309	-	3288	3299	3309	3310	3310
G32	2000	1410	1410	1410	1410	1406	1406	1410	1410	1410
G33	2000	1382	1382	1382	1382	1378	1374	1382	1382	1382
G34	2000	1384	1384	1384	1384	1378	1376	1384	1384	1384
G35	2000	7687	7686	7684	7686	7678	7661	7686	7687	7687
G36	2000	7680	7680	7678	7679	7670	7660	7678	7680	7680
G37	2000	7691	7691	7689	7690	7682	7670	7689	7691	7691
G38	2000	7688	7687	7687	-	7683	7670	7688	7688	7688
G39	2000	2408	2408	2408	-	2397	2397	2408	2408	2408
G40	2000	2400	2400	2400	-	2390	2392	2400	2400	2400
G41	2000	2405	2405	2405	-	2400	2398	2405	2405	2405
G42	2000	2481	2481	2481	-	2469	2474	2481	2481	2481
G43	1000	6660	6660	6660	6660	6660	6660	6659	6660	6660
G44	1000	6650	6650	6650	6650	6639	6649	6650	6650	6650
G45	1000	6654	6654	6654	6654	6652	6654	6654	6654	6654
G46	1000	6649	6649	6649	-	6649	6649	6649	6649	6649
G47	1000	6657	6657	6657	-	6656	6656	6657	6657	6657
G48	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000
G49	3000	6000	6000	6000	6000	6000	6000	6000	6000	6000
G50	3000	5880	5880	5880	5800	5880	5880	5880	5880	5880
G51	1000	3848	3848	3848	-	3847	3847	3847	3848	3848
G52	1000	3851	3851	3851	-	3849	3850	3851	3851	3851
G53	1000	3850	3850	3850	-	3848	3848	3850	3850	3850
G54	1000	3852	3852	3852	-	3851	3850	3851	3852	3852
G55	5000	10299	-	10294	10299	10236	-	10299	10299	10299
G56	5000	4017	-	4012	4016	3934	-	4016	4017	*4016*
G57	5000	3494	-	3492	-	3460	-	3488	3494	3494
G58	5000	19293	-	19263	-	19248	-	19276	19276	*19288*
G59	5000	6086	-	6078	-	6019	-	6085	6085	**6087**
G60	7000	14188	-	14176	14186	14057	-	14186	14186	**14190**
G61	7000	5796	-	5789	-	5680	-	5796	5796	**5798**
G62	7000	4870	-	4868	-	4822	-	4866	4866	*4868*
G63	7000	27045	-	26997	-	26963	-	26754	27018	*27033*
G64	7000	8751	-	8735	-	8610	-	8731	8735	*8747*
G65	8000	5562	-	5558	5550	5518	-	5556	5560	*5560*
G66	9000	6364	-	6360	6352	6304	-	6352	6364	*6360*
G67	10000	6950	-	6940	6934	6894	-	6934	6944	*6942*
G70	10000	9591	-	9541	-	9458	-	9580	9548	*9544*
G72	10000	7006	-	6998	-	6922	-	6990	6990	*6998*
G77	14000	9938	-	9926	-	-	-	9900	9902	*9928*
G81	20000	14048	-	14030	-	-	-	13978	14010	*14036*
3dl101000	1000	896	896	-	-	-	-	-	896	896
3dl102000	1000	900	900	-	-	-	-	-	900	900
3dl103000	1000	892	892	-	-	-	-	-	892	892
3dl104000	1000	898	898	-	-	-	-	-	898	898
3dl105000	1000	886	886	-	-	-	-	-	886	886
3dl106000	1000	888	888	-	-	-	-	-	888	888
3dl107000	1000	900	900	-	-	-	-	-	900	900
3dl108000	1000	882	882	-	-	-	-	-	882	882
3dl109000	1000	902	902	-	-	-	-	-	902	902
3dl1010000	1000	894	894	-	-	-	-	-	894	894
3dl141000	2744	2446	2446	-	-	-	-	-	2446	2446
3dl142000	2744	2458	2458	-	-	-	-	-	2458	2458
3dl143000	2744	2442	2442	-	-	-	-	-	2442	**2444**
3dl144000	2744	2450	2450	-	-	-	-	-	2450	2450
3dl145000	2744	2446	2446	-	-	-	-	-	2446	2446
3dl146000	2744	2452	2452	-	-	-	-	-	2452	2452
3dl147000	2744	2444	2444	-	-	-	-	-	2444	2444
3dl148000	2744	2448	2448	-	-	-	-	-	2448	2448
3dl149000	2744	2428	2426	-	-	-	-	-	2428	2428
3dl1410000	2744	2460	2458	-	-	-	-	-	2460	*2458*

Table 2.6 – continued from previous page

Instance	$\|V\|$	f_{pre}	GES	BLS	MACUT	TS-UBQP	TS/PM	MAMBP	TSHEA	MOH
Better		4/91/91	4/74/91	20/71/91	7/30/91	47/69/91	29/54/91	33/71/91	11/91/91	
Equal		74/91/91	70/74/91	51/71/91	23/30/91	22/69/91	25/54/91	37/71/91	75/91/91	
Worse		13/91/91	0/74/91	0/71/91	0/30/91	0/69/91	0/54/91	1/71/90	5/91/91	

2.4 Discussion

In this section, we investigate the role of several important ingredients of the proposed algorithm, including the bucket sorting data structure, the descent improvement search operators O_1 and O_2 and the diversified improvement search operators O_3 and O_4.

2.4.1 Impact of the bucket sorting technique

As described in Section 2.2.5, the bucket sorting technique is utilized in the MOH algorithm for the purpose of quickly identifying a suitable move with the best objective gain. To verify its effectiveness, we implemented another MOH version where we replaced the bucket sorting data structure with a simple vector and conducted an experimental comparison on the max-3-cut problem. For this experiment, we used 20 representative Gxx instances and ran 20 times both MOH versions to solve each chosen instance with a time limit of 300 seconds.

Table 2.7 reports the average of the best objective values and the total number of iterations of each MOH version for each instance. From Table 2.7, we observe that the MOH algorithm using the bucket sorting structure conducted 3.3 times more iterations on average than using the vector structure within the given time span. Moreover, the former is able to find better results for 16 instances and only one worse result. In conclusion, this experiment confirms that using the devised bucket sorting technique is able to considerably improve the computational efficiency and search capacity of the MOH algorithm.

Table 2.7: Computational assessment of bucket sorting compared to an implementation using a vector applied to the max-3-cut problem

Instance	bucket sorting structure		vector structure		differences	
	f_{bss}	$iter_{bss}$	f_{vs}	$iter_{vs}$	$f_{bss} - f_{vs}$	$iter_{bss}/iter_{vs}$
G22	17135.65	87,068,095.55	17132.7	55,940,769.45	2.95	1.56
G26	17128.1	89,044,944.75	17121.65	50,698,801.15	6.45	1.76
G28	3943.4	81,621,472.45	3942.9	49,226,453.00	0.5	1.66
G30	4091.95	89,369,709.35	4095.85	52,714,888.95	-3.9	1.70
G32	1654.85	212,255,042.05	1652.75	59,712,070.05	2.1	3.55
G34	1605.4	216,409,597.50	1604.2	51,582,268.90	1.2	4.20
G36	10024.1	136,113,904.60	10015	48,257,118.45	9.1	2.82
G38	10027.1	147,998,869.05	10021.5	53,182,934.85	5.6	2.78
G40	2841.85	137,242,801.85	2831.75	53,555,508.15	10.1	2.56
G44	8556.75	99,472,399.80	8557.1	102,758,227.95	-0.35	0.97
G46	8555.1	100,453,139.40	8555.35	100,251,434.60	-0.25	1.00
G54	5028.65	170,660,709.15	5026.9	98,723,794.70	1.75	1.73
G56	4709.05	105,834,778.80	4662.45	14,561,723.95	46.6	7.27
G58	25144.4	88,340,858.10	25092.5	14,574,161.75	51.9	6.06
G60	17019.6	37,339,981.15	16963.55	8,873,616.55	56.05	4.21
G62	5685.7	101,427,430.65	5656.7	9,955,135.45	29	10.19
G64	10318.1	68,975,406.10	10175.75	8,846,430.90	142.35	7.80
G66	7417.3	92,758,417.20	7353.45	7,508,205.95	63.85	12.35
G70	9999	4,336,200.40	9999	4,046,618.05	0	1.07
G72	8189.35	77,034,721.40	8109.9	6,998,747.65	79.45	11.01

Table 2.8: Comparative results for max-cut with varying combination strategies of O_1 and O_2

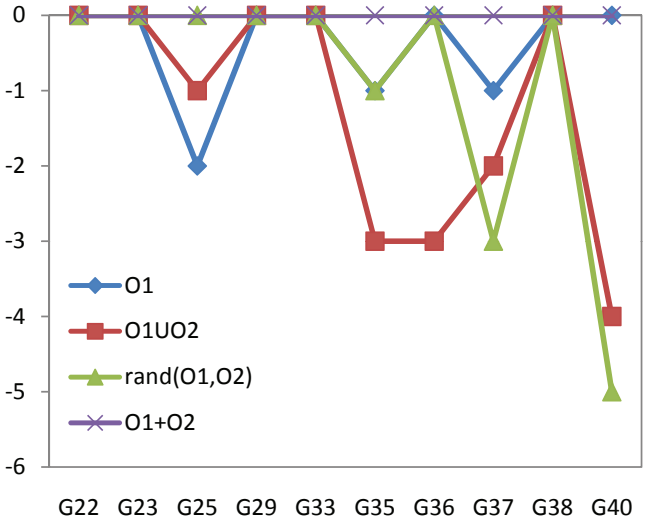
Instance	O_1			$O_1 \cup O_2$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G22	13359	13357.6	381.6	13359	13355.8	357.3
G23	13344	13343.6	473.4	13344	13344	550.9
G25	13338	13334	442.8	13339	13335.8	690.4
G29	3405	3398.22	211.1	3405	3396.4	254.2
G33	1382	1381.4	553.5	1382	1382	716.5
G35	7686	7681.3	755.4	7684	7679.1	449.6
G36	7680	7672	1367.1	7677	7672.5	408.1
G37	7690	7685.5	1039.2	7689	7683.4	1099.0
G38	7688	7684	135.2	7688	7681.2	177.8
G40	2400	2384.7	453.5	2396	2381.6	427.2
Instance	$rand(O_1, O_2)$			$O_1 + O2$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G22	13359	13356	365.3	13359	13357	438.2
G23	13344	13343.9	584.9	13344	13344	302.1
G25	13340	13336.4	408.8	13340	13335.5	451.5
G29	3405	3398.4	403.9	3405	3398.1	569.9
G33	1382	1381.8	585.2	1382	1381.4	667.4
G35	7686	7683.1	628.0	7687	7684.3	968.3
G36	7680	7672	944.8	7680	7675.3	1075.6
G37	7688	7681.7	1078.3	7691	7687.5	1133.2
G38	7688	7680.8	153.6	7688	7685.7	333.0
G40	2395	2388.8	412.4	2400	2385.2	467.1

2.4.2 Impact of the descent improvement search operators

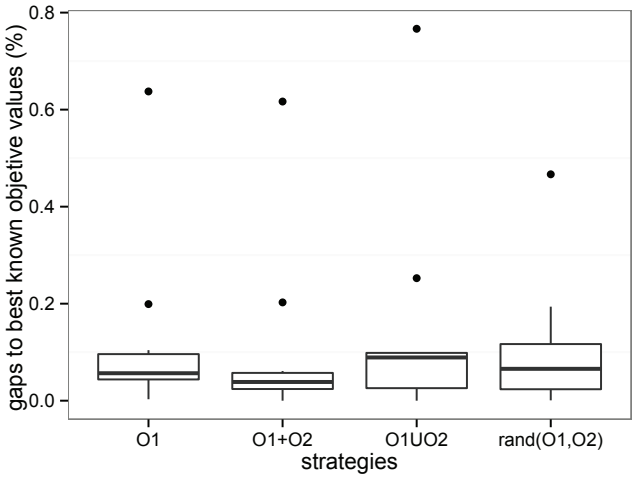
As described in Section 2.2.6, the proposed algorithm employs operators O_1 and O_2 for its descent improvement phase to obtain local optima. To analyze the impact of these two operators, we implement three variants of our algorithm, the first one using the operator O_1 alone, the second one using the union $O_1 \cup O_2$ such that the descent search procedure always chooses the best move among the O_1 *and* O_2 moves [Lü *et al.*, 2011a], the third one using operator $rand(O_1, O_2)$ where the descent procedure applies randomly and with equal probability O_1 or O_2, while keeping all the other ingredients and parameters fixed as described in Section 2.3.3. The strategy used by our original algorithm, detailed in Section 2.2.6, is denoted as $O_1 + O_2$.

This study was based on the max-cut problem and the same 10 challenging instances used for parameter tuning of Section 2.3.3 . Each selected instance was solved 10 times by each of these variants and our original algorithm. The stopping criterion was a timeout limit of 30 minutes. The obtained results are presented in Table 2.8, including the best objective value f_{best}, the average objective value f_{avg} over the 10 independent runs, as well as the CPU times in seconds to reach f_{best}. To evaluate the performance, we display in Fig. 2.2(a) the gaps between the best objective values obtained by different strategies and the best objective values by our original algorithm. We also show in Fig. 2.2(b) the box and whisker plots which indicate, for different O_1 and O_2 combination strategies, the distribution and the ranges of the obtained results for the 10 tested instances. The results are expressed as the additive inverse of percent deviation of the averages results from the best known objective values obtained by our original algorithm.

From Fig. 2.2(a), one observes that for the tested instances, other combination strategies obtain fewer best known results compared to the strategy $O_1 + O_2$, and produce large gaps to the best known results on some instances. From Fig. 2.2(b), we observe a clear difference in the distribution of the results with different strategies. For the results with the strategies of $O_1 + O_2$, the plot indicates a smaller mean value and significantly smaller variation compared to the results obtained by other strategies. We thus conclude that the strategy used by our algorithm ($O_1 + O_2$) performs better than other strategies.



(a) $f_{best-strategy} - f_{bestknown}$, gaps to best known objective values



(b) $(f_{bestknown} - f_{avg-strategy})/f_{bestknown} * 100\%$, gaps to best known objective values

Figure 2.2: Analysis of the move operators O_1, O_2

Table 2.9: Comparative results for max-cut with varying parameter ρ

Instance	$\rho = 1$			$\rho = 0$			$\rho = 0.5$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G22	13359	13350.1	352.7	13356	13355.2	440.6	13359	13357	438.2
G23	13344	13344	441.4	13338	13335.6	340.1	13344	13344	302.1
G25	13339	13335.1	426.1	13337	13333.5	412.9	13340	13335.5	451.5
G29	3405	3395.2	614.5	3402	3399.8	593.5	3405	3398.1	569.9
G33	1376	1373.6	519.9	1382	1382	609.2	1382	1381.4	667.7
G35	7686	7680.7	832.1	7680	7678.2	850.8	7687	7684.3	968.3
G36	7676	7669.2	1540.8	7671	7667.6	1304.8	7680	7675.3	1075.6
G37	7690	7681.2	1167.8	7685	7679.6	1053.8	7691	7687.5	1133.2
G38	7688	7681.4	275.1	7685	7679	257.3	7688	7685.7	333.0
G40	2394	2375.3	453.0	2399	2390.5	529.8	2400	2385.2	467.1

2.4.3 Impact of the diversified improvement search operators

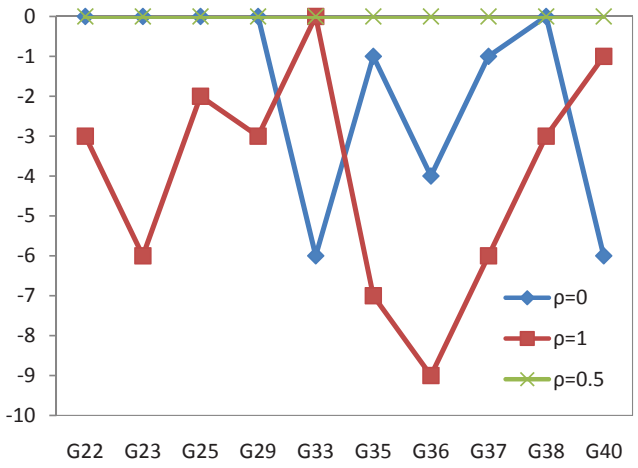
As described in Section 2.2.7, the proposed algorithm employs two diversified operator O_3 and O_4 to enhance the search power of the algorithm and make it possible for the search to visit new promising regions. The diversified improvement procedure uses probability ρ to select O_3 or O_4. To analyze the impact of operators O_3 and O_4, we tested our algorithm with $\rho = 1$ (using the operator O_3 alone), $\rho = 0.5$ (equal application of O_3 and O_4 used in our original MOH algorithm), $\rho = 0$ (using the operator O_4 alone), while keeping all the other ingredients and parameters fixed as described before. The stopping criterion was a timeout limit of 30 minutes. We then independently solved each selected instance 10 times with those different values of ρ. The obtained results on the max-cut problem for the 10 challenging instances used for parameter tuning of Section 2.3.3 are presented in Table 2.9, including the best objective value f_{best}, the average objective value f_{avg} over the 10 independent runs, as well as the CPU times in seconds to reach f_{best}. To evaluate the performance, we again calculate the gaps between different best objective values shown in Fig. 2.3(a) and average objective values shown in Fig. 2.3(b), where the set of values f_{best}, f_{avg}, when $\rho = 0.5$, are set as the reference values.

As in Section 2.4.2, to evaluate the performance, we show in Fig. 2.3(a) the gaps between the best objective values obtained with different values of ρ and the best objective values by our original MOH algorithm ($\rho = 0.5$). We also show in Fig. 2.3(b) the box and whisker plots which indicates, for different values of ρ, the distribution and the ranges of the obtained results for the 10 tested instances. The results are expressed as the additive inverse of percent deviation of the averages results from the best known objective values obtained by our original algorithm.

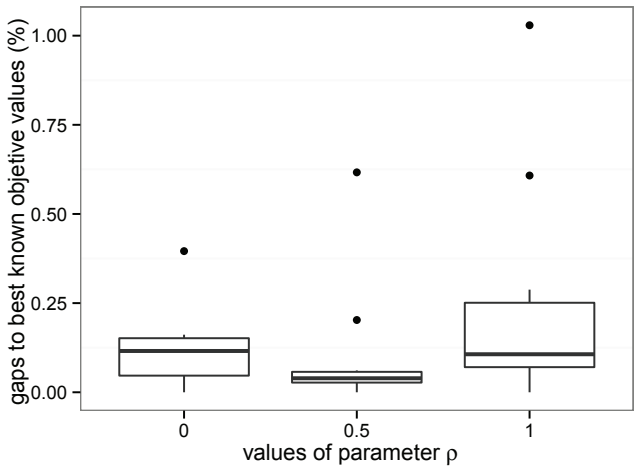
Fig. 2.3(a) discloses that using O_3 or O_4 alone obtains fewer best known results than using them jointly and achieves significantly worse results on some particular instances. From Fig. 2.3(b), we observe a visible difference in the distribution of the results with different strategies. For the results with the parameter $\rho = 0.5$, the plot indicates a smaller mean value and significantly smaller variation compared to the results obtained by other strategies. We thus conclude that jointly using O_3 and O_4 with $\rho = 0.5$ is the best choice since it produces better results in terms of both best and average results.

2.5 Conclusion

This chapter proposed an effective multiple operator heuristic (MOH) for approximating the general max-k-cut problem, which coordinates five distinct search operators to be organized in three search phases. Computational study on the two sets of well-known benchmarks composed of 91 instances demonstrates that the proposed MOH algorithm not only performs well on the general max-k-cut problem, but also



(a) $f_{best-\rho} - f_{bestknown}$, gaps between f_{best} obtained with different ρ values to best known objective values



(b) $(f_{bestknown} - f_{avg-\rho})/f_{bestknown} * 100\%$, gaps to best known objective values

Figure 2.3: Analysis of the move operators O_3, O_4

remains highly competitive for the special case of the popular max-cut problem. In addition, we investigated the importance of the bucket sorting technique as well as alternative strategies for combing search operators and justified the combinations adopted in the proposed MOH algorithm.

In the next chapter, we will consider the max-bisection problem, which is a cardinality constrained max-cut problem. To solve this problem, we go along the line of multiple search operators based local search to conduct extensive exploitation of the search space and develop an effective iterated tabu search for the max-bisection problem.

3

An effective iterated tabu search for the max-bisection problem

In this chapter, we present an Iterated Tabu Search (ITS) algorithm to solve the max-bisection problem. ITS employs two distinct search operators organized into three search phases to effectively explore the search space. Bucket sorting is used to ensure a high computational efficiency of the ITS algorithm. Experiments on 71 well-known benchmark instances of the literature demonstrate that the proposed algorithm is highly competitive compared to the state-of-the-art approaches and discovers improved best-known results (new lower bounds) for 10 benchmark instances. The content of this chapter is based on an article submitted to Computers & Operations Research which was revised in Feb. 2016.

Contents

3.1 Introduction

In this chapter, we present an effective heuristic algorithm for the max-bisection problem based on the iterated local search (ILS) framework [Lourenço *et al.*, 2010], which has been applied with success to a number of combinatorial optimization problems (for some recent application examples, see [Benlic and Hao, 2013a; Cordeau and Maischberger, 2012; Palubeckis *et al.*, 2014; Qin *et al.*, 2015; Silva *et al.*, 2015]). The proposed iterated tabu search algorithm relies on two distinct local search operators for solution transformations. The algorithm is composed of three different search phases (descent-based improvement, diversifying improvement and perturbation) to ensure an effective examination of the search space. The basic idea of our approach can be summarized as follows. The descent-based improvement procedure aims to locate a local optimum from an initiating solution (Section 3.2.6). This is achieved by a fast descent procedure with the conventional *1-move* operator (Sections 3.2.3). Then the diversifying improvement phase applies a tabu search procedure (with the *1-move* and *constrained swap* operators) to examine nearby search areas around the obtained local optimum with the purpose of discovering improved solutions (Section 3.2.7). Each time an improved solution is found, the search switches back to the descent-based improvement phase to make an intensive exploitation of the area. If the search is trapped in a deep local optimum, the perturbation phase applies a random search operator to definitively lead the search process to a distant region from which a new round of the search procedure starts. This process is iterated until a stopping condition is met. To ensure the computational efficiency of the search operators, we employ streamlining techniques based on dedicated and efficient data structures.

The proposed ITS algorithm includes the following original features. First, ITS relies on a joint use of two complementary search operators to conduct an extensive exploitation of the search space. The *1-move* operator is used to quickly discover a local optimal solution from which improved solutions are sought by employing the more advanced *c-swap* operator. Second, in addition to an improvement phase and a perturbation phase used in conventional ILS algorithms, the proposed ITS algorithm additionally includes a fast descent procedure to quickly attain a promising search area which is deeply examined with the powerful tabu search procedure. This combination prevents the search procedure from running the more expensive tabu search procedure in an unpromising area and thus helps to increase the search efficiency of the algorithm.

We assess the performance of the proposed algorithm on 71 well-known benchmark graphs in the literature which were commonly used to test new max-cut and max-bisection algorithms. Computational results show that ITS competes favorably with respect to the existing best performing max-bisection heuristics, by improving the current best-known results (new lower bounds) on 10 instances.

This chapter is organized as follows. In Section 3.2, we present the general scheme and main components of the designed ITS algorithm (search space, move operators, descent procedure, tabu search procedure and perturbation procedure). Section 3.3 provides computational results and comparisons with other state-of-the-art algorithms in the literature. Section 3.4 is dedicated to an analysis of essential parts of the proposed algorithm. Concluding remarks are given in Section 3.5.

3.2 Iterated tabu search for max-bisection

3.2.1 General working scheme

The general working procedure of the proposed ITS algorithm for the max-bisection problem is described in Algorithm 2 whose components are explained in the following subsections. The algorithm explores the search space of bisections (Section 3.2.2) by alternately applying two distinct and complementary move operators (*1-move* and *constrained_swap*) to make transitions from the current solution to a neighboring solution (Section 3.2.3). Basically, from an initial solution (i.e., a bisection) which is randomly sampled from the search space, the algorithm first applies, with operator *1-move*, a descent local search to attain a local optimum I (Alg. 2, lines 8 - 20, descent-based improvement phase, Section 3.2.6). Since the returned solution I makes an additional *1-move* operation with respect to the local optimal solution I^*, a roll back scheme that makes a reverse *1-move* operation is used to get back to the search status when I^* is reached. Another alternative operation to achieve the same purpose is to make a copy of I^* to I and initialize the bucket data structure for the diversifying improvement phase, but this method is considered as more expensive than the roll back scheme. Given that each roll back actually performs two consecutive *1-move* operations, the iteration counter is thus increased by 2. Then the algorithm continues to the diversifying improvement phase (Alg. 2, lines 25 - 44, Section 3.2.7) which uses a tabu-based procedure to explore new solutions around the local optimum I. This search phase relies on both *1-move* and *constrained_swap* which are applied in a probabilistic way. The second search phase ends when a maximum number ω of consecutive iterations is reached without improving the best solution found. In this case, the search is judged to be trapped in a deep local optimum. To escape this deep local optimum, the search turns into a perturbation phase (Alg. 2, line 46), which strongly transforms the current solution by randomly swapping γ vertices (Section 3.2.8). The perturbed solution serves then as a new starting solution of the next round of the descent-based improvement phase. This process is iterated until a stopping criterion (e.g., a given cutoff time) is met and the best solution found during the search is returned as the outcome of the algorithm.

3.2.2 Search space and evaluation solution

Given the purpose of max-bisection (i.e., to partition the vertex set V into two equal-sized subsets such that the weight sum of the edges crossing the two subsets is maximized), we define the search space Ω to be composed of all possible *bisections* (i.e., balanced two-way partitions) $\{S_1, S_2\}$ of vertex set V:

$$\Omega = \{\{S_1, S_2\} : S_1, S_2 \subset V, S_1 \cup S_2 = V, S_1 \cap S_2 = \emptyset, |S_1| = |S_2|\}. \tag{3.1}$$

For a given bisection $I = \{S_1, S_2\} \in \Omega$, its objective value $f(I)$ is the weight sum of the crossing edges which connect S_1 and S_2:

$$f(I) = \sum_{i \in S_1, j \in S_2} w_{ij}. \tag{3.2}$$

Then, for two candidate bisections $I' \in \Omega$ and $I'' \in \Omega$, I' is better than I'' if and only if $f(I') > f(I'')$. The goal of our algorithm is to find a solution $I_{best} \in \Omega$ with $f(I_{best})$ as large as possible. Our algorithm only samples feasible solutions within the above search space.

Algorithm 2 General procedure for the max-bisection problem.

Require: Graph $G = (V, E)$, max number ω of consecutive non-improvement iterations in diversified phase, probability ρ for selecting *1-move* and $c - swap()$.
Ensure: the best solution I_{best} found
$I \leftarrow$ Random_Initial_solution() ▷ A random bisection from the search space Ω, see Section 3.2.2
$I_{best} \leftarrow I$ ▷ I_{best} records the best solution found so far
$iter \leftarrow 0$ ▷ Iteration counter
while stopping condition not satisfied **do**
  /* lines 8 to 20: Descent local search phase, see Section 3.2.6 */
  repeat
    $I^* \leftarrow I$
    $I \leftarrow I \oplus$ *1-move*(u, S_1) ▷ Select a vertex with the best move gain and perform the *1-move*
    Update move gains ▷ Move gains recorded in a bucket data structure, see Section 3.2.4
    $iter \leftarrow iter + 1$
    $I \leftarrow I \oplus$ *1-move*(v, S_2)
    Update move gains; $iter \leftarrow iter + 1$
  until $f(I) < f(I^*)$
  /* lines 17 to 20: Roll back to recover the search status when the local optimum I^* is reached */
  $I \leftarrow I \oplus$ *1-move*(v, S_1)
  Update move gains; $iter \leftarrow iter + 1$
  $I \leftarrow I \oplus$ *1-move*(u, S_2)
  Update move gains; $iter \leftarrow iter + 1$
  if $f(I^*) > f(I_{best})$ **then**
    $I_{best} \leftarrow I^*$ ▷ Update the best solution found so far
  end if
  /* lines 25 to 44: Diversifying improvement phase, see Section 3.2.7 */
  $c \leftarrow 0$ ▷ Counter of non-improvement iterations
  while $c < \omega$ **do**
    if $Random(0, 1) < \rho$ **then** ▷ $Random(0, 1)$ returns a random real number between 0 to 1
      $I \leftarrow I \oplus$ *c-swap*(u, v) ▷ Perform the best *c-swap* considering tabu status
      Add $\{u, v\}$ to tabu list
      Update move gains; $iter \leftarrow iter + 1$
    else
      $I \leftarrow I \oplus$ *1-move*(u, S_1) ▷ Perform the best *1-move* considering tabu status
      Add u to tabu list
      Update move gains; $iter \leftarrow iter + 1$
      $I \leftarrow I \oplus$ *1-move*(v, S_2)
      Add v to tabu list
      Update move gains; $iter \leftarrow iter + 1$
    end if
    if $f(I) > f(I_{best})$ **then**
      $I_{best} \leftarrow I; c \leftarrow 0$ ▷ Update the best solution found so far
    else
      $c \leftarrow c + 1$
    end if
  end while
  /* Perturbation phase, see Section 3.2.8 */
  $I \leftarrow Perturb(I)$
end while

3.2.3 Move operators and neighborhood

From the incumbent solution which is necessarily a feasible solution (i.e., a bisection), the proposed algorithm explores its neighboring solutions by applying two different move operators. Formally, let $I = \{S_1, S_2\}$ be the incumbent solution and let mv be a move operator, we use $I' \leftarrow I \oplus mv$ to denote the neighboring solution I' obtained by applying mv to I.

For a given move operator mv, we define the notion of *move gain* Δ_{mv}, which indicates the variation in the objective value when the incumbent solution I is transformed to a neighboring solution I' by applying the move operator, i.e.,

$$\Delta_{mv} = f(I') - f(I) \tag{3.3}$$

where f is the optimization objective defined in Eq. (3.2).

Our algorithm employs two move operators: *1-move* and *constrained_swap* (*c-swap* for short) which are defined as follows.

- *1-move*: Given a bisection $I = \{S_1, S_2\}$, *1-move*(v, S_i) displaces a vertex v from its current subset S_i $(i = 1, 2)$ to the other subset S_{3-i}. Note that one application of *1-move* always leads to an unbalanced partition (thus an infeasible bisection). To maintain the balance of the partition, two consecutive applications of *1-move* are always jointly performed by moving first a vertex u from subset S_1 to S_2 (denoted by *1-move*(u, S_1)), accompanied by moving another vertex v from S_2 to S_1 (denoted by *1-move*(v, S_2)). Such a combined application of *1-move* ensures a balanced partition (thus a feasible bisection).
- *c-swap*: Given a bisection $I = \{S_1, S_2\}$, *c-swap*(v_1, v_2) exchanges two vertices $v_1 \in S_1$ and $v_2 \in S_2$ belonging to two subsets subject to the constraint that v_1 and v_2 is linked by an edge $(v_1, v_2) \in E$. In other words, our *c-swap* operator only considers pairs of vertices such that they not only belong to the two subsets of the bisection, but also are linked by an edge crossing the subsets.

Based on these two move operators (*1-move* and *c-swap*), two neighborhoods $N1$ and $N2$ are defined as follows:

$$N1 = \{I \oplus \textit{1-move}(v, S_i) : v \in S_i\}$$

$$N2 = \{I \oplus \textit{c-swap}(v_1, v_2) : v_1 \in S_1, v_2 \in S_2, \{v_1, v_2\} \in E\}$$

Clearly, $N1$ and $N2$ are bounded in size by $O(|V|)$ and $O(|E|)$ respectively.

As stated above, since the neighboring solutions of I in $N1$ are infeasible, two consecutive applications of *1-move* are performed to maintain the feasibility of the new neighboring solution. We also note that the *1-move* operator was commonly used in the literature [Fiduccia and Mattheyses, 1982; Lin and Zhu, 2014; Wu and Hao, 2013].

On the contrary, few studies investigate the *swap* operator. When it was employed, it was usually applied in an *unconstrained* way in the sense that each possible pair of vertices (v_1, v_2) such that $v_1 \in S_1$ and $v_2 \in S_2$ was considered [Kernighan and Lin, 1970]. Note that the unconstrained swap operator will lead to a neighborhood of size $O(|V|^2)$ which is typically much larger than our $N2$ neighborhood (bounded by $O(|E|)$ in size) induced by the constrained *c-swap* operator. This is particularly true for sparse graphs.

After an application of either of the two move operators, the move gains of the impacted solutions are updated according to the dedicated streamlining techniques explained below.

3.2.4 Bucket sorting for fast move gain evaluation and updating

As we show in Sections 3.2.6 and 3.2.7, our algorithm iteratively makes transitions from the incumbent solution to a particular neighboring solution by applying a selected move operation. Typically, to make the right choice, the algorithm needs to identify the most favorable move operation with an increased move gain among many candidates. To ensure a high search efficiency, it is crucial for the algorithm to be able to rapidly evaluate all the candidate moves at each iteration of its search process. In this section, we describe fast incremental evaluation techniques based on bucket data structures to streamline the calculations. These specific techniques allow the algorithm to efficiently keep and update the move gains after each move application.

1-move: For each *1-move(v,S)* application, let Δ_v be the move gain of moving vertex $v \in S$ to the other subset $V \setminus S$ (We use the notation $\Delta_{v \to S}$ if the destination subset S needs to be emphasized). Then initially, each move gain can be determined by the following Formula:

$$\Delta_v = \sum_{i \in S, i \neq v} \omega_{vi} - \sum_{j \in V \setminus S} \omega_{vj} \tag{3.4}$$

where ω_{vi} and ω_{vj} are respectively the weights of edges $\{v, i\}$ and $\{v, j\}$.

Then, once a *1-move(v,S)* is performed, the move gain of each vertex can be updated by performing the following calculation:

1. $\Delta_v = -\Delta_v$
2. for each $u \in V \setminus \{v\}$,

$$\Delta_u = \begin{cases} \Delta_u - 2\omega_{uv}, & \text{if } u \in S \\ \Delta_u + 2\omega_{uv}, & \text{if } u \in V \setminus S \end{cases} \tag{3.5}$$

Now we explain how the factor 2 in Eq. (3.5) comes. Let us first consider the objective gain of moving a vertex $u \in S$ ($u \neq v$), which is $\Delta_u = \sum_{i \in S, i \neq u} w_{ui} - \sum_{j \in V \setminus S} w_{uj}$ according to the definition of the objective function. After the vertex v is moved from S to $V \setminus S$, the objective gain of moving vertex u is updated as $\Delta_u = \sum_{i \in S \setminus \{v\}, i \neq u} w_{ui} - \sum_{j \in V \setminus S \cup \{v\}} w_{uj} = \Delta_u - w_{uv} - (w_{uv}) = \Delta_u - 2w_{uv}$. Similarly, the objective gain of moving a vertex $u \in V \setminus S$ ($u \neq v$) is given by $\Delta_u = \sum_{j \in V \setminus S, j \neq u} w_{uj} - \sum_{i \in S} w_{ui}$. After the vertex v is moved from S to $V \setminus S$, the objective gain of moving $u \in V \setminus S$ is updated as $\Delta_u = \sum_{j \in V \setminus S \cup \{v\}, j \neq u} w_{uj} - \sum_{i \in S \setminus \{v\}} w_{ui} = \Delta_u + w_{uv} - (-w_{uv}) = \Delta_u + 2w_{uv}$.

Notice that if there is no edge between the vertices u and v, the edge weight ω_{uv} equals 0, in which case the associated Δ_u value will not change. One observes that only the move gains of vertices affected by this move (i.e., the displaced vertex and its adjacent vertices) will be updated, which reduces the computation time significantly.

Usually the move gains can be stored in an array, with which the time for finding the best move with maximum move gain grows linearly with the number of vertices ($O(n)$). For large problem instances (very large n), the required time can still be quite high. To avoid unnecessary searching for the best move, we

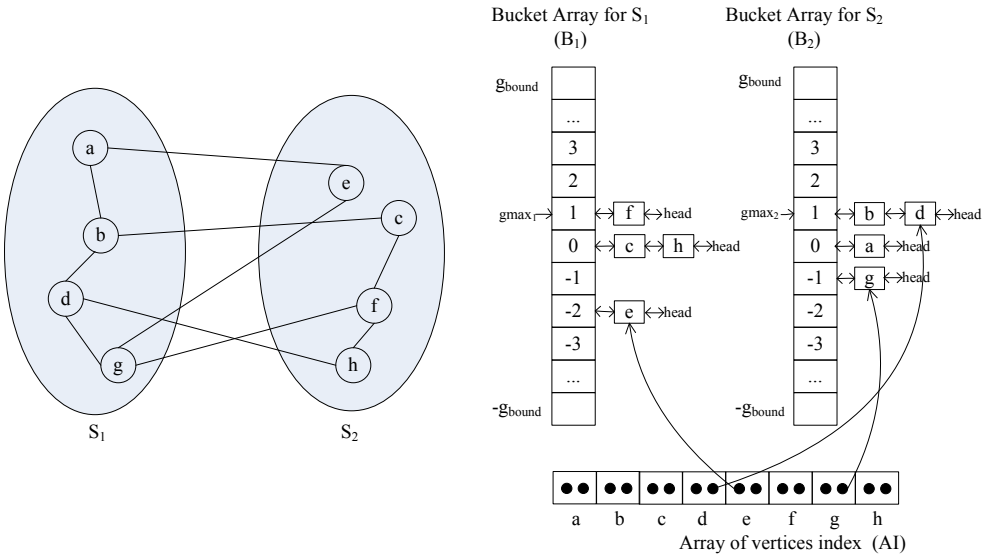


Figure 3.1: An example of bucket structure for max-bisection

adopt a bucket structure which is inspired by the bucket sorting proposed by [Fiduccia and Mattheyses, 1982] for graph partition. With this technique, we always keep the vertices ordered by their move gains in decreasing order, so that the most favorable one can be identified quickly as we explain below.

Our bucket sorting for *1-move* relies on two arrays of buckets, one for each partition subset $S_i \in \{S_1, S_2\}$. In each bucket array i, $i \in \{1, 2\}$, the j^{th} entry stores the vertices with the move gain $\Delta_{v \to S_i}$ currently equaling to j, where the vertices are maintained by a circular double linked list. To ensure a direct access to the vertex in the circular double linked lists, as suggested in [Fiduccia and Mattheyses, 1982], the algorithm also maintains another array for all vertices, where each element points to its corresponding vertex in the circular double linked list. The use of a circular doubly linked list instead of a doubly linked list like in [Fiduccia and Mattheyses, 1982] aims to ease the implementation of our tie-breaking scheme which is needed to select the vertex when several candidates exist (see Section 3.2.5 for more details on this issue).

Fig. 3.1 shows an illustrative example of the bucket structure for max-bisection. The graph (Fig. 3.1, left) has 8 vertices belonging to the two subsets S_1 and S_2 (edge weights are supposed to be equal to 1). The bucket structure for this graph is shown in Fig. 3.1 (right). One observes that the gain of moving vertex c or h to subset S_1 equals 0, then those two vertices are stored in the entry of B_1 with index 0. Notice that vertices c and h are managed as a circular double linked list. The array AI shown at the bottom of Fig. 3.1 manages position indexes for all vertices. For simplicity, we do not show all the links in the figure.

After each *1-move*, the bucket structure is updated by recomputing the move gains (see Formula (3.5)) of the affected vertices which include the moved vertex and its adjacent vertices, and shifting them to appropriate buckets.

3.2.5 Selection of the best vertex with a tie breaking scheme

For each array of buckets, finding the best vertex with maximum move gain is equivalent to finding the first non-empty bucket from the top of the array and then selecting a vertex in its circular double linked list. If there are more than one vertex with maximum move gain in the circular double linked list (see Fig. 3.1), a tie occurs. In particular, we observed experimentally that many ties may occur during the runs of our ITS algorithm, which reveals the importance of a suitable tie-breaking scheme. Three tie-breaking schemes

(random selection, FIFO (first-in-first-out) selection and LIFO (last-in-first-out) selection) are often used to break ties. The work of [Hagen and Kahng, 1997] showed that the LIFO selection of gain buckets was superior to the FIFO selection and random selection. A possible explanation given by the authors was that clustered vertices tend to move together.

In our algorithm, we use the LIFO selection scheme to break ties. However, given that our algorithm employs a tabu mechanism to forbid a vertex to move back to its original subset (see Section 3.2.7), it is inappropriate to insert the forbidden vertices at the head of the list, since doing this will cause useless computations when seeking a proper vertex for a move operation. To adapt the LIFO selection scheme to tabu search, we make the following improvements.

To update the move gain of an impacted vertex after a move, ITS first checks the tabu status of the vertex. If the vertex is in the tabu list, ITS inserts the vertex to the *tail* of the corresponding gain bucket, otherwise, ITS inserts the vertex to the *head* of the gain bucket. To decide the vertex for a *1-move* operation, ITS always selects the first vertex which is not in the tabu list from the head of the gain bucket. This strategy reduces the computing time for checking those forbidden vertices, as we show in Section 3.4.1.

***c-swap*:** For each $c - swap(u, v)$ operation, let $\Delta_{u,v}$ be the move gain of exchanging vertices u and v between the two subsets of the bisection. Then $\Delta_{u,v}$ can be calculated by a combination of the move gains of its two underlying *1-move* (Δ_u and Δ_v) as follows:

$$\Delta_{u,v} = \Delta_u + \Delta_v + 2\omega_{uv} \tag{3.6}$$

According to the definition of the neighborhood N2, N2 only considers the endpoints (vertices) of the edges crossing two subsets S_1 and S_2. Then it is clear that for a given incumbent solution, there are at most $|E|$ candidate *c-swap* moves to evaluate. Seeking directly the move with the maximum move gain among all these possible moves would be too computationally expensive. In order to mitigate this problem, we maintain another bucket structure for *c-swap* moves to accelerate the move evaluation process. The bucket structure for *c-swap* is similar to that for *1-move*. This is achieved by keeping an array of buckets and in each bucket, the i^{th} entry stores the edge $\{u, v\}$ with the move gain $\Delta_{u,v}$ currently being equal to i, where the edges are maintained by a circular double linked list. To ensure a direct access to the edges in the circular double linked lists, as described above, the algorithm also maintains another array for all edges, where each entry points to its corresponding edge in the circular double linked lists.

Similarly, after each move, the bucket structure is updated by recomputing the move gains (see Formula (3.5)) of the affected vertices (i.e., each swapped vertex and its adjacent vertices), and shifting them to appropriate buckets.

Complexity: The complexity of each move comes from searching for the vertex or a pair of vertices with maximum move gain, recomputing the move gain for the affected vertices and updating the bucket structure. The vertex with maximum move gain can be simply obtained in constant time $(O(1))$. Recomputing move gain is in linear time relative to the number of affected vertices $(O(n))$. The time of updating the bucket structure is also only related to the number of affected vertices bounded by $(O(d_{max}))$ where d_{max} is the maximum degree of the graph.

3.2.6 Descent local search phase to locate local optima

The descent local search (DLS) phase is used to obtain a local optimum from a given starting solution (see Algorithm 2, lines 10 - 19). For this, DLS employs the *1-move* operator defined in Section 3.2.3 to iteratively improve the incumbent solution until a local optimum is reached. At each iteration of the descent procedure, a best *1-move* (i.e., with the maximum move gain) is selected by using the bucket structures explained in Section 3.2.4 and displaced from its current subset to the other subset. As explained in Section 3.2.3, to maintain the balance of the two subsets of the bisection, DLS always jointly performs two consecutive *1-move* operations.

First, DLS selects a vertex u with the largest move gain (i.e., Δ_v is maximum), displaces u from its subset (say S_1) to the other subset and updates the bucket structure of move gains according to the technique described in Section 3.2.4. Then, DLS selects another vertex v in the other subset (say S_2) with the largest move gain, transfers v from S_2 to S_1 and updates the bucket structure again. In the case where two or more vertices have the same largest move gain, the LIFO tie-breaking strategy described in Section 3.2.4 is used to choose the applied vertex.

After each combined application of two consecutive *1-move* operations, if the new objective value is better (larger) than the objective value of the former incumbent solution, the current descent iteration is achieved and DLS continues its descent process with the newly attained solution as its new current solution. Otherwise, DLS stops after rolling back to the previous solution prior to the last two-consecutive *1-move* operations (see Algorithm 2, lines 17 - 20). This solution corresponds to a local optimum with respect to the $N1$ neighborhood and serves as the input solution of the diversifying improvement search phase which is explained in the next section.

3.2.7 Diversifying improvement phase to discover promising region

The descent local phase described in Section 3.2.6 alone cannot go beyond the first local optimum it encounters. The diversifying improvement search phase, which is based on the tabu search method [Glover and Laguna, 1999], 1) to jump out of this local optimum and 2) to intensify the search around this local optimum with the purpose of discovering solutions better than the input local optimum.

The diversifying improvement search procedure jointly uses the *1-move* and *c-swap* operators defined in Section 3.2.3. To apply these two operators, we employ a probabilistic combination technique which extends the existing combination schemes described in [Lü *et al.*, 2011b]. The application of *1-move* or *c-swap* is determined probabilistically at each iteration: with probability ρ (a parameter), *c-swap* is applied; with probability $1 - \rho$, *1-move* is applied (see Algorithm 2, lines 28 - 38).

When *1-move* is selected, the algorithm performs the combined *1-move* operations in a way similar to that described in Section 3.2.6 except that here a tabu list H is considered [Glover and Laguna, 1999]. The tabu list is a memory which keeps track of displaced vertices to prevent them from being moved back to their initial subsets. Precisely, the algorithm first selects an eligible vertex (see below) with the maximum move gain and transfers it from its current subset (say S_1) to the other subset, then it updates the bucket structure of move gains according to the technique described in Section 3.2.4. After that, it selects another eligible vertex in the other subset (say S_2) with the best move gain and moves it from S_2 to S_1. The bucket structure is updated to actualize the impacted move gains accordingly.

After the transfer of a vertex v, the vertex is added to the tabu list H and forbidden to join again its

original subset for the next H_v iterations. H_v (called the tabu tenure) is determined dynamically as follows:

$$H_v = 3 + rand(|V|/10) \tag{3.7}$$

where $rand(k)$ is a random number from 0 to k.

Note that a move leading to a solution better than all solutions ever found is always performed even if the underlying vertex is forbidden by the tabu list (This is called the aspiration criterion in the terminology of tabu search). A vertex is said to be *eligible* if it is not forbidden by the tabu list or if the aspiration criterion is satisfied.

Similarly, when *c-swap* is selected, two vertices $v_1 \in S_1$ and $v_2 \in S_2$ with maximum move gain are selected subject to $\{v_1, v_2\} \in E$. Another tabu list H^c is maintained for *c-swap*. After each *c-swap* move, the edge $\{v_1, v_2\}$ is added to the tabu list H^c and it is forbidden to swap v_1 and v_2 back to their original subsets for the next H^c iterations, which, like for the *1-move*, is dynamically determined by formula (3.7). The same aspiration criterion as that used by *1-move* is also applied. After each *c-swap* move, the bucket structure is updated to actualize the impacted move gains. Note that when multiple best *c-swap* moves are available, the LIFO selection strategy is used to choose the applied *c-swap* move (see Section 3.2.4).

The tabu search procedure iteratively applies *1-move* and *c-swap* to improve the incumbent solution. If the best solution found so far (f_{best}) cannot be improved during a maximum number ω of consecutive iterations, the search is judged to be trapped in a deep local optimum. In this case, the perturbation phase (Section 3.2.8) is invoked to move the search to a distant region.

3.2.8 Perturbation phase for strong diversification

The diversifying improvement phase allows the search to escape some local optima. However, the algorithm may still get stuck in a non-promising search zone. This is the case when the best-found solution f_{best} cannot be improved after ω consecutive iterations. To help the search to escape from such deep local optima, we apply a simple perturbation mechanism to the current solution to diversify the search. The perturbation swaps a number of pairs of vertices in the following way. For each swap, we randomly choose one vertex v from S_1 and another vertex u from S_2, and then swap v and u. This process is repeated γ times where γ is a parameter which indicates the strength of the perturbation. After the perturbation phase, the search returns to the descent-based improvement phase with the perturbed solution as its new starting solution.

3.3 Experimental results and comparisons

3.3.1 Benchmark instances

To assess the performance of the proposed ITS approach, we carried out intensive computational experiments on the set of 71 well-known benchmark graphs in the literature. These graphs have 800 to 20000 vertices and an edge density from 0.02% to 6%. They were generated by a machine-independent graph generator including toroidal, planar and random weighted graphs. These instances are available from: http://www.stanford.edu/yyye/yyye/Gset or from the authors of this paper. These

well-known benchmark graphs were frequently used to evaluate the performances of max-bisection and max-cut algorithms [Benlic and Hao, 2013a; Festa *et al.*, 2002; Lin and Zhu, 2014; Shylo *et al.*, 2012; Shylo *et al.*, 2015; Wang *et al.*, 2013; Wu and Hao, 2012; Wu and Hao, 2013; Xu *et al.*, 2011].

3.3.2 Experimental protocol

Our ITS algorithm was programmed in C++ and compiled with GNU g++ (optimization flag "-O2"). Our computer is equipped with a Xeon E5440 (2.83GHz, 2GB RAM). When running the DIMACS machine benchmark [1], our machine requires 0.43, 2.62 and 9.85 CPU time in seconds respectively for graphs r300.5, r400.5, and r500.5 compiled with g++ -O2.

3.3.3 Parameters

The proposed algorithm requires three parameters: maximum allowed number ω of non-improvement iterations, probability ρ for move operator selection, and number γ of perturbation moves. To achieve a reasonable tuning of the parameters, we adopted the irace package [López-Ibánez *et al.*, 2011] which implements the Iterated F-race (IFR) method [Bartz-Beielstein *et al.*, 2010] and allows an automatic parameter configuration. We used the following parameter value ranges for this tuning: $\omega = \{1500, 2500, 3500, 4500, 5500\}$, $\rho = [0.1, 0.5]$, $\gamma = \{50, 200, 400, 600\}$. We performed the parameter tuning experiment on a selection of 5 representative and challenging instances from the 71 benchmark graphs: G22, G23, G37, G55, G62. This calibration experiment led to the following parameter values: $(\omega = 3500, \rho = 0.3, \gamma = 200)$, which were used in all our experiments throughout the paper.

Considering the stochastic nature of our ITS algorithm, each of the 71 benchmark instance was independently solved 20 times with different random seeds. For the purpose of fair comparisons reported in Sections 3.3.4 and 3.3.5, we followed the reference algorithms and used a timeout limit as the stopping criterion of our ITS algorithm. The timeout limit was set to be 30 minutes for graphs with $|V| < 5000$ and 120 minutes for graphs with $|V| \geq 5000$.

To fully evaluate the performance of the proposed algorithm, we performed a comparison with the three most recent and best performing state-of-the-art max-bisection algorithms [Lin and Zhu, 2014; Wu and Hao, 2013; Xu *et al.*, 2011]. The current best results of the literature were reported in [Lin and Zhu, 2014; Wu and Hao, 2013] recently in 2013 and 2014.

3.3.4 Comparison with the current best-known solutions

Table 3.1 shows the computational results of our ITS algorithm on the 71 benchmark graphs [2] in comparison with the previous best-known results f_{pre}, which are taken from the two most recent studies [Lin and Zhu, 2014; Wu and Hao, 2013]. The first two columns of the table indicate the name and the number of vertices of the graphs. Columns 4 to 7 present the computational statistics attained by our algorithm, where f_{best} and f_{avg} show the best objective value and the average objective value over 20 runs, std gives the standard deviation and $time(s)$ indicates the average CPU time in seconds to reach f_{best}.

1. dfmax:ftp://dimacs.rutgers.edu/pub/dsj/clique/
2. Our best results are available at: http://www.info.univ-angers.fr/pub/hao/maxbisection/ITSresults.zip.

From Table 3.1, we observe that our ITS algorithm, evaluated under the same cutoff time limit as the best performing reference algorithm MA-WH, is able to improve the previous best-known results for 10 large benchmark graphs (indicated in bold) and match the best-known results for all the other graphs. This performance is remarkable given that the current best results were reported recently. Moreover, the results of the proposed algorithm show small standard deviations across different runs and different graphs, indicating a good robustness of the algorithm.

Table 3.1: Computational results of the proposed ITS algorithm on the set of 71 benchmark graphs in comparison with the current best results ever reported in the literature.

Instance	$\|V\|$	f_{pre}	f_{best}	f_{avg}	std	$time(s)$
G1	800	11624	11624	11624.00	0.00	1.50
G2	800	11617	11617	11617.00	0.00	3.24
G3	800	11621	11621	11621.00	0.00	1.02
G4	800	11646	11646	11646.00	0.00	1.77
G5	800	11631	11631	11631.00	0.00	0.76
G6	800	2177	2177	2177.00	0.00	1.50
G7	800	2002	2002	2002.00	0.00	0.53
G8	800	2004	2004	2004.00	0.00	3.50
G9	800	2052	2052	2052.00	0.00	1.88
G10	800	1998	1998	1998.00	0.00	4.99
G11	800	564	564	564.00	0.00	0.12
G12	800	556	556	556.00	0.00	0.56
G13	800	582	582	582.00	0.00	4.52
G14	800	3062	3062	3062.00	0.00	90.68
G15	800	3050	3050	3050.00	0.00	55.84
G16	800	3052	3052	3052.00	0.00	32.82
G17	800	3047	3047	3047.00	0.00	200.67
G18	800	992	992	992.00	0.00	14.50
G19	800	905	905	905.00	0.00	3.51
G20	800	941	941	941.00	0.00	1.52
G21	800	930	930	930.00	0.00	50.41
G22	2000	13359	13359	13355.52	5.47	432.10
G23	2000	13344	13344	13342.10	2.09	168.24
G24	2000	13336	13336	13335.02	1.67	300.75
G25	2000	13340	13340	13338.20	1.98	149.21
G26	2000	13328	13328	13327.41	1.54	433.68
G27	2000	3341	3341	3340.65	1.75	140.64
G28	2000	3298	3298	3298.00	0.00	198.23
G29	2000	3403	3403	3403.00	0.00	3.26
G30	2000	3412	3412	3412.00	0.00	54.22
G31	2000	3309	3309	3309.00	0.00	242.19
G32	2000	1410	1410	1410.00	0.00	425.70
G33	2000	1382	1382	1382.00	0.00	485.83
G34	2000	1384	1384	1384.00	0.00	189.27
G35	2000	7686	7686	7684.10	2.04	448.35
G36	2000	7678	7678	7676.45	2.16	634.11
G37	2000	7689	7689	7687.74	2.09	627.86
G38	2000	7688	7688	7686.56	3.04	688.32
G39	2000	2408	2408	2406.87	2.56	242.60
G40	2000	2400	2400	2398.82	3.02	354.50
G41	2000	2405	2405	2404.21	0.99	82.55
G42	2000	2481	2481	2476.86	5.85	286.18
G43	1000	6659	6659	6659.00	0.00	5.25
G44	1000	6650	6650	6650.00	0.00	2.09
G45	1000	6654	6654	6654.00	0.00	3.99
G46	1000	6649	6649	6649.00	0.00	30.12
G47	1000	6657	6657	6657.00	0.00	4.88
G48	3000	6000	6000	6000.00	0.00	0.97
G49	3000	6000	6000	6000.00	0.00	1.57
G50	3000	5880	5880	5880.00	0.00	50.64
G51	1000	3847	3847	3847.00	0.00	101.43
G52	1000	3851	3851	3851.00	0.00	98.43
G53	1000	3850	3850	3850.00	0.00	109.50
G54	1000	3851	3851	3851.00	0.00	177.89
G55	5000	10299	10299	10290.83	4.54	2596.84
G56	5000	4016	4016	4013.13	2.28	1926.45
G57	5000	3488	**3490**	3487.76	1.88	610.16
G58	5000	19276	19276	19265.90	3.18	5102.34
G59	5000	6085	6085	6074.34	2.35	4902.13
G60	7000	14186	**14187**	14176.54	4.01	5678.63
G61	7000	5796	5796	5780.18	5.08	4072.54
G62	7000	4866	4866	4860.12	2.69	1472.10
G63	7000	26754	**26988**	26985.32	1.18	2256.66
G64	7000	8731	**8737**	8712.10	6.28	6032.55
G65	8000	5556	5556	5550.87	2.42	2350.98
G66	9000	6352	**6356**	6352.01	1.93	1323.15
G67	10000	6934	**6938**	6935.46	1.34	1023.40

Table 3.1 – continued from previous page

Instance	$\|V\|$	f_{pre}	f_{best}	f_{avg}	std	$time(s)$
G70	10000	9580	**9581**	9576.32	0.98	1154.32
G72	10000	6990	**6994**	6992.50	0.84	1201.97
G77	14000	9900	**9918**	9915.14	1.02	2013.44
G81	20000	13978	**14030**	14025.45	1.36	1953.23

3.3.5 Comparison with state-of-the-art max-bisection algorithms

In this section, we further evaluate the performance of the proposed algorithm by comparing it with three best performing algorithms of the literature that achieved state-of-art performances:

1. A Lagrangian net algorithm (LNA) [Xu *et al.*, 2011] integrating the discrete Hopfield neural network and the penalty function method (relaxing the bisection constraints in the objective function). The reported results of LNA were obtained on a PC with a 2.36GHz CPU and 1.96GB RAM. The algorithm was programmed in Matlab 7.4.

2. A memetic algorithm for the max-bisection problem (MA-WH) [Wu and Hao, 2013] integrating a grouping crossover operator and a tabu search procedure. The results reported in the paper were obtained on a PC with a 2.83GHz Intel Xeon E5440 CPU and 2.0GB RAM (the same platform was used in our study). The program was coded in C.

3. Another memetic algorithm for the max-bisection problem (MA-LZ) [Lin and Zhu, 2014] integrating a grouping crossover operator and an improved FM [Fiduccia and Mattheyses, 1982] based local search procedure. The reported results of MA-LZ were obtained on a PC with a 2.11GHz AMD CPU and 1.0GB RAM. The algorithm was programmed in C++.

Both the MA-WH algorithm and our ITS algorithm used the same computing platform while LNA and MA-LZ were run on different computing platforms. In order to make a fair comparison of the computing time, we measured the differences among the three computing platforms according to the Standard Performance Evaluation Cooperation (SPEC) (`www.spec.org`), which indicated that the computers used by LNA and MA-LZ are respectively 1.2 and 1.4 times slower than the computer we used for our experiments.

Table 3.2 shows the comparative results of our ITS algorithm on the whole set of 71 benchmark graphs with respect to the three reference algorithms LNA, MA-WH and MA-LZ. For each reference algorithm, we report the best objective values (f_{best}), the consumed CPU times ($time$) in seconds to attain the best objective values (f_{best}), and the differences (gap) between each reference algorithm and our ITS algorithm. As mentioned above, to harmonize the computing times, we divided the times of LNA and MA-LZ by the factor provided by SPEC, i.e., 1.2 and 1.4 respectively. The last two columns reporting the results of our ITS algorithm are extracted from Table 3.1. The entries marked as "-" in the table indicate that the results are not available in the literature.

From Table 3.2, we first observe that our proposed ITS algorithm performs the best in terms of the best objective values among all the compared algorithms. Specifically, ITS dominates LNA for all the tested instances. MA-LZ matches the results of ITS for 10 instances and obtains inferior results than ITS for all the other reported instances. ITS reaches larger f_{best} objective values than MA-WH for 10 instances and equal objective values for the other 61 instances. In terms of the computational time, it is not obvious to make a fair comparison given that the competing algorithms lead to solutions of quite different quality. This is particularly the case for LNA and MA-LZ which performs the worst and the second worst in terms of solution quality. Compared to the most powerful existing MA-WH algorithm, we observe that ITS has a

similar computing performance to attain solutions of equal or better quality for large instances. Moreover, in Table 3.3 we show the time information of ITS to attain solutions of the same quality as MA-WH for the 12 largest instances with 7000 to 20000 vertices. The table discloses that our ITS algorithm is much faster than MA-WH for these large instances (except G61). For 6 instances, ITS is even 10 to 20 times faster. To conclude, the comparisons with the current state-of-the-art algorithms demonstrate that our proposed ITS algorithm is highly effective in terms of both solution quality and computing time, in particular on large instances for which ITS scales well.

Table 3.2: Comparative results of ITS with three state of the art and best performing algorithms: LNA, MA-LZ and MA-WH.

Instance	$\|V\|$	LNA			MA-LZ			MA-WH			ITS	
		f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$
G1	800	11490	22.22	-134	11624	13.38	0	11624	2.40	0	11624	1.50
G2	800	11505	21.95	-112	11617	11.66	0	11617	5.20	0	11617	3.24
G3	800	11511	21.95	-110	11621	14.77	0	11621	1.32	0	11621	1.02
G4	800	11554	22.04	-92	11641	16.29	-5	11646	1.77	0	11646	1.77
G5	800	11521	21.80	-110	11630	14.30	-1	11631	0.88	0	11631	0.76
G6	800	2037	22.08	-140	2177	10.35	0	2177	1.16	0	2177	1.50
G7	800	1889	22.00	-113	2000	14.72	-2	2002	0.82	0	2002	0.53
G8	800	1873	21.94	-131	2001	16.66	-3	2004	4.26	0	2004	3.50
G9	800	1907	21.86	-145	2046	11.94	-6	2052	1.19	0	2052	1.88
G10	800	1875	21.96	-123	1998	14.99	0	1998	5.59	0	1998	4.99
G11	800	560	3.18	-4	564	11.67	0	564	12.10	0	564	0.12
G12	800	546	3.17	-10	554	11.29	-2	556	11.54	0	556	0.56
G13	800	572	3.17	-10	578	11.12	-4	582	32.52	0	582	4.52
G14	800	3023	7.02	-39	3058	17.76	-4	3062	799.00	0	3062	90.68
G15	800	2996	7.01	-54	3049	15.20	-1	3050	692.96	0	3050	55.84
G16	800	2994	7.02	-58	3047	15.83	-5	3052	82.82	0	3052	32.82
G17	800	2997	6.99	-50	3043	17.16	-4	3047	778.67	0	3047	200.67
G18	800	909	7.03	-83	991	10.82	-1	992	16.36	0	992	14.50
G19	800	823	7.00	-82	905	8.59	0	905	40.31	0	905	3.51
G20	800	865	6.98	-76	941	6.09	0	941	2.48	0	941	1.52
G21	800	849	6.98	-81	930	9.97	0	930	34.71	0	930	50.41
G22	2000	13105	57.48	-254	13346	25.97	-13	13359	303.20	0	13359	432.10
G23	2000	13120	57.36	-224	13319	27.67	-25	13344	132.13	0	13344	168.24
G24	2000	13115	57.34	-221	13322	25.87	-14	13336	102.75	0	13336	300.75
G25	2000	13125	57.41	-215	13314	26.36	-26	13340	308.51	0	13340	149.21
G26	2000	13160	57.25	-168	13300	27.64	-28	13328	366.09	0	13328	433.68
G27	2000	3109	57.16	-232	3317	26.74	-24	3341	109.49	0	3341	140.64
G28	2000	3063	58.13	-235	3289	26.96	-9	3298	217.84	0	3298	198.23
G29	2000	3179	58.06	-224	3376	26.54	-27	3403	1.36	0	3403	3.26
G30	2000	3139	58.18	-273	3397	26.11	-15	3412	44.82	0	3412	54.22
G31	2000	3092	58.13	-217	3296	25.43	-13	3309	263.21	0	3309	242.19
G32	2000	1382	16.88	-28	1410	61.07	0	1410	887.50	0	1410	425.70
G33	2000	1344	17.01	-38	1378	59.80	-4	1382	856.80	0	1382	485.83
G34	2000	1350	16.88	-34	1382	52.09	-2	1384	536.12	0	1384	189.27
G35	2000	7548	39.22	-138	7659	34.26	-27	7686	1312.42	0	7686	448.35
G36	2000	7530	39.08	-148	7655	33.79	-23	7678	1259.10	0	7678	634.11
G37	2000	7541	39.21	-148	7669	33.86	-20	7689	1543.36	0	7689	627.86
G38	2000	7537	39.23	-151	7662	34.63	-26	7688	922.66	0	7688	688.32
G39	2000	2255	40.11	-153	2382	23.11	-26	2408	976.95	0	2408	242.60
G40	2000	2189	40.00	-211	2386	24.82	-14	2400	1198.28	0	2400	354.50
G41	2000	2234	40.03	-171	2383	25.78	-22	2405	546.57	0	2405	82.55
G42	2000	2290	40.11	-191	2456	26.74	-25	2481	1513.96	0	2481	286.18
G43	1000	6580	15.34	-79	-	-	-	6659	1.25	0	6659	5.25
G44	1000	6548	15.33	-102	-	-	-	6650	1.18	0	6650	2.09
G45	1000	6513	15.33	-141	-	-	-	6654	4.23	0	6654	3.99
G46	1000	6538	15.33	-111	-	-	-	6649	10.48	0	6649	30.12
G47	1000	6529	15.34	-128	-	-	-	6657	5.97	0	6657	4.88
G48	3000	-	-	-	-	-	-	6000	1.42	0	6000	0.97
G49	3000	-	-	-	-	-	-	6000	1.28	0	6000	1.57
G50	3000	-	-	-	-	-	-	5880	33.89	0	5880	50.64
G51	1000	3773	10.58	-74	-	-	-	3847	292.60	0	3847	101.43
G52	1000	3788	10.61	-63	-	-	-	3851	814.96	0	3851	98.43
G53	1000	3784	10.60	-66	-	-	-	3850	516.28	0	3850	109.50
G54	1000	3789	10.63	-62	-	-	-	3851	551.51	0	3851	177.89
G55	5000	-	-	-	-	-	-	10299	2396.84	0	10299	2596.84
G56	5000	-	-	-	-	-	-	4016	1886.98	0	4016	1926.45
G57	5000	-	-	-	-	-	-	3488	4883.34	-2	3490	610.16
G58	5000	18931	268.71	-345	19213	120.67	-63	19276	4276.67	0	19276	5102.34
G59	5000	5578	260.91	-507	5978	88.69	-107	6085	4446.16	0	6085	4902.13
G60	7000	-	-	-	-	-	-	14186	5508.45	0	14187	5678.63
G61	7000	-	-	-	-	-	-	5796	3755.71	0	5796	4072.54
G62	7000	-	-	-	-	-	-	4866	4652.00	0	4866	1472.10
G63	7000	-	-	-	-	-	-	26754	5670.30	-234	26988	2256.66
G64	7000	-	-	-	-	-	-	8731	5793.56	-6	8737	6032.55

Table 3.2 – continued from previous page

Instance	$\|V\|$	LNA			MA-LZ			MA-WH			ITS	
		f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$	gap	f_{best}	$time(s)$
G65	8000	5418	290.72	-138	5534	463.44	-22	5556	5385.86	0	5556	2350.98
G66	9000	6194	391.03	-162	6324	850.69	-32	6352	6267.15	-4	6356	1323.15
G67	10000	6782	512.62	-156	6912	797.09	-26	6934	6203.44	-4	6938	1023.40
G70	10000	-	-	-	-	-	-	9580	7032.70	-1	9581	1154.32
G72	10000	-	-	-	-	-	-	6990	7046.03	-4	6994	1201.97
G77	14000	-	-	-	-	-	-	9900	6752.26	-18	9918	2013.44
G81	20000	-	-	-	-	-	-	13978	7023.49	-52	14030	1953.23

Table 3.3: ITS needs much less time to attain the best objectives of the current best performing MA-WH algorithm on the 12 largest instances with 7000 to 20000 vertices.

Instance	MA-WH		ITS
	f_{best}	$time(s)$	$time(s)$
G60	14186	5508.45	5678.63
G61	5796	3755.71	4072.54
G62	4866	4652.00	1472.1
G63	26754	5670.30	238.16
G64	8731	5793.56	5532.55
G65	5556	5385.86	2350.98
G66	6352	6267.15	930.15
G67	6934	6203.44	1223.4
G70	9580	7032.70	1154.32
G72	6990	7046.03	970.92
G77	9900	6752.26	530.71
G81	13978	7023.49	486.70237

3.3.6 Comparison with a recent state-of-the-art exact algorithm for the minimum bisection problem

A bisection of an unweighted graph $G = (V, E)$ ($|V|$ even) is a pair of disjoint subsets $S_1 \subset V$, $S_2 \subset V$ of equal cardinality. The cost of a bisection is the number of cutting edges $\{u, v\} \in E$ such that $u \in S_1$ and $v \in S_2$. The *minimum bisection* problem (or graph bisection) is to determine a bisection of minimum cost. The minimum bisection problem can be considered as a special case of the maximum bisection problem studied in this paper. In fact, for the given unweighted graph $G = (V, E)$, create a weighted graph where each edge has a weight value of -1 (call this weighted graph G'), then the objective value of the maximum bisection problem of G' multiplied by -1 corresponds to the objective value of the minimum bisection problem of G. Consequently, to solve the minimum bisection, we can run our ITS algorithm on the graph where each edge is given the weight -1 and return the resulting objective value multiplied by -1.

To test the performance of our ITS algorithm on the minimum bisection problem, we carried out a comparative study with a very recent and powerful exact algorithm specifically designed for the minimum bisection problem [Delling *et al.*, 2015]. This study was based on 3 sets of benchmarks with a total of 20 graphs used in the reference paper, including cgmesh graphs (meshes representing various objects), steinlib graphs (sparse benchmark instances for the Steiner problem in graphs) and walshaw graphs (mostly finite-element meshes). Notice that we did not test all the graphs used in [Delling *et al.*, 2015] given that the current implementation of the ITS algorithm does not allow us to solve very large graphs with more than 70,000 vertices.

We carry out 10 independent runs of our ITS algorithm for each tested instance within a cutoff time limit of 3600 seconds and terminate each run once the best known result is found. The parameter settings for the instances in this experiment are the same as used for the maximum bisection benchmark instances. Without bothering to show a detailed tabulation of computational results, we summarize the main findings

obtained from this experiment as follows. Our ITS algorithm is able to attain the optimal solutions for the walshaw and cgmesh graphs. In particular, for the cgmesh graphs, ITS reaches the optimal solutions with a computing time ranging from 5 to 10 times shorter than the time needed by the exact algorithm to complete its search. On the other hand, ITS fails to reach the optimal solutions for large steinlib graphs. An interesting observation is that ITS works well for graphs with a large minimum bisection value while the exact algorithm performs well for graphs with a small minimum bisection value (the latter is confirmed in [Delling *et al.*, 2015]). In this sense, we can consider that both algorithms complement each other, suitable to solve graphs of different characteristics. The inferiority of ITS for solving graphs with small minimum bisection values is partly attributed to the ineffectiveness of the *c-swap* operator for this type of special graphs. Essentially, the *c-swap* operator only concentrates on swapping cutting edges, which proved to be effective for the graphs used to benchmark max-bisection algorithms, but becomes inefficient when the cutting edges are very limited as it is the case for the steinlib graphs.

3.4 Discussion

In this section, we investigate the roles of the Last In First Out (LIFO) tie breaking strategy based on bucket sorting and the combined neighborhood in the proposed ITS algorithm. The experiments of this section were based on a selection of 17 challenging instances while the tested ITS variants used the same stopping conditions as in the previous experiments.

3.4.1 Impact of the bucket-sorting based tie breaking strategies

The adopted bucket sorting is a crucial data structure to the effectiveness of the proposed algorithm, in particular to the LIFO tie breaking strategy. Recall that each bucket in the bucket array generally includes multiple vertices (organized into a circular doubly link list) from which moving any vertex will lead to the same objective gain. Apparently, no difference occurs among vertices in the same bucket. However, we assume that potential connections among vertices exist and the order of vertices being inserted into a bucket is worthy of a careful consideration. Based on this assumption, we proposed an improved LIFO insertion strategy (see Section 3.2.4), where a vertex is inserted at the head of the circular doubly link list whenever its move gain is changed (i.e., its inserted position in the bucket array is changed accordingly), to ensure that this vertex will be first selected when a tie break happens. The reason lies in the fact that if the move gain of a vertex u is changed because of moving a vertex v, then u has a higher opportunity to be moved during the following iterations. An exception is to insert tabu vertices at the tail of the circular doubly link lists in order to penalize the recently moved vertices.

To verify the role of the bucket sorting structure to the performance of the ITS algorithm, we tested an ITS variant which disables the bucket sorting structure and only keeps a vector to record objective gains resulted from performing each *1-move*. In this case, the identification of a vertex with the maximum objective gain in the ITS variant has to scan the whole vector instead of looking at the top of the bucket array. When the maximum objective gain is held by more than one vertex, ties are broken randomly. Table 3.4 (upper part) compares the standard ITS algorithm (with the bucket sorting structure and the LIFO tie breaking strategy, named ITS_{LIFO}) and the ITS variant which excludes the bucket sorting structure (named $\text{ITS}_{No-bucket}$). From the results, we observe that removing the bucket sorting structure degrades considerably the performance of the ITS algorithm both in terms of the best and average solutions, which is confirmed by a small p-value of 3.738e-05 from the Friedman test for both cases. Moreover, compared to ITS_{LIFO}, $\text{ITS}_{No-bucket}$ generally requires more computing time to reach its best results (which are worse

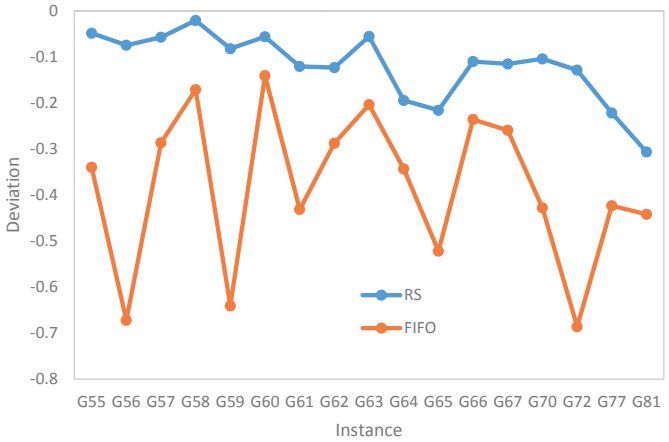
than those of ITS$_{LIFO}$). In conclusion, the experiment demonstrates the usefulness of the bucket sorting structure technique in the proposed ITS algorithm.

To further verify the adopted LIFO tie breaking strategy, we compared LIFO with the Random Strategy (Random) and the First In First Out strategy (FIFO). The random strategy scans vertices of the same bucket according to a random order, no matter if a new vertex is inserted at the head or the tail of a circular doubly link list. The FIFO strategy uses a queue structure, with the vertices in a bucket being scanned from the head to the tail like the LIFO strategy but with any vertex being inserted at the tail of the circular doubly link list. For this experiment, we kept all the other components of the proposed ITS algorithm unchanged except the tie breaking strategy.

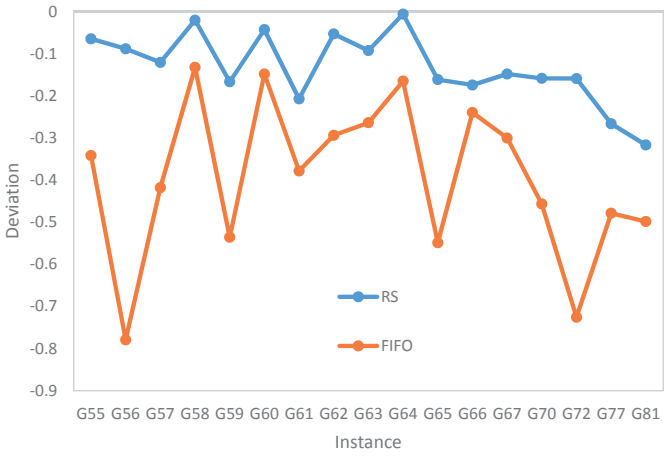
Table 3.4 (lower part) reports the best objective value f_{best} and average objective value f_{avg} over 20 runs as well as the average time $time$ to reach f_{best}. From this table, we observe that the LIFO tie breaking strategy dominates the Random and FIFO strategies both in terms of solution quality and computing time. In order to clearly observe the superiority of the LIFO strategy, we plot in Figure 3.2(a) and 3.2(b) the deviation of the best and average objective values obtained by Random and FIFO from that of LIFO for each tested instance. Notice that if the absolute value of the deviation is smaller, then the corresponding objective value is better. From Figures 3.2(a) and 3.2(b), we clearly observe that deviation values are all negative, meaning both Random and FIFO are inferior to LIFO in terms of the best and average objective values. In conclusion, this experiment demonstrates the interest of the adopted LIFO tie breaking strategy.

Table 3.4: Assessment of the bucket sorting structure and comparisons among the different tie-breaking strategies

Instance	ITS$_{LIFO}$			ITS$_{No-bucket}$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G55	10299	10290.83	2596.84	10296	10285.18	3755.78
G56	4016	4013.13	1926.45	4012	4007.97	4237.44
G57	3490	3487.76	610.16	3488	3478.17	5451.93
G58	19276	19265.9	5102.34	19272	19264.14	4759.58
G59	6085	6074.34	4902.13	6078	6063.64	4192.53
G60	14186	14176.54	5678.63	14170	14162.91	6012.47
G61	5796	5780.18	4072.54	5786	5770.43	3699.49
G62	4866	4860.12	1472.1	4860	4848.72	4275.62
G63	26988	26985.32	2256.66	26976	26967.02	5071.38
G64	8737	8712.1	6032.55	8725	8707.16	3975.71
G65	5556	5550.87	2350.98	5542	5535.63	4217.56
G66	6356	6352.01	1323.15	6345	6334.45	5274.54
G67	6938	6935.46	1023.4	6927	6920.46	4057.85
G70	9581	9576.32	1154.32	9564	9540.34	4538.75
G72	6994	6992.5	1201.97	6980	6975.2	5638.47
G77	9918	9915.14	2013.44	9890	9880.14	6972.68
G81	14030	14025.45	1953.23	13978	13950.45	7001.35
Instance	ITS$_{FIFO}$			ITS$_{Random}$		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G55	10264	10255.62	5389.34	10294	10284.13	5560.1
G56	3989	3981.85	6883.49	4013	4009.57	5895.28
G57	3480	3473.17	5573.11	3488	3483.54	4560.34
G58	19243	19240.30	5991.60	19272	19261.88	6832.53
G59	6046	6041.78	7137.13	6080	6064.19	6102.8
G60	14166	14155.48	5365.37	14178	14170.41	6016.74
G61	5771	5758.29	5966.63	5789	5768.18	5319.93
G62	4852	4845.82	6084.48	4860	4857.52	6087.27
G63	26933	26914.02	5274.82	26973	26960.2	5752.03
G64	8707	8697.68	6462.01	8720	8711.54	5143.31
G65	5527	5520.40	6587.86	5544	5541.89	6136.65
G66	6341	6336.77	6728.68	6349	6340.91	7056.77
G67	6920	6914.62	5612.06	6930	6925.16	6835.17
G70	9540	9532.55	6177.37	9571	9561.09	6326.62
G72	6946	6941.74	6567.88	6985	6981.35	6964.13
G77	9876	9867.64	7139.18	9896	9888.72	6587.06
G81	13968	13955.49	5581.10	13987	13980.98	7019.52



(a) The best objective deviation of the tie breaking strategies Random and FIFO from LIFO



(b) The average objective deviation of the tie breaking strategies Random and FIFO from LIFO

Figure 3.2: Analysis of the tie breaking strategies

3.4.2 Impact of the combined use of *1-move* and *c-swap* operators

Our proposed ITS algorithm employs both the *1-move* and *c-swap* operators, which are combined in a probabilistic way as described in Section 3.2.3. To verify the effectiveness of the combined use of these operators, we developed two algorithmic variants. The first ITS variant disables *c-swap* and uses *1-move* (i.e., by removing lines 27-31 in Algorithm 1). The second ITS variant just replaces *c-swap* by the conventional swap operator (denoted as *s-swap*, see Section 3.2.3). In both variants, we keep the other ITS components unchanged. We run ITS (denoted by *1-move* + *c-swap*) as well as these two variants (denoted by *1-move* and *1-move* + *s-swap*) under the same experimental conditions as before to solve the 17 selected instances and report the results in terms of f_{best}, f_{avg} and $time$ in Table 3.5.

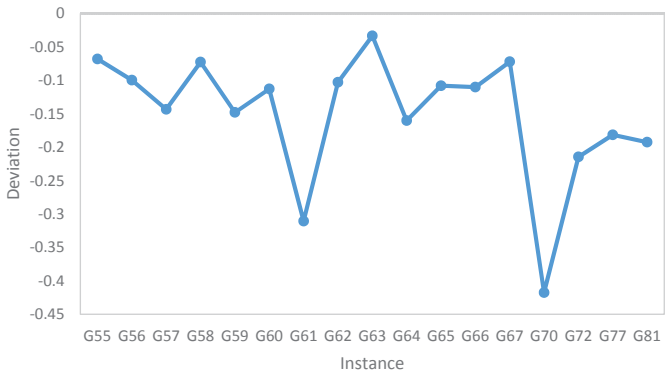
From Table 3.5, we observe that the ITS algorithm with *1-move* + *c-swap* obtains better f_{best} and f_{avg} values for each tested instance. In addition, the joint use of *1-move* and *c-swap* takes the shortest time while obtaining results of much better quality. We also observe that the variant using *1-move* alone performs better than the variant jointly using *1-move* + *s-swap*. This indicates that contrary to our fast *c-swap* operator, the expensive *s-swap* operator is not suitable here due to the high time complexity needed to explore the induced huge neighborhood of quadratic size $O(|V|^2)$. Furthermore, Figures 3.3(a) and 3.3(b) plot respectively the best and average deviation with *1-move* from the corresponding objective values with *1-move* + *c-swap*, which clearly discloses the merit of the joint use of the *1-move* and *c-swap* operators. Even if we do not provide additional figures for the *1-move* + *s-swap* variant, we understand that the observations made for the *1-move* variant hold as well. Moreover, Friedman statistical tests confirm that ITS algorithm with *1-move* + *c-swap* performs significantly better than the other two ITS variants in terms of both best and average solution values. This experiment demonstrates thus the contribution of the constrained *c-swap* operator to the performance of the proposed ITS algorithm.

Table 3.5: Computational comparisons of the ITS algorithm using the *1-move* operator and the constrained swap operator (*c-swap*) with an ITS variant using *1-move* alone and another ITS variant using *1-move* and the conventional swap operator (*s-swap*)

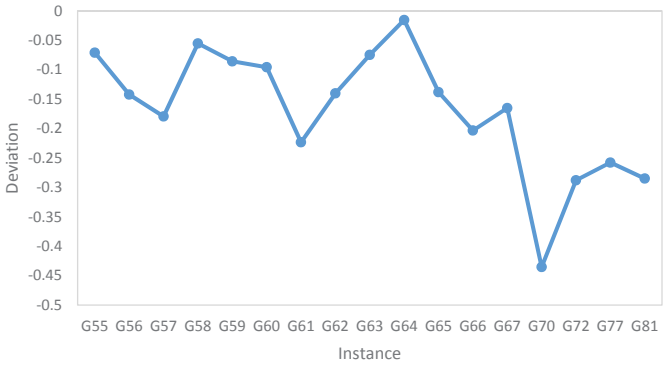
Instance	*1-move + c-swap*			*1-move*			*1-move + s-swap*		
	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$	f_{best}	f_{avg}	$time(s)$
G55	10299	10290.83	2596.84	10292	10283.53	6192.26	10254	10231.45	6587.15
G56	4016	4013.13	1926.45	4012	4007.43	5595.33	4008	3988.9	5697.57
G57	3490	3487.76	610.16	3485	3481.50	5013.16	3466	3460.4	4989.16
G58	19276	19265.9	5102.34	19262	19255.25	6382.23	19190	19175.55	7014.74
G59	6085	6074.34	4902.13	6076	6069.13	4637.32	6043	6030.7	6910.92
G60	14186	14176.54	5678.63	14170	14162.97	5435.35	14101	14079.3	6514.35
G61	5796	5780.18	4072.54	5778	5767.28	6816.18	5709	5684.35	5638.13
G62	4866	4860.12	1472.1	4861	4853.31	4267.49	4821	4810.1	4968.75
G63	26988	26985.32	2256.66	26979	26965.14	4758.43	26910	26803.35	5017.68
G64	8737	8712.1	6032.55	8723	8710.75	6026.28	8705	8692.1	6987.14
G65	5556	5550.87	2350.98	5550	5543.21	5472.34	5318	5301.8	6541.25
G66	6356	6352.01	1323.15	6349	6339.09	5262.37	6036	6012.2	5746.28
G67	6938	6935.46	1023.4	6933	6924.01	6465.22	6714	6683.4	6357.17
G70	9581	9576.32	1154.32	9541	9534.64	4785.42	9013	8981.3	7104.38
G72	6994	6992.5	1201.97	6979	6972.36	6679.44	6034	5986.45	6879.32
G77	9918	9915.14	2013.44	9900	9889.59	6944.30	9062	9013.4	6245.84
G81	14030	14025.45	1953.23	14003	13985.48	7004.45	12002	11946.45	7008.46

3.5 Conclusion

In this chapter, we developed an iterated tabu search algorithm for the maximum bisection problem, which achieved a high level performance by including two distinct search operators applied into three



(a) The best objective deviation of the ITS variant using the *1-move* operator alone from the ITS algorithm using both the *1-move* and *c-swap* operators



(b) The average objective deviation of the ITS variant using the *1-move* operator alone from the ITS algorithm using both the *1-move* and *c-swap* operators

Figure 3.3: Analysis of the combined use of the *1-move* operator and the constrained swap (*c-swap*) operator

search phases. The descent-based improvement phase uses the vertex move operator (*1-move*) to discover a first local optimum from a starting solution. The diversifying improvement phase jointly employs the *1-move* operator and a constrained swap operator in a probabilistic way (under the tabu search framework) to discover better solutions. The perturbation phase is applied as a means of strong diversification to get out of deep local optimum traps. To obtain an efficient implementation of the proposed algorithm, we developed streamlining techniques and a LIFO tie-breaking strategy based on dedicated bucket structures.

Experimental assessments on the 71 well-known benchmark instances with up to 20000 vertices indicated that the proposed ITS algorithm was able to obtain improved best results (new lower bounds) for 10 large instances and match the best-known results for all the other instances. Comparisons with state-of-the-art algorithms showed that the ITS algorithm was superior to the reference algorithms both in terms of solution quality and computational efficacy. Furthermore, the main ingredients of the ITS algorithm were analyzed to shed lights on their influences over the performance of the algorithm.

In the next chapter, we will consider the vertex separator problem, which receives more attention in recent several years. To solve this problem, we resort to the powerful path relinking search metaheuristic approach that builds a good balance between search intensification and diversification.

4

An effective path relinking algorithm for the vertex separator problem

This chapter presents the first path relinking algorithm for solving the NP-hard vertex separator problem in graphs. The proposed algorithm employs iterated tabu search for solution improvement, in which the typical *1-move* operator and a complementary new *swap-move* operator are jointly used to conduct neighborhood exploration. The dedicated path generation method creates a path starting from an initiating solution, on which a sequence of intermediate solutions gradually approach the guiding solution by performing moves based on a greedy selection criterion. Extensive experiments are conducted on four benchmark sets of 365 instances with up to 20000 vertices. Computational comparisons with state-of-the-art algorithms reveal that our algorithm, within a highly competitive computational time, is capable of discovering new best solutions (improved upper bounds) for 67 instances and matching the previous best solutions for all but one instance. The content of this chapter is based on an article submitted to Knowledge-Based Systems in April 2016.

Contents

4.1 Introduction

Path relinking is a population-based general framework which was originally proposed for enhancing the tabu search method [Glover *et al.*, 2000; Glover *et al.*, 2003; Glover *et al.*, 2004]. PR has recently shown outstanding performances in solving a number of challenging combinatorial optimization problems [Chen and Glover, 2016; Lacomme *et al.*, 2015; Lai and Hao, 2015; Peng *et al.*, 2015; Wang *et al.*, 2012]. A path relinking algorithm generally includes the following components: a reference set initialization and updating method, a path generation method, a path solution selection method and a solution improvement method, where the common purpose of path generation and solution selection methods lies in producing potential solutions with good quality and diversity.

Consider that no study has been reported on applying path relinking to VSP, this chapter presents the first path relinking algorithm for VSP (named PR-VSP), which is composed of a reference set initialization and updating method, a solution improvement method, a path generation method and a solution selection method. The method to initialize and update a reference set is capable of maintaining a set of elite solutions with high quality and good diversity. The solution improvement method follows the framework of iterated tabu search, which alternates between a dedicated tabu search phase and a random perturbation phase. The tabu search procedure employs two complementary search operators (*1-move* and *swap-move*) to collectively perform neighborhood exploration, where the innovative *swap-move* operator is applied to solve VSP for the first time. The path generation method builds a solution path from an initiating solution to a guiding solution, on which a sequence of intermediate solutions are created by performing local moves based on a greedy selection mechanism. The solution selection method picks one or multiple solutions on the path which are submitted to the solution improvement method for quality optimization.

Experimental assessments on four sets of benchmarks with a total of 365 instances disclose that our PR-VSP algorithm is able to find new best solutions (updated upper bounds) for 67 instances and matches previous best solutions for all but one instance.

The rest of the chapter is organized as follows. Section 4.2 presents the general scheme and each component of the proposed PR-VSP. Section 4.3 is dedicated to experimental results and comparisons with state-of-the-art algorithms in the literature. Concluding remarks are given in Section 4.4.

4.2 The proposed path relinking algorithm for VSP

4.2.1 Main scheme

Algorithm 3 shows the general scheme of the PR-VSP algorithm. It first creates a reference set $RefSet$ consisting of a set of elite (feasible) solutions $\{S_1, S_2, \ldots, S_p\}$ ($p = |RefSet|$) and constructs a set $PairSet$ composed of indexes of all pairwise solutions in $RefSet$. Then, for each pair of solutions (S_i and S_j), a path generation method is utilized to build a solution path (i.e., a sequence of intermediate solutions) that connects the initiating solution where the path starts from (say S_i) and the guiding solution where the path

Algorithm 3 Outline of the path relinking algorithm

Input: $G = (V, E)$: an undirected graph, c: a vector of weights for each vertex in V, b: an upper limit for the size of each shore subset
Output: the best solution S^* found and its objective value $f(S^*)$
repeat
  Initialize $RefSet$ and $PairSet$ (see Section 4.2.3)
  Record the best solution S^* in $RefSet$ and the objective value $f(S^*)$
  while $(PairSet \neq \emptyset)$ **do**
    Pick an index pair $(i, j) \in PairSet$ to get a pair of solutions (S^i, S^j) from $RefSet$
    Apply the Path Relinking Method to build a path from S^i to S^j and another path from S^j to S^i (see Section 4.2.5)
    Apply the Solution Selection Method to select solutions on each path (see Section 4.2.6)
    Apply the Solution Improvement Method to selected solutions (see Section 4.2.4)
    Update the best solution S^* and its objective value $f(S)^*$
    Update $RefSet$ and $PairSet$ (see Section 4.2.3)
  end while
until the elapsed time surpasses a given time limit

ends (say S_j). By interchanging the initiating and guiding solutions, another path is built in the same way. A solution selection method is then applied to pick one or multiple solutions from the path for further improvement. Each time a new solution is found, the $RefSet$ update method is triggered and the set $PairSet$ is accordingly updated. When $Pairset$ becomes empty, the algorithm re-initializes $RefSet$ and then repeats the above-mentioned procedure until a stopping condition (e.g., a cutoff time limit) is reached.

4.2.2 Search space

Given $G = (V, E)$, a candidate solution to the VPS problem is any partition of the vertex set V into a separator C and two shores A and B satisfying constraints (2) and (3) defined in the introduction. Thus, we define the search space Ω explored by the PR-VSP algorithm to be the set of all such possible three-way partitions $\{A, B, C\}$ of vertex set V, i.e.,

$$\Omega = \{\{A, B, C\} : A, B \subset V, C = V \setminus (A \cup B), (A \times B) \cap E = \emptyset, A \cap B = \emptyset, \max\{|A|, |B|\} \leq b\}. \tag{4.1}$$

For a given candidate solution $S = \{A, B, C\} \in \Omega$, its quality is directly given by its objective value, i.e., the weight sum of the vertices in the separator C, $f(S) = \sum_{i \in C} w_i$. For two given candidate solutions S' and S'' in the search space, S' is better than S'' if and only if $f(S') < f(S'')$.

Notice that for a graph of reasonable size (say several hundreds of vertices), the number of possible solutions in Ω can be already quite large. Moreover, the search space Ω will increase very rapidly with the increase of the number of vertices of the graph. The purpose of the proposed PR-VSP algorithm is to locate a solution as good as possible in this highly combinatorial search space within a given computing effort by sampling some promising candidate solutions as effectively as possible. To reach this goal, PR-VSP calls for a number of dedicated search operators and strategies that are explained below.

4.2.3 RefSet and PairSet initialization and updating

The reference set $RefSet$ contains the working solutions of the PR-VSP algorithm and is composed of a set (or population) of elite solutions with high quality and good diversity (See Alg. 1, line 4). $RefSet$ is created by employing a randomized initialization procedure to acquire diverse solutions and a tabu search based solution improvement method to assure high quality of the acquired solutions. Each initial solution is generated by the procedure presented in [Benlic and Hao, 2013b], which applies the following steps. The first step is to randomly assign all the vertices into the shore subsets A and B. For each cutting edge $(v_i, v_j) \in E$ ($v_i \in A, v_j \in B$), the second step displaces randomly v_i or v_j to the separator C. The last step randomly displaces vertices in the shore subset whose size surpasses the upper limit b into the separator C to satisfy constraint (3). Once a new solution is generated, it is immediately improved by the tabu search procedure of Section 4.2.4. We repeat the above procedure to produce $2p$ improved solutions, from which p non-identical solutions with the best objective values are chosen to form $RefSet$. Two comments are in order. First, one notes that the worst-case complexity of producing an initial solution is $O(2|V| + |E|)$. Second, this initialization procedure might produce an empty shore subset and thus an infeasible solution. If this happens, the feasibility will be ensured by the tabu search method which follows.

The $RefSet$ updating procedure decides the way of inserting a newly generated solution in $RefSet$ and removing an existing solution from $RefSet$ (See Alg. 1, line 12). To maintain a healthy $RefSet$, the updating mechanism requires that the new solution S_n considered for insertion satisfies both a specified distance threshold τ and a solution quality criterion [Lai and Hao, 2015]. Specifically, we first determine a solution S_c in $RefSet$ such that S_c has the minimum distance d_{min} to the solution S_n, the distance between S_c and S_n being the number of vertices not shared in the two separators. If $d_{min} \leq \tau$, then S_n replaces the solution S_c if S_c is no better than S_n; otherwise S_n is directly discarded. If $d_{min} > \tau$, then S_n replaces the worst solution S_w in $RefSet$ if S_n is no worse than S_w or is discarded otherwise. The complexity of each $RefSet$ updating operation is $O(p \cdot |C|)$.

$PairSet$ is used to mark each pairwise solutions which will experience a path relinking procedure (See Alg. 1, lines 4 and 7). It is initialized as the index pair of each pair of solutions in $RefSet$. Each time an index pair experiences a path relinking, it is removed from $PairSet$. Moreover, if a newly produced solution replaces a solution in $RefSet$, all the index pairs related to this replaced solution are removed from $PairSet$ and new index pairs composed of the new solution and each other solution in $RefSet$ are added into $PairSet$. When $RefSet$ is not updated for a certain consecutive number of times, all the index pairs are removed and $PairSet$ becomes empty.

4.2.4 The solution improvement method - iterated tabu search

Moves and calculation of move gain

As explained in Section 4.2.2, a solution of VSP is represented by a partition $S = \{A, B, C\}$ satisfying the two problem constraints $(A \times B) \cap E = \emptyset$ and $\max\{|A|, |B|\} \leq b$. To generate neighbor solutions from the current solution, the following two move operators are employed.

The first move operator (called *1-move*) displaces a vertex v_i in the separator C to either the shore subset A or B, without violating the constraint $\max\{|A|, |B|\} \leq b$. To enable the resulting neighbor solution further satisfy the constraint $(A \times B) \cap E = \emptyset$, a repair operation is followed to displace to the separator C all the vertices in the opposite shore which are adjacent to v_i. This *1-move* operator is shown to be effective in state-of-the-art algorithms [Benlic and Hao, 2013b; Sánchez-Oro *et al.*, 2014]. The objective

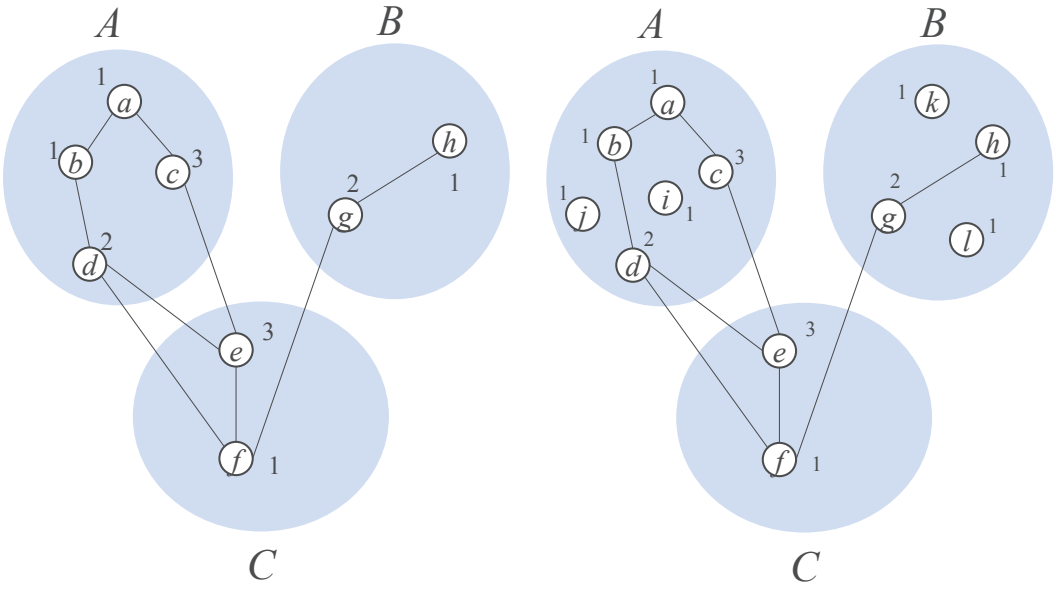


Figure 4.1: Two examples showing the benefit of the *swap-move* operator

gain of performing a *1-move* operation (i.e., the objective variation between its neighbor solution and the current solution S, also called move gain) is calculated as:

$$mg^1(v_i, S) = \begin{cases} -w_i + \sum_{v_j \in B, (v_i, v_j) \in E} w_j & \text{if } v_i \in C \text{ moves to } A \\ -w_i + \sum_{v_j \in A, (v_i, v_j) \in E} w_j & \text{if } v_i \in C \text{ moves to } B \end{cases} \tag{4.2}$$

The second move operator (called *swap-move*) is a new operator introduced in this work which is targeted to the case where the size of a shore subset reaches the upper limit b (i.e., $|A| = b$ or $|B| = b$). The *swap-move* operator displaces a vertex v_i from the separator C to the shore subset whose size is equal to the upper limit b (thus momentarily violating the constraint $\max\{|A|, |B|\} \leq b$) and then displaces another vertex w_{min} with the minimum weight from this shore subset to the separator C (satisfying the constraint $\max\{|A|, |B|\} \leq b$). To satisfy the constraint $(A \times B) \cap E = \emptyset$, the same repair operation as for *1-move* is employed to produce a feasible neighbor solution. The objective gain of performing a *swap-move* operation is calculated according to Eq. 4.3.

$$mg^2(v_i, S) = \begin{cases} -w_i + w^A_{min} + \sum_{v_j \in B, (v_i, v_j) \in E} w_j & \text{if } |A| = b \text{ and } v_i \in C \text{ moves to } A \\ -w_i + w^B_{min} + \sum_{v_j \in A, (v_i, v_j) \in E} w_j & \text{if } |B| = b \text{ and } v_i \in C \text{ moves to } B \end{cases} \tag{4.3}$$

To show the interest of the newly introduced *swap-move* operator with respect to the conventional *1-move* operator, let us consider two illustrative examples (Fig. 4.1).

The left graph in Fig. 4.1 ($|V| = 8$ and $b = 4$) shows a candidate solution $S = \{A = \{a, b, c, d\}, B = \{g, h\}, C = \{e, f\}\}$ with an objective value of 4 ($f(S) = 4$). If we use *1-move* to displace vertex $e \in C$, then e must be displaced from C to B since the number of vertices in A has already reached the given upper limit b. Once e is displaced in B, the repair operation displaces its adjacent vertices c and d from A to C. Therefore, the move gain obtained by this *1-move* operation is $-w_e + w_c + w_d = -3 + 2 + 3 = 2$ (i.e.,

the objective function value of the resulting solution S' is $f(S') = f(S) + 2 = 6$). However, if we apply *swap-move* to exchange vertex e from separator C against a from shore A, the resulting solution S'' has an objective gain of $-w_e + w_a = -2$, with leading to a better objective value of $f(S'') = f(S) - 2 = 2$).

The right graph in Fig. 4.1 ($|V| = 12$ and $b = 6$) shows a candidate solution $S = \{A = \{a, b, c, d, i, j\}, B = \{g, h, l, k\}, C = \{e, f\}\}$ which includes 4 isolated vertices $I = \{i, j, k, l\}$. If we use *1-move* to displace e from C to B, the resulting solution S' gets an objective increase of 2. Note that there is no chance for the vertices in I to be moved into the separator C by using the *1-move* operator since these vertices are not connected to any other vertex (including those of C). On the other hand, displacing the isolated vertices can reduce the size of the shore subsets, which enables vertices moving out of the separator to produce an improved solution. Here we can use *swap-move* to exchange e against any of the four vertices in I to obtain an improved solution S'' with an objective increase of 2.

It is noted that using *swap-move* is particularly useful when the graph contains isolated vertices or vertices with low degrees.

Bucket sorting

To quickly calculate the move gain for *1-move* or *swap-move*, we use an n-dimensional vector Δ^A, where each entry Δ_i^A records the total weight of all vertices of the shore subset A which are adjacent to a vertex v_i (i.e., $\Delta_i^A = \sum_{v_j \in A, (v_i, v_j) \in E} w_j$). With Δ_i^A preliminarily computed, the objective gains of performing both *1-move* and *swap-move* shown in Eq. (4.2) and (4.3) can be instantly obtained in constant time. Similarly, another vector Δ^B is employed in the same way for the shore subset B. By simply replacing all the occurrences of A by B, we obtain the updating equations of Δ^B.

In addition, a bucket sorting technique is utilized to quickly identify the best move in $O(1)$ time instead of scanning all the move gains of vertices in the separator C. Specifically, we use two arrays of buckets Bkt^A and Bkt^B to record the objective gain of displacing any vertex from the separator C to each shore subset A or B. Notice that when the condition for performing a *swap-move* is satisfied, the corresponding entry in the bucket actually represents the objective gain of *swap-move*. In each bucket array, the jth entry stores all the vertices with the objective gain currently equaling to j, which are managed by a doubly linked list. To ensure a direct access to the vertex in the doubly linked list, another index array is also employed, in which each entry stores the address that points to its vertex in the doubly linked list. For each array of buckets, identifying the best vertex with the maximum objective gain equals to the identification of the first non-empty bucket from the top of the bucket array, from which a vertex is randomly chosen from the doubly linked list.

Fig. 4.2 shows an illustrative example of the bucket structure for VSP. The graph (Fig. 4.2, left) has 11 vertices, where all the vertices have a weight of 1 for simplicity. Given the solution $S = \{A = \{a, b, c, d\}, B = \{i, j, k\}, C = \{e, f, g, h\}\}$, the bucket sorting structure is shown in Fig. 4.2 (right). To see how the vertices are arranged in the structure, we consider vertex e as an example. The move gain of displacing e from C to A is calculated as $-w_e + \Delta_e^B = 0$, so e is stored in the position $j = 0$ of the bucket array Bkt^A. In the same token, the vertex index e is stored in the position $j = -1$ of the bucket array Bkt^B. For each vertex in the separator C, we store it in the right entries of the buckets Bkt^A and Bkt^B in the same way. In addition, each entry of the index array of vertices shown at the bottom of Fig. 4.2 points to the position of this vertex in the buckets Bkt^A and Bkt^B. From the top of buckets, we see that displacing f to A, h to B, g to A and g to B leads to the same maximum gain of -1, from which one move will be chosen at random.

To perform a *1-move* or *swap-move* operation, the following three steps are concerned: 1) a vertex is

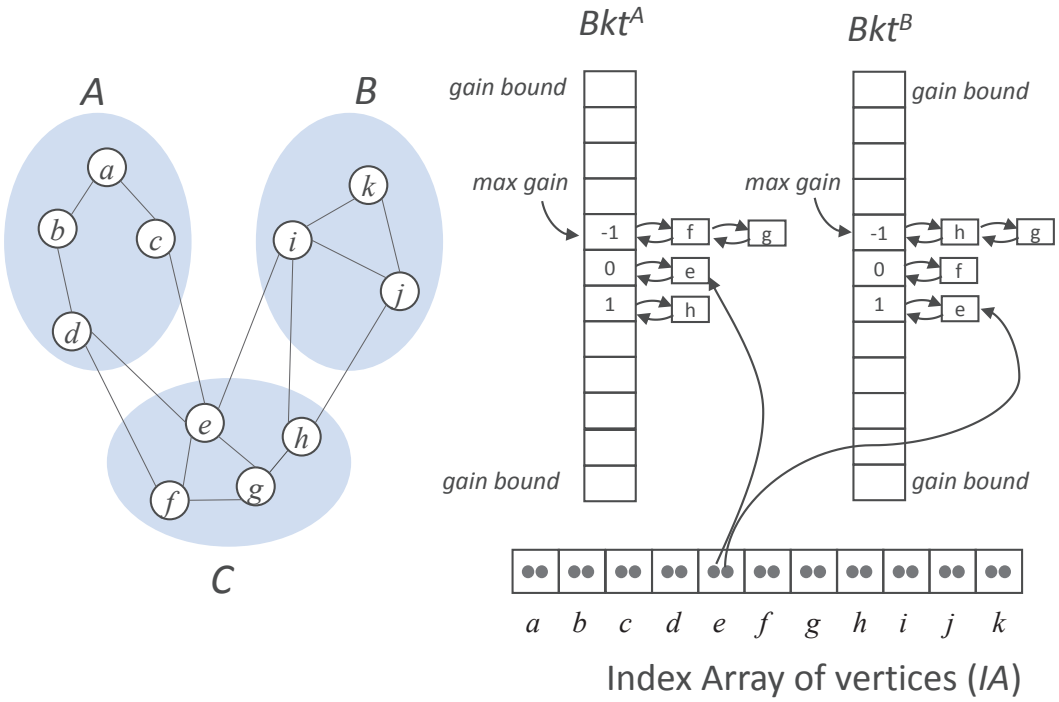


Figure 4.2: An example of the bucket structure for the vertex separator problem

displaced from the separator to either shore subset; 2) the adjacent vertices in the opposite shore subset of the displaced vertex are displaced to the separator; 3) a vertex is displaced from a shore subset to the separator. Now, let us take the shore subset A as an example to illustrate how to quickly update the Δ^A vector.

- If a vertex v_i is displaced from C to A, the Δ^A vector is updated as
  $\Delta^A_j = \Delta^A_j + w_i$, for all $v_j \in V$ where $(v_i, v_j) \in E$
- If all the adjacent vertices of a vertex $v_i \in B$ (denoting the set of these vertices as $SubA$) are displaced from A to C, Δ^A is updated as
  $\Delta^A_j = \Delta^A_j - \sum_{v_k \in SubA, (v_k, v_j) \in E} w_k$, for all $v_j \in V$ where $v_k \in SubA$, $(v_k, v_j) \in E$
- If a vertex v_i is displaced from A to C, Δ^A is updated as
  $\Delta^A_j = \Delta^A_j - w_i$, for all $v_j \in V$ where $(v_i, v_j) \in E$

The method to update Δ^B for the operations on the shore subset B is obtained by replacing all the appearances of A by B and B by A.

The operations on the bucket structure with regard to the above mentioned move operations are as follows.

- Delete: delete a vertex from Bkt^A and Bkt^B if it is displaced from C to A
- Add: add a vertex into Bkt^A and Bkt^B if it is displaced from A to C
- Shift: shift a vertex to the correct entry in each bucket array according to its updated objective gains

Since the bucket size is given by the number of possible objective gains, the bucket sorting technique is limited to problem instances with integral objective gains. Moreover, a large varying range for objective gains may be out of memory, which is another potential limitation. Despite these potential limitations, our experimental results on the well-known benchmark instances indicate that the devised bucket sorting technique considerably improves the computational efficiency, thus the performance of our path relinking algorithm.

Iterated tabu search

Within the proposed PR-VSP algorithm, we use an iterated tabu search (ITS) procedure as the solution improvement method. Basically, this ITS procedure alternates between a tabu search phase [Glover and Laguna, 1999] and a perturbation phase. Each tabu search phase continues until the best solution cannot be improved for a consecutive number of iterations (called *iteration cutoff*, set as $\beta * |C|$ where β is a parameter). At this moment, the perturbation phase is triggered to generate a perturbed solution which serves as the starting solution of the next ITS run.

The tabu search phase uses both the *1-move* and *swap-move* operators to exploit the search space. At each iteration, TS performs a best move among the set of eligible moves. A move is eligible if it is not forbidden by the tabu list (see below), or if it leads to a solution better than any solution visited so far. Precisely, if the size of each shore subset is less than the upper limit b, then only the *1-move* operator is used during the search. Otherwise, both *1-move* and *swap-move* have a chance to be applied. Specifically, if the objective gain of performing a *swap-move* is better than that of performing a *1-move*, then each type of move will be selected with an equal probability of 50%. This rule is overridden if performing a *swap-move* leads to a solution better than the best solution found so far. For this case, the *swap-move* is always performed. The idea to take a worse *1-move* into consideration is to reduce the shore subset whose size reaches b, which to some extent enhances the search diversification. Note that if a shore subset becomes empty during a certain tabu search iterations, the next iteration will force a vertex to be displaced to this empty subset. In this way, we assure that the tabu search focuses its exploration on the feasible search area.

Tabu search uses a short memory called tabu list to prohibit recently performed moves from being performed for the next tt iterations (called tabu tenure) [Glover and Laguna, 1999]. The tabu tenure is adaptively tuned according to the search status. Specifically, let $|C|$ denote the size of the separator and $dmax$ denote the average value of the highest 5% vertex degrees, then the tabu tenure is set as $tt = min(dmax, |C|/2) + min(Rand(\alpha \times dmax), |C|/2)$, where $Rand(\alpha \times dmax)$ returns a random integer no greater than $\alpha \times dmax$ and α is a parameter. For a *1-move* which displaces a vertex v_i from C to A, given that v_i may go back to C due to the change of vertices in B, we prohibit v_i from joining A for the next tt iterations. For a *swap-move* which exchanges a vertex v_i of C against a vertex v_j of A, we prohibit both v_i and v_j from moving to A for the next tt iterations. The other vertices involved in a move are not concerned by the tabu list.

The perturbation phase performs a consecutive number of *1-move* operations (called perturbation strength) on the local optimum from the last tabu search phase. Specifically, each perturbation step randomly displaces a vertex from the separator C to either shore subset with equal probability, followed by a repair operation to make the resulting solution feasible. A strong perturbation deteriorates a large part of the input solution. On the other hand, a weak perturbation fails to allow the search to jump out of the attractor around the local optimum. In our experiment, the perturbation strength is set as $\rho * |C|$, where ρ is a parameter.

4.2.5 The path relinking method

The path relinking method constructs a path connecting an initiating solution and a guiding solution (both from $RefSet$), where each intermediate solution on the path gradually incorporates attributes from the guiding solution and finally matches the guiding solution [Glover *et al.*, 2000]. According to this scheme, the path relinking method first calculates the distance between the initiating solution and the guiding solution. For two solutions $S_1 = (A_1, B_1, C_1)$ and $S_2 = (A_2, B_2, C_2)$, let $Cnd_1 = C_1 \setminus C_2$, $Cnd_2 = C_2 \setminus C_1$ and $Cnd = Cnd_1 \cup Cnd_2$. Then, the distance d between S_1 and S_2 is defined as the size of the set Cnd, i.e. $d = |Cnd|$, which corresponds to the number of vertices not shared by C_1 and C_2.

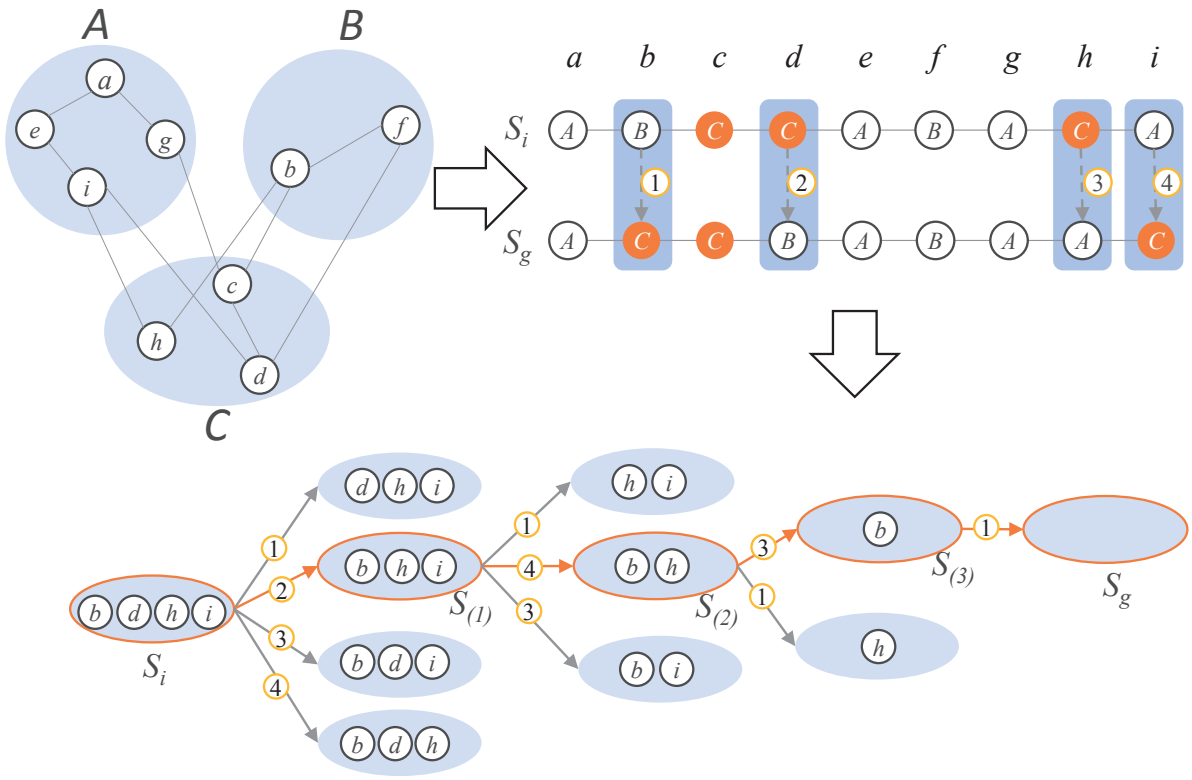


Figure 4.3: An illustrative example of the path relinking procedure

To produce a sequence of intermediate solutions $S(1), S(2), \ldots, S(d)$ on the path where $S(1)$ is the initiating solution and $S(d)$ is the guiding solution, the following mechanism is employed. For vertices in Cnd_1, the operation OP_1 is to displace a vertex from the separator C_1 to the shore subset O it lies in the solution S_2. For vertices in Cnd_2, the operation OP_2 is to displace a vertex from the shore subset it lies in the solution $S1$ to the separator $C2$. Consider that applying the operation OP_1 generally leads to an infeasible path solution $S(t)$, a repair operation that displaces all the adjacent vertices in the opposite shore subset of the displaced vertex v_i to the separator C_1 is followed. To repair the next solution $S(t+1)$ on the path, we can directly utilize its precedent solution $\hat{S}(t)$ by a repair operation to get $\hat{S}(t+1)$ rather than directly repairing $S(t+1)$. The repair operation has a complexity of $O(|V| \cdot c)$ where c represents the density of the graph $G(V, E)$. On the other hand, an operation OP_2 always produces a feasible solution and thus no repair operation is needed.

Each step $t = \{1, 2, \ldots, d\}$ for building the path selects a vertex from Cnd such that it results in a feasible solution with the best objective value after performing OP_1 and OP_2 operations. This can be achieved in $O(|Cnd| \cdot |V| \cdot c)$. Each time a vertex in Cnd is displaced, it is removed from Cnd and the distance between the resulting path solution and the guiding solution is decreased by 1. The next solution $S(t+1)$ is obtained by performing a OP_1 or OP_2 operation on the solution $S(t)$. After d steps, the set Cnd becomes empty and the path generation method arrives at the guiding solution and thus terminates.

Fig. 4.3 provides an example to illustrate the path generation procedure. Two solutions $S_i = \{A_i = \{a, e, g, i\}, B_i = \{b, f\}, C_i = \{c, d, h\}\}$ and $S_g = \{A_g = \{a, e, g, h\}, B_g = \{d, f\}, C_g = \{b, c, i\}\}$ are given. To build a path starting from the solution S_i (initiating solution) and ending at the solution S_g (guiding solution), we first identify the set of uncommon vertices in the separators C_i and C_g, denoted as $Cnd = \{b, d, h, i\}$. Then for each step in the path generation procedure, a vertex from Cnd goes through a OP_1 or OP_2 operation. Hence, four vertices can be chosen in the first path generation step, by displacing the vertex b from B_i to C_i, d from C_i to B_i, h from C_i to A_i or i from A_i to C_i, and produces four candidate

path solutions. Among them the solution $S(1) = \{A_i = \{a, e, g, i\}, B_i = \{b, d, f\}, C_i = \{c, h\}\}$ is chosen as the path solution because it leads to a feasible solution with the best objective value. Then starting from $S(1)$, three vertices can be chosen in the second path generation step and creates three candidate path solutions, from which the solution $S(2) = \{A_i = \{a, e, g\}, B_i = \{b, d, f\}, C_i = \{c, h, i\}\}$ is chosen to be on the path. After a total of four steps, the path generation procedure arrives at the guiding solution S_g and stops.

4.2.6 The solution selection method

The solution selection method aims to identify solutions from the sequence of intermediate solutions produced by the path generation method for further improvement by iterative tabu search. In general, several solutions can be selected for improvement. Considering that the solutions on the path are quite close to each other, running ITS on multiple solutions would lead to the same locally optimal solution. Therefore, we just select one solution from the path with the best objective value.

4.3 Experimental results

This section is dedicated to a large experimental assessment of the proposed PR-VSP algorithm. For this purpose, we present computational results on four sets of benchmark instances and compare our results with those reported by the state-of-the-art algorithms in the literature.

4.3.1 Experimental protocols

We use the following four sets of benchmarks with a total of 365 instances which are commonly tested in the literature.

- **Traditional benchmarks:** This set [1] contains 104 small instances with $11 \leq |V| \leq 191$ and $20 \leq |E| \leq 13992$ with known optimal solutions. This set of instances was first introduced and studied in [de Souza and Balas, 2005] and tested in [Benlic and Hao, 2013b; Biha and Meurs, 2011].
- **Hermberg and Rendl benchmarks:** This set [2] is composed of 71 structured and random instances with $|V|$ ranging from 800 to 20000 and graph density ranging from 0.000131 to 0.06. Note that the last 17 large graphs are investigated for the first time in this work. This set of instances was first tested in [Benlic and Hao, 2013b].
- **Barabasi-Albert benchmarks:** This set [3] includes 95 instances with $100 \leq |V| \leq 1000$ and node degree randomly selected from $[1, |V|]$. Graphs of this type are widely observed in the Internet, the World Wide Web, citation networks and some social networks. This set of instances was tested in [Sánchez-Oro *et al.*, 2014].
- **Erdos-Renyi benchmarks:** This set [4] contains 95 random instances with $100 \leq |V| \leq 1000$ and each pair of vertices connected with a probability randomly chosen from $[0.2, 1.0]$. This set of instances was tested in [Sánchez-Oro *et al.*, 2014].

1. http://www.ic.unicamp.br/ cid/Problem-instances/VSP.html#VSP
2. http://www.optsicom.es/maxcut/#instances
3. http://www.optsicom.es/vs
4. http://www.optsicom.es/vs

Table 4.1: Parameter setting of the PR-VSP algorithm

Parameters	Section	Description	Value
p	4.2.3	$RefSet$ size	20
α	4.2.4	coefficient used in the tabu tenure	1.6
β	4.2.4	coefficient used in the iteration cutoff	2.4
ρ	4.2.4	coefficient used in the perturbation strength	rand(0.05,0.25)
τ	4.2.3	coefficient used in the distance threshold	0.3

Table 4.2: Post-hoc statistical tests for the parameter α

α	0.4	0.8	1.2	1.6
0.8	0.8853			
1.2	0.0474	0.1612		
1.6	0.0015	0.0019	0.3327	
2.0	0.0231	0.1291	0.5298	0.6241

Our PR-VSP algorithm was programmed in C++ and compiled using GNU g++ on a Xeon E5440 (2.83 GHz CPU and 8 GB of RAM). The following time limits were used as stopping conditions of our experiments: 1 second for the traditional benchmarks, 3600 seconds for the Hermberg and Rendl benchmarks and 10 seconds for both the Barabasi-Albert and Erdos-Renyi benchmarks. Given the stochastic nature of the PR-VSP algorithm, we run PR-VSP to solve each problem instance 100 times independently and report computational statistics based on the outcomes of the 100 runs.

4.3.2 Parameter setting

Table 4.1 shows the parameter setting of the PR-VSP algorithm used for our experiments. To identify the adopted parameter values, we conducted a parameter sensitivity analysis on a set of 20 representative instances by comparing different values for each parameter: $p \in \{10, 15, 20, 25, 30\}$, $\alpha \in \{0.4, 0.8, 1.2, 1.6, 2.0\}$, $\beta \in \{1.0, 1.5, 2.0, 2.5, 3.0\}$, $\rho \in \{rand(0.05, 0.20), rand(0.05, 0.25), rand(0.10, 0.25), rand(0.15, 0.25), rand(0.15, 0.30)\}$ and $\gamma \in \{0.2, 0.25, 0.3, 0.35, 0.4\}$. By varying the values of one parameter and keeping the values of the other parameters unchanged, we ran the PR-VSP algorithm 20 times to solve each chosen instance and recorded the average solution values. Hence, we obtained a table for each parameter where the columns represent different values for this parameter and the rows represent the average solution values for each instance. Furthermore, we employed Friedman statistical tests to verify if different values for a specific parameter present statistical differences.

Experimental results indicated that varying values of the parameters p, β, ρ and γ present no significant differences with p-values of 0.7925, 0.5374, 0.4147 and 0.8769, respectively. This means that the algorithm is not sensitive to these four parameters. However, the p-value of 0.0007 for the parameter α indicates that the algorithm is sensitive to the tabu tenure. Furthermore, we conducted a post-hoc analysis to check statistical differences between each pair of α values and showed the results in Table 4.2. As it can be seen in Table 4.2, four pairs of values present significant differences with a p-value < 0.05, among which two pairs are related to the setting $\alpha = 1.6$. In order to choose the best parameter setting, we also evaluated the number of best solutions achieved by each setting as a secondary criterion. The results showed that the setting $\alpha = 1.6$ obtains the best solution for 18 out of the 20 tested instances and performed the best among all the settings. In conclusion, this experiment reveals the rationality of the chosen parameter setting of Table 4.1.

4.3.3 Reference algorithms

For the purpose of our comparative study, we use the following state-of-the-art algorithms as our references.

- Breakout local search (BLS) [Benlic and Hao, 2013b] is a heuristic algorithm which reported results on the 104 traditional benchmarks as well as the 71 Hermberg and Rendl benchmarks. Like our PR-VSP algorithm, BLS was written in C++ and compiled with GNU g++ under GNU/Linux running on an Intel Xeon E5440 (2.83 GHz and 2 GB of RAM). The stopping condition was a maximum running time of 10 seconds for the 104 traditional benchmarks and 3600 seconds for the 71 Hermberg and Rendl benchmarks.
- General variable neighborhood search (GVNS) [Sánchez-Oro *et al.*, 2014] is a heuristic algorithm which reports results on the 104 traditional benchmarks, the 95 Barabasi-Albert benchmarks and the 95 Erdos-Renyi benchmarks. GVNS was implemented in Java SE7 and the results were obtained on a computer with an Intel Core i7 2600 CPU (3.4 GHz) and 4 GB of RAM. The stopping condition used was a maximum running time of 5 seconds for the 104 traditional benchmarks and 1800 seconds for the other benchmarks.
- B-S [de Souza and Balas, 2005] is a branch-and-cut exact algorithm based on the results of an in-depth polyhedral study. Computational reports were reported on the 104 traditional benchmarks on a Pentium 4 computer (2.5 GHz and 2 GB of RAM) with a time limit of 1800 minutes.
- B-M [Biha and Meurs, 2011] is another exact approach which applies the general CPLEX 9.0 solver to a mixed-integer program. The results on the 104 traditional benchmarks were obtained on a Pentium M740 computer with 1.73 GHz and 1 GB of RAM. The stopping condition was not explicitly indicated in [Biha and Meurs, 2011].

Given that the compared algorithms (except BLS) were run on computing platforms which are different from our computer, it is difficult to make a fair comparison of the computing times. For this reason, we focused our comparisons on the solution quality criterion while providing the timing information only for indicative purposes. To make the time comparison meaningful, we used the CPU performance measurement suits from the well-known SPEC (https://www.spec.org/benchmarks.html) to normalize the computing times of the compared algorithms with our machine as the reference. As such, we multiplied the computing times reported by GVNS, B-S and B-M by 1.2, 0.8 and 0.6 respectively.

4.3.4 Computational results and comparisons

Table 4.3 shows the computational results on the 104 transitional instances obtained by our PR-VSP algorithm along with the results of four reference algorithms: breakout local search (BLS) [Benlic and Hao, 2013b], general variable neighborhood search (GVNS) [Sánchez-Oro *et al.*, 2014] and the two exact algorithms presented in [de Souza and Balas, 2005; Biha and Meurs, 2011]. Since this set of benchmark instances have known optimal solutions, we report the number of instances for which the optimal solutions are obtained by each algorithm and the computational time. For the two heuristics (PR-VSP and BLS), we indicate the best time, the average time and the worst time in seconds. From Table 4.3, we find that our algorithm is able to reach the optimal solutions for all the 104 instances, with a worst time of 0.82 seconds and an average time of 0.03 seconds, which is the shortest among all the compared algorithms. By considering the results of all the algorithms, we conclude that these 104 transitional instances can be considered to be trivial for the state-of-the-art methods.

Table 4.4 is dedicated to the set of 71 Hermberg and Rendl benchmark instances and presents the comparative results between the PR-VSP algorithm and the state-of-the-art BLS algorithm. The second column (f_{prev}) indicates the current best known results ever reported in the literature. The results of PR

Table 4.3: Computational results of the PR-VSP algorithm on the set of 104 small traditional instances in comparison with three reference algorithms

Algorithms	t_{avg}	t_{best}	t_{worst}	#solved instances
PR-VSP	0.03	0.00	0.82	104/104
BLS[Benlic and Hao, 2013b]	0.08	0.00	3.06	104/104
GVNS[Sánchez-Oro *et al.*, 2014]	4.81	0.55	10.81	104/104
B-S[de Souza and Balas, 2005]	62.18	-	1131.60	97/104
B-M[Biha and Meurs, 2011]	140.28	-	9783.08	104/104

and BLS are respectively shown in column 3-5 and columns 6-8 in terms of the best solution value $Best$, the average solution values Avg and the average time $Time$ to reach $Best$. To make a fair comparison, we reran BLS on our computer under the same time limit as our PR-VSP algorithm. From Table 4.4, we observe that PR-VSP is able to find new best solutions (displayed in bold) for 22 out of 71 instances and fails to reach the best known results for only one instance (G46). Moreover, PR obtains better, equal and worse average solution values relative to the top BLS algorithm for 45, 14 and 12 instances, respectively, demonstrating its competitiveness compared to BLS in terms of solution quality. Finally, the computational time taken by PR-VSP to reach better solutions is competitive with the time taken by BLS.

Table 4.5 and 4.6 compare the PR-VSP and GVNS algorithms on 90 Barabasi-Albert instances and 90 Erdos-Renyi instances, respectively. For the PR algorithm, we report the best solution value $Best$, average solution value Avg and the computational time $Time$ to reach $Best$ obtained for each instance. The results of the GVNS algorithm are taken directly from [Sánchez-Oro *et al.*, 2014]. As shown in Tables 4.5 and 4.6, our PR algorithm robustly attains better solutions (displayed in bold) than GVNS for 26 Barabasi-Albert instances and 20 Erdos-Renyi instances, respectively. For the other instances, our PR algorithm matches the best solution values found by the GVNS algorithm. In particular, the computational time of PR is 50 times shorter than that of GVNS on average, revealing the efficacy of our PR algorithm.

Table 4.4: Computational results of the PR-VSP algorithm on the set of 71 Hermberg and Rendl instances in comparison with the state-of-the-art BLS algorithm

Instances	f_{prev}	PR-VSP			BLS		
		Best	*Avg*	Time	*Best*	*Avg*	Time
G1	257	257	257	0.84	257	257	8.23
G2	257	257	257	0.38	257	257	7.49
G3	257	257	257	1.43	257	257.05	76.35
G4	363	363	363	11.54	363	363.5	1735.65
G5	257	257	257	5.18	257	257	180.59
G6	257	257	257	0.41	257	257	7
G7	257	257	257	0.63	257	257	5.78
G8	257	257	257	1.92	257	257	153.27
G9	257	257	257	1.37	257	257	29.89
G10	257	257	257	3.56	257	257	220.92
G11	16	16	16	0.15	16	16	0.14
G12	32	32	32	0.08	32	32	0.05
G13	45	45	46.8	69.75	45	45	5.02
G14	146	146	146.3	386.15	146	146	1009.69
G15	144	144	144	12.98	144	144	13.83
G16	144	144	144	11.29	144	144	8.38
G17	144	144	144	55.89	144	144	54.88
G18	146	146	146.1	184.52	146	146	632.42
G19	144	144	144	8.97	144	144	19.47
G20	144	144	144	14.73	144	144	16.24
G21	144	144	144.1	67.01	144	144	16.08
G22	588	**587**	587	826.47	588	588.4	1023.94
G23	590	590	590	10.06	590	590.4	1342.36
G24	589	**587**	587.9	1228.16	589	589.5	1384.47
G25	589	**588**	588.3	1515.4	589	589.2	841.67
G26	587	587	587	671.46	588	588.15	1005.26
G27	820	**818**	818.7	815.85	820	820.05	798.99
G28	822	**821**	821.7	996.89	822	822.95	163.71
G29	820	**819**	819	1246.36	820	820.75	1922.14
G30	821	**820**	820.6	1716.18	821	821.75	1041.71
G31	819	819	819	976.61	819	819.65	1771.11
G32	40	40	40	0.44	40	40	0.66

Table 4.4 – continued from previous page

Instances	f_{prev}	PR-VSP			BLS		
		Best	*Avg*	Time	*Best*	*Avg*	Time
G33	50	50	50	0.26	50	50	0.2
G34	80	80	82	0.21	80	82	0.14
G35	436	**435**	435.2	2025.4	436	436.35	1696.19
G36	441	**440**	440.4	1105.17	441	442.05	1302.49
G37	435	**434**	434.7	2307.84	435	438.2	2211.1
G38	439	439	439	1010.22	439	440.3	2156.96
G39	436	**435**	435.3	1415.07	436	437.8	1843.45
G40	440	440	440.4	1129.67	440	442.1	2365.89
G41	435	**434**	434.5	1160.87	435	437.05	1400.01
G42	439	**438**	438.8	650	439	440.8	2080.13
G43	411	411	411	5.28	411	411	11.52
G44	411	411	411	1.86	411	411	247.13
G45	410	410	410	4.04	410	410	53.78
G46	411	*412*	412	0.96	412	412	3.92
G47	411	411	411	17.88	411	411.95	0.62
G48	100	100	104	0.49	100	102	0.82
G49	60	60	60	1.25	60	60	0.52
G50	50	50	50	1.02	50	50	0.99
G51	224	224	224	38.18	224	224	59.11
G52	223	223	223.4	383.27	223	223.25	1140.31
G53	221	221	221.2	187.11	221	221.35	626.08
G54	219	219	219	18.14	219	219	32.88
G55	995	**979**	987	2752.88	997	1006.95	3170.29
G56	999	**972**	987.7	3345.15	999	1010.4	2357.58
G57	100	100	110	8.11	100	100	1.25
G58	1109	**1085**	1101	3352.99	1109	1133.35	3433.27
G59	1105	**1088**	1102.2	776.48	1105	1127.1	3396
G60	1376	**1354**	1372	3375.54	1386	1397.15	3343.69
G61	1385	**1350**	1368.2	3561.93	1385	1397.8	2653.87
G62	140	140	146	1.67	140	149	43.09
G63	1575	**1546**	1560.4	3321.63	1575	1596.85	2139.52
G64	1582	**1549**	1566.6	3363.47	1582	1602.6	3205.26
G65	160	160	164	7.72	160	160	34.39
G66	180	180	184	39.73	180	181	35.56
G67	194	194	197	782.94	194	196.7	411.21
G70	605	**320**	328.1	2977.13	609	633.25	2503.34
G72	194	194	197.5	487.76	194	195.5	643.86
G77	200	200	219.6	136.63	200	206.2	632.68
G81	200	200	220	4.58	200	213.85	39.27
Better		22					
Equal		48					
Worse		1					

Table 4.5: Computational results of the PR-VSP on the set of 95 Barabasi-Albert instances in comparison with the state-of-the-art GVNS algorithm

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
barabasi_albert_1(100,65)	43	43	0.02	43	5.13
barabasi_albert_1(1000,878)	564	564	3.24	564	93.57
barabasi_albert_1(150,137)	86	86	0.04	86	7.64
barabasi_albert_1(200,175)	112	112	0.06	112	10.11
barabasi_albert_1(250,146)	99	99	0.26	99	13.01
barabasi_albert_1(300,255)	160	160	0.19	160	16.01
barabasi_albert_1(350,320)	198	198	0.23	198	17.98
barabasi_albert_1(400,376)	234	234	0.24	234	20.86
barabasi_albert_1(450,326)	218	218	0.55	218	23.45
barabasi_albert_1(500,277)	204	204	0.51	204	25.16
barabasi_albert_1(550,499)	314	314	1	314	33.41
barabasi_albert_1(600,541)	**348**	348	1.1	349	32.74
barabasi_albert_1(650,465)	320	320	0.71	320	45.84
barabasi_albert_1(700,649)	**409**	409	1.54	415	40.76
barabasi_albert_1(750,422)	303	303	1.11	303	59.74
barabasi_albert_1(800,627)	418	418	1.37	418	59.07
barabasi_albert_1(850,619)	418	418	2.86	418	76.26
barabasi_albert_1(900,817)	**522**	522	4.63	527	59.72
barabasi_albert_1(950,626)	**442**	442	2	444	57.83
barabasi_albert_2(100,69)	45	45	0.02	45	5.05
barabasi_albert_2(1000,856)	556	556	4.64	556	100.28
barabasi_albert_2(150,94)	65	65	0.04	65	7.66
barabasi_albert_2(200,161)	105	105	0.09	105	10.17
barabasi_albert_2(250,235)	147	147	0.12	147	12.5
barabasi_albert_2(300,220)	**147**	147	0.17	148	15.11
barabasi_albert_2(350,182)	129	129	0.16	129	21.29
barabasi_albert_2(400,227)	**164**	164	0.32	165	23.78

Table 4.5 – continued from previous page

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
barabasi_albert_2(450,392)	252	252	0.78	252	25.52
barabasi_albert_2(500,288)	205	205	0.46	205	36.97
barabasi_albert_2(550,355)	**242**	242	0.74	247	31.19
barabasi_albert_2(600,520)	335	335	0.7	335	35.86
barabasi_albert_2(650,485)	327	327	1.02	327	41.45
barabasi_albert_2(700,545)	368	368	1.85	368	47.1
barabasi_albert_2(750,395)	**285**	285	0.66	293	61.85
barabasi_albert_2(800,617)	**416**	416	1.14	419	54.16
barabasi_albert_2(850,739)	478	478	4.07	478	53.72
barabasi_albert_2(900,576)	**404**	404	1.04	411	49.2
barabasi_albert_2(950,744)	**501**	501	4.72	505	91.86
barabasi_albert_3(100,64)	43	43	0.02	43	5.1
barabasi_albert_3(1000,601)	430	430	2.34	430	72.14
barabasi_albert_3(150,129)	83	83	0.04	83	7.62
barabasi_albert_3(200,111)	81	81	0.07	81	10.66
barabasi_albert_3(250,191)	124	124	0.15	124	13.26
barabasi_albert_3(300,260)	159	159	0.11	159	15.56
barabasi_albert_3(350,251)	166	166	0.3	166	17.98
barabasi_albert_3(400,284)	**191**	191	0.21	193	22.92
barabasi_albert_3(450,243)	**177**	177	0.25	179	24.86
barabasi_albert_3(500,273)	200	200	0.47	200	25.12
barabasi_albert_3(550,294)	217	217	0.29	217	35.87
barabasi_albert_3(600,435)	293	293	0.82	293	35.01
barabasi_albert_3(650,642)	387	387	0.76	387	41.02
barabasi_albert_3(700,678)	417	417	0.83	418	37.65
barabasi_albert_3(750,643)	416	416	0.74	416	38.4
barabasi_albert_3(800,595)	**399**	399	5.16	409	59.2
barabasi_albert_3(850,693)	**453**	453	2.3	458	65.65
barabasi_albert_3(900,851)	535	535	1.81	535	54.22
barabasi_albert_3(950,553)	**398**	398	1.48	401	62.09
barabasi_albert_4(100,87)	51	51	0.02	51	5.1
barabasi_albert_4(1000,509)	**381**	381	3.31	391	78.21
barabasi_albert_4(150,111)	71	71	0.05	71	7.55
barabasi_albert_4(200,197)	127	127	0.07	127	10.11
barabasi_albert_4(250,133)	98	98	0.16	98	13.23
barabasi_albert_4(300,205)	139	139	0.23	139	17.08
barabasi_albert_4(350,294)	188	188	0.27	188	17.53
barabasi_albert_4(400,350)	225	225	0.22	225	22.92
barabasi_albert_4(450,229)	165	165	0.19	165	26.05
barabasi_albert_4(500,496)	305	305	0.3	305	26.78
barabasi_albert_4(550,347)	245	245	0.46	245	31.13
barabasi_albert_4(600,305)	226	226	0.43	226	31.05
barabasi_albert_4(650,535)	347	347	0.59	347	36.09
barabasi_albert_4(700,621)	395	395	1.18	395	45.1
barabasi_albert_4(750,722)	**447**	447	1	453	42.79
barabasi_albert_4(800,750)	477	477	1.08	477	59.9
barabasi_albert_4(850,646)	434	434	1.44	434	72.54
barabasi_albert_4(900,768)	**504**	504	3.97	510	68.77
barabasi_albert_4(950,758)	507	507	0.99	507	89.51
barabasi_albert_5(100,89)	55	55	0.02	55	5.05
barabasi_albert_5(1000,578)	413	413	1.19	413	74.64
barabasi_albert_5(150,103)	67	67	0.05	67	7.64
barabasi_albert_5(200,199)	132	132	0.05	132	10.15
barabasi_albert_5(250,132)	94	94	0.18	94	13.07
barabasi_albert_5(300,211)	139	139	0.13	139	16.11
barabasi_albert_5(350,249)	**164**	164	0.16	168	18.01
barabasi_albert_5(400,233)	162	162	0.35	162	23.82
barabasi_albert_5(450,424)	**269**	269	0.38	270	25.17
barabasi_albert_5(500,408)	270	270	1.48	270	27.48
barabasi_albert_5(550,495)	317	317	0.48	317	31.76
barabasi_albert_5(600,475)	316	316	0.73	316	30.97
barabasi_albert_5(650,434)	**298**	298	0.68	304	47.87
barabasi_albert_5(700,501)	**341**	341	0.67	346	35.7
barabasi_albert_5(750,744)	453	453	1.71	453	43.28
barabasi_albert_5(800,663)	432	432	0.75	432	49.66
barabasi_albert_5(850,635)	**430**	430	3.16	433	71.7
barabasi_albert_5(900,662)	**446**	446	1.27	452	88.55
barabasi_albert_5(950,818)	534	534	2.48	534	83.49
Better	25				
Equal	70				
Worse	0				

Table 4.6: Computational results the PR-VSP on the set of 95 Erdos-Renyi instances in comparison with the state-of-the-art GVNS algorithm

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
erdos_renyi_1(100,0.89)	82	82	0.02	82	5.004
erdos_renyi_1(1000,0.27)	333	333	0.4	333	66.929
erdos_renyi_1(150,0.86)	118	118	0.06	118	7.531
erdos_renyi_1(200,0.82)	147	147	0.07	147	10.202
erdos_renyi_1(250,0.89)	205	205	0.1	205	12.583
erdos_renyi_1(300,0.34)	99	99	0.06	99	16.779
erdos_renyi_1(350,0.32)	116	116	0.07	116	20.186
erdos_renyi_1(400,0.79)	290	290	0.34	290	20.517
erdos_renyi_1(450,0.25)	149	149	0.09	149	22.897
erdos_renyi_1(500,0.95)	454	454	0.47	454	25.097
erdos_renyi_1(550,0.64)	316	316	0.45	316	36.505
erdos_renyi_1(600,0.59)	**316**	316	0.56	318	39.628
erdos_renyi_1(650,0.24)	216	216	0.17	216	38.103
erdos_renyi_1(700,0.41)	**243**	243	0.46	254	57.626
erdos_renyi_1(750,0.57)	**394**	394	1.05	401	54.65
erdos_renyi_1(800,0.31)	266	266	0.3	266	76.686
erdos_renyi_1(850,0.91)	746	746	1.77	746	44.4
erdos_renyi_1(900,0.37)	299	299	0.43	299	55.387
erdos_renyi_1(950,0.81)	733	733	2.39	733	54.825
erdos_renyi_2(100,0.12)	28	28	0.01	28	5.095
erdos_renyi_2(1000,0.30)	333	333	0.41	333	69.413
erdos_renyi_2(150,0.51)	60	60	0.03	60	7.87
erdos_renyi_2(200,0.23)	65	65	0.38	65	10.825
erdos_renyi_2(250,0.81)	183	183	0.1	183	12.812
erdos_renyi_2(300,0.52)	132	132	0.09	132	15.142
erdos_renyi_2(350,0.19)	115	115	0.05	115	20.724
erdos_renyi_2(400,0.40)	133	133	0.13	133	25.606
erdos_renyi_2(450,0.10)	**144**	144	1.04	145	29.39
erdos_renyi_2(500,0.05)	**153**	153	4.03	155	34.925
erdos_renyi_2(550,0.33)	183	183	0.15	183	36.158
erdos_renyi_2(600,0.21)	**198**	198.1	0.19	199	44.549
erdos_renyi_2(650,0.36)	216	216	0.24	216	46.489
erdos_renyi_2(700,0.49)	300	300	0.45	300	52.715
erdos_renyi_2(750,0.94)	678	678	0.95	678	38.341
erdos_renyi_2(800,0.36)	266	266	0.36	266	41.346
erdos_renyi_2(850,0.64)	**506**	506	3.06	511	61.056
erdos_renyi_2(900,0.61)	**507**	507	0.92	511	79.593
erdos_renyi_2(950,0.83)	**754**	754	1.62	755	50.552
erdos_renyi_3(100,0.78)	61	61	0.02	61	5.009
erdos_renyi_3(1000,0.92)	891	891	6.2	891	52.155
erdos_renyi_3(150,0.38)	49	49	0.03	49	7.88
erdos_renyi_3(200,0.35)	66	66	0.03	66	10.753
erdos_renyi_3(250,0.37)	83	83	0.05	83	14.101
erdos_renyi_3(300,0.25)	99	99	0.05	99	15.334
erdos_renyi_3(350,0.55)	161	161	0.14	161	18.249
erdos_renyi_3(400,0.11)	**129**	129	2.46	130	22.915
erdos_renyi_3(450,0.75)	309	309	0.41	309	22.976
erdos_renyi_3(500,0.50)	**211**	211	0.27	223	36.001
erdos_renyi_3(550,0.87)	452	452	0.57	452	27.84
erdos_renyi_3(600,0.25)	199	199	0.14	199	30.911
erdos_renyi_3(650,0.45)	246	246	0.28	246	46.143
erdos_renyi_3(700,0.44)	**265**	265	0.39	278	55.913
erdos_renyi_3(750,0.94)	676	676	0.86	676	38.087
erdos_renyi_3(800,0.61)	437	437	1.07	437	65.796
erdos_renyi_3(850,0.27)	283	283	0.29	283	84.293
erdos_renyi_3(900,0.81)	686	686	1.77	686	47.637
erdos_renyi_3(950,0.80)	726	726	1.53	726	55.15
erdos_renyi_4(100,0.32)	33	33	0.02	33	5.01
erdos_renyi_4(1000,0.55)	**507**	507	1.59	512	67.073
erdos_renyi_4(150,0.69)	89	89	0.05	89	7.6
erdos_renyi_4(200,0.61)	102	102	0.06	102	10.423
erdos_renyi_4(250,0.69)	153	153	0.1	153	12.757
erdos_renyi_4(300,0.35)	99	99	0.06	99	16.594
erdos_renyi_4(350,0.22)	**115**	115.4	0.06	116	18.767
erdos_renyi_4(400,0.86)	307	307	0.23	307	20.179
erdos_renyi_4(450,0.94)	407	407	0.38	407	22.679
erdos_renyi_4(500,0.75)	**343**	343	0.65	344	27.087
erdos_renyi_4(550,0.83)	432	432	0.51	432	29.448
erdos_renyi_4(600,0.76)	425	425	0.74	425	35.145
erdos_renyi_4(650,0.59)	**347**	347	1.13	348	33.057
erdos_renyi_4(700,0.62)	**390**	390	0.71	397	39.808
erdos_renyi_4(750,0.57)	380	380	1.1	380	55.152
erdos_renyi_4(800,0.98)	765	765	1.14	765	40.252
erdos_renyi_4(850,0.74)	582	582	3.57	592	52.937
erdos_renyi_4(900,0.35)	299	299	0.4	299	57.307
erdos_renyi_4(950,0.45)	371	371	0.67	371	67.883

Table 4.6 – continued from previous page

Instances	PR-VSP			GVNS	
	Best	*Avg*	Time	*Best*	Time
erdos_renyi_5(100,0.71)	58	58	0.02	58	5.004
erdos_renyi_5(1000,0.86)	821	821	2.7	821	57.026
erdos_renyi_5(150,0.07)	**40**	40	0.02	42	8.252
erdos_renyi_5(200,0.44)	69	69	0.04	69	10.815
erdos_renyi_5(250,0.68)	149	149	0.09	149	12.991
erdos_renyi_5(300,0.36)	99	99	0.07	99	16.772
erdos_renyi_5(350,0.55)	**170**	170	0.14	172	20.441
erdos_renyi_5(400,0.38)	133	133	0.12	133	25.181
erdos_renyi_5(450,0.25)	149	149	0.09	149	22.936
erdos_renyi_5(500,0.21)	165	165.1	0.34	165	30.532
erdos_renyi_5(550,0.60)	290	290	0.44	290	31.482
erdos_renyi_5(600,0.24)	199	199	0.15	199	30.702
erdos_renyi_5(650,0.65)	**386**	386	1.09	390	41.257
erdos_renyi_5(700,0.94)	633	633	1.01	633	35.291
erdos_renyi_5(750,0.70)	473	473	0.98	473	52.227
erdos_renyi_5(800,0.38)	266	266	0.72	266	42.812
erdos_renyi_5(850,0.33)	283	283	0.33	283	44.077
erdos_renyi_5(900,0.22)	299	299	0.29	299	47.898
erdos_renyi_5(950,0.29)	316	316	0.36	316	59.508
Better	20				
Equal	75				
Worse	0				

4.3.5 Analysis

To complement the computational results presented in the last section, we now provide an analysis of some key ingredients of the proposed PR-VSP algorithm to shed light on their impact over the performance of the algorithm. As explained in Section 4.2, relative to the existing leading heuristics like [Benlic and Hao, 2013c; Sánchez-Oro *et al.*, 2014], PR-VSP includes two distinguishing features: a new local search operator (i.e., *swap-move*) and a dedicated path relinking procedure. In order to assess their contributions, we create two PR-VSP variants by disabling the *swap-move* operator (denoted by PR_non-swap) and the path relinking procedure (denoted by ITS). We compare PR-VSP with these two variants based on a selection of 31 representative instances.

For this experiment, we ran the two PR variants under the same condition as the PR-VSP algorithm. The results are summarized in Table 4.7 where we indicate the best solution values $Best$, the average solution values Avg and the average time $Time$ to reach $Best$ obtained by PR-VSP, PR_non-swap and ITS for each tested instance. Note that the results of PR-VSP are directly extracted from Table 4.4. As shown in Table 4.7, PR-VSP performs better than PR_non-swap and ITS by reaching better $Best$ values for 18 and 20 more instances. Moreover, PR-VSP also performs better in terms of the average solution values, with an average of 680.4 against 691.76 for PR_non-swap and 685.82 for ITS. In addition, the computing time of PR-VSP is quite competitive with those of PR_non-swap and ITS, while attaining better solution quality.

In conclusion, this experiment demonstrates the effectiveness of the designed new local search operator and the dedicated path relinking procedure to the performance of the PR-VSP algorithm.

4.4 Conclusion

In this chapter, we presented an effective path relinking algorithm for solving the vertex separator problem. The proposed PR-VSP algorithm integrates a reference set initialization and updating method, a solution improvement method, a path generation method as well as a solution selection method. The iterated tabu search based solution improvement method applies complementary *1-move* and *swap-move* operators to cooperatively explore the search space, where the benefit of the innovative *swap-move* operator

Table 4.7: Comparative results on 31 instances between PR-VSP and two variants

Instances	PR-VSP			PR_non-swap			ITS		
	Best	*Avg*	*Time*	*Best*	*Avg*	*Time*	*Best*	*Avg*	*Time*
G1	257	257	0.84	257	257	0.25	257	257	0.34
G10	257	257	3.56	257	257	1.21	257	257	5.16
G14	146	146.3	386.15	146	146.65	502.14	147	147	280.09
G21	144	144.1	67.01	144	144.2	115.4	144	144.95	619.74
G22	587	587	826.47	588	588.65	1520.39	588	589.66	1745.23
G23	590	590	10.06	590	590	50.57	590	590.95	315.59
G24	587	587.9	1228.16	588	588.35	1821.29	589	590.64	1564.18
G25	588	588.3	1515.4	589	589.5	1747.17	589	589.65	1573.66
G26	587	587	671.46	587	588.3	837.26	587	588.57	2552.6
G27	818	818.7	815.85	819	819.95	967.38	820	820.75	2136.41
G28	821	821.7	996.89	822	822.4	1372.29	822	822.95	1947.28
G29	819	819	1246.36	820	820.3	724.32	819	820.3	2841.23
G30	820	820.6	1716.18	820	820.6	1936.48	820	820.8	1901.36
G35	435	435.2	2025.4	435	435.65	2825.71	435	436.65	1576.71
G36	440	440.4	1105.17	440	440.95	1247.53	441	441.6	1647.73
G37	434	434.7	2307.84	435	436.7	539.92	435	436.7	1969.17
G38	439	439	1010.22	439	441.1	1828.57	440	441.5	1975.54
G39	435	435.3	1415.07	436	438.37	1521.74	437	438.95	1724.79
G40	440	440.4	1129.67	440	441.95	1019.46	440	441.95	2104.56
G41	434	434.5	1160.87	435	438.39	1437.12	435	441.15	2017.85
G47	411	411	17.88	411	411.3	135.58	411	412.3	887.64
G51	224	224	38.18	224	224.4	121.52	224	225.8	472.4
G55	979	987	2752.88	989	992.8	3301.25	984	990.45	3017.85
G56	972	987.7	3345.15	989	997.05	3470.67	976	998.12	3214.04
G58	1085	1101	3352.99	1092	1105.78	3102.21	1090	1103.72	2974.58
G59	1088	1102.2	776.48	1090	1109.13	1034.11	1094	1101.45	2457.45
G60	1354	1372	3375.54	1369	1385.8	1509.75	1359	1375.8	2434.87
G61	1350	1368.2	3561.93	1380	1398.12	2087.28	1356	1368.3	2641.79
G63	1546	1560.4	3321.63	1552	1565.4	2935.84	1563	1575.55	3017.75
G64	1549	1566.6	3363.47	1553	1564.36	3157.48	1559	1579.65	2974.47
G70	320	328.1	2977.13	505	584.26	3015.65	401	410.6	2748.64
AVG	676.00	680.4	1500.71	685.19	691.76	1480.24	680.94	685.82	1849.7
best/total	31/31			13/31			11/31		

is demonstrated both qualitatively and experimentally. The path generation method employs a dedicated strategy based on a greedy selection criterion to build path solutions.

The proposed algorithm was assessed on four sets of 365 instances with up to 20000 vertices. Comparisons with the best performing algorithms in the literature disclosed that our algorithm finds improved best solutions for 67 large instances and matches previously best known results for all but one instance, while attaining this remarkable performance within competitive or an order of magnitude shorter time.

Several interesting perspectives can be considered for future studies. First, the quality of the results attained by the proposed algorithm depends on the adopted setting of its parameters. Though a fine-tuning of the parameters is possible, it is interesting to investigate automatic tuning techniques. Second, designing new search operators and operator cooperation strategies would be another interesting work in future. Finally, the path relinking framework integrated with well designed multiple complementary local search operators could be adapted to design effective heuristics for other graph partitioning problems like those studied in [Benlic and Hao, 2011b; Zadegan *et al.*, 2013; Zhou *et al.*, 2015].

General Conclusion

Conclusions

This thesis focuses on designing effective approaches for solving several NP-hard graph partitioning problems, i.e., the max-k-cut problem, the max-bisection problem and the vertex separator problem. Due to high computational complexity and widespread applications of these problems, we adopted heuristic and metaheuristic methods to find "good-enough" sub-optimal solutions for large scale problem instances in acceptable computing time. Particularly, our algorithms employ multiple search operators to collaboratively perform space exploration, which present a good search balance between intensification and diversification. Assessed on multiple sets of well-known benchmarks, the proposed algorithms are shown to be highly competitive with respect to the best performing algorithms in the literature.

The multiple search operator algorithm (MOH) for the general max-k-cut problem achieved a high level performance by including five distinct search operators which are applied in three search phases. The descent-based improvement phase aims to discover local optima of increasing quality with two intensification-oriented operators. The diversified improvement phase combines two other operators to escape local optima and discover promising new search regions. The perturbation phase is applied as a means of strong diversification to get out of deep local optimum traps. To obtain an efficient implementation of the proposed algorithm, we developed streamlining techniques based on bucket sorting. We demonstrated the effectiveness of the MOH algorithm both in terms of solution quality and computational efficiency on the two sets of well-known benchmarks composed of 91 instances. For the general max-k-cut problem, the proposed algorithm is able to improve 90 percent of the current best known results available in the literature. Moreover, for the very popular special case with $k = 2$, i.e., the max-cut problem, MOH also performs well by discovering 4 improved best results which were never reported by any max-cut algorithm of the literature. We also investigated the importance of the bucket sorting technique as well as alternative strategies for combing search operators and justified the combinations adopted in the proposed MOH algorithm.

The iterated tabu search (ITS) algorithm for the maximum bisection problem achieved a high level performance by including two distinct search operators which are applied in three search phases. The descent-based improvement phase uses the vertex move operator (*1-move*) to discover a first local optimum from a starting solution. The diversifying improvement phase jointly employs the *1-move* operator and a constrained swap operator in a probabilistic way (under the tabu search framework) to discover better solutions. The perturbation phase is applied as a means of strong diversification to get out of deep local optimum traps. To obtain an efficient implementation of the proposed algorithm, we developed streamlining techniques and a LIFO tie-breaking strategy based on dedicated bucket structures. Experimental assessments on the 71 well-known benchmark instances with up to 20000 vertices indicated that the proposed ITS algorithm was able to obtain improved best results (new lower bounds) for 10 large instances and match the best-known results for all the other instances. Comparisons with state-of-the-art algorithms showed that the ITS algorithm was superior to the reference algorithms both in terms of solution quality and computational efficacy. Furthermore, the main ingredients of the ITS algorithm were analyzed to shed lights on their impacts over the performance of the algorithm.

Finally, we presented an effective path relinking algorithm for solving the vertex separator problem, which integrates a reference set initialization and updating method, a solution improvement method, a path generation method as well as a solution selection method. The iterated tabu search based solution

improvement method applies complementary *1-move* and *swap-move* operators to cooperatively exploit search space, where the benefit of the innovative *swap-move* operator is demonstrated both qualitatively and experimentally. The path generation method employs a dedicated strategy based on a greedy selection criterion to build path solutions. The proposed algorithm is benchmarked on four sets consisting of a total of 355 instances. Comparisons with the best performing algorithms in the literature disclose that our algorithm finds improved solutions for 67 large instances and matches previously best known results for all but one instance, while attaining this remarkable performance within competitive or an order of magnitude shorter time.

Perspectives

From the work presented in this thesis, several interesting perspectives can be contemplated for future studies to reinforce the performance of the proposed algorithms.

The multilevel graph partitioning framework, which approximates the initial problem by solving a series of smaller (and easier) problems, is shown to be dramatically effective for finding near optimal k-way partition on large graphs. In addition to the coarsening phase used for producing a series of coarser graphs, another key ingredient in the multilevel framework is the refinement phase for acquiring improved partitions. Furthermore, no previous study has developed heuristic or metaheuristic algorithms within a multilevel framework for coping with max-k-cut, max-bisection and vertex separator problems. As such, it would be interesting to integrate algorithms proposed in this thesis as the refinement phase into a multilevel framework and assess its performance.

For the purpose of achieving search diversification, the core element of producing initial solutions in our proposed algorithms is randomness. Despite its effectiveness, this rudimentary strategy has the drawback of being not able to promptly and precisely migrate search into more promising area. Hence, it's worthy of investigating more advanced initial solution generation methods to discover best solutions in reduced computational efforts. A possible advancement is to make use of auxiliary memories that collect history information to guide the search, including an elite set of high quality solutions, a memory to summarize frequency of each vertex lying in the same subset in the elite set, as well as long-term and short-term memories to record occurrence of each vertex lying in each subset, etc. Dependent on a collection of memories, a probability model is built to determine initial solutions for instance.

Mathematical programming approaches are able to provide robust solutions while metaheuristic approaches are capable of returning sub-optimal solutions with time effectiveness. Hence, developing a matheuristic approach that exploits mathematical programming techniques in a metaheuristic framework is another promising search direction. For example, we can use metaheuristic methods to produce a set of sub-optimal solutions, based on which variable fixing techniques operate to reduce the original problem to the size exact methods are able to effectively tackle with. Alternatively, high quality lower bounds can be computed by use of metaheuristic approaches, with which exact methods can better prune its search tree to enhance search efficacy.

The high and robust performance of the proposed algorithms depend critically on a set of good parameters, whose optimal settings are usually instance independent. However, parameter tuning is normally a hard task, especially when a number of sensitive parameters exist. Hence, developing an automatic parameter tuning method based on the characteristics of the current instance to be solved could be a very favorable research for our future work.

Our proposed multiple operator heuristic establishes an original framework and presents attractive per-

formance. We hope to adapt this search scheme to other combinatorial problems so as to evaluate its usefulness.

List of Figures

List of Tables

List of Publications Submitted journal papers

- Fuda Ma and Jin-Kao Hao, A multiple search operator heuristic for the max-k-cut problem. Submitted to Annals of Operations Research (Submission Feb. 2015, Revision March 2016).
- Fuda Ma, Jin-Kao Hao and Yang Wang, An effective iterated tabu search for the maximum bisection problem. Submitted to Computers & Operations Research (Submission Nov. 2015, Revision Feb. 2016).
- Fuda Ma, Jin-Kao Hao and Yang Wang, An effective path relinking algorithm for the vertex separator problem. Submitted to Knowledge-Based Systems (Submission April 2016).

References

[Anjos *et al.*, 2013] M.F. Anjos, F. Liers, G. Pardella, and A. Schmutzer. Engineering branch-and-cut algorithms for the equicut problem. In *Discrete Geometry and Optimization*, pages 17–32. 2013. 12

[Armbruster *et al.*, 2008] M. Armbruster, M. Fügenschuh, C. Helmberg, and A. Martin. A comparative study of linear and semidefinite branch-and-cut methods for solving the minimum graph bisection problem. In *Integer Programming and Combinatorial Optimization, IPCO'08*, pages 112–125, 2008. 12

[Armbruster *et al.*, 2012] M. Armbruster, M. Fügenschuh, C. Helmberg, and A. Martin. LP and SDP branch-and-cut algorithms for the minimum graph bisection problem: a computational comparison. *Mathematic Programming Computation*, 4:275–306, 2012. 12

[Arráiz and Olivo, 2009] E. Arráiz and O. Olivo. Competitive simulated annealing and tabu search algorithms for the max-cut problem. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pages 1797–1798, 2009. 10

[Balas and de Souza, 2005] E. Balas and C.C. de Souza. The vertex separator problem: a polyhedral investigation. *Mathematical Programming*, 3:583–608, 2005. 15

[Barahona *et al.*, 1988] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36:493–513, 1988. 6, 12

[Bartz-Beielstein *et al.*, 2010] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss. *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, 2010. 53

[Battiti and Bertossi, 1997] R. Battiti and A. Bertossi. Differential greedy for the 0-1 equicut problem. In *Proceedings of DIMACS Workshop Network Design: Connectivity and Facilities Location*, 1997. 13

[Battiti and Bertossi, 1999] R. Battiti and A. Bertossi. Greedy, prohibition, and reactive heuristics for graph partitioning. *Computers, IEEE Transactions on*, 48:361–385, 1999. 12

[Benlic and Hao, 2011a] U. Benlic and J.K. Hao. An effective multilevel tabu search approach for balanced graph partitioning. *Computers & Operations Research*, 38:1066–1075, 2011. 8

[Benlic and Hao, 2011b] U. Benlic and J.K. Hao. A multilevel memetic approach for improving graph k-partitions. *Evolutionary Computation, IEEE Transactions on*, 15:624–642, 2011. 8, 80

[Benlic and Hao, 2013a] U. Benlic and J.K. Hao. Breakout local search for the max-cut problem. *Engineering Applications of Artificial Intelligence*, 26:1162–1173, 2013. 11, 26, 27, 28, 29, 44, 53

[Benlic and Hao, 2013b] U. Benlic and J.K. Hao. Breakout local search for the vertex separator problem. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 461–467, 2013. 15, 66, 72, 74, 75

[Benlic and Hao, 2013c] U. Benlic and J.K. Hao. Hybrid metaheuristics for the graph partitioning problem. In *Hybrid Metaheuristics. Studies in Computational Intelligence*, volume 434, pages 157–184. 2013. 8, 79

[Biha and Meurs, 2011] M.D. Biha and M.J. Meurs. An exact algorithm for solving the vertex separator problem. *Journal of Global Optimization*, 49:425–434, 2011. 14, 15, 72, 74, 75

[Bodlaender and Jansen, 2000] H.L. Bodlaender and K. Jansen. On the complexity of the maximum cut problem. *Nordic Journal of Computing*, 7:14–31, 2000. 9

[Boros and Hammer, 1991] E. Boros and P. Hammer. The max-cut problem and quadratic 0-1 optimization: Polyhedral aspects, relaxations and bounds. *Annals of Operations Research*, 33:151–180, 1991. 6

[Brunetta *et al.*, 1997] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78:243–263, 1997. 12

[Bui and Jones, 1992] T.N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, 1992. 14

[Bui and Moon, 1996] T.N. Bui and B.R. Moon. Genetic algorithm and graph partitioning. *Computers, IEEE Transactions on*, 45:841–855, 1996. 13

[Burer and Monteiro, 2001] S. Burer and R.D. Monteiro. A projected gradient algorithm for solving the maxcut SDP relaxation. *Optimization Methods and Software*, 15:175–200, 2001. 9

[Burer *et al.*, 2002] S. Burer, R.D.C. Monteiro, and Y. Zhang. Rank-two relaxation heuristics for max-cut and other binary quadratic programs. *SIAM Journal on Optimization*, 12:503–521, 2002. 9, 26

[Chang and Du, 1987] K.C. Chang and D.H.C. Du. Efficient algorithms for layer assignment problem. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6:67–78, 1987. 6, 12

[Chardaire *et al.*, 2007] P. Chardaire, M. Barake, and G.P. McKeown. A probe-based heuristic for graph partitioning. *Computers, IEEE Transactions on*, 56:1707–1720, 2007. 13

[Chen and Glover, 2016] Y. Chen and F. Glover. An evolutionary path relinking approach for the quadratic multiple knapsack problem. *Knowledge-Based Systems*, 92:23–34, 2016. 64

[Chen *et al.*, 1983] R.W. Chen, Y. Kajitani, and S.P. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *Circuits and Systems, IEEE Transactions on*, 30:284–299, 1983. 6

[Cho *et al.*, 1998] J.D. Cho, S. Raje, and M. Sarrafzadeh. Fast approximation algorithms on maxcut, k-coloring, and k-color ordering for VLSI applications. *Computers, IEEE Transactions on*, 47:1253–1266, 1998. 6, 12

[Conforti *et al.*, 1990a] M. Conforti, M.R. Rao, and A. Sassano. The equipartition polytope I: Formulations, dimention and basic facets. *Mathematical Programming*, 49:49–70, 1990. 12

[Conforti *et al.*, 1990b] M. Conforti, M.R. Rao, and A. Sassano. The equipartition polytope II: Valid inequalities and facets. *Mathematical Programming*, 49:71–90, 1990. 12

[Cordeau and Maischberger, 2012] J.F. Cordeau and M. Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39:2033–2050, 2012. 44

[Cormen *et al.*, 2001] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press Cambridge, 2001. 23

[Croce *et al.*, 2007] F. Della Croce, M.J. Kamiński, and V.Th. Paschos. An exact algorithm for max-cut in sparse graphs. *Operations Research Letters*, 35:403–408, 2007. 9

[Dang *et al.*, 2002] C. Dang, L. He, and I.K. Hui. A deterministic annealing algorithm for approximating a solution of the max-bisection problem. *Neural Networks*, 15:441–458, 2002. 13

[de Souza and Balas, 2005] C.C. de Souza and E. Balas. The vertex separator problem: algorithms and computations. *Mathematical Programming*, 3:609–631, 2005. 15, 72, 74, 75

[de Souza and Cavalcante, 2011] C.C. de Souza and V.F. Cavalcante. Exact algorithms for the vertex separator problem in graphs. *Networks*, 57:212–230, 2011. 15

[Delling *et al.*, 2015] D. Delling, D. Fleischman, A.V. Goldberg, I. Razenshteyn, and R.F. Werneck. An exact combinatorial algorithm for minimum graph bisection. *Mathematical Programming*, 153:417–458, 2015. 12, 57, 58

[Ding *et al.*, 2001] C.H. Ding, X. He, H. Zha, M. Gu, and H.D. Simon. A min-max cut algorithm for graph partitioning and data clustering. *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 107–114, 2001. 6, 12

[Eisenblätter, 2002] A. Eisenblätter. The semidefinite relaxation of the k-partition polytope is strong. *Integer Programming and Combinatorial Optimization*, 25:273–290, 2002. 6

[Elf *et al.*, 2003] M. Elf, M. Jünger, and G. Rinaldi. Minimizing breaks by maximizing cuts. *Operations Research Letters*, 31:343–349, 2003. 12

[Evrendilek, 2008] C. Evrendilek. Vertex separator for partitioning a graph. *Sensors*, 8:635–657, 2008. 14

[Falkner *et al.*, 1994] J. Falkner, F. Rendl, and H. Wolkowicz. A computational study of graph partitioning. *Mathematical Programming*, 66:211–239, 1994.

[Fan and Pardalos, 2010] N. Fan and Panos M. Pardalos. Linear and quadratic programming approaches for the general graph partitioning problem. *Journal of Global Optimization*, 48:57–71, 2010. 7

[Feige *et al.*, 2005] U. Feige, M. Hajiaghayi, and J.R. Lee. Improved approximation algorithms for minimum weight vertex separators. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 563–572, 2005. 14

[Ferreira *et al.*, 1996] C.E. Ferreira, A. Martin, C.C. deSouza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem: formulations and valid inequalities. *Mathematical Programming*, 74:247–266, 1996. 7

[Ferreira *et al.*, 1998] C.E. Ferreira, A. Martin, C.C. deSouza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem: a computational study. *Mathematical Programming*, 81:229–256, 1998. 7

[Festa *et al.*, 2002] P. Festa, P. M. Pardalos, M. G. C. Resende, and C. C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 7:1033–1058, 2002. 10, 13, 26, 53

[Fiduccia and Mattheyses, 1982] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. *Design Automation, 1982. 19th Conference on*, pages 175–181, 1982. 6, 7, 23, 47, 49, 55

[Frieze and Jerrum, 1995] A. Frieze and M. Jerrum. Improved approximation algorithms for max-k-cut and max bisection. *Proceedings of the Fourth IPCO Conference*, pages 1–13, 1995. 12

[Fu and Chen, 2006] B. Fu and Z. Chen. Sublinear time width-bounded separators and their application to the protein sidechain packing problem. In *Algorithmic Aspects in Information and Management, LNCS*, volume 4041, pages 149–160. 2006. 14

[Fukuyama, 2006] J. Fukuyama. NP-completeness of the planar separator problems. *Journal of Graph Algorithms and Applications*, 10:317–328, 2006. 14

[Ghaddar *et al.*, 2011] B. Ghaddar, M.F. Anjos, and F. Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, 188:155–174, 2011. 6

[Glover and Laguna, 1999] F. Glover and M. Laguna. *Tabu search*. Springer, 1999. 22, 51, 70

[Glover *et al.*, 2000] F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path-relinking. *Control and Cybernetics*, 39:654–684, 2000. 64, 70

[Glover *et al.*, 2003] F. Glover, M. Laguna, and R. Marti. Scatter search and path relinking: advances and applications. *Handbook of Metaheuristics*, 57:1–35, 2003. 64

[Glover *et al.*, 2004] F. Glover, M. Laguna, and R. Marti. Scatter search and path relinking: foundations and advanced designs. *New Optimization Technologies in Engineering*, 141:87–100, 2004. 64

[Goemans and Williamson, 1995] M. X. Goemans and D.P. Williamson. Improved approximation algorithms for max-cut and satisfiability problems using semidefinite programming. *Journal of the Acm*, 42:1115–1145, 1995. 9, 12

[Hagen and Kahng, 1997] L. W. Hagen and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16:1199–1205, 1997. 50

[Hager and Hungerford, 2015] W.W. Hager and J.T. Hungerford. Continuous quadratic programming formulations of optimization problems on graphs. *European Journal of Operational Research*, 240:328–337, 2015. 15

[Halperin and Zwick, 2002] E. Halperin and U. Zwick. A unified framework for obtaining improved approximation algorithms for maximum graph bisection problem. *Random Structures & Algorithms*, 20:382–402, 2002. 12

[Hendrickson and Leland, 1995] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on super-computing (CDROM)*, 1995. 8

[Homer and Peinado, 1997] S. Homer and M. Peinado. Design and performance of parallel and distributed approximation algorithms for maxcut. *Journal of Parallel and Distributed Computing*, 46:48–61, 1997. 9

[Inayoshi and Manderick, 1994] H. Inayoshi and B. Manderick. The weighted graph bi-partitioning problem: A look at GA performance. In *Parallel Problem Solving From Nature – PPSN III, Lecture Notes in Computer Science*, volume 866, pages 617–625, 1994. 13

[Kahruman *et al.*, 2007] S. Kahruman, E. Kolotoglu, S. Butenko, and I.V. Hicks. On greedy construction heuristics for the max-cut problem. *International Journal of Computational Science and Engineering*, 3:211–218, 2007. 9

[Kann *et al.*, 1997] V. Kann, S. Khanna, J. Lagergren, and A. Panconesi. On the hardness of approximating max k-cut and its dual. *Chicago Journal of Theoretical Computer Science*, 2:151–180, 1997. 6

[Karisch *et al.*, 2000] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *Informs Journal on Computing*, 12:177–191, 2000. 12

[Karp, 1972] R.M. Karp. *Reducibility among combinatorial problems*. Springer, 1972. 6

[Karypis and Kumar, 1998] G. Karypis and V. Kumar. Metis 4.0: unstructured graphs partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, 1998. 7, 8

[Karypis and Kumar, 1999] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Review*, 41:278–300, 1999. 6

[Karypis and Kumar, 2000] G. Karypis and V. Kumar. Multilevel k-way hypergraph partitioning. *VLSI Design*, 11:285–300, 2000.

[Kernighan and Lin, 1970] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–308, 1970. 7, 47

[Kochenberger *et al.*, 2013] G. Kochenberger, J.K. Hao, Z. Lü, H. Wang, and F. Glover. Solving large scale max cut problems via tabu search. *Journal of Heuristics*, 19:565–571, 2013. 11, 27, 29

[Kohmoto *et al.*, 2003] K. Kohmoto, K. Katayama, and H. Narihisa. Performance of a genetic algorithm for the graph partitioning problem. *Mathematical and Computer Modelling*, 38:1325–1332, 2003. 13

[Krislock *et al.*, 2014] N. Krislock, J. Malick, and F. Roupin. Improved semidefinite bounding procedure for solving max-cut problems to optimality. *Mathematical Programming*, 143:61–86, 2014. 10

[Lacomme *et al.*, 2015] P. Lacomme, C. Prins, C. Prodhon, and L. Ren. A multi-start split based path relinking MSSPR approach for the vehicle routing problem with route balancing. *Engineering Applications of Artificial Intelligence*, 38:237–251, 2015. 64

[Lai and Hao, 2015] X. Lai and J.K. Hao. Path relinking for the fixed spectrum frequency assignment problem. *Expert Systems with Applications*, 42:4755–4767, 2015. 64, 66

[Leighton and Rao, 1999] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46:787–832, 1999. 14

[Leighton, 1983] F.T. Leighton. *Complexity issues in VLSI: optimal layout for the shuffle-exchange graph and other networks*. MIT Press, Cambridge, 1983. 14

[Liers *et al.*, 2004] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard ising spin glass problems by branch-and-cut. *New Optimization Algorithms in Physics*, 50:47–68, 2004. 6

[Lin and Zhu, 2012] G. Lin and W. Zhu. A discrete dynamic convexized method for the max-cut problem. *Annals of Operations Research*, 196:371–390, 2012. 10

[Lin and Zhu, 2014] G. Lin and W. Zhu. An efficient memetic algorithm for the max-bisection problem. *Computers, IEEE Transactions on*, 63:1365–1376, 2014. 13, 47, 53, 55

[Ling *et al.*, 2008] A. Ling, C. Xu, and L. Tang. A modified VNS metaheuristic for max-bisection problems. *Journal of Computational and Applied Mathematics*, 220:413–421, 2008. 13

[Lipton and Tarjan, 1979] R.J. Lipton and R.E Tarjan. a separator theorem for planar graphs. *SIAM Journal on Numerical Analysis*, 36:177–189, 1979. 14

[Lisser and Rendl, 2003] A. Lisser and F. Rendl. Telecommunication clusterig using linear and semidefinite programming. *Mathematical Programming*, 95:91–101, 2003.

[López-Ibánez *et al.*, 2011] M. López-Ibánez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical report, 2011. 53

[Lourenço *et al.*, 2010] H.R. Lourenço, O. Martin, and T. & Stützle. Iterated local search: framework and applications. *International Series in Operations Research & Management Science*, 146:363–397, 2010. 44

[Lü *et al.*, 2011a] Z. Lü, F. Glover, and J.K. Hao. Neighborhood combination for unconstrained binary quadratic problems. In *MIC Post-Conference Book*, pages 49–61. 2011. 38

[Lü *et al.*, 2011b] Z.P. Lü, J.K. Hao, and F. Glover. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, 17:97–118, 2011. 51

[Marti *et al.*, 2009] R. Marti, A. Duarte, and M. Laguna. Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing*, 21:26–38, 2009. 10

[Merz and Freisleben, 2000] P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8:61–91, 2000. 13

[Mitchell, 2001] J.E. Mitchell. Branch-and-cut for the k-way equipartition problem. *Technical report, Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180*, 2001. 7

[Mitchell, 2003] J.E. Mitchell. Realignment in the national football league: Did they do it right? *Naval Research Logistics (NRL)*, 50:683–701, 2003. 6, 7

[Mohar and Poljak, 1990] B. Mohar and S. Poljak. Eigenvalues and the max-cut problem. *Czechoslovak Mathematical Journal*, 40:343–353, 1990. 9

[Monien *et al.*, 2000] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. *Parallel Computing*, 26:1609–1634, 2000. 8

[Murty and Kabadi, 1987] K.G. Murty and S.N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical programming*, 39:117–129, 1987. 12

[Orlova and Dorfman, 1972] G.I. Orlova and Y.G. Dorfman. Finding the maximum cut in a graph. *Engineering Cybernetics*, 10:502–506, 1972. 9

[Palubeckis *et al.*, 2014] G. Palubeckis, A. Ostreika, and D. Rubliauskas. Maximally diverse grouping: an iterated tabu search approach. *Journal of the Operational Research Society*, 66:579–592, 2014. 44

[Palubeckis, 2004] G. Palubeckis. Application of multistart tabu search to the maxcut problem. *Information Technology and Control*, 2:29–35, 2004. 10

[Peng *et al.*, 2015] B. Peng, Z. Lü, and T. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53:154–164, 2015. 64

[Pinter, 1984] R.Y. Pinter. Optimal layer assignment for interconnect. *Journal of VLSI and Computer Systems*, 1:123–137, 1984. 6

[Qin *et al.*, 2015] T. Qin, B. Peng, U. Benlic, T.C.E. Cheng, Y. Wang, and Z.P. Lü. Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling. *Computers & Operations Research*, 61:81–88, 2015. 44

[Rahimian *et al.*, 2015] F. Rahimian, A.H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi. A distributed algorithm for large-scale graph partitioning. *ACM Transactions on Automatic and Adaptive Systems*, 10:12:1–12:24, 2015. 7

[Rendl *et al.*, 2007] F. Rendl, G. Rinaldi, and A. Wiegele. A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations. In *International Conference on Integer Programming and Combinatorial Optimization*, volume 4513, pages 295–309, 2007. 9

[Rendl *et al.*, 2010] F. Rendl, G. Rinaldi, and A. Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121:307–335, 2010. 10

[Sahni and Gonzalez, 1976] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Acm*, 23:555–565, 1976. 9

[Sánchez-Oro *et al.*, 2014] J. Sánchez-Oro, N. Mladenović, and A. Duarte. General variable neighborhood search for computing graph separators. *Optimization Letters*, 2014. 15, 66, 72, 74, 75, 79

[Scott and Sorkin, 2004] A.D. Scott and G.B. Sorkin. Solving sparse semi-random instances of max-cut and max-CSP in linear expected time. Research report 23417 (w0411-056), IBM Research division, Thomas J. Watson Research Center, 2004. 9

[Shylo and Shylo, 2010] V.P. Shylo and O.V. Shylo. Solving the maxcut problem by the global equilibrium search. *Cybernetics and System Analysis*, 46:744–754, 2010. 10

[Shylo *et al.*, 2012] V.P. Shylo, O.V. Shylo, and V.A. Roschyn. Solving weighted max-cut problem by gloabl equilibrium search. *Cybernetics and System Analysis*, 48:563–567, 2012. 10, 26, 27, 28, 29, 53

[Shylo *et al.*, 2015] V. Shylo, F. Glover, and I. Sergienko. Teams of global equilibrium search algorithms for solving the weighted maximum cut problem in parallel. *Cybernetics and Systems Analysis*, 51:16–24, 2015. 26, 29, 53

[Silva *et al.*, 2015] M.M. Silva, A. Subramanian, and L.S. Ochi. An iterated local search heuristic for the split delivery vehicle routing problem. *Computers & Operations Research*, 53:234–249, 2015. 44

[Soper *et al.*, 2004] A.J. Soper, C. Walshaw, and M. Cross. A combined evolutionary search and multilevel optimization approach to graph-partitioning. *Journal of Global Optimization*, 29:225–241, 2004. 8

[Steenbeek *et al.*, 1998] A.G. Steenbeek, E. Marchiori, and A.E. Eiben. Finding balanced graph bi-partitions using a hybrid genetic algorithm. In *Proceedings of the IEEE International Conference on Evolutionary Computation ICEC'98*, pages 90–95, 1998. 13

[Wang *et al.*, 2012] Y. Wang, Z. Lu, F. Glover, and J.K. Hao. Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, 223:595–604, 2012. 11, 64

[Wang *et al.*, 2013] Y. Wang, Z. Lü, F. Glover, and J.K. Hao. Probabilistic grasp-tabu search algorithms for the ubqp problem. *Computers & Operations Research*, 40:3100–3107, 2013. 26, 27, 28, 29, 53

[Wu and Hao, 2012] Q. Wu and J.K. Hao. A memetic approach for the max-cut problem. In *Parallel Problem Solving from Nature - PPSN XII*, volume 7492 of *Lecture Notes in Computer Science*, pages 297–306, 2012. 11, 26, 27, 28, 29, 53

[Wu and Hao, 2013] Q. Wu and J.K. Hao. Memetic search for the max-bisection problem. *Computers & Operations Research*, 40:166–179, 2013. 13, 26, 27, 28, 29, 47, 53, 55

[Wu *et al.*, 2015] Q. Wu, Y. Wang, and Z. Lü. A tabu search based hybrid evolutionary algorithm for the max-cut problem. *Applied Soft Computing*, 34:827–837, 2015. 26, 29

[Xu *et al.*, 2011] F. Xu, X. Ma, and B. Chen. A new Lagrangian net algorithm for solving max-bisection problems. *Journal of Computational and Applied Mathematics*, 235:3718–3723, 2011. 13, 53, 55

[Ye, 2001] Y. Ye. A .699-approximation algorithm for max-bisection. *Mathematical Programming*, 90:101–111, 2001. 12

[Zadegan *et al.*, 2013] S.M.R. Zadegan, M. Mirzaie, and F. Sadoughi. Ranked k-medoids: A fast and accurate rank-based partitioning algorithm for clustering large datasets. *Knowledge-Based Systems*, 39:133–143, 2013. 80

[Zhou *et al.*, 2015] Y. Zhou, J.K. Hao, and A. Goëffon. A three-phased local search approach for the clique partitioning problem. *Journal of Combinatorial Optimization*, pages 1–23, 2015. 80

[Zhu *et al.*, 2013] W. Zhu, G. Lin, and M.M. Ali. Max-k-cut by the discrete dynamic convexized method. *INFORMS Journal on Computing*, 25:27–40, 2013. 6, 26, 27

UNIVERSITE BRETAGNE LOIRE

Thèse de Doctorat

Fuda MA

Heuristiques à opérateurs multiples pour des problèmes de partitionnement de graphe

Multiple Operator Metaheuristics for Graph Partitioning Problems

Résumé
Les problèmes de partitionnement de graphe sont une classe bien connue des problèmes d'optimisation combinatoire NP-difficiles avec un large éventail d'applications, telles que la conception de circuits intégrés, la physique statistique, la planification d'équipe sportive, la segmentation d'images et la structuration de protéines. En raison de la grande complexité de ces problèmes, les approches heuristiques et métaheuristiques sont couramment utilisées pour trouver des solutions approchées. Cette thèse considère trois problèmes représentatifs de cette famille, incluant le problème "max-k-cut", le problème "max-bisection" et le problème de séparation de sommets (VSP). Elle vise à élaborrer des algorithmes heuristiques efficaces basés sur une ensemble d'opérateurs de recherche complémentaires. Plus précisément, nous développons une heuristique à opérateurs multiples (MOH) pour "max-k-cut", un algorithme de recherche Tabu itérée (ITS) pour "max-bisection" et un algorithme "path relinking" (PR-VSP) pour VSP. Des résultats exprimentaux sur des jeux de test standard démontrent que les algorithmes proposés rivalisent favorablement avec les approches existantes de la littérature. L'utilisation combinée de plusieurs opérateurs de recherche est analysée afin de mettre en évidence l'influence de ces opérateurs sur la performance des algorithmes.

Mots clés
Optimisation combinatoire, Problèmes de partitionnement de graphe, Stratégies recherche à opérateur multiple, Heuristiques et métaheuristiques.

Abstract
Graph partitioning problems are a class of well-known NP-hard combinatorial optimization problems with a wide range of applications, such as VLSI layout design, statistical physics, sports team scheduling, image segmentation, and protein conformation for instances. This thesis considers three representative problems in this family, including the max-k-cut problem, the max-bisection problem and the vertex separator problem (VSP). Due to high computational complexity, heuristic and metaheuristic approaches are commonly used for approximating the challenging problems. This thesis is devoted to developing efficient metaheuristic algorithms based on a collection of complementary search operators. Specifically, we develop a multiple operator heuristic (MOH) for max-k-cut, an iterated tabu search (ITS) algorithm for max-bisection and a path relinking (PR-VSP) algorithm for VSP. Extensive computational experiments and comparisons demonstrate that the proposed algorithms compete favorably with state-of-the-art approaches in the literature. The combined use of multiple search operators is analyzed to shed lights on the influence over the performance of the algorithms.

Key Words
Combinatorial optimization, Graph partitioning problems, Multiple search strategies, Heuristics and metaheuristics.