Metaheuristics for large binary quadratic optimization and its applications

Yang Wang

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

université angers

Thèse de Doctorat

Yang WANG

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Informatique
Laboratoire : Laboratoire d'Étude et de Recherche en Informatique d'Angers (LERIA)

Soutenue le fevrier 2013

École doctorale : 503 (STIM)
Thèse n° : 1306

Metaheuristics for large binary quadratic optimization and its applications

Métaheuristiques pour l'optimization quadratique en 0/1 à grande échelle et ses applications

JURY

Rapporteurs :	**M. Alexandre CAMINADA**, Professeur, Université de Technologie de Belfort-Montbéliard **M. Gary KOCHENBERGER**, Professor, University of Colorado Denver
Examinateurs :	**M. Jean-Charles BILLAUT**, Professeur, Ecole Polytechnique de l'Université de Tours **M. David LESAINT**, Professeur, Université d'Angers
Directeur de thèse :	**M. Jin-Kao HAO**, Professeur, Université d'Angers

Contents

General Introduction

Context

Given a set of objects with profits (including negative profits) assigned not only to separate objects but also to pairs of them, the Binary Quadratic Optimization (BQO) problem consists in finding a subset of objects to maximize the overall profits.

BQO is a well-known NP-hard combinatorial optimization problem and provides a variety of applications, including financial analysis, social psychology, machine scheduling, computer aided design, statistical mechanics, traffic management, molecular conformation and cellular radio channel allocation. Moreover, BQO can be served as a unified model for many combinatorial optimization problems, such as graph coloring, maximum cut, set packing, maximum independent set, maximum clique, maximum edge weight clique, linear ordering and generalized independent set problems, etc.

Exact methods (e.g. branch and bound, branch and cut, lagrangean decompositions and column generation) are quite useful to obtain optimal solutions for problem instances of limited sizes. However, because of the high computational complexity, heuristic and metaheuristic algorithms are practically used to produce approximate solutions to larger problem instances.

This thesis is devoted to developing effective metaheuristic algorithms for solving the BQO problem. Meantime, we undertake to tackle combinatorial optimization problems that can be transformed into the form of BQO, with a direct application or a trivial adaptation of our developed algorithms for BQO.

Objectives

The first objective of this thesis is to solve large BQO problem instances by drawing on approaches (e.g., variable fixation or multilevel framework) that are capable of reducing the scale of an original problem so as to carry out extensive exploitation in a decreased search area. As a result, we proposed backbone guided tabu search (BGTS) algorithms on the basis of variable fixation technique and developed a backbone multilevel memetic algorithm (BMMA) following the traditional multilevel framework.

The second objective is focused on generating preferable initial solutions for efficiently exploring search space of BQO. For this end, we allow for the greedy random adaptive construction, the restart/recovery strategy and the path relinking approach, where both restart/recovery and path relinking are fundamental principles underlying tabu search and especially attract us given that it was never studied before on BQO. The use of each foregoing method gave rise to GRASP-Tabu Search (GRASP-TS), GRASP-Tabu Search with Population Management (GRASP-TS/PM) as well as Path Relinking (PR) algorithms.

The third objective consists in investigating new applications of BQO. To achieve this goal, we consider problems including maximum cut (MaxCut), maximum clique (MCP), maximum vertex weight clique (MVWCP) and minimum sum coloring (MSCP). We transformed them into the formulation of BQO and then effectively solved them with our proposed algorithms for BQO.

Our final objective is to propose a highly effective search algorithm for dealing with the cardinality constrained binary quadratic optimization problem (also known as maximum diversity problem (MDP)). For this purpose, we devised a tabu search based memetic algorithm (TS/MA) in which the tabu search component utilizes a successive filter candidate list strategy and is joined with a solution combination strategy based on identifying strongly determined and consistent variables.

Contributions

The main contributions of this thesis are the following:

- We proposed a backbone guided tabu search framework that interweaves a tabu search phase with a variable fixing/freeing phase. Within this framework, we developed four BGTS algorithms for solving BQO by combining different variable scoring and fixing/freeing rules. Specifically, we designed two variable scoring rules based on the variable contribution to a set of solutions, with distinction whether each solution in the set is treated equally. Meanwhile, we devised two variable fixing/freeing rules, one inheriting the backbone components obtained from the historic fixing/freeing phases while the other reconsidering all the backbone components according to the current fixing/freeing phase. Experimental results showed that one of the developed BGTS algorithms is capable of matching the best known results for all the tested instances and improving the performance of the basic TS in terms of both solution quality and computational efficiency. A further analysis provided explanation why one particular variable fixing/freeing and scoring rule led to better computational results than another one.

- We developed a backbone multilevel memetic algorithm to tackle large BQO problem instances. The proposed BMMA algorithm incorporates a backbone based coarsening phase, an asymmetric uncoarsening phase and a memetic refinement phase, where the backbone based procedure and the memetic refinement procedure make use of tabu search to obtain improved solutions. Experimental results and comparisons with other state-of-the-art algorithms indicated that BMMA is able to attain all the best known values with a computing effort less than any existing approach.

- We devised GRASP-TS and GRASP-TS/PM algorithms that hybrid GRASP with tabu search for BQO, where tabu search is used for solution improvement. GRASP-TS uses an adaptive random greedy function to construct an initial solution from the scratch. GRASP-TS/PM makes use of a restart/recovery strategy to produce a solution, in which partial solution components inherit corresponding elements of an elite

solution fetched from a population and the remaining solution components are rebuilt as in the GRASP-TS procedure. We also directly applied GRASP-TS and GRASP-TS/PM to solve the MaxCut problem after transforming MaxCut into the form of BQO. Furthermore, we conducted an adaptation and extension of the GRASP-TS algorithm (denoted by GRASP-TS/MCPs) to solve MCP and MVWCP reformulated as the form of BQO. Experiments on BQO, MaxCut, MCP and MVWCP problem instances indicated that our proposed algorithms are very competitive when comparing with other best algorithms in the literature, although being not special purpose algorithms tailored for MaxCut, MCP and MVWCP.

- We implemented two path relinking algorithms for BQO that comprise a reference set initialization method, a tabu search based improvement method, a reference set update method, a relinking method and a path solution selection method. The proposed algorithms differ from each other mainly on the way they generate the path, one employing a greedy strategy (PR1) and the other employing a random strategy (PR2). In addition, our PR algorithms were also employed to solve MaxCut and MSCP after transforming them into the formulation of BQO. We evaluated the performance of the proposed PR algorithms on BQO, MaxCut and MSCP problem instances and demonstrated that both PR1 and PR2 yielded highly competitive outcomes in comparison with the previous best known results from the literature.

- We presented an effective memetic algorithm based on tabu search for tackling cardinality constrained BQO. The tabu search component uses a successive filter candidate list strategy and the solution combination component employs a combination operator based on identifying strongly determined and consistent variables. Analysis of comparisons with state-of-the-art algorithms demonstrate statistically that our TS/MA algorithm competes very favorably with the best performing algorithms. Key elements and properties of TS/MA are also analyzed to disclose the source of its success.

Organization

The manuscript is organized in the following way:

- In the first chapter, we introduce the BQO problem and present its applications both in practical and combinatorial optimization problems. Then, we summarize basic ingredients of most local search implementations targeted at BQO. In addition, we place an emphasis on reviewing various heuristic and metaheuristic algorithms proposed for solving BQO. Finally, we present standard BQO benchmark families that are frequently used to evaluate performance of algorithms.

- In the second chapter, we first present a backbone guided tabu search framework which mainly consists of a tabu search phase and a variable fixing/freeing phase, where the variable fixing/freeing phase operates based on variables scoring rules and

variable fixing/freeing rules. Then we describe the tabu search procedure, two variable scoring rules and two variable fixing/freeing rules, where different combinations of scoring rules with fixing/freeing rules produce four BGTS algorithms. Finally, we evaluate our proposed algorithms on the challenging BQO benchmark instances and report experimental results.

- In the third chapter, we investigate a backbone multilevel memetic algorithm aiming at solving large BQO problem instances. Following the general multilevel scheme, we detail each component of our multilevel algorithm adapted for BQO and demonstrate the effectiveness of the proposed BMMA algorithm by providing experimental comparison between BMMA and other state-of-the-art algorithms.

- In the fourth chapter, we first describe GRASP-TS and GRASP-TS/PM algorithms for solving BQO. Then, we describe a GRASP-TS/MCPs algorithm, which adapts the proposed GRASP-TS, to solve the MCP and MVWCP problems transformed into the BQO form. Finally, we evaluate our proposed algorithms on benchmark instances from BQO, MaxCut, MCP and MVWCP problems and show comparisons with best preforming algorithms.

- In the fifth chapter, we present two path relinking algorithms for BQO. A general path relinking scheme for BQO is displayed, followed by the implementation of each ingredient in it. Besides, we illustrate how to recast the minimum sum coloring problem into the BQO formulation. Extensive computational results on BQO, MaxCut and MSCP benchmark instances are shown to demonstrate the effectiveness of the proposed path relinking algorithms.

- In the last chapter, we describe an effective memetic algorithm based on tabu search for tackling MDP. First, we present the main scheme of TS/MA and detail each component in it. Then we show our computational results and comparisons with the current best performing approaches. Finally, we analyze the key elements and properties of TS/MA.

Chapter 1

Introduction

This chapter introduces the binary quadratic optimization (BQO) problem and various domains where BQO is applied. Considering that local search is frequently incorporated as a major component into most algorithms we review for dealing with BQO, we then summarize basic ingredients of local search implementations used for BQO. Afterwards, we review various heuristic and metaheuristic algorithms proposed in the literature. Finally, we provide standard BQO benchmark families that are most often used to evaluate performance of various algorithms.

Contents

1.1 Binary quadratic optimization problem

1.1.1 Problem description

Given a symmetric $n \times n$ matrix $Q = (q_{ij})$, where $q_{ij} \in \Re$, the BQO problem is to identify a binary vector x of length n for the following objective [Beasley, 1998]:

$$\begin{aligned} &\text{Maximize:} \quad f(x) = x^t Q x = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j, \\ &\text{subject to:} \quad x_i \in \{0,1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{1.1}$$

Actually, the matrix Q corresponds to a graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E = [e_{ij}]$, where $e_{ij} = q_{ij}$ if $q_{ij} \neq 0$. Hence, an alternative objective of BQO is to partition V into two subsets V_0 and V_1 such that $\sum_{i \in V_1} e_{ii} + 2 \sum_{i \in V_1, j \in V_1} e_{ij}$ is maximized.

Fig. 1.1 shows an illustrative example to make transformation between the two above-mentioned definitions. The objective function value of this example equals $q_{22} + q_{44} + q_{66} + 2q_{24} + 2q_{26}$.

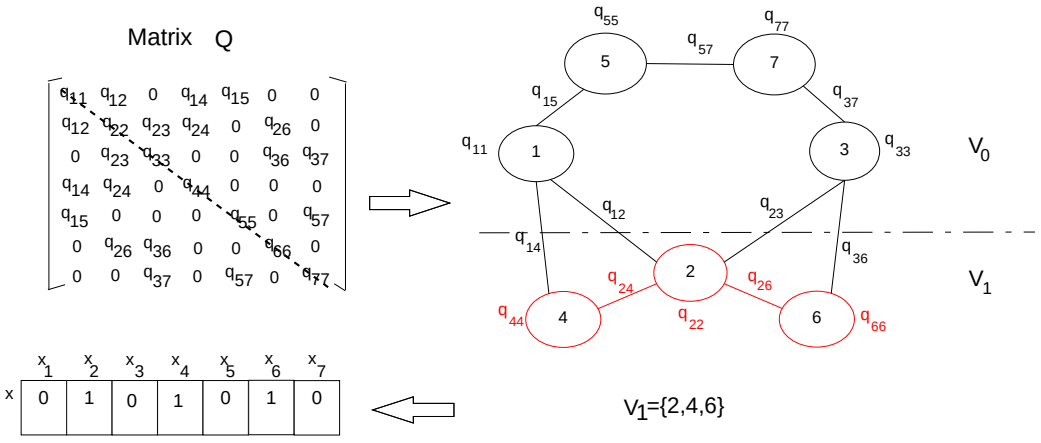


Figure 1.1: A graph example of illustrating the BQO problem

Binary quadratic optimization is also named as (unconstrained) quadratic bivalent programming, (unconstrained) quadratic zero-one programming, (unconstrained) quadratic pseudo boolean programming, unconstrained binary quadratic programming, binary quadratic programs, quadratic unconstrained binary optimization.

Imposing a cardinality constraint to BQO results in the notable maximum diversity problem (MDP) [Kuo *et al.*, 1993]. The objective of MDP can be formulated as follows:

$$\begin{aligned} \text{Maximize:} \quad & f(x) = x^t Q x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j, \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = m, \quad x_i \in \{0, 1\}, \quad i = 1, \ldots, n \end{aligned} \tag{1.2}$$

BQO is a canonical NP-hard combinatorial optimization problem. There is a limited number of its subclass that are polynomially solvable [Barahona, 1986; Jha and Pardalos, 1987; Picard, 1974]. Also, the problem of determining if an BQO problem has a unique global optimal solution remains NP-hard [Pardalos and Jha, 1992]. Furthermore, even if we know BQO contains a unique global optimum, this problem still belongs to the class of NP-hard [Pardalos and Jha, 1992].

1.1.2 Application domains

1.1.2.1 Practical problems

The formulation of BQO can represent problems in a variety of domains, including:

- financial analysis [Laughunn, 1970; McBride and Yormark, 1980]
- statistical mechanics [Barahona *et al.*, 1988]
- social psychology [Harary, 1953]
- computer aided design [Krarup and Pruzan, 1978]
- traffic management [Gallo *et al.*, 1980; Witzgall, 1975]
- machine scheduling [Alidaee *et al.*, 1994]
- cellular radio channel allocation [Chardaire and Sutter, 1994]
- molecular conformation [Phillips and Rosen, 1994]

For example, in the capital-budgeting problem, given a set of intercorrelated investment proposals and the covariances of each pairwise proposals, a decision-maker needs to select a portfolio of proposals to minimize the investment risk that is measured as covariances between proposal returns. In the statistical physics, given magnetic impurities and the energy interaction between two impurities, the mathematical model of minimizing the energy of the spin glass corresponds to the BQO formulation.

1.1.2.2 Combinatorial optimization problems

The BQO formulation has been served as a common model for many combinatorial optimization problems pertaining to graphs, such as:

- Graph coloring problem [Kochenberger *et al.*, 2005]
- Maximum cut problem [Kochenberger *et al.*, 2011]
- Set packing problem [Alidaee *et al.*, 2008]
- Set partitioning problem [Lewis *et al.*, 2008]
- Maximum independent set problem [Pajouh *et al.*, 2011]
- Maximum edge weight clique problem [Alidaee *et al.*, 2007]
- Linear ordering problems [Lewis *et al.*, 2009]
- Generalized independent set [Kochenberger *et al.*, 2007]

For example, the maximum cut problem can be naturally transformed into the BQO model. Given an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E \subseteq V \times V$, each edge $e(i, j)$ is associated with a weight w_{ij}, the maximum cut problem (MaxCut) asks for a partition of V into two disjoint subsets such that the total weight of cut (edges crossing the two subsets) is maximized. Formally, the objective function of MaxCut is:

$$\begin{aligned} &\text{Maximize:} \quad f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i (1 - x_j), \\ &\text{subject to:} \quad x_i \in \{0, 1\}, \quad i = 1, \ldots, n. \end{aligned} \tag{1.3}$$

The following corresponding relation is apparent in comparison with the formulation of BQO shown in Eq. 1.1:

$$q_{ii} = \sum_{j=1, j \neq i}^{n} w_{ij}, \qquad q_{ij} = -w_{ij}, \quad (i \neq j) \tag{1.4}$$

More generally, the method of reformulating the original problem into the BQO model consists in identifying a proper quadratic penalty function to replace each problem constraint and then adding all the penalty functions to the linear or quadratic objective function. A proper penalty function assures that its function value is zero when the constraint is satisfied. In particular, the BQO model mainly deals with the following two types of constraints.

$$\#1 : Ax = b \tag{1.5}$$

$$\#2 : x_i + x_j <= 1 \tag{1.6}$$

For the transformation of the constraints #1 and #2, the following penalty functions $p1$ and $p2$ are used, respectively:

$$p1(x) = P(Ax - b)^t (Ax - b) \tag{1.7}$$

$$p2(x) = P x_i x_j \tag{1.8}$$

where a positive scalar P is selected for the minimization problem while a negative scalar is used for the maximization problem.

Once a problem at hand is reformulated into the form of BQO, a general purpose algorithm for BQO can be used to address this specific problem, which avoids the necessity to design a new method for each new problem.

1.2 Previous work

Due to its theoretical significance as an NP-hard problem and immense potential applications, BQO has attracted researchers to design various solution procedures to tackle it. Exact algorithms (e.g., branch and bound [Pardalos, 1990; Billionnet and Sutter, 1994; Palubeckis, 1995; Hansen *et al.*, 2000], decomposition method [Chardaire and Sutter, 1994; Mauri and Lorena, 2011; Mauri and Lorena, 2012], linearization followed by linear 0-1 programming [Glover and Woolsey, 1973; Glover and Woolsey, 1974] and semidefinite relaxation based method [Helmberg and Rendl, 1998; Rendl *et al.*, 2006; Billionnet and Elloumi, 2007]) are capable of finding optimal solutions for problem instances with a maximum of 500 variables. However, because of the high computational complexity, various heuristics are commonly used to create approximate solutions to larger problem instances. Since this thesis is not focused on the exact algorithms, the following sections mainly review various heuristic and metaheuristic algorithms reported in the literature.

1.2.1 Basic preliminaries

Since local search is either directly used to solve BQO or a vital sub-routine embedded in most algorithms to improve solution quality, we first summarize basic ingredients of the local search procedure proposed for BQO before introducing various heuristics and metaheuristics.

1.2.1.1 Search space and evaluation function

A solution of BQO is a boolean vector of length n; i.e., $x = \{x_1, \ldots, x_n\} \in \Psi = \{0, 1\}^n$. Thus, the solution space Ψ is of size 2^n. Given a solution $x \in \Psi$, its quality or fitness is usually directly measured by the objective function $f(x)$ of Eq. 1.1.

1.2.1.2 Neighborhood structure

Neighborhood structure determines a set of solutions that can be directly reached from the current solution. The following two types of neighborhood structures are principally used for BQO:

1-flip: flip a variable x_i to its complementary value $1 - x_i$;

k-flip: flip k ($2 <= k <= n$) variables $x_{i_1}, \ldots, x_{i_k}$ simultaneously to the corresponding complementary values $1 - x_{i_1}, \ldots, 1 - x_{i_k}$, respectively.

The *1-flip* neighborhood is bounded by O(n). The size of *k-flip* neighborhood increases exponentially with k and is bounded by $\binom{n}{k}$. A local search algorithm with the complete *k-flip* neighborhood consumes enormous amounts of computational efforts when contrasted with *1-flip* neighborhood.

1.2.1.3 Move selection

To perform a move that transforms a solution x to its neighborhood solution, the following move selection strategies are generally utilized:

first improvement (FirstImp): scan the neighborhood of the current solution x and pick the first solution found better than x;

best improvement (BestImp): exhaustively explore the neighborhood of x and return the solution with the best solution quality.

1.2.1.4 Fast move evaluation

To enhance the efficiency of neighborhood exploitation, it is vital to quickly evaluate each solution in the neighborhood. In other words, one needs to find an efficient method to calculate move gain; i.e., the objective difference between a solution and its neighboring solution. In the following we first present how to quickly calculate the move gain for *1-flip* move [Glover and Hao, 2010] and then generalize it to the *k-flip* move [Katayama and Narihisa, 2004].

1.2.1.4.1 Fast evaluation for *1-flip* move Let x and x' denote two solutions where x' is a solution in the neighborhood of x and let $N = \{1, \ldots, n\}$ denote the index set for the set of variables $x = \{x_1, \ldots, x_n\}$. Then the move gain upon flipping a variable x_i can be calculated as follows.

First, we maintain a vector Δ to record move gains of transforming from the solution x to each neighboring solution x' when performing *1-flip* moves. Specifically, when flipping the variable x_i, the move gain Δ_i is calculated as follows:

$$\Delta_i = (1 - 2x_i)(q_{ii} + 2 \sum_{j \in N, j \neq i, x_j = 1} q_{(i,j)}) \tag{1.9}$$

Once a move is performed, we just need to update a subset of move gains affected by the move. Specifically, the following abbreviated calculation can be performed to update Δ upon flipping a variable x_i:

$$\Delta_i = \begin{cases} -\Delta_i & if \quad i = j \\ \Delta_i + 2q_{ij}(1 - 2x_i)(1 - 2x_j) & otherwise \end{cases} \tag{1.10}$$

1.2.1.4.2 Fast evaluation for k-flip move The generalized *k-flip* move can be considered as a sequence of *1-flip* moves, thus it is not hard to infer the following equation for the calculation of the move gain produced by a *k-flip* move. Formally, let $x_{i_1}, \ldots, x_{i_k}$ are k variables to flip, then the move gain $\Delta_{i_1,\ldots,i_k}$ for simultaneously flipping the k variables can be computed as follows:

$$
\begin{aligned}
\Delta_{i_1,\ldots,i_k} &= \Delta_{i_1} && \text{(1-flip)}\\
&+\Delta_{i_2} + 2q_{i_1 i_2}(1-2x_{i_1})(1-2x_{i_2}) && \text{(2-flip)}\\
&+\Delta_{i_3} + 2q_{i_1 i_3}(1-2x_{i_1})(1-2x_{i_3}) + 2q_{i_2 i_3}(1-2x_{i_2})(1-2x_{i_3}) && \text{(3-flip)}\\
&+\Delta_{i_4} + 2q_{i_1 i_4}(1-2x_{i_1})(1-2x_{i_4}) + 2q_{i_2 i_4}(1-2x_{i_2})(1-2x_{i_4})\\
&+2q_{i_3 i_4}(1-2x_{i_3})(1-2x_{i4}) && \text{(4-flip)}\\
&\qquad\vdots && \vdots\\
&= \sum_{r=1}^{k}\Delta_{i_r} + 2\sum_{r=1}^{k-1}\sum_{s=r+1}^{k} q_{i_r i_s}(1-2x_{i_r})(1-2x_{i_s}) && \text{(k-flip)}
\end{aligned}
$$

1.2.2 Heuristic and metaheuristic algorithms

Many heuristic and metaheuristic algorithms emerged in the BQO literature, which we can mainly categorize into the several classes: greedy construction search method, simulated annealing, tabu search, population based approach (e.g. evolutionary algorithms, memetic algorithms and scatter search) and other approaches. In the following sections we present them in detail.

1.2.2.1 Fast solving heuristics

[Boros *et al.*, 1989] developed a Devour Digest Tidy-up procedure, also known as DDT method. On the basis of the posiform representation Z of BQO, DDT includes the Devour, Digest and Tidy-up phases. Devour identifies a term T from L (L denotes the set of all the elements of Z) with the largest coefficient and places it into S. Digest draws logical conclusions by assigning the disjunctive equation of all the elements in S equaling to 0 (in terms of minimization). If no logical conclusion can be drawn, then T is simply removed from L to S, and return to Devour. Otherwise, Tidy-up begins to substitute the logical conclusions previously drawn into Z. The above DDT procedure repeats until L is an empty set. Experiments indicated that DDT performs especially effective on problems of low density.

Consider the case that the DDT method simultaneously set several variables with value 1 or 0 would result in worse result than to give inferred assignment to only one variable, [Glover *et al.*, 2002] proposed several one-pass heuristics to guarantee that in each pass only one variable gets the implied assignment. The differences among the proposed one-pass heuristics lies in the different strategies of evaluating contributions of variables. Experimental comparisons among the proposed one-pass heuristics showed that some of them perform quite effectively for certain problem instances, but no single method dominates on every problem instance.

[Hanafi *et al.*, 2011] devised five alternative DDT heuristics based on different representation of the BQO formulation, where DDT1 to DDT4 methods respectively have standard, posiform, bi-form and negaform representations and DDT5 has a posiform representation along with a one-pass mechanism. An obviously additional difference of their

DDT alternatives from [Boros *et al.*, 1989; Glover *et al.*, 2002] lies in the use of a *r-flip* local search procedure to improve solutions obtained by DDT constructions. Extensive tests on small, medium and large benchmark instances disclosed that (1) DDT3 with the bi-form representation generally produces the best results for medium and large instances; (2) the proposed *r-flip* local search contributes to significant improvement of the results of the proposed DDT methods with only a slight increase of time consumption.

[Boros *et al.*, 2007] presented a local search scheme which is operated as below. Starting from an initial solution, each iteration constructs a candidate set and then picks a variable from this set and changes its value to its complement, thus moving to the next solution. This iterative procedure repeats until the candidate set is empty. Based on the above scheme, they studied five initialization methods, two candidate set construction methods and four variable selection methods, thus reaching up to 40 local search alternatives. Experiments on multiple benchmark instances indicated that the local search alternative combining the following methods achieved the best performance. The initial method assigns each variable with a fractional value equaling to the proportion of the sum of all the positive entries of the matrix in the sum of the absolute value of each entry of the matrix. The candidate set construction method consists of such variable that flips its value would bring improvement to the current solution no matter whether it was already flipped in the previous iteration. The variable selection methods selects the smallest-index variable from the candidate set with the largest improvement to the current solution.

1.2.2.2 Greedy construction search method

[Merz and Freisleben, 2002] proposed a randomized greedy construction heuristic to quickly obtain an improved solution. The greedy construction procedure starts from a solution with all variables assigned to be 0.5 (the so called third state). The first step randomly picks a variable and randomly assign a value 0 or 1 to it. Each successive construction step considers all the variables with value 0.5 and pick a variable from them with probability proportional to the gain value when changing the variable's value from 0.5 to 0 or 1. The construction procedure ends when no variable lies in the third state. In addition, two local search heuristics named *1-opt* and *k-opt* are presented. The *1-opt* local search is a simple ascent algorithm based on *1-opt* neighborhood. Each iteration for the *k-opt* local search repeats performing the best *1-flip* move subject to requiring that this move has not been performed during this iteration until all *1-flip* moves are performed and picks the best solution to start the next iteration. The above iteration is repeated until no improved solution can be obtained. The complexity of this *k-opt* local search for per iteration is $O(n^2)$. Furthermore, they analyzed the performance of the multi-start randomized greedy algorithm, the multi-start *1-opt* local search, the multi-start *k-opt* local search and the multi-start *k-opt* local search with randomized greedy initial solutions and observed that the superiority of the proposed *k-opt* local search over *1-opt* local search is more obvious for small scale problem instances than for medium and large scale problem instances.

[Palubeckis and Tomkevicius, 2002] applied a greedy random adaptive search procedure, GRASP-PT, that basically switches between a construction phase and a local search phase. For each step in the construction phase, GRASP-PT maintains a candidate list

that contains a certain number of variables with the largest gain values calculated according to a specific gain function and picks a variable from this candidate list with probability proportional to its gain value. The local search phase implements a simple ascent algorithm. In addition, an enhanced version of GRASP-PT implemented by replacing local search with tabu search, called GMSTS-PT and a classic random restarting procedure combined with tabu search in the improvement phase, called MSTS-PT, are also tested for the comparative purpose. Computational results demonstrated that (1) GRASP-PT is not competitive with GMSTS-PT and MSTS-PT; (2) the greedy construction phase of GMSTS-PT is superior to the random restart procedure of MSTS-PT.

1.2.2.3 SA-Simulated Annealing

[Alkhamis *et al.*, 1998] presented a simulated annealing based heuristic, SA-AHA, according to a traditional simulated annealing algorithm framework. It begins with a randomly generated solution and an initial temperature. At each iteration, SA-AHA generates a random *1-flip* move. If this is an improving move, it is automatically performed; Otherwise, this move is performed with a probability $e^{-\Delta/T}$ where Δ indicates the move gain and T is the current temperature constant. After a certain number of iterations, the temperature is decreased according to a cooling function from [Aarts *et al.*, 1988]. The above procedure is repeated until either no solution is accepted in 10 consecutive temperatures or the temperature has fallen below a pre-specified value. Experiments on problem instances with 100 variables and comparisons with several bounding techniques based algorithms indicated that SA-AHA outperforms the reference algorithms. Especially, SA-AHA is able to solve hard problem instances very efficiently while bounding algorithms can not solve them in a reasonable computation time. Additional analysis demonstrated that matrix density does not affect the efficiency of the SA-AHA algorithm.

In [Beasley, 1998], a simulated annealing algorithm, SA-B, was presented. Basically, the iterative procedure of SA-B is the same as SA-AHA. However, in SA-B each iteration applies a different temperature constant to determine probability of accepting a non-improving move. In addition, at the end of the annealing process, a local search procedure based on the first improvement strategy is utilized to further improve solution quality. Experimental results for the GKA benchmark indicated that SA-B generally converges faster to best solutions than the reference algorithms although it obtains inferior solution quality for several instances. In addition, tested on the ORLIB benchmark showed that SA-B is especially effective for the 10 largest instances with 2500 variables.

[Katayama and Narihisa, 2001] designed an implementation of simulated annealing methodology similar to SA-AHA, called SA-KN. However, SA-KN adopts multiple annealing processes, each of which starts with a different initial temperature and takes the best solution found in the previous annealing process as the initial solution, to enhance its search ability. Experimental results for ORLIB problem instances with variables ranging from 500 to 2500 indicated that SA-KN achieves competitive performances, especially for the large instances, as demonstrated in SA-B.

1.2.2.4 TS-Tabu Search

[Glover *et al.*, 1998] introduced an adaptive memory tabu search algorithm, AMTS that uses recency and frequency information to affect a move. Particularly, recency information is used to penalize a move that is recently conducted while frequency information is mainly used to break ties when many moves have the same best evaluation value. Strategic oscillation is employed to alternate between constructive phases (progressively setting variables to 1) and destructive phases (progressively setting variables to 0), which are triggered by critical events, i.e., when the next move causes the objective function to decrease. Oscillation amplitude is adaptively controlled by a span parameter. Tests on GKA benchmark showed AMTS outperforms the best exact and heuristic methods previously reported in the literature.

[Beasley, 1998] proposed a tabu search algorithm, TS-B, based on *1-flip* neighborhood. It begins from an initial solution with each variable assigned to be 0 and marked as non-tabu. During each iteration it conducts a best non-tabu move. This performed move is then marked as tabu for the next T consecutive iterations (T is known as tabu tenure and set as $T = min(20, n/4)$). If the current iteration finds a better solution than the best solution found so far, a local search procedure with first-improvement strategy is launched to further improve this new solution. TS-B repeats the above procedure until the current iteration reaches the maximum allowed iteration. Notice that TS-B does not incorporate the fast evaluation technique and also neglects an aspiration criterion.

[Palubeckis, 2004b] examined five multistart tabu search strategies, with names from MSTS1 to MSTS5. Each multistart tabu search algorithm employs a tabu search procedure, called TS-P to enhance solution quality and a multi-start strategy to produce a new initial solution located in a more promising area. Notice that TS-P is very similar to TS-B except that TS-P employs a tactic to get *1-flip* moves fast evaluated. The first restart strategy produces a new initial solution in a random way. The second restart strategy identifies a candidate set of variables that are prone to change their values when moving from the current solution to an optimal one and applies a steepest ascent algorithm that only takes variables in this candidate set into consideration and keeps the other variables fixed at specific values. The third restart strategy is the same as the constructive phase of GRASP-PT [Palubeckis and Tomkevicius, 2002]. The fourth restart strategy incorporates a set of elite solutions that is used to calculate the probability of each variable with the assigned value 1. If the probability for a given variable is larger than 0.5, then this variable is assigned to be 1 in the resulting new solution; otherwise it is assigned to be 0. The last restart strategy uses a perturbation scheme of changing the problem instance at hand, followed by a short run of tabu search on the modified instance. Experiments on largest ORLIB instances and Palubeckis instances demonstrated that the algorithm with the second restart strategy incorporated performs best among the several proposed multistart algorithms.

[Palubeckis, 2006] developed an iterated tabu search algorithm (ITS) that combines a tabu search procedure to improve the solution quality and a perturbation mechanism to create a new initial solution. The tabu search procedure is exactly the one used in [Palubeckis, 2004b]. The perturbation mechanism is operated as follows. First, it con-

structs a candidate list of a specified size which consists of variables with the largest *1-flip* move gains with regard to a local optimal solution. Then it randomly selects a variable from this set and flips this variable to move toward a new solution. Finally, it updates corresponding move gains of variables caused by the move. The above procedure is repeated until the number of perturbed variables reaches the specified count. Experimental results indicated that although the simplicity of ITS, it is very competitive with other state-of-the-art algorithms.

[Liu *et al.*, 2006] proposed a hybrid *r-opt/1-opt* tabu search algorithm, HLS, which switches among three phases: a hybrid local search phase, a destruction phase and a construction phase. First, the hybrid local search phase that hybrids *1-opt* and *r-opt* local search is launched. This phase behaves like a basic variable neighborhood search procedure [Hansen and Mladenović, 2003] but excludes useless *r-opt* moves according to a theorem that is capable of reducing the number of moves needed to be considered by several orders. When no improved move is found, the hybrid local search phase terminates. Meantime, the destruction phase is followed to carry out the *1-flip* move with the least damage to the current solution. The conducted move is tagged tabu and the destruction phase continues until there occurs a non-tabu move that can improve the current solution. At this time, a construction phase is triggered to perform the best non-tabu move and this performed move is immediately tagged tabu. If the obtained solution is better than the best solution ever found, the algorithm returns to the hybrid local search phase. If no variable exists that can make further improvement, the algorithm then returns to the destruction phase. Tested results showed the proposed genetic hybrid *r-opt/1-opt* tabu search generally outperforms *1-opt* tabu search and performs better than a *1-opt* based multistart tabu search algorithm for problem instances with very large size and density.

[Glover *et al.*, 2010] presented a diversification-driven tabu search algorithm, D^2TS, for BQO. D^2TS alternates between a basic tabu search procedure, named TS-GLH, and a memory-based perturbation strategy guided by a long-term memory. TS-GLH uses *1-flip* neighborhood and best improvement strategy. A tabu list is included to prevent solutions visited within a certain number of iterations, known as tabu tenure (set as $n/100 + rand[1, 10]$ where rand[1,10] takes an random integer from the interval [1,10]), from being revisited. Furthermore, an aspiration is used to permit a move to be selected in spite of being tabu if it leads to a solution better than the current best solution. For the perturbation, three memory structures are introduced: (1) a flipping frequency vector to record the number of times a variable has been flipped from the initial iteration until the current iteration; (2) an elite set of solutions to record a certain number of best local optimal solutions; (3) a consistency vector to count the times of each variable that is assigned with the common value in the set of elite solutions. By use of these information, the perturbation operator operates an elite solution and favors variables with low flipping frequency and high consistency to flip. Computational results showed that D^2TS is capable of matching or improving the previously reported results for the challenging ORLIB and Palubeckis instances.

[Lü *et al.*, 2012] studied neighborhood union and token-ring search methods to combine *1-flip* and *2-flip* neighborhoods within a tabu search algorithm. The *1-flip* based tabu search (N1) is the same as the one in TS-GLH [Glover *et al.*, 2010]. The *2-flip* based

tabu search (N2) constrains consideration to such *2-flip* moves that separately flipping each variable involved in this *2-flip* move would lead to the move gain ranked top $3\sqrt{n}$ among all the *1-flip* moves, in order to reduce the computational efforts of identifying the best move from the complete N2 neighborhood. The neighborhood union includes the strong neighborhood union ($N1 \bigsqcup N2$) that picks each move from both N1 and N2 while selective neighborhood union ($N1 \bigcup N2$) that at each iteration makes a move selected from N1 with probability p and N2 with probability $1 - p$. The token ring search (N1 $\rightarrow$ N2) continuously performs moves within a single neighborhood until no improvement is possible and then switches to the other neighborhood to carry out moves in the same fashion. Experimental results lead to the following rankings: for a single neighborhood $N2 > N1$ while for neighborhood union $N1 \bigcup N2 > N1 \rightarrow N2 > N1 \bigsqcup N2$.

1.2.2.5 Population based search methods

[Lodi *et al.*, 1999] introduced an evolutionary heuristic, called EH, for solving BQO. EH is characterized by the following special features. First, EH contains a preprocessing phase with the purpose of fixing certain variables at their optimal values and reducing the problem size. This type of fixation belongs to permanent fixation since for each successive round of local search, these variables are excluded from consideration. Second, a local search procedure based on the alternation between construction phase and destructive phases like in [Glover *et al.*, 1998] is employed to get an improved solution. Finally, EH uses a uniform crossover operator to generate offspring solutions, where variables with common values in parental solutions are temporarily fixed in this round of local search. Experimental results indicate that EH can match the best known results for problem instances with up to 500 variables in a very short computing time. A further analysis demonstrates that the preprocessing phase is effective for small problem instances but is impossible to reduce the problem size for large ones.

[Merz and Freisleben, 1999] devised a hybrid genetic algorithm, GLS-MF, in which a simple local search is incorporated into the traditional genetic algorithm. The local search procedure uses *1-flip* neighborhood and best improvement strategy. The crossover operator is a variant of uniform crossover, requiring the generated offspring solution has the same hamming distance away from parents. Once the newly generated offspring solution satisfies the updating criterion, it becomes a member of the population and replaces the solution with the worst solution quality. A diversification component is launched when the average hamming distance of the population drops below a threshold $d = 10$ or the population is not updated for more than 30 consecutive generations. Experimental results showed that a simple evolutionary algorithm is sufficient to find best known results for problem instances with less than 200 variables but for those with a high number of variables, it is essential to incorporate local search to attain high quality solutions.

[Katayama *et al.*, 2000] proposed an alternative genetic local search algorithm, named GLS-KTN. The local search procedure of GLS-KTN combines *k-opt* local search proposed in [Merz and Freisleben, 2002] and a *1-opt* local search of [Katayama and Narihisa, 2001]. A traditional uniform crossover and a mutation operator are exploited to generate a suitable offspring solution. A similar diversification/restart strategy as GLS-MF is integrated to

maintain a diversified population. Tested on ORLIB benchmark, GLS-KTN attained best known results in a short running time and better average solution quality than GLS-MF.

[Merz and Katayama, 2004] conducted landscape analysis and observed that (1) local optima of BQO problem instances are concentrated in a small fraction of the search space; (2) the fitness of local optima and the distance between local optima and the global optimum are correlated. Based on this, they designed a memetic algorithm, MA-MK, in which an innovative variation operator is utilized to generate an offspring solution and the *k-opt* local search proposed in [Katayama *et al.*, 2000] is utilized to improve solution quality. Specifically, the initial population is generated with a randomized greedy heuristic introduced in [Merz and Freisleben, 2002]. The innovative variation operator introduces new alleles not contained in both parents by referring to the move gain of performing *1-flip* move, avoiding the rediscovery of local optima already visited to the utmost extent. The selection rule for maintaining a new population is similar to $(u+\lambda)$-ES evolutionary strategy. Evaluated on ORLIB benchmark instances, the proposed approach is demonstrated especially effective in solving large ORLIB instances.

[Lü *et al.*, 2010b] developed a hybrid genetic-tabu search with multi-parent crossover, named GTA to solve BQO. GTA jointly uses traditional uniform crossover and logic multi-parent combination operators to generate suitable and diversified offspring solutions. In the multi-parent crossover operator, a variable's strength is defined as the sum of its weights for an elite set of solutions, where its weight for a single solution is measured as the inversion of the number of variables with assigned value 1 in this solution. If a variable's strength is higher than average strength, its value in the generated offspring solution is 1; otherwise its value is 0. In addition, GTA applies a pool updating strategy that depends on both the solution quality and the Hamming distance between this solution and the elite set. Evaluated on 25 Palubeckis benchmark instances with 2500 to 5000 variables, GTA obtained highly competitive results in comparison with the previous best known results from the literature.

[Lü *et al.*, 2010a] proposed a hybrid metaheuristic approach, called HMA, which integrates a basic tabu search procedure into a genetic search framework. First, HMA combines a traditional uniform crossover operator with a diversification guided path relinking operator to guarantee the quality and diversity of an offspring solution. Second, HMA defines a new distance by reference to variable's importance instead of treating all the variables the same as the Hamming distance and employs a quality-and-distance criterion to update the population as in GTA. Finally, a tabu search procedure is responsible for intensified examination around the offspring solutions. Computational results showed HMA is among the best performing procedures for solving challenging BQO problem instances from Palubeckis family.

[Amini *et al.*, 1999] presented a scatter search approach, SS, including a diversification generation method, a solution improvement method, a reference set update method, a subset generation method and a solution combination method. The diversification generation method systematically generates a collection of diverse trial solutions based on a seed solution in a way of setting an incremental parameter that determines at which bits of the seed solution should be flipped. The improvement method performs a compound move that sequentially cycles among *1-flip*, *2-flip* and *3-flip* candidate moves until no at-

tractive move can be identified. The reference set update method replaces solutions in the reference set with new candidate solutions according to the quality measurement. In order to build a new solution, a linear combination of selected solutions from the reference set is applied. Since some variables would receive fractional values in the combined solution, a rounding procedure is followed to make this solution feasible. Experiments on three classes of problems showed that the proposed scatter search method is very robust, especially for large problem instances.

1.2.2.6 Other algorithms

[Shylo and Shylo, 2011] developed a global equilibrium search, named GES, which performs multiple temperature cycles. Each temperature cycle alternates between an initial solution generation phase and a tabu search phase. The use of historical facilitates to determine the probability that a variable receives value 1 in the generated solution. The tabu search procedure is similar to the one used in [Glover *et al.*, 2010] except requiring that each admissible move produces a solution with hamming distance to a reference set surpassing a distance threshold. Computational tests indicate that GES performs quite well in terms of both the solution quality and computing speed.

[Cai *et al.*, 2011] presented a memetic clonal selection algorithm (MCSA) with estimation of distribution algorithm (EDA) vaccination, called MCSA-EDA, for solving BQO. MCSA-EDA adopts EDA vaccination, fitness uniform selection scheme (FUSS) and adaptive TS to overcome the deficiencies of traditional CSA algorithm. Experimental comparisons indicate that MCSA-EDA enhances the performance of CSA.

[Wang *et al.*, 2011a] provided a tabu Hopfield neural network with estimation of distribution algorithm, THNN-EDA, based on ideas from EDA and TS. The cooperation between long term memory of EDA with the short term memory of TS avoids the network trapped in local optima and thus provides a good performance for THNN-EDA. Experimental results on standard BQO problem instances and MaxCut problems instances reformulated as BQO showed that THNN-EDA is better than or competitive with other HNN based algorithms and some metaheuristic algorithms.

1.3 Benchmark instances

To examine the performance of algorithms, the following benchmarks with a total of 126 instances are most often used in the BQO literature. The detailed characteristics of these problem instances are also shown in Table 1.1.

- [Glover *et al.*, 1998] (GKA family): 45 small scale instances, ranging in size from 25 to 500 variables and in density from 6.5% to 100%. These instances were generated from the P&R routine.
- [Beasley, 1998] (Beasley family): 60 small and medium scale instances, ranging in size from 50 to 2500 variables and with the density of 10%. These instances are available from OR-Library [Beasley, 1996] and can be downloaded at: http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/bqpinfo.html.

Table 1.1: Benchmark instances for BQO

Family	Sub-family	Variables n	Number instances	Density (d)(%)	Q diagonal elements	Q off-diagonal elements
GKA	GKA-a	30-100	8	6.5 to 50	[-100, 100]	[-100,100]
	GKA-b	20-125	10	100	[-63, 0]	[0, 100]
	GKA-c	40-100	7	10 to 80	[-50, 50]	[-100,100]
	GKA-d	100	10	6.5 to 50	[-50, 50]	[-75,75]
	GKA-e	200	5	10 to 50	[-50, 50]	[-100,100]
	GKA-f	500	5	10 to 100	[-75, 75]	[-50,50]
ORLIB	b50	50	10	10	[-100, 100]	[-100,100]
	b100	100	10	10	[-100, 100]	[-100,100]
	b250	250	10	10	[-100, 100]	[-100,100]
	b500	500	10	10	[-100, 100]	[-100,100]
	b1000	1000	10	10	[-100, 100]	[-100,100]
	b2500	2500	10	10	[-100, 100]	[-100,100]
Palubeckis	p3000	3000	5	50-100	[-100, 100]	[-100,100]
	p4000	4000	5	50-100	[-100, 100]	[-100,100]
	p5000	5000	5	50-100	[-100, 100]	[-100,100]
	p6000	6000	3	50-100	[-100, 100]	[-100,100]
	p7000	7000	3	50-100	[-100, 100]	[-100,100]

- [Palubeckis, 2004b] (Palubeckis family): 21 large instances, ranging in size from 3000 to 7000 variables and in density from 50% to 10%. The sources of the generator and input files to replicate these problem instances can be found at: http://www.soften.ktu.lt/∼gintaras/ubqop_its.html.

In this thesis, we mainly focus on hard instances consisting of the 10 largest instances from ORLIB with 2500 variables and those from the Palubeckis family with 3000 to 7000 variables. In particular, we are interested in the most challenging Palubeckis instances with no less than 5000 variables since the other small and medium scale instances can be solved very easily by many algorithms in the literature.

Chapter 2

Backbone Guided Tabu Search

We present a backbone guided tabu search (BGTS) framework that alternates between two phases: (1) a basic tabu search procedure to optimize the objective function as far as possible; (2) a variable fixing/freeing procedure using the notion of strongly determined variables to alternately fix backbone components of the solutions which likely share values in common with an optimal solution. Based on such a fact that our variable fixing/freeing procedure needs to decide how to score variables and which variables should be identified as backbone variables to be fixed, we investigate two rules for scoring variables and two rules for fixing variables. The different combinations of these rules constitute four BGTS algorithms. Experimental comparisons dispose that one of the proposed BGTS algorithms is capable of matching the best known results for all the tested instances and boosts the performance of the basic TS in terms of both solution quality and computational efficiency. Additional analysis of deviations from the best known solution and the correlations between the fitness distances of high-quality solutions illustrates why one particular variable fixing and scoring rule leads to better computational results than another one. This chapter is based on the paper published in EvoCOP2011 International Conference [Wang *et al.*, 2011b].

Contents

2.1 Introduction

The *backbone* terminology comes from the context of the well-known satisfiability problem (SAT). Informally, the backbone of a SAT instance is the set of literals (A literal is a boolean variable or the negation of a boolean variable) which are true in every satisfying truth assignment ([Monasson *et al.*, 1998; Kilby *et al.*, 2005]). [Zhang, 2004] presents an effective backbone-based heuristic for SAT and an example of a similar strategy is reported for the multi-dimensional knapsack problem in [Wilbaut *et al.*, 2009]. Such a strategy was also proposed in connection with exploiting *strongly determined* and *consistent* variables in [Glover, 1977], and has come to be one of the basic strategies associated with tabu search. (Discussions of this strategy in multiple contexts appear in [Glover and Laguna, 1997; Glover, 2005].)

We restrict attention to the "strongly determined" aspect of strongly determined and consistent variables, and borrow the "backbone" terminology from the SAT literature as a vehicle for naming our procedure. The SAT notion of a backbone refers to a set of variable assignments that are shared by all the global optima of an instance. From a practical standpoint this definition clearly has little utility since we do not know these global optima in advance and our goal is to find one of them. Hence we instead take an approach based on available knowledge by keeping track of a set of solutions generated during the course of the search that exhibit the highest quality, and use the criterion of being strongly determined as an indicator of those assignments that likely to be shared in common with a global optimum. In particular, we use a simplification of the notion of [Glover, 1977] by considering a variable to be strongly determined if changing its assigned value in a high quality solution will cause the quality of that solution to deteriorate significantly. Identifying a backbone according to this criterion, we then "instantiate" the backbone by fixing the values of those variables that qualify for membership.

We proposed four backbone guided tabu search algorithms based on different rules for scoring variables and fixing variables. Experimental results showed that the performance of the proposed BGTS algorithms strongly depend on the variable fixing strategies employed but are not very sensitive to the variable scoring methods. Further analysis sheds light on how different fixing and scoring strategies are related with the search behavior and the search space characteristics.

2.2 Backbone guided tabu search algorithms

In order to describe the BGTS algorithms more precisely, we first give the following denotations. Let F denote the index set for the fixed variables and U the index set for the

free (unfixed) variables. Note that F and U partition the index set $N = \{1, \ldots, n\}$, i.e., $F \cup U = N$, $F \cap U = \emptyset$. Let na be the number of variables to be added when new variables are fixed and nd the number of variables to be dropped when new variables are freed. In addition, let x_i^F, for $i \in F$, denote the current values assigned to the fixed variables, and let x^0 denote the starting solution at each run of TS.

2.2.1 General scheme

Algorithm 2.1: Outline of the BGTS framework for BQO

Input: matrix Q
Output: the best binary n-vector x^* found so far
Initialization: $F = \{\}$, $U = \{x_1, \ldots, x_n\}$, $f(x^*) = -\infty$, $f_p = -\infty$
repeat
  Construct an initial solution x^0: $x_i^0 = x_i^F$ for $i \in F$ and $x_i^0 = Rand\{0, 1\}$ for $i \in U$
  $x \leftarrow$ Tabu_Search(x^0, U) /* Section 2.2.2*/
  Obtain a population of reference solutions P from current TS procedure /* Section 2.2.3 */
  if $f(x) > f(x^*)$ **then**
    $x^* = x$, $f(x^*) = f(x)$
  end if
  if $f(x) > f_p$ **then**
    (**Variable Fixing Phase**)
    Apply Variable Scoring Rule to score variables /* Section 2.2.4 */
    Apply Variable Fixing Rule to fix na backbone variables /* Section 2.2.5 */
    $|F| = |F| + na$, $|U| = |U| - na$
  else
    (**Variable Freeing Phase**)
    Apply Variable Scoring Rule to score variables /* Section 2.2.4 */
    Apply Variable Freeing Rule to free nd backbone variables /* Section 2.2.5 */
    $|F| = |F| - nd$, $|U| = |U| + nd$
  end if
  $f_p = f(x)$
until a stop criterion is met

Algorithm 2.1 describes the framework of our BGTS algorithms. It begins with a randomly constructed initial solution x^0 and repeatedly alternates between a tabu search (TS) procedure and a phase that either fixes or frees variables until a stop criterion is satisfied. The TS procedure is employed to improve the input solution and to obtain a population of reference solutions with the purpose of scoring and fixing variables.

If the objective value $f(x)$ obtained by the current round of TS is better than the previous one f_p, a variable fixing phase is launched. Specifically, the fixing phase first uses Variable Scoring Rule to give score values to variables. Then Variable Fixing Rule sorts the variables according to these values and determines a number na of variables that go into a set F of variables to be fixed. Consequently, the set of variables F will not be allowed to change its composition during the next round of TS, although conditionally changing the value of a fixed variable is another interesting strategy worthy of further

investigation. It is understood that the values of variables x_i^0 in the starting solution x^0 are selected randomly except for $i \in F$.

On the contrary, if the TS procedure fails to find an improved solution relative to f_p, the algorithm performs the freeing phase that permit reduced number of variables fixed during the next round of TS, thus freeing some variables. The freeing phase selects variables freed based on variable scoring and freeing rules.

The proposed BGTS framework cycles between a tabu search phase and a variable fixing/freeing phase and it terminates when a stopping condition arrives.

2.2.2 Tabu search

As demonstrated in [Glover *et al.*, 1998] and more recently in [Palubeckis, 2004b; Palubeckis, 2006; Glover *et al.*, 2010; Lü *et al.*, 2010b; Lü *et al.*, 2010a], tabu search based on *1-flip neighborhood* is one of the most successful local search algorithms for solving BQO. Recall that the *1-flip move* consists in changing (flipping) the value of a single variable x_i to its complementary value $1 - x_i$. Hence, we employ a tabu search algorithm to carry out *1-flip* neighborhood exploitation.

To increase search efficiency, it is critical to quickly evaluate neighborhood moves. We apply the technique incorporated in [Glover and Hao, 2010] that maintains a vector Δ to record move gain of transferring the solution x to its neighboring solution x' when performing a *1-flip* move, this vector Δ can be initialized as follows:

$$\Delta_i = (1 - 2x_i)(q_{ii} + 2 \sum_{j \in N \setminus \{i\}, x_j = 1} q_{ij}) \tag{2.1}$$

After a move is performed, we just need to update those elements in Δ affected by the move. Specifically, the following abbreviated calculation is used to update Δ upon flipping a variable x_i:

$$\Delta_i = \begin{cases} -\Delta_i & if \quad i = j \\ \Delta_i + 2q_{ij}(1 - 2x_i)(1 - 2x_j) & otherwise \end{cases} \tag{2.2}$$

To ensure solutions visited within a certain span of iterations will not be revisited, tabu search typically incorporates a short-term memory, known as *tabu list* [Glover and Laguna, 1997]. In our implementation, each time a variable x_i is flipped, a random integer is taken from an interval $tt = [a, b]$ (where a and b are chosen integers) as the tabu tenure of variables x_i to prevent x_i from being flipped for a specified number of iterations. In the current study, we select to set $a = 0.007n$ and $b = a + 10$. Specifically, our tabu list is defined by a n-element vector T. When x_i is flipped, we assign the sum of a random integer from tt and the current iteration count $Iter$ to the i_{th} element $T[i]$ of T. Subsequently, for any iteration $Iter$, a variable x_k is forbidden to take part in a swap move if $T[k] > Iter$.

Our tabu search algorithm then restricts consideration to variables not currently tabu, and selects a variable to flip that produces the best (largest) Δ_i value. In the case that two or more moves have the same best move gain, ties are broken randomly.

Accompanying this rule, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution. By convention we speak of "better" and "best" in relation to the objective function value $f(x)$. (Similarity, we refer to the objective function value when speaking of solution *quality*.) The tabu search procedure stops when the best solution cannot be improved within a given number α of moves that called *improvement cutoff*. We set $\alpha = 100000$ in the experiments.

The pseudo-code of the tabu search procedure is shown in Algorithm 2.2.

Algorithm 2.2: Pseudo-code of the tabu search procedure for BQO

Input: a given solution x and its objective function value $f(x)$
Output: an improved solution x^* and its objective function value $f(x^*)$
Initialize vector Δ according to Eq. 2.1, initialize tabu list vector T by assigning each element with value 0, $Iter = 0$, $NonImpIter = 0$
while $NonImpIter < \alpha$ **do**
    Identify the index i^*_{nt} from all non-tabu moves that leads to the maximum Δ value (break ties randomly); Similarly identify i^*_t from tabu moves
    if $\Delta_{i^*_t} > \Delta_{i^*_{nt}}$ and $f(x^*) + \Delta_{i^*_t} > f(x^*)$ **then**
        $i^* = i^*_t$
    else
        $i^* = i^*_{nt}$
    end if
    $x_{i^*} = 1 - x_{i^*}$, $f(x) = f(x) + \Delta_{i^*}$
    Update Δ according to Eq. 2.2
    Update T by assigning $T[i] = Iter + rand(tt)$
    if $f(x) > f(x^*)$ **then**
        $x^* = x, f(x^*) = f(x)$
        $NonImpIter = 0$
    else
        $NonImpIter = NonImpIter + 1$
    end if
    $Iter = Iter + 1$
end while

Notice that underlying the BGTS framework, the moves in TS only considers variables from the set U of unfixed variables to be flipped while keeping backbone variables in the set F fixed at specific value, either 0 or 1.

2.2.3 Reference solutions

Reference solutions are used for fixing or freeing variables. We conjecture that there exists a subset of variables, of non-negligible size, whose optimal values are often also assigned to these same variables in high quality solutions. Thus, our goal is to identify such a critical set of variables and infer their optimal values from the assignments they receive in high quality solutions. Our expectation is that this will reduce the search space sufficiently to enable optimal values for the remaining variables to be found more readily. On the basis

of this conjecture, we maintain a set of reference solutions consisting of good solutions obtained by TS. Specifically, we take a given number p of the best solutions from the current round of TS (subject to requiring that these solutions differ in a minimal way), which then constitute a solution population P for the purpose of fixing or freeing variables. In our implementation, we empirically set $p = 20$. (A more refined analysis is possible by a strategy of creating clusters of the solutions in the reference set and of considering interactions and clusterings among subsets of variables as suggested in [Glover, 1977].)

2.2.4 Rules for scoring variables

Definition 1. Relative to a given solution $x = \{x_1, x_2, ..., x_n\}$ and a variable x_i, the (objective function) *contribution* of x_i in relation to x is defined as:

$$VC_i(x) = (1 - 2x_i)(q_{ii} + \sum_{j \in N \setminus \{i\}} 2q_{ij}x_j) \tag{2.3}$$

As noted in [Glover *et al.*, 1998] and in a more general context in [Glover *et al.*, 2010], $VC_i(x)$ identifies the change in $f(x)$ that results from changing the value of x_i to 1 - x_i; i.e.,

$$VC_i(x) = f(x') - f(x) \tag{2.4}$$

where $x'_j = x_j$ for $j \in N - \{i\}$ and $x'_i = 1 - x_i$. We observe that under a maximization objective if x is a locally optimal solution, as will typically be the case when we select x to be a high quality solution, then $VC_i(x) \leq 0$ for all $i \in N$, and the current assignment $x_i = x_i$ will be more strongly determined as $VC_i(x)$ is "more negative".

Definition 2. Relative to a given population of solutions $P = \{x^1, \ldots, x^p\}$ and their corresponding objective function values $F = \{f(x^1), \ldots, f(x^p)\}$ indexed by $I = \{1, \ldots, p\}$, and relative to a chosen variable x_i, let $P_i(0) = \{k \in I : x_i^k = 0\}$ and $P_i(1) = \{k \in I : x_i^k = 1\}$, the (objective function) *contribution* of x_i in relation to P is defined as follows.

Contribution for $x_i = 0$:

$$VC_i(P:0) = \sum_{k \in P_i(0)} (\beta \cdot VC_i(x^k) + (1 - \beta) \cdot \tilde{A}(f(x^k)) \cdot VC_i(x^k)) \tag{2.5}$$

Contribution for $x_i = 1$:

$$VC_i(P:1) = \sum_{k \in P_i(1)} (\beta \cdot VC_i(x^k) + (1 - \beta) \cdot \tilde{A}(f(x^k)) \cdot VC_i(x^k)) \tag{2.6}$$

where f_{min} and f_{max} are respectively the minimum and maximum objective values of the set F and $\tilde{A}(\cdot)$ represents the normalized function:

$$\tilde{A}(f(x^k)) = (f(x^k) - f_{min})/(f_{max} - f_{min} + 1) \tag{2.7}$$

Definition 3. Relative to a variable x_i and a population of solutions P, the score of x_i with respect to P is then defined as:

$$Score(i) = min\{VC_i(P:0), VC_i(P:1)\} \tag{2.8}$$

Rule 1 for scoring variables (SR1):
Only consider the contribution of variables to the reference solutions by setting β equaling to 1.0. Obviously, the second part of Definition 2 that is multiplied by $1-\beta$ is equal to 0 if assigning $\beta = 1.0$, thus the objective function values of the reference solutions are neglected.
Rule 2 for scoring variables (SR2):
Simultaneously consider the contribution of variables and the solution quality of the reference solutions by assigning a value to β from the interval $[0, 1)$. Preliminary experiments suggests $\beta = 0.4$.

2.2.5 Rules for fixing and freeing variables

Rule 1 for fixing variables (FIX1):
Order the elements of $i \in U$ such that $score(i_1) \leq \ldots \leq score(i_{|U|})$
Let $F(+) = i_1, \ldots, i_{na}$
$F := F \cup F(+)$ $(|F| := |F| + na)$
$U := U - F(+)$ $(|U| := |U| - na)$
$x_i^F = x_i^0$ for $i \in F(+)$, (x_i^F is already determined for $i \in$ "previous F" $:= F - F(+)$ and x_i^0 represents the value that x_i should be assigned to according to Eq. 2.8, i.e., $x_i^0 = 0$ if $VC_i(P:0) < VC_i(P:1)$ and $x_i^0 = 1$ otherwise.)

Rule 1 for freeing variables (FREE1):
Order the elements of $i \in F$ such that $score(i_1) \geq \ldots \geq score(i_{|F|})$
Let $F(-) = i_1, \ldots, i_{nd}$
$F := F - F(-)(|F| := |F| - nd)$
$U := U \bigcup F(-)(|U| := |U| + nd)$

Rule 2 for fixing variables (FIX2):
Set $|F| := |F| + na$
Order the elements of $i \in N$ such that $score(i_1) \leq \ldots \leq score(i_n)$
(We only need to determine the first $|F|$ elements of this sorted order.)
Let $F = i_1, \ldots, i_{|F|}$
$U := N - F(|U| := |U| - na)$
$x_i^F = x_i^0$ for $i \in F$

Rule 2 for freeing variables (FREE2):
Set $|F| := |F| - nd$
Order the elements of $i \in N$ such that $score(i_1) \leq \ldots \leq score(i_n)$
(We only need to determine the first F elements of this sorted order.)
Let $F = i_1, \ldots, i_{|F|}$

$U := N - F(|U| := |U| + nd)$
$x_i^F = x_i^0$ for $i \in F$

Remarks: FIX1 differs in two ways from FIX2. At each fixing phase, FIX2 fixes $|F|$ variables, while FIX1 only fixes na new variables since $|F| - na$ variables are already fixed. In other words, once a variable is fixed by FIX1, its value cannot be changed unless a freeing phase frees this variable. Instead of inheriting the previously fixed variable assignment as in FIX1, FIX2 selects all $|F|$ variables to be fixed at each fixing phase. FREE1 only needs to score variables belonging to F and then to select those with the highest scores to be freed, while FREE2 redetermines the variables to be freed each time.

Based on the fact that in the initial stage, the number of unfixed variables is large while this number becomes smaller and smaller through a series of passes when the TS method finds progressively improved solutions, we employ a strategy that gradually reduced number of newly added backbone variables na throughout such a succession of improvements. Specifically, the number of backbone variables at the first fixing phase is relatively large and is then gradually reduced with a geometric ratio when successive improvements occur, as follows.

Let $Fix(h)$ denote the number of new variables (na) that are assigned fixed values and added to the fixed variables at fixing phase h. We begin with a chosen value $Fix1$ for $Fix(1)$, referring to the number of backbone variables at the first fixing phase and then generate values for higher fixing phases by making use of an "attenuation fraction" g as follows.

$$Fix(h) = \begin{cases} Fix1 & (h = 1) \\ Fix(h-1) \cdot g & (h > 1) \end{cases} \tag{2.9}$$

We select the value $Fix1 = 0.25n$ and the fraction $g = 0.4$ in our experiments.

Contrary to the fixing phase, the number of the backbone variables released from their assignments at each freeing phase is not adjusted, due to the fact that at each trial only a small number of backbone variables are wrongly fixed and need to be freed. Specifically, we set the number nd of backbone variables as follows.

$$nd = \begin{cases} 60 & (nd <= |F|) \\ |F| & otherwise \end{cases} \tag{2.10}$$

2.2.6 Four derived algorithms

Our four BGTS algorithms consist of four different combinations of the two variable fixing rules and the two variable scoring rules. Specifically, using $\beta = 1.0$ as our scoring rule, we employ the variable fixing rules FIX1 and FIX2 to get the first two algorithms, respectively. Likewise, the third and fourth algorithms are derived by combining the scoring rules $\beta = 0.4$ with FIX1 and FIX2, respectively.

Table 2.1: Results of BGTS algorithms with variable fixing rules FIX1 and FIX2 ($\beta = 1.0$)

Instance	BKR	FIX1					FIX2					$Diff?$
		g_{best}	g_{avg}	$hits$	t_{b_avg}	t_{best}	g_{best}	g_{avg}	$hits$	t_{b_avg}	t_{best}	
p3000.1	3931583	0	413	15	172	40	0	3193	5	54	63	Y
p3000.2	5193073	0	0	20	62	2	0	397	12	26	5	Y
p3000.3	5111533	0	71	18	115	6	0	1144	2	43	4	Y
p3000.4	5761822	0	114	16	93	5	0	3119	7	61	7	Y
p3000.5	5675625	0	372	8	86	5	0	1770	2	147	16	Y
p4000.1	6181830	0	0	20	65	14	0	319	19	74	16	N
p4000.2	7801355	0	1020	11	295	64	0	2379	5	81	59	Y
p4000.3	7741685	0	181	18	201	17	0	1529	9	58	20	Y
p4000.4	8711822	0	114	18	171	56	0	1609	9	209	39	Y
p4000.5	8908979	0	1376	9	231	58	0	2949	2	231	134	Y
p5000.1	8559680	0	670	1	999	999	368	2429	0	1800	1800	Y
p5000.2	10836019	0	1155	6	740	47	582	2528	0	1800	1800	Y
p5000.3	10489137	0	865	3	1037	279	354	4599	0	1800	1800	Y
p5000.4	12252318	0	1172	3	1405	1020	608	4126	0	1800	1800	Y
p5000.5	12731803	0	268	13	1003	192	0	2941	3	588	279	Y
p6000.1	11384976	0	914	6	451	68	0	4694	4	550	209	Y
p6000.2	14333855	0	1246	1	739	739	88	3332	0	1800	1800	Y
p6000.3	16132915	0	2077	2	1346	1267	2184	8407	0	1800	1800	Y
p7000.1	14478676	0	2315	1	2470	2470	744	4155	0	3000	3000	Y
p7000.2	18249948	716	2340	0	3000	3000	2604	6164	0	3000	3000	Y
p7000.3	20446407	0	2151	7	981	478	0	8150	5	1836	149	Y
Av.		34	897	9.3	746	516	359	3330	4.0	988	848	Y

2.3 Experimental results

2.3.1 Benchmark instances and experimental protocol

To evaluate the performance of the proposed four BGTS algorithms, we test them on 21 benchmark instances from the Palubeckis family. The characteristics of these instances can be found from Section 1.3. Given the stochastic nature of the algorithms, each problem instance is independently solved 20 times. Our BGTS algorithms are programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.83GHz CPU and 8GB Memory. The stop condition for a single run is respectively set to be 5, 10, 30, 30, 50 minutes for instances with 3000, 4000, 5000, 6000 and 7000 variables, respectively.

2.3.2 Comparison among 4 different BGTS algorithms

We present in Tables 2.1 and 2.2 the computational results with β equaling to 1.0 and 0.4, respectively. Each table reports the results using both FIX1 and FIX2 variable fixing rules. Column 2 gives the best known results (BKR) obtained by all previous methods applied to these problems. The remaining columns give the results of one of the two versions (FIX1 and FIX2) according to four criteria: (1) the best solution gap, g_{best}, to the best known results (i.e., $g_{best} = BKR - f_{best}$ where f_{best} denotes the best solution value obtained by our algorithm), (2) the average solution gap, g_{avg}, to the best known results (i.e., $g_{avg} = BKR - f_{avg}$ where f_{avg} represents the average objective value), (3) the number of times over 20 runs, *hits*, for reaching BKR and (4) the CPU time, consisting of the average time and the best time, t_{b_avg} and t_{best} (in seconds), for reaching BKR. The

Table 2.2: Results of BGTS algorithms with variable fixing rules FIX1 and FIX2 ($\beta = 0.4$)

Instance	BKR	FIX1					FIX2					$Diff?$
		g_{best}	g_{avg}	$hits$	t_{b_avg}	t_{best}	g_{best}	g_{avg}	$hits$	t_{b_avg}	t_{best}	
p3000.1	3931583	0	308	16	98	4	0	3315	5	75	2	Y
p3000.2	5193073	0	0	20	59	9	0	488	13	50	3	Y
p3000.3	5111533	0	166	17	108	2	0	1355	4	28	5	Y
p3000.4	5761822	0	19	19	109	24	0	1684	10	74	2	Y
p3000.5	5675625	0	275	11	147	14	0	1796	3	154	40	Y
p4000.1	6181830	0	0	20	61	13	0	354	19	78	3	N
p4000.2	7801355	0	783	11	369	44	0	2722	3	382	106	Y
p4000.3	7741685	0	254	17	234	29	0	1474	8	75	29	Y
p4000.4	8711822	0	75	19	250	13	0	2537	7	158	12	Y
p4000.5	8908979	0	1769	8	361	275	0	3112	3	101	41	Y
p5000.1	8559680	0	791	2	721	228	325	2798	0	1800	1800	Y
p5000.2	10836019	0	860	4	540	37	0	2397	1	45	45	Y
p5000.3	10489137	0	1698	5	702	292	354	4939	0	1800	1800	Y
p5000.4	12252318	0	1123	2	103	76	444	3668	0	1800	1800	Y
p5000.5	12731803	0	455	12	747	261	0	3250	3	145	114	Y
p6000.1	11384976	0	1450	9	1014	432	0	5405	2	1178	768	Y
p6000.2	14333855	0	1079	3	911	515	0	4923	1	192	192	Y
p6000.3	16132915	0	2320	3	1000	642	0	6137	1	147	147	Y
p7000.1	14478676	0	1784	2	1519	785	1546	4556	0	3000	3000	Y
p7000.2	18249948	0	2743	1	2238	2238	1710	5986	0	3000	3000	Y
p7000.3	20446407	0	3971	3	1457	870	0	11604	1	1113	1113	Y
Av.		0	1044	9.7	607	324	209	3548	4.0	733	668	Y

last column $Diff?$ indicates the superiority of FIX1 over FIX2 when a 95% confidence t-test is performed in terms of the objective values. Furthermore, the last row "Av." indicates the summary of the algorithm's average performance.

Table 2.1 shows the computational results of variable fixing strategies FIX1 and FIX2 where $\beta = 1.0$. One observes that for all the considered criteria, FIX1 outperforms FIX2 for almost all the instances. Specifically, FIX1 is able to reach the best known results for all instances except one (p7000.2) while FIX2 fails for 8 cases. Moreover, FIX1 has an average hits of 9.3 over 20 runs, more than two times larger than FIX2's 4.0. FIX1 is also superior to FIX2 when it comes to the average gap to the best known results. In addition, FIX1 performs slightly better than FIX2 in terms of the CPU time. The T-test also demonstrates that FIX1 is significantly better than FIX2 except only one case (p4000.1).

Table 2.2 gives the computational results of variable fixing strategies FIX1 and FIX2 when β is set to be 0.4 instead of 1.0. From Table 2.2, we observe that FIX1 outperforms FIX2 in terms of all the considered criteria, including g_{best}, g_{avg}, $hits$, t_{b_avg} and t_{best}. One also notices that this is quite similar to the case of $\beta = 1.0$. Therefore, we can conclude that the variable fixing strategy FIX1 is generally superior to FIX2 when using the two variable scoring strategies considered. In other words, the two variable scoring strategies have a similar influence on the computational results. The ability of the tabu search method using FIX1 together with SR2 to obtain all of the best known results in the literature places this method on a par with the best methods like [Glover *et al.*, 2010; Lü *et al.*, 2010a], while its solution times are better than those obtained in [Glover *et al.*, 2010].

Table 2.3: Results of the basic TS algorithm

Instance	BKR	Basic TS Algorithm					
		f_{best}	g_{best}	g_{avg}	$hits$	t_{b_avg}	t_{best}
p3000.1	3931583	3931583	0	207	12	50	50
p3000.2	5193073	5193073	0	306	12	29	50
p3000.3	5111533	5111533	0	679	12	67	50
p3000.4	5761822	5761822	0	394	18	44	50
p3000.5	5675625	5675625	0	675	5	61	50
p4000.1	6181830	6181830	0	13	16	76	50
p4000.2	7801355	7801355	0	1766	5	108	50
p4000.3	7741685	7741685	0	526	9	204	50
p4000.4	8711822	8711822	0	175	14	231	50
p4000.5	8908979	8908979	0	1148	11	323	50
p5000.1	8559680	8559680	0	925	1	1650	50
p5000.2	10836019	10836019	0	1628	1	23	50
p5000.3	10489137	10489137	0	2799	2	869	50
p5000.4	12252318	12251403	915	2202	0	1800	50
p5000.5	12731803	12731803	0	1011	3	531	50
p6000.1	11384976	11384976	0	1097	4	1244	50
p6000.2	14333855	14333257	598	3180	0	3600	50
p6000.3	16132915	16132915	0	1642	6	2279	50
p7000.1	14478676	14477845	831	2400	0	3600	50
p7000.2	18249948	18249799	149	2875	0	3600	50
p7000.3	20446407	20446407	0	4426	2	1134	50
Av.			118	1432	6.34	1025	1025

2.3.3 Comparison between BGTS and its underlying TS

We now assess the effect of the proposed variable fixing strategies on the performance of TS by comparing our BGTS algorithms with its underlying TS procedure on the set of 21 instances. For this purpose, we run the TS procedure described in Section 2.2.2 under the same time limit as our BGTS algorithms. The results are shown in Table 2.3. From Tables 2.2, 2.1 and 2.3, one observes that two BGTS algorithms with variable fixing rules FIX1 do boost the performance of the basic TS in terms of the criteria (1)-(5) for almost all the instances.

2.4 Discussion and analysis

In this section, we discuss and analyze some key factors which may explain the performance differences among BGTS algorithms with different variable fixing and scoring rules. For this purpose, we examine the Variables Fixing Errors (number of wrongly fixed variables) relative to the putative optimal solution and show a fitness landscape analysis of high-quality solutions.

2.4.1 Variable fixing errors

As previously demonstrated, the variable fixing rules FIX1 dominates FIX2 for both scoring rules (with $\beta = 1.0$ and $\beta = 0.4$). In order to ascertain why this is the case, we conduct an experiment to compare the total number of wrongly fixed variables during the search using these two variable fixing rules. For this, we carry out our experiment on instance

p5000.5 and repeat the experiment 20 times. For each run, we count, after each fixing or freeing phase, the number of mismatched variables of the current (possibly partial) solution with respect to the best known solution[1]. Figure 2.1, where each point represents the accumulated Variable Fixing Errors over 20 runs, shows how the variable fixing rules affect the Variable Fixing Errors at each fixing or freeing phase under two variable scoring rules: the left one is for $\beta = 0.4$ and the right is for $\beta = 1.0$. From Figure 2.1, one observes that the number of variable fixing errors induced by FIX1 and FIX2 (with both scoring strategies) increases rapidly at the beginning of the search and then decreases gradually when the search progresses. However, the number of the Variable Fixing Errors of FIX1 is much smaller than that of FIX2 throughout the search process. This observation together with the results in Tables 2.1 and 2.2 demonstrate that the variable fixing strategy plays a vital role in our BGTS algorithms for both $\beta = 1.0$ and $\beta = 0.4$.

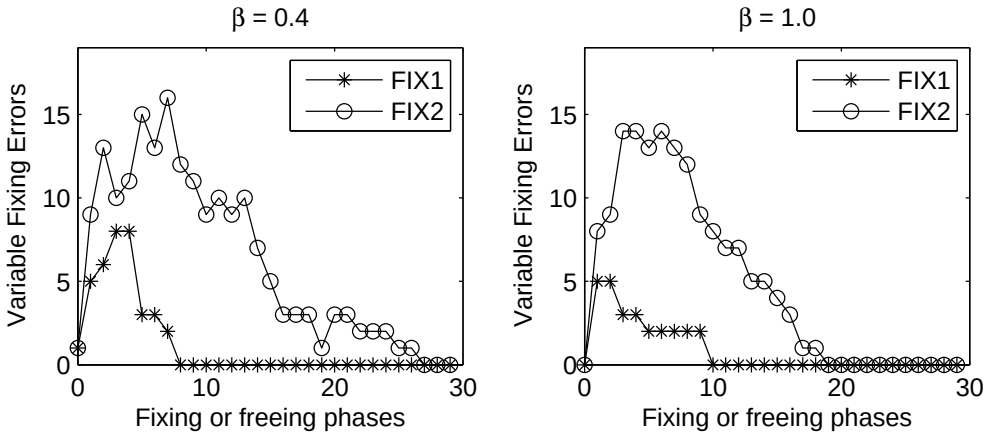


Figure 2.1: Comparison of variable fixing errors between two fixing rules

2.4.2 Fitness distance correlation analysis

In this section, we show a search landscape analysis using the fitness distance correlation [Jones and Forrest, 1995], which estimates how closely the fitness and distance are related to the nearest optimum in the search space. For this purpose, we collect a large number of high-quality solutions by performing 20 independent runs of our BGTS algorithms, each run being allowed 30 fixing and freeing phases, where each phase has 20 elite solutions recorded in the population P. Thus, $20 * 30 * 20 = 12,000$ solutions are collected and plotted. Figures 2.2 and 2.3 show the hamming distance between these solutions to the best known solution against the fitness difference $\Delta f = BKR$ - $f(x^k)$ of these high-quality solutions for instances p5000.1 and p5000.5, respectively.

Figure 2.2 discloses that the majority of the high quality solutions produced by variable fixing rule FIX1 (two upper sub-figures) has a much wider distance range than the solutions

[1]The best known solutions are obtained by different algorithms, sharing exactly the same assignment. Thus, we assume that it is very likely to be the optimal solution.

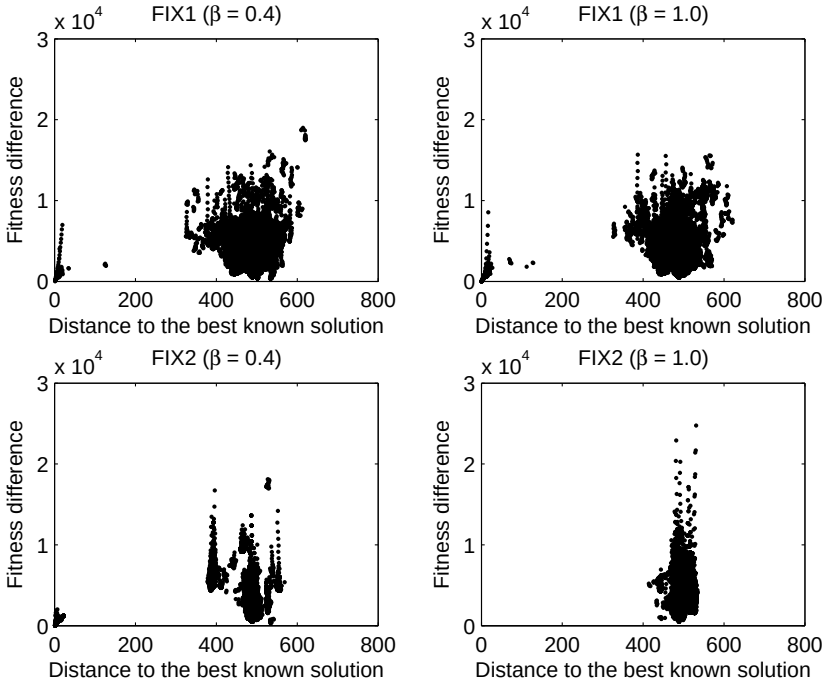


Figure 2.2: Fitness distance correlation: instance p5000.1

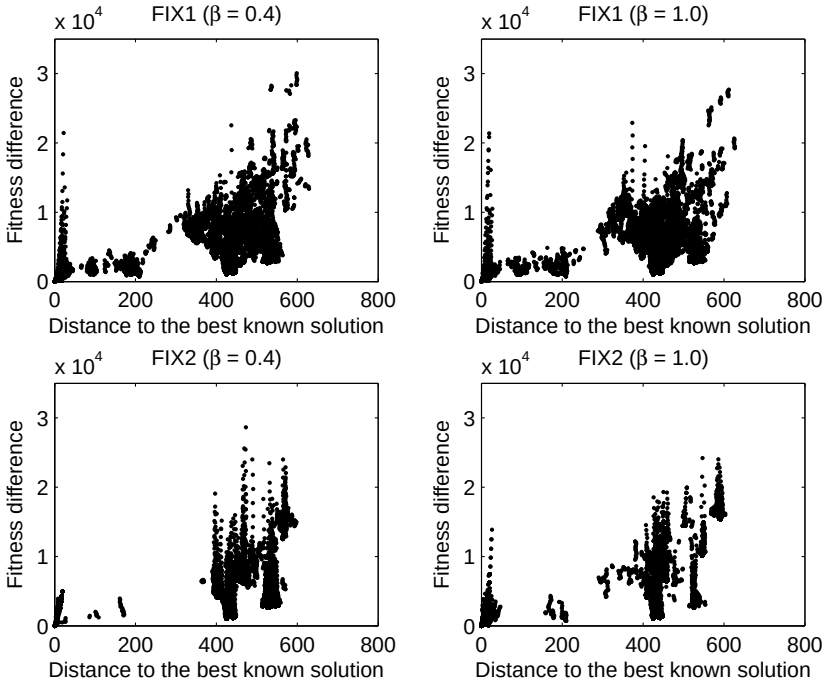


Figure 2.3: Fitness distance correlation: instance p5000.5

produced by rule FIX2 (two bottom sub-figures), which indicates that the search space of FIX1 is more dispersed than that of FIX2. Moreover, the high-quality solutions of FIX1 are much closer to the x-axis than FIX2, implying that FIX1 can obtain better objective values than FIX2. In sum, this indicates the higher performance of the FIX1 rule.

Figure 2.3 presents a trend quite similar to that of Figure 2.2 in terms of the solutions' distance range and the percentage of high quality solutions when comparing the two variable fixing rules FIX1 (two upper sub-figures) and FIX2 (two bottom sub-figures). However, a clear difference from Figure 2.2 is that high quality solutions are distributed in a wider range. In particular, the distribution of solutions is more continuous and does not produce the "isolated cluster effect" shown in Figure 2.2. This indicates that instance p5000.5 is much easier than p5000.1 to solve as shown in Tables 2.1 and 2.2. Indeed, for instance p5000.5, the search space seems smoother, enabling the search to traverse easily from solutions that are far from optimal to the best known solution.

2.5 Conclusions

In this chapter, we propose a backbone guided tabu search (BGTS) framework that alternates between a basic tabu search phase and a variable fixing/freeing phase for handling the binary quadratic optimization problem. The variable fixing/freeing phase dynamically enlarges/reduces the backbone (fixed variables), enabling the successive tabu search to exploit a reduced search area consisting only of those non-fixed variables. Within the BGTS framework, we investigated four BGTS algorithms with different combinations of two variable fixing rules and two variable scoring rules.

Using 21 standard instances from the Palubeckis family, we showed that one of the proposed BGTS algorithms obtained highly competitive outcomes in comparison with the previous best known results from the literature. A direct comparison between BGTS and the underlying TS procedure confirms that incorporating backbone boosts the performance of the basic tabu search algorithm.

To analyze the intrinsic differences of the proposed four BGTS algorithms, we counted the errors of fixed variables in each fixing/freeing phase in comparison with a (near) optimal solution and identified the correlations between fitness distances of high quality solutions to characterize the search behavior of the variable fixing and scoring rules. Our experimentation discloses that our TS method indeed performs differently according to the variable fixing rule employed, but is much less sensitive to the variable scoring rule. The finding that one of the BGTS algorithms obtains the best solutions in the literature to the challenging test problems examined underscores the value of analyzing their impacts.

In the next chapter, we develop a new algorithm underlying the multilevel framework for handling the large BQO problem instances, in which we hierarchically simplify the initial problem by means of extracting backbone variables, apply an enhanced memetic algorithm to refine a solution for each intermediate level problem and an asymmetric uncoarsening phase that go back in a level-by-level manner to the initial problem to correct the erroneously extracted backbone variables.

Chapter 3

Backbone Multilevel Memetic Algorithm

This chapter presents a backbone multilevel memetic algorithm (BMMA) designed to approximate large BQO instances. The proposed algorithm is composed of a backbone-based coarsening phase, an asymmetric uncoarsening phase and a memetic refinement phase, where the backbone-based procedure and the memetic refinement procedure make use of tabu search to obtain improved solutions. Evaluated on 11 largest instances from the Palubeckis family, the proposed algorithm proves to be able to attain all the best known values with a computing effort less than any existing approach. The content of this chapter is published in CPAIOR2012 International Conference [Wang *et al.*, 2012e].

Contents

3.1 Introduction

In this chapter, we are interested in investigating the so-called multilevel approach to handling large BQO instances. The multilevel approach is known to be useful to tackle large instances of several other types of combinatorial optimization problems [Walshaw, 2004]. For example, multilevel algorithms are among the best performing approaches for large graph partitioning problems [Toulouse *et al.*, 1999; Walshaw and Cross, 2000; Meyerhenke *et al.*, 2009; Benlic and Hao, 2011].

Generally, the multilevel paradigm consists of three phases [Walshaw, 2004]: (1) a coarsening phase to create a hierarchy of coarser (smaller and intermediate) problems through grouping or extracting problem variables; (2) an initial optimization phase to obtain a solution to the coarsest (smallest) problem using an optimization procedure; (3) an uncoarsening phase (also called projection) to recover progressively each intermediate problem and apply to it the optimization procedure to further improve the solution quality.

We investigate the multilevel approach applied to BQO. The proposed multilevel algorithm integrates a coarsening phase based on the backbone notion (Section 3.2.2), a population-based memetic optimization procedure (Section 3.2.4) and an asymmetric uncoarsening phase (Section 3.2.5). Experiments on a set of 11 largest BQO benchmark instances from the literature demonstrate that our proposed algorithm is able to attain the current best known results with much less computing time than any other existing algorithm (Section 3.3).

3.2 Backbone multilevel memetic algorithm

3.2.1 The general multilevel scheme

The general scheme of our multilevel algorithm for BQO is shown in Algorithm 3.1. To begin with, the initial matrix Q_0 is transformed into a sequence of coarser matrices $Q_1, \ldots, Q_q$ such that $n_1 > \ldots > n_q$ where each n_i $(i = 1, \ldots, q)$ is the number of variables in Q_i. To obtain each intermediate matrix, we apply the idea of creating and extracting backbone variables, as explained in Section 3.2.2). This coarsening phase stops when q reaches a prefixed value called the threshold level. For the series of matrices $Q_0, \ldots, Q_q$, we call Q_0 the highest level problem and Q_q the lowest level problem.

The next phase aims to generate an initial (optimized) solution to the lowest level problem Q_q. In our case, we employ the population-based hybrid metaheuristic approach (HMA) presented in [Lü *et al.*, 2010a]. Here, an initial population of solutions P_q for Q_q is generated and improved by HMA.

Finally, the uncoarsening phase successively selects and adds some previously extracted variables to the current problem Q_i $(0 < i < q)$, leading to a higher level (and larger) problem Q_{i-1}. The solutions P_i of the current problem together with the newly added variables are projected to the new problem Q_{i-1} and further optimized by HMA to obtain an improved population P_{i-1} of solutions. The uncoarsening phase stops when the highest level $i = 0$ is reached. At this point, the best solution found during the search is returned as the final solution to the problem Q_0.

The following sections detail each phase of our multilevel algorithm.

Algorithm 3.1: Outline of the BMMA algorithm for BQO

Input: $n_0 \times n_0$ matrix Q_0; maximum coarsening level q
Output: the best solution and its objective function value
$i = 0$
while $i < q$ **do**
  $Q_{i+1} \leftarrow$ Coarsen(Q_i) /* Create coarser intermediate matrices; see Section 3.2.2 */
  $i = i + 1$
end while
$P_i \leftarrow$ Initial_Solution(Q_i) /* Generate initial solutions to the coarsest (lowest level) problem; see Section 3.2.3 */
$P_i \leftarrow$ Memetic_Refinement(P_i, Q_i) /* Apply the memetic algorithm to optimize the initial solutions; see Section 3.2.4 */
while $i > 0$ **do**
  $i = i - 1$
  $P_i \leftarrow$ Projection(P_{i+1}, Q_i) /* Back to a higher level matrix; see Section 3.2.5 */
  $P_i \leftarrow$ Memetic_Refinement(P_i, Q_i) /* Apply the memetic algorithm to optimize the current solutions */
end while

3.2.2 The backbone-based coarsening phase

The backbone multilevel memetic algorithm employs a coarsening phase to cluster backbone variables. From a given matrix Q_i ($i = 0, \ldots, q$), our coarsening procedure repeats the following steps: 1) build a solution (an approximation of the global optimum) of problem Q_i, 2) use the solution to identify a set of backbone variables and, 3) create a simplified (or lower level) problem (i.e., a smaller matrix Q_{i+1}) by extracting from Q_i the rows and columns corresponding to the backbone variables. Algorithm 3.2 gives the pseudo-code of this backbone-based coarsening phase.

Algorithm 3.2: Pseudo-code of the backbone-based coarsening phase

Input: an $n_0 \times n_0$ matrix Q_0; maximum coarsening level q
Output: a series of coarser matrices $Q_1, Q_2, \ldots, Q_q$
$i = 0$
while $i < q$ **do**
  $S_i \leftarrow$ Initial_Solution(n_i)
  $S_i \leftarrow$ Tabu_Search(S_i, Q_i)
  Record the best solution S^* and its objective function value $f(S^*)$
  Identify the backbone variables B_i in level i with regard to the solution $S_i^{\#}$ /* Formula (2) */
  Remove the corresponding row and column of each variable in B_i from Q_i to get a lower level matrix Q_{i+1}
  $i = i + 1$
end while

The coarsening phase mainly consists of a while loop which starts from the highest

level problem with $i = 0$. During the loop, we first construct an initial solution S_i by randomly assigning a value 0 or 1 to each variable of the current level problem and employ tabu search (see Section 2.2.2) to find a good local optimum for backbone identification. We additionally record the best solution S^* found so far and its objective function value $f(S^*)$.

To identify the set of backbone variables of Q_i, we use V_i to denote the set of the variables of Q_i and S_i a solution to Q_i. We first calculate the contribution $VC_k(S_i^{\#})$ of each variable x_k in V_i (see Section 2.2.4), where $S_i^{\#}$ is a solution composed of S_i and the assignment of each backbone variable acquired prior to the level i.

$$VC_k(S_i^{\#}) = (1 - 2x_k)(Q_0(k,k) + \sum_{m \in N_0 \setminus \{k\}, x_m = 1} 2Q_0(k,m)) \tag{3.1}$$

where $N_0 = \{1, 2, \ldots, n_0\}$ and x_m is the value of each variable in $S_i^{\#}$.

Then we use these $VC_k(S_i^{\#})$ values to sort the variables in a non-decreasing order and select the top na_i variables with respect to their contribution values. According to the preliminary experiments, we set $na_i = n_i \times 0.2$ if $i = 0$ and $na_i = na_{i-1} \times 0.4$ otherwise ($i > 0$). These variables constitute the set of our backbone variables denoted by B_i and are extracted from the matrix Q_i, leading to a new and simplified lower level problem Q_{i+1}.

Finally, we set $i = i + 1$ and repeat the while loop until i reaches the maximal level q (set to be equal to 3 in our experiments).

Obviously, each lower level problem Q_i ($i > 0$) is a sub-problem of the highest level problem Q_0 and the solution of Q_i plus the value assignments of the backbone variables extracted prior to level i constitute a solution of Q_0.

3.2.3 Initial population of solutions

After the coarsening phase, a solution is sought for the problem of the lowest level (Q_q). For this, an initial population of solutions P_q is first constructed as follows. Each solution in P_q is generated in such a way that each variable receives randomly either 0 or 1. If this solution is not a duplicate of any solution in the population, it becomes a member of P_q. The above procedure repeats until the number of solutions reaches the population size which we set to 8 in the algorithm. The solutions are then optimized by applying the population-based memetic algorithm HMA which is explained below.

3.2.4 The population-based memetic algorithm

The original population-based memetic algorithm HMA uses jointly the well-known uniform and a path-relinking crossover operators [Lü *et al.*, 2010a]. In this work, only the uniform crossover (UX) [Syswerda, 1989] is employed since experimental studies show that UX performs well under the multilevel framework. UX operates on two parent solutions randomly selected from the population and generates an offspring solution such that each of its variables takes the value of the corresponding variable in either parent one or parent two with equal probability.

For each offspring solution, HMA applies the tabu search procedure (see Section 2.2.2) to improve the solution. To maintain the diversity of its population, HMA uses a dedicated rule to decide whether an offspring solution is added to the population. For this, HMA introduces a quality-and-distance goodness score for the offspring solution. If this goodness score is higher than the lowest score in the population, then the offspring solution is inserted into the population and replaces the solution with the lowest goodness score. Otherwise, this offspring solution remains inserted into the population with a small probability. More details about the memetic algorithm can be found in [Lü *et al.*, 2010a].

3.2.5 The asymmetric uncoarsening phase

In a multilevel approach, the uncoarsening phase carries out the inverse of the coarsening phase and typically traverses level by level the intermediate problems from the problems of the lowest level q to the highest level 0. For each level, each coarsened variable is uncoarsened to restore the original variables of the immediate upper level $i-1$. In this section, we explain how our uncoarsening phase is realized with regard to our backbone-based coarsening phase.

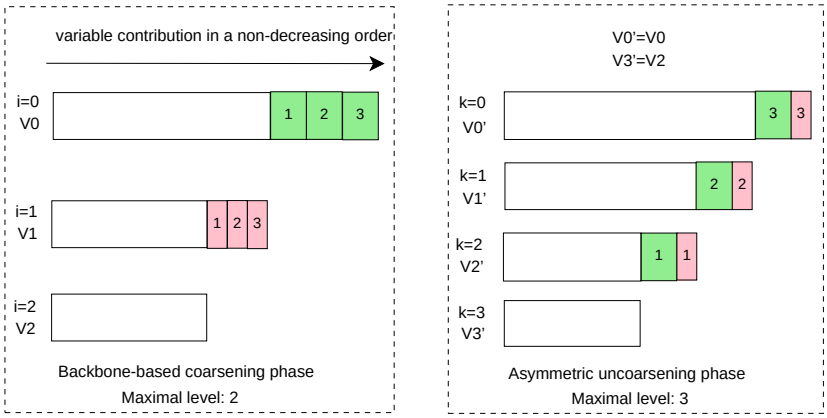


Figure 3.1: Illustration of the asymmetric uncoarsening phase

Our uncoarsening phase progressively brings back the backbone variables extracted during the coarsening phase and allows them to take part in the subsequent optimizations. To achieve this, several strategies can be applied. For example, we can add back in a systematic way the extracted backbone variables in the strict reverse order of their extraction. We will discuss this systematic uncoarsening method in Section 3.4. Here we adopt another uncoarsening strategy (called asymmetric uncoarsening) which our experiments have shown to be more effective.

The idea of our asymmetric uncoarsening phase is based on the hypothesis that the values of the backbone variables with a high contribution will have a higher probability of being optimal than the values of variables with a lower contribution. Therefore, it is desirable to freeze highly contributing variables at their assigned values as long as possible during the uncoarsening phase and to restore first those backbone variables with small

contributions. These restored variables become part of the variables considered by the optimization process applied at each uncoarsening step. Since the backbone variables are restored according to contribution values instead of the order in which they are extracted, we refer to this strategy as an *asymmetric* uncoarsening phase. Notice that asymmetric uncoarsening may lead to a number of levels different from that created by the coarsening phase.

Figure 3.1 illustrates our asymmetric uncoarsening strategy. Each box represents the set V_i of all the variables of Q_i and the length of the box represents the size of V_i. The left portion of the figure shows a coarsening phase with 2 levels which extracts the backbone variables to simplify the highest level problem Q_0 into two lower level problems Q_1 and Q_2 in sequence. The right portion of the figure shows an asymmetric uncoarsening phase with 3 levels by adding back progressively the backbone variables from the lowest level problem $Q_3^{'}$ to a series of intermediate levels and finally to the highest level problem $Q_0^{'}$.

The process is achieved as follows. As mentioned in the backbone-based coarsening phase, the variables at each coarsening step are sorted in a non-decreasing order with regard to their contribution values and a certain number of variables are selected as backbone variables. Based on this, we separate the set of the backbone variables extracted at each coarsening step into K subsets, marked as $1, \ldots, K$ (In our example, $K = 3$, see below for the meaning of K). During the uncoarsening phase, we first select the subsets marked as 1 (which contain the backbone variables with small contributions) and add the variables contained in these subsets into set $V_3^{'}$ to create the set $V_2^{'}$, leading to the higher level problem $Q_2^{'}$. The same operations are successively applied to variable subsets marked as 2 and K (In our example, $K = 3$). In this way, we finally go back to the highest level problem Q_0.

Algorithm 3.3: Pseudo-code of the asymmetric uncoarsening phase

```
1: Input: The lowest problem Q_q, a fixed uncoarsening level K > 1
2: Output: The best binary n_0-vector S* and the objective function value f(S*)
3: Divide the set of backbone variables extracted at each coarsening level into K subsets with
   equal size
4: Fetch one subset from each coarsening level and combine them to generate the set UC_k for
   each uncoarsening level k = K, ..., 1
5: k = K
6: while k > 0 do
7:     k = k − 1
8:     Uncoarsen the variables in UC_{k+1} to obtain the matrix Q_k by inserting the row and
       column of each variable in UC_{k+1} into the matrix Q_{k+1}
9:     Project each solution in population P_{k+1} to the corresponding solution in P_k
10:    P_k ← Memetic_Refinement(P_k, Q_k)
11:    Record the best solution found so far S* and its objective function f(S*)
12: end while
```

The pseudo-code of the asymmetric uncoarsening phase is shown in Algorithm 3.3. To begin with, we separate the set of backbone variables extracted at each coarsening level into K subsets where K defines the number of the uncoarsening steps needed to go back to

Table 3.1: Computational results of the BMMA algorithm

Instance	BKR	BMMA					
		f_{best}	f_{avg}	σ	t_{best}	t_{b_avg}	t_{avg}
p5000.1	8559680	8559680(1)	8558912	424	86	86	645
p5000.2	10836019	10836019(2)	10835253	527	92	219	607
p5000.3	10489137	10489137(2)	10488450	1057	344	351	630
p5000.4	12252318	12252318(2)	12251122	809	98	275	584
p5000.5	12731803	12731803(11)	12731423	493	158	326	554
p6000.1	11384976	11384976(5)	11384566	854	170	400	878
p6000.2	14333855	14333855(5)	14333101	1132	341	416	939
p6000.3	16132915	16132915(3)	16130610	1147	179	545	848
p7000.1	14478676	14478676(4)	14477235	1423	656	944	1349
p7000.2	18249948	18249948(1)	18247518	1424	951	951	1289
p7000.3	20446407	20446407(9)	20444603	3414	550	761	1132
Av.	13626885	13626885	13625708	1155	330	479	860
Deviation%.		0.000000	0.008633				

Q_0. Then we fetch one subset from each coarsening level and combine them to construct the set UC_k for each uncoarsening step k ($k = K, \ldots, 1$). This is a preparatory step for the uncoarsening phase (Alg. 3.3, lines 3-4).

From this point, an uncoarsening loop is launched with k starting at K. For each step, we reduce k by 1 and uncoarsen the variables in UC_{k+1} by including them into the set V_{k+1} to construct the set V_k and by inserting the row and column of each variable in UC_{k+1} into the matrix Q_{k+1} to obtain the matrix Q_k. In addition, the solutions of population P_k are obtained by projecting the solutions of P_{k+1} plus the added backbone variables in UC_{k+1} with their corresponding values. Finally, the memetic optimization algorithm is used to refine the population P_k. The above loop continues until the highest level $k = 0$ is reached. The best solution found so far S^* and its objective function $f(S^*)$ are always recorded.

3.3 Experimental results

3.3.1 Benchmark instances and experimental protocol

In this section, we carry out extensive experiments to evaluate the performance of our backbone multilevel memetic algorithm (BMMA). Since the multilevel scheme is designed to cope with large problem instances, we take 11 largest instances with variables from 5000 to 7000 from the Palubeckis family that are known to be very difficult to solve for several algorithms. The stopping criteria is the completion of a round of the multilevel procedure rather than a time limit and each problem instance is independently solved 20 times.

3.3.2 Computational results of the BMMA algorithm

Table 3.1 presents the results of our BMMA algorithm. Columns 1 and 2 give the instance names and the best known results in the literature. Columns 3 to 8 report respectively BMMA's best solution values f_{best} and the number of times to reach f_{best} over 20 runs

Table 3.2: Computational results of the HMA algorithm

Instance	BKR	HMA					
		f_{best}	f_{avg}	σ	t_{best}	t_{b_avg}	t_{avg}
p5000.1	8559680	8559355(1)	8558671	783	349	349	600
p5000.2	10836019	10836019(1)	10835298	262	452	452	600
p5000.3	10489137	10489137(2)	10488711	637	518	555	600
p5000.4	12252318	12252275(1)	12250982	637	589	589	600
p5000.5	12731803	12731803(9)	12731195	684	251	434	600
p6000.1	11384976	11384807(1)	11384506	812	884	884	900
p6000.2	14333855	14333855(1)	14332723	1456	761	761	900
p6000.3	16132915	16132915(2)	16130419	1098	603	641	900
p7000.1	14478676	14478676(1)	14476628	1300	1072	1072	1300
p7000.2	18249948	18249948(2)	18247600	1403	1086	1119	1300
p7000.3	20446407	20446407(6)	20444120	3728	508	855	1300
Av.	13626885	13626836	13625532	1164	643	701	873
Deviation%.		0.000358	0.009928				

in parentheses, the average solution values f_{avg}, the standard deviation σ, the best time t_{best} and the average time t_{b_avg} to reach the best solution values f_{best}, and the average time t_{avg} consumed for a BMMA run (in seconds). The last two rows report the average over the 11 instances for each evaluation criteria and the average percent deviation of the solution values from the best known values.

From Table 3.1, we find that the average objective values attained by BMMA are very close to the best known results, with an average percent deviation 0.008633%. Finally, the best and average time to reach our best solution values are only 330 and 479 seconds, respectively. In sum, our BMMA algorithm is quite effective in finding the best known values for these challenging instances.

3.3.3 Comparison with hybrid metaheuristic approach

We now assess the advantage of the multilevel scheme by comparing the BMMA algorithm with its optimization algorithm HMA which is applied at each uncoarsening level (see Section 3.2.4). For this purpose, we run HMA within the time limit t_{avg} (see Table 3.1), i.e., the time of a BMMA run. The results are shown in Table 3.2.

From Tables 3.1 and 3.2, one observes that the BMMA algorithm outperforms the HMA algorithm in terms of several different criteria. Specifically, when it comes to the best solution values found, HMA is inferior to BMMA on 3 instances (5000.1, 5000.4 and 6000.1). In addition, HMA's best and average solution deviation from the best known results are 0.000358% and 0.009928%, in comparison with BMMA's deviation values 0.000000% and 0.008633%. Furthermore, the best and average time for BMMA to find the best solution values are respectively 330 and 479 seconds which are 49% and 32% less than that of HMA. These outcomes must be qualified by observing that, as shown in [Lü *et al.*, 2010a], given longer time limits HMA consistently attains the best known results of the literature.

3.3.4 Comparison with other state-of-art algorithms

In order to further evaluate our BMMA algorithm, we compare it with several best-performing algorithms in the literature. These methods are respectively named ITS

Table 3.3: Comparison between BMMA and other algorithms : Gap to the best known result

Instance	BMMA	BGTS	D^2TS	HMA	ITS	MST2	SA	DHNN-EDA
p5000.1	0	0	325	0	700	325	1432	2244
p5000.2	0	0	0	0	0	582	582	1576
p5000.3	0	0	0	0	0	0	354	813
p5000.4	0	0	0	0	934	1643	444	1748
p5000.5	0	0	0	0	0	0	1025	1655
p6000.1	0	0	0	0	0	0	430	453
p6000.2	0	0	0	0	88	0	675	4329
p6000.3	0	0	0	0	2729	0	0	4464
p7000.1	0	0	0	0	340	1607	2579	4529
p7000.2	0	0	104	0	1651	2330	5552	5750
p7000.3	0	0	0	0	0	0	2264	1707
Av.	0	0	39	0	586	589	1394	2661

Table 3.4: Comparison between BMMA and other algorithms : Best time (seconds)

Instance	BMMA	BGTS	D^2TS	HMA	ITS	MST2	SA	DHNN-EDA
p5000.1	86	556	2855	587	507	540	605	1572
p5000.2	219	1129	1155	464	421	649	691	1572
p5000.3	351	874	1326	758	672	788	945	1572
p5000.4	275	379	838	1453	596	935	1059	1572
p5000.5	326	629	623	686	551	719	1057	1572
p6000.1	400	597	509	994	978	1037	615	2378
p6000.2	416	428	1543	1332	839	887	1085	2378
p6000.3	545	601	2088	1406	957	1218	1474	2378
p7000.1	944	1836	1217	1435	1771	1449	1952	3216
p7000.2	951	1569	849	1770	1013	1722	1738	3216
p7000.3	761	703	3520	2456	1446	2114	2138	3216
Av.	479	846	1502	1213	886	1096	1214	2240

[Palubeckis, 2006], MST2 [Palubeckis, 2004b], SA [Katayama and Narihisa, 2001], D^2TS [Glover *et al.*, 2010], HMA [Lü *et al.*, 2010a], BGTS [Wang *et al.*, 2011b] and DHNN-EDA [Wang *et al.*, 2011a]. Given the fact that all these algorithms were run under different environments, often with larger time limits, it is thus hard to make a completely fair comparison. Nevertheless, this experiment indicates that our proposed algorithm performs exceedingly well in relation to these reference state-of-the-art algorithms.

Table 3.3 compares the best solution values reported by each reference algorithm. To highlight the difference among the reference algorithms, we show the gap between the best solution of each algorithm and the best known solution. From Table 3.3, we observe that the BMMA, BGTS and HMA algorithms perform similarly well in that they are all able to attain the best known results on all the instances. In addition, the BMMA algorithm outperforms the other four reference algorithms, named ITS, MST2, SA and DHNN-EDA and is slightly better than the D^2TS algorithm. To be specific, the four reference algorithms have an average solution gap from 586 to 2661 and the D^2TS algorithm has an average solution gap of 39 to the best known values.

Table 3.4 compares the average time to reach the best solution values. The BGTS, HMA and D^2TS algorithms are run on a PC with a Pentium 2.66GHz CPU and DHNN-

Table 3.5: Comparison between the symmetric and asymmetric uncoarsening methods

Instance	BKR	Symmetric			Asymmetric		
		f_{best}	f_{avg}	σ	f_{best}	f_{avg}	σ
p5000.1	8559680	8559075	8558510	**412**	**8559680**	**8558912**	424
p5000.2	10836019	**10836019**	10834954	707	**10836019**	**10835253**	**527**
p5000.3	10489137	**10489137**	10487669	1247	**10489137**	**10488450**	**1057**
p5000.4	12252318	**12252318**	12250980	**662**	**12252318**	**12251122**	809
p5000.5	12731803	**12731803**	12731247	525	**12731803**	**12731423**	**493**
p6000.1	11384976	11384733	11384026	1285	**11384976**	**11384566**	**854**
p6000.2	14333855	14333727	14332568	**997**	**14333855**	**14333101**	1132
p6000.3	16132915	16130915	16129770	**683**	**16132915**	**16130610**	1147
p7000.1	14478676	**14478676**	14475669	**1344**	**14478676**	**14477235**	1423
p7000.2	18249948	18249844	18246763	1513	**18249948**	**18247518**	**1424**
p7000.3	20446407	**20446407**	20441970	3971	**20446407**	**20444603**	**3414**
Av.	13626885	13626605	13624921	1213	**13626885**	**13625708**	**1155**
Deviation%.	–	0.002055	0.014415	–	**0.000000**	**0.008633**	–

EDA is run on a comparable PC with a Pentium 2.8GHz CPU. The ITS, MST2 and SA algorithms are run on a Pentium III 800 PC. We transformed their original times by dividing them by 3 given that our computer is about 3 times faster than the Pentium III 800 PC [Glover *et al.*, 2010].

From Table 3.4, we can make the following observations. First, among the three algorithms (BMMA, BGTS and HMA) which reach the best known results for all the 11 instances, our proposed BMMA algorithm needs an average time of 479 seconds to reach the best solution values, against 846 and 1213 seconds for the BGTS and HMA algorithms respectively.

Second, for the 4 other algorithms (D^2TS, ITS, MST2, SA, DHNN-EDA) which fail to find the best known solutions for at least two instances, our BMMA algorithm clearly dominates all of them both in terms of the best solution values and computational efficiency. In particular, BMMA needs one fifth of the time needed by the most recent DHNN-EDA algorithm to attain much better solutions.

In sum, this experimental study demonstrates the merit of our BMMA algorithm for solving the large instances of the BQO problem.

3.4 A short discussion

In order to verify the proposed asymmetric backbone uncoarsening phase indeed works well compared to a more customary type of multilevel procedure, we also implemented a symmetric backbone uncoarsening phase, which adds back progressively the backbone variables from the lowest level Q_q to the highest level Q_0 by following the strict reverse order the backbone variables are extracted during the coarsening phase. For this experiment, we kept other components of our BMMA algorithm unchanged except the uncoarsening component. Table 3.5 shows the computational results of the two different uncoarsening methods.

As we can see in Table 3.5, the asymmetric uncoarsening performs better than the symmetric one in terms of the best, average and standard deviation values. Specifically, the asymmetric uncoarsening obtains the best known values for all the instances while

the symmetric uncoarsening leads only to 6 best known results. Moreover, the asymmetric uncoarsening reaches better average values with a smaller deviation from the best known results (0.008633% versus 0.014415% for symmetric uncoarsening). In addition, the asymmetric uncoarsening is also superior to the symmetric uncoarsening in terms of the standard deviation, with the value 1155 versus 1213.

3.5 Conclusions

Solving large random BQO problem instances is a challenging task. We have shown the multilevel approach constitutes an effective approach to cope with these large random instances. The proposed algorithm combines a backbone-based coarsening phase, an asymmetric uncoarsening phase and a memetic refinement procedure, each incorporating tabu search to obtain improved solutions. Experiments on the most challenging instances (with 5000 to 7000 variables) demonstrate that the proposed algorithm is able to find all the best known results while using much less computing time than the previous state-of-the-art algorithms. We anticipate that our approach can be further refined by investigating alternative strategies for the coarsening and uncoarsening phases.

So far, we have developed backbone guided tabu search algorithms and a backbone multilevel memetic algorithm, with the common core of reducing the problem scale to conduct intensive exploitation in a smaller search area. In two follow-up chapters, we will focus on devising new strategies of constructing initial solutions for BQO, either used in a multistart mechanism or an evolutionary framework, to direct search into newly promising area. Meantime, we will also concentrate on solving applications of BQO such as maximum cut, maximum clique, maximum vertex weight clique and minimum sum coloring problems with our devised algorithms for BQO.

Chapter 4

Probabilistic GRASP-Tabu Search

In this chapter, we propose a simple GRASP-Tabu Search algorithm working with a single solution (denoted by GRASP-TS) and an enhanced version by combining GRASP-Tabu Search algorithm with Population Management strategy (denoted by GRASP-TS/PM) to solve the BQO problem. Furthermore, we conduct an adaptation and extension of the GRASP-TS algorithm (denoted by GRASP-TS/MCPs) to solve the maximum clique problem (MCP) and maximum vertex weight clique problem (MVWCP) by recasting them into the BQO formulation. The first experiment with both GRASP-TS and GRASP-TS/PM algorithms on 31 large random BQO problem instances and 54 MaxCut instances indicate that the proposed GRASP-TS and GRASP-TS/PM are very competitive with state-of-the-art algorithms, where GRASP-TS/PM is capable of improving the best known results for 19 MaxCut instances. The second experiment with GRASP-TS/MCPs on a total of 160 DIMACS and DIMACS-VW benchmark instances indicate that GRASP-TS/MCP is competitive with or better than the other reference algorithms aiming at these two problems, obtaining improved results for 13 DIMACS-VW instances. The content of this chapter is based on the paper [Wang *et al.*, 2012g] accepted to Computers & Operations Research and the paper [Wang *et al.*, 2012b] submitted to Discrete Optimization.

Contents

4.1 Introduction

The construction of an initial solution is essential when designing an algorithm. One popular method of producing an initial solution is the so-called greedy random adaptive search procedure (GRASP). This method generally starts from an empty solution and each step enlarges this solution according to a greedy random construction heuristic until a complete solution is obtained. Another method is the use of the restart/recovery strategy, which constitutes a major principle underlying tabu search. Instead of creating an initial solution from the scratch, the restart/recovery method usually employs a destructive/constructive process that dismantles only part of a previously visited elite solution and rebuilds the remaining portion.

In this chapter, we propose two algorithms for solving BQO that combine GRASP and Tabu Search. The first, GRASP-TS, uses a basic GRASP algorithm with single solution search while the other, GRASP-TS/PM, launches each tabu search by introducing a population management strategy based on an elite reference set. In GRASP-TS/PM we pick a single solution at a time from the reference set, and operate on it, utilizing the ideas of "elite solution recovery" and "probabilistic evaluation" proposed in [Glover, 1989; Xu *et al.*, 1998]. The tabu search procedure uses the one described in Algorithm 2.2 to improve solution quality.

In addition, consider that BQO has served as a unified model for numerous combinatorial optimization problems, we investigate its applications to the MaxCut, MCP and MVWCP problems. For the solving of MaxCut, we use the GRASP-TS and GRASP-TS/PM algorithms without any adaptation after transforming it into the BQO form. For the solving of MCP and MVWCP, we extend our GRASP-TS algorithm to better address them in the form of BQO and denote the extended version as GRASP-TS/MCPs.

We evaluate the proposed GRASP-TS and GRASP-TS/PM algorithms on BQO and MaxCut benchmarks and evaluate the GRASP-TS/MCPs algorithm on MCP and MVWCP benchmarks. Comparisons with other state-of-the-art algorithms demonstrate the efficiency of the proposed algorithms.

4.2 GRASP-Tabu Search

4.2.1 General GRASP-TS procedure

The GRASP algorithm is usually implemented as a multistart procedure [Resende and Ribeiro, 2003; Resendel and Ribeiro, 2005], consisting of a randomized greedy solution construction phase and a sequel local search phase to optimize the objective function as far as possible. These two phases are carried out iteratively until a stopping condition is satisfied.

Our basic GRASP-Tabu Search algorithm (denoted by GRASP-TS) for the BQO follows this general scheme (see Algorithm 4.1) and uses a dedicated greedy heuristic for solution construction (see Section 4.2.2) as well as tabu search (see Section 2.2.2) as its local optimizer.

Algorithm 4.1: Outline of GRASP-TS for BQO

```
1: Input: matrix Q
2: Output: the best binary n-vector x* found so far and its objective value f*
3: f* = −∞
4: repeat
5:    Construct a greedy randomized solution x^0          /* Section 4.2.2 */
6:    x ← Tabu_Search(x^0)                                /* Section 2.2.2 */
7:    if f(x) > f* then
8:       x* = x,  f* = f(x)
9:    end if
10: until a stopping criterion is met
```

4.2.2 Solution construction

GRASP-TS constructs a new solution at each step according to a greedy random construction heuristic, which was originally used in probabilistic Tabu Search (PTS) [Glover, 1989; Xu *et al.*, 1996; Xu *et al.*, 1998] and motivated by the fact that the GRASP construction resembles this PTS approach.

The construction procedure consists of two phases: one is to adaptively and iteratively select some variables to receive the value 1; the other is to assign the value 0 to the left variables. Starting with an empty solution, a variable x_i with the maximum q_{ii} is picked as the first element of the partial solution.

Given the partial solution $px = \{x_{k_1}, x_{k_2}, ..., x_{k_\alpha}\}$, indexed by $pi = \{k_1, k_2, ..., k_\alpha\}$, we calculate its objective function value (OFV) as:

$$OFV(px) = \sum_{i \in pi, x_i = 1} (q_{ii} + \sum_{j \in pi, j \neq i} q_{ij} \cdot x_j) \tag{4.1}$$

At each iteration of the first phase we choose one unassigned variable according to a greedy function and then assign value 1 to it. Specifically, we calculate the objective

function increment (OFI) to the partial solution px if one unassigned variable x_j ($j \in N \setminus pi$) is added into px with value 1.

$$OFI_j(px) = q_{jj} + \sum_{i \in pi}(q_{ij} \cdot x_i) \tag{4.2}$$

At each step, all the unassigned variables are sorted in an non-increasing order according to their OFI values. Note that we only consider the first rcl variables, where rcl is called the restricted candidate list size. The r-th ranked variable is associated with a bias $b_r = 1/e^r$. Therefore, the r-th ranked variable is selected with probability $p(r)$ that is calculated as below:

$$p(r) = b_r / \sum_{j=1}^{rcl} b_j \tag{4.3}$$

Once a variable x_j is selected and added into the partial solution px with the assignment value 1, the partial solution px and its index pi, its objective function value $OFV(px)$ and the left variables' OFI values are updated correspondingly as follows:

$$px = px \cup \{x_j\},\ pi = pi \cup \{j\} \tag{4.4}$$

$$OFV(px) = OFV(px) + OFI_j(px) \tag{4.5}$$

For any variable x_k ($k \in N \setminus pi$),

$$OFI_k(px) = OFI_k(px) + q_{jk} \tag{4.6}$$

This procedure repeats until all the OFI values of the unassigned variables are negative. Then, the new solution is completed by assigning the value 0 to all the left variables.

4.3 GRASP-Tabu Search with Population Management

4.3.1 General GRASP-TS/PM procedure

Starting from the basic GRASP-TS algorithm, we introduce additional enhancements using the idea of maintaining a pool of elite solutions. By combining GRASP-TS with the population management strategy, our reinforced GRASP-TS/PM algorithm offers a better balance between intensification and diversification as a means of exploiting the search space. The general architecture of the GRASP-TS/PM algorithm is described in Alg. 4.2, which is composed of four main components: a reference set construction procedure (lines 4, 23 in Alg. 4.2, Section 4.3.2), a randomized greedy solution reconstruction operator (line 11 in Alg. 4.2, Section 4.3.3), a tabu search procedure (line 12 in Alg. 4.2, Section 2.2.2) and a reference set updating rule (lines 16-21 in Alg. 4.2, Section 4.3.4).

GRASP-TS/PM starts from an initial reference set ($RefSet$) consisting of b local optimum solutions (line 4), from which the worst solution x^w in terms of the objective

value is identified (line 6). Then, $Examine(x) = True$ indicates that solution x is to be examined (line 7). At each GRASP-TS/PM iteration, one solution x^0 is randomly chosen from the solutions to be examined in $RefSet$ ($Examine(x^0) = True$), reconstructed according to the randomized greedy heuristic and optimized by the tabu search procedure to local optimality (lines 9-12). If the improved solution x meets the criterion of updating $RefSet$, the worst solution x^w is replaced by x in $RefSet$ and $Examine(x)$ is set to be $True$ (lines 16-19). Then, the new worst solution x^w is identified (line 20). This procedure repeats until all the solutions in $RefSet$ have been examined. When this happens, $RefSet$ is rebuilt as the initial reference set construction except that the best solution x^* becomes a member of the new $RefSet$ (line 23).

Algorithm 4.2: Outline of GRASP-TS/PM for BQO

```
 1: Input: matrix Q
 2: Output: the best binary n-vector x* found so far and its objective value f*
 3: f* = −∞
 4: RefSet ← Initialize_RefSet( )                          /* Section 4.3.2 */
 5: while The stopping criterion is not satisfied do
 6:    Find the worst solution x^w in RefSet in terms of the objective value
 7:    Let Examine(x^i) = True, i = 1, ..., b (|RefSet| = b)
 8:    repeat
 9:       Randomly choose one individual x^0 from RefSet with Examine(x^0) = True
10:       Examine(x^0) = False
11:       x ← Reconstruct_Solution(x^0)                    /* Section 4.3.3 */
12:       x ← Tabu_Search(x)                               /* Section 2.2.2 */
13:       if f(x) > f* then
14:          x* = x,  f* = f(x)
15:       end if
16:       UpdateSucc ← Update_RefSet(RefSet, x)            /* Section 4.3.4 */
17:       if UpdateSucc is TRUE then
18:          RefSet ← RefSet ∪ {x} \ {x^w}
19:          Examine(x) = True
20:          Record the new worst solution x^w in RefSet
21:       end if
22:    until (∀x ∈ RefSet, Examine(x) = False)
23:    RefSet ← Reconstruct_RefSet(RefSet)                 /* Section 4.3.2 */
24: end while
```

4.3.2 RefSet initialization and reconstruction

The initial reference set contains b different local optimum solutions and is constructed as follows. Starting from scratch, we randomly generate a solution, improve it to local optimality by our tabu search procedure (Alg. 2.2, Section 2.2.2) and then add it into the reference set if this solution does not occur in $RefSet$. The procedure repeats until the size of $RefSet$ reaches b.

As shown in Algorithm 4.2, the reference set is recreated when all the solutions in

$RefSet$ have been examined. In this case, the best solution x^* becomes a member of the new $RefSet$ and the remaining solutions are generated in the same way as in constructing the initial $RefSet$.

The initial or the renewed reference set can also be obtained by applying the randomized greedy construction heuristic described in Section 4.2.2. However, experimental studies showed although there are no significant performance differences, random generation generally leads to slightly better results. For this reason, we adopt random generation of reference sets.

4.3.3 Solution reconstruction

In GRASP-TS/PM, a new solution is reconstructed based on an elite solution, borrowing the idea of elite solution recovery strategy described in [Glover, 1989; Xu *et al.*, 1998]. More specifically, GRASP-TS/PM creates a new solution by first inheriting parts of the "good" assignments of one elite solution in $RefSet$ to form a partial solution and then completing the remaining parts as GRASP-TS does. We describe how the partial elite assignments are inherited as follows.

Given an elite solution x in $RefSet$, we reconstruct a new solution by the strategic oscillation, which was proposed in the early literature [Glover, 1977] in a multi-start role to replace the customary multi-start design by using a destructive/constructive process that dismantles only part of a selected solution and rebuilds the remaining portion. Specifically, it exploits critical variables identified as *strongly determined*, and has come to be one of the basic strategies associated with tabu search.

For the identification of *strongly determined* variables, we first evaluate the *objective function contribution* of a given variable x_i for a reference solution x, denoted by $VC_i(x)$ (see Section 2.2.4). After calculating each variable's VC value, we sort all variables in a non-decreasing order according to their VC values. Then the top α variables are selected and assigned the same values as in x. Thus, the assignments of these α strongly determined variables form a partial solution. Note that, instead of using the "strongly determined" move evaluations described above, an alternative way to make the probabilistic assignments can be based on the "consistent variables" evaluations given by the population of elite solutions as shown in [Glover, 1977]. In addition, a combination of the population-based determination and the move value-based determination would also be possible, as shown in Section 2.2.4.

With the partial elite solution, we determine the assigned values of the remaining variables in the new solution using the randomized greedy heuristic as in GRASP-TS (see Section 4.2.2). Notice that GRASP-TS starts with an empty solution to construct an initial solution.

4.3.4 RefSet updating

The updating procedure of $RefSet$ is invoked each time a newly constructed solution is improved by tabu search. Specifically, the improved solution is added into $RefSet$ if it is distinct from any solution in $RefSet$ and better than the worst solution x^w in terms of

the objective function value. Under this circumstance, x^w is replaced and thus $RefSet$ is updated.

4.4 An extension of GRASP-TS to maximum clique problems

In this section, we adapt our GRASP-TS algorithm to the maximum clique and maximum vertex weight clique problems (MCPs). For this purpose, the adapted GRASP-TS/MCPs algorithm uses the union of *1-flip* and *2-flip* neighborhoods to explore the solution space, instead of depending only on *1-flip* neighborhood of GRASP-TS.

4.4.1 Maximum clique problems

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, a clique is a set of vertices $C \subseteq V$ such that every pair of distinct vertices is connected with an edge in G, i.e., the subgraph generated by C is complete. The maximum clique problem (MCP) is to find a clique of maximum cardinality. An important generalization of the MCP, known as the maximum vertex weight clique problem (MVWCP), arises when each vertex i in G is associated with a positive weight w_i. The MVWCP calls for a clique of G with the maximum $\sum_{i \in C} w_i$. It is clear that the MCP is a special case of the MVWCP with $w_i = 1$ for each vertex.

The MCP is known to be NP-hard and the associated decision problem is NP-complete [Garey and Johnson, 1979]. Furthermore, no polynomial time algorithm is known to be able to approximate the clique within a factor of $n/2^{(\log n)^{(1-\epsilon)}}$ for any $\epsilon > 0$ where n is the number of nodes in graph [Knot, 2001]. Besides its theoretical significance, the MCP provides practical applications mainly including information retrieval, signal transmission, computer vision and bioinformatics [Balus and Yu, 1986; Ji *et al.*, 2004]. Given the interest of the MCP, a large number of solution procedures has been reported in the literature, such as continuous-based heuristic [Busygin, 2006], iterated local search [Grosso *et al.*, 2008], k-opt local search [Katayama *et al.*, 2005; Pajouh *et al.*, 2011], reactive local search [Battiti and Protasi, 2001; Battiti and Mascia, 2010; Wu and Hao, 2011], phased local search [Pullan, 2006], dynamic local search [Pullan and Hoos, 2006], simulated annealing [Geng *et al.*, 2007], ant colony optimization [Solnon and Fenet, 2005] and a hybrid algorithm [Singh and Gupta, 2006].

As a generalization of the MCP, it is obvious that the MVWCP has at least the same computational complexity as the MCP. Moreover, the MVWCP has important applications in the domains of computer vision, pattern recognition and robotics [Ballard and Brown, 1983]. To solve the MVWCP, a variety of algorithms has been reported in the literature, comprising several exact algorithms [Babel, 1994; Östergård, 2001], an augmentation algorithm [Manninno and Stefanutti, 1999], a distributed computational network algorithm [Bomze *et al.*, 2000], a trust region technique algorithm [Busygin, 2006] and an effective phased local search algorithm [Pullan, 2008].

4.4.2 Transformation of MCPs to the BQO model

4.4.2.1 Linear model for the MCP and MVWCP

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, each vertex associated with a positive weight w_i, the linear programming model for the MVWCP can be formulated as follows [Sengor *et al.*, 1999]:

$$\begin{aligned} Max \quad & f(x) = \sum_{i=1}^{n} w_i x_i \\ \text{subject to:} \quad & x_i + x_j \leq 1, \ \forall \{v_i, v_j\} \in \overline{E} \\ & x_i \in \{0, 1\} \end{aligned} \tag{4.7}$$

where $n = |V|$, x_i is the binary variable associated to vertex v_i, $\overline{E}$ denotes the edge set of the complimentary graph $\overline{G}$.

Notice that if $w_i = 1$ ($i \in \{1, \ldots, n\}$), Equation (1) turns into the linear model of the MCP.

4.4.2.2 Nonlinear BQO alternative

The linear model of the MVWCP can be recast into the form of the BQO by utilizing the quadratic penalty function $g(x) = Px_ix_j$ to replace the inequality constraint of the MVWCP where P is a negative penalty scalar [Kochenberger *et al.*, 2004]. This infeasibility penalty function is considered to be valid given that $g(x) = 0$ once the inequality constraint is satisfied. To construct the nonlinear BQO model, each inequality constraint is replaced by the penalty function $g(x)$ which is added to the linear objective of Eq. 1. The resulting BQO alternative will have the same objective value as the linear form subject to all penalty items equaling to 0, indicating that all constraints are satisfied. Hence, the nonlinear BQO model can be formulated as follows:

$$Max \ \ xQx = \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} w_{ij} x_i x_j \tag{4.8}$$

where $w_{ij} = P$ if $\{v_i, v_j\} \in \overline{E}$ and 0 otherwise.

Further, a penalty scalar P is considered to be suitable as long as its absolute value $|P|$ is larger than half of the maximum linear objective function coefficient ($|P| > w_i/2$). Consider that the quadratic penalty function should be negative under the case of a maximal objective, we select $P = -1000$ and $P = -1$ for the MVWCP and MCP benchmark instances tested in our experiments, respectively. The optimized solution x obtained by solving the nonlinear BQO formulation indicates that such a choice for P enables the sum of all the quadratic penalty functions $g(x)$ to equal to 0. In other words, the subgraph constructed by the variables with the assignment of 1 in the optimized solution x forms a clique.

4.4.2.3 An example of the transformation

To illustrate the transformation from the MVWCP to the BQO, we consider the following graph:

Its linear formulation according to Equation (4.7) is:

$$\begin{aligned} Max \quad & f(x) = 2x_1 + 3x_2 + 4x_3 + 5x_4 + 2x_5 + 3x_6 \\ \text{s.t.} \quad & x_1 + x_3 \leq 1; \qquad x_1 + x_4 \leq 1; \\ & x_1 + x_6 \leq 1; \qquad x_2 + x_4 \leq 1; \\ & x_2 + x_6 \leq 1; \qquad x_3 + x_5 \leq 1; \\ & x_3 + x_6 \leq 1; \qquad x_5 + x_6 \leq 1. \end{aligned} \tag{4.9}$$

Choosing the scalar penalty $P = -15$, we obtain the following BQO model:

$$\begin{aligned} Max \quad & f(x) = 2x_1 + 3x_2 + 4x_3 + 5x_4 + 2x_5 + 3x_6 - 30x_1x_3 - 30x_1x_4 \\ & -30x_1x_6 - 30x_2x_4 - 30x_2x_6 - 30x_3x_5 - 30x_3x_6 - 30x_5x_6 \end{aligned} \tag{4.10}$$

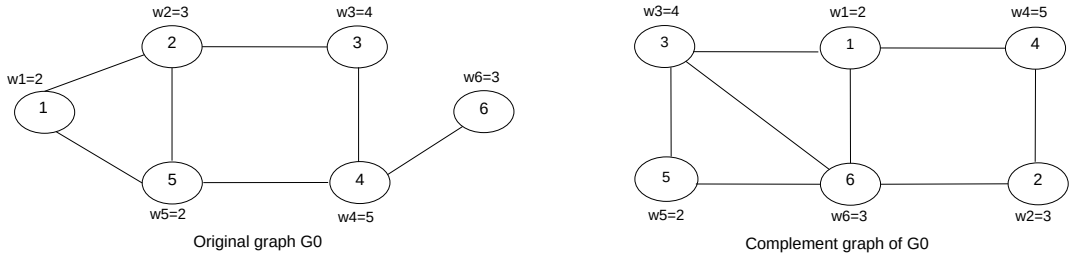


Figure 4.1: A graph sample of illustrating the transformation method of MCPs into BQO

which can be re-written as:

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{pmatrix} \times \begin{pmatrix} 2 & 0 & -15 & -15 & 0 & -15 \\ 0 & 3 & 0 & -15 & 0 & -15 \\ -15 & 0 & 4 & 0 & -15 & -15 \\ -15 & -15 & 0 & 5 & 0 & 0 \\ 0 & 0 & -15 & 0 & 2 & -15 \\ -15 & -15 & -15 & 0 & -15 & 3 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} \tag{4.11}$$

Solving this BQO problem yields $x_3 = x_4 = 1$ (all other variables equal zero) and the optimal objective function value is 9.

4.4.3 Solution construction

For the MCP, the initial solution is constructed with the same method in Section 4.2.2. For the MVWCP, each step we build a restricted candidate list RCL in which each variable

has a positive OFI value. Then we choose one variable from RCL with a probability of $1/|RCL|$ and add it assigned with value 1 into the partial solution. This process continues until RCL becomes empty. Although its simplicity, this strategy is demonstrated to be effective for the MVWCP.

4.4.4 Tabu search

For each initial solution generated by the greedy probabilistic construction, we apply an extended version of Alg. 2.2 to further improve its quality. The tabu search in Alg. 2.2 uses a simple *1-flip* move (flipping the value of a single variable x_i to its complementary value $1 - x_i$) to conduct the neighborhood search. Here we not only exploit the *1-flip* move but also incorporate a *2-flip* move (flipping the values of a pair of variables (x_i, x_j) to their corresponding complementary values $(1-x_i, 1-x_j)$) [Glover and Hao, 2010]. The above two types of moves constitute the neighborhood structures N1 and N2.

One drawback of an N2 move is the amount of time it consumes. Considerable effort is required to evaluate all the *2-flip* moves because the neighborhood structure N2 generates $n(n-1)/2$ solutions at each iteration. To overcome this obstacle, we employ an instance of the successive filter candidate list strategy of [Glover and Laguna, 1997] by restricting our attention to moves in N2 that can be obtained as follows. The first step is to examine all the *1-flip* moves of the current solution x, and if any of these moves is improving we go ahead and select it. But if no *1-flip* move is improving, we limit attention to *1-flip* moves that produce an objective function value no worse than $f(x) + 2P$, where $f(x)$ is the objective function value of x. (Recall that we are maximizing and the penalty P is negative. This implies that the candidate *1-flip* moves can violate at most a single additional constraint beyond those violated by x, since a single constraint is penalized as $Px_{ij} + Px_{ji}$ and hence incurs a penalty of $2P$.) Finally, only the *1-flip* moves that pass this filtering criterion are allowed to serve as the source of potential *2-flip* moves.

Tabu search typically introduces a tabu list to assure that solutions visited within a certain number of iterations, called the tabu tenure, will not be revisited. In the present study, each time a variable x_i is flipped, this variable enters into the tabu list and cannot be flipped for the next $TabuTenure$ iterations. For the neighborhood structure N1, our tabu search algorithm then restricts consideration to variables not forbidden by the tabu list. For the neighborhood structure N2, we consider a move to be non-tabu only if both variables associated with this move are not in the tabu list and only such moves are considered during the search process.

For each iteration in our tabu search procedure, a non-tabu move is chosen according to the following rules: (1) if the best move from N1 leads to a solution better than the best solution obtained in this round of tabu search, we select the best move from N1, thus bypassing consideration of N2; (2) if no such a move in N1 exists, we select the best move from the combined pool of N1 and N2. A simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution. Once a move is performed, we fast update the set of variables affected by this move (see Section 1.2.1.4). The tabu search procedure stops when the best solution cannot be improved within a given number μ of moves.

4.5 Experimental results

4.5.1 Results on BQO benchmark

The first experiment is to evaluate the GRSAP-TS and GRASP-TS/PM algorithms for the 31 challenging BQO instances. There are six parameters in the proposed algorithms, i.e., time limit to terminate algorithms, restricted candidate list size (rcl), the size of $RefSet$ (b), the inheriting parts of the "good" assignments of one elite solution in $RefSet$ (α), tabu tenure (tt), improvement cutoff of tabu search (μ). We set $rcl = 50$, $b = 10$ and $\alpha = 0.25n$ for this set of BQO benchmark. Notice that these parameter settings are also used for the MaxCut and MCPs benchmarks. In addition, we set the other parameters as follows: (1) time limit: 1 minute for the 10 ORLIB instances and 5, 10, 20, 30 and 50 minutes, respectively for the 21 Palubeckis instances with 3000, 4000, 5000, 6000 and 7000 variables (this time cutoff is the same as in [Lü *et al.*, 2010a]) (2) $tt = [n/100, n/100 + 10]$ (3) $\mu = 5n$.

These parameter values were determined based on preliminary experiments. For instance, we experimented with selecting $rcl \in \{50,\ 0.1 \cdot n,\ 0.2 \cdot n,\ 0.3 \cdot n,\ 0.4 \cdot n,\ 0.5 \cdot n,\ 1.0 \cdot n\}$ on a preliminary set of problem instances and observed that $rcl = 50$ is a good compromise in terms of the best objective value, average average objective value, standard deviation and CPU time. Better parameter values would be possible in some cases, but as we see below, the proposed algorithms with the given parameter values are able to achieve a highly competitive performance.

Table 4.1 shows the computational statistics of the GRASP-TS and GRASP-TS/PM algorithms on the 31 BQO instances. Columns 1 and 2 respectively give the instances names and the best known results BKR in the literature. Note that these best results were first reported in [Palubeckis, 2004b; Palubeckis, 2006] and recently improved in [Glover *et al.*, 2010; Lü *et al.*, 2010a]. The columns under headings "GRASP-TS" and "GRASP-TS/PM" list the best objective value f_{best}, the average objective value f_{avg}, the standard variance of the objective value σ and the average CPU time $time$ (seconds) for reaching f_{best} over the 20 runs. Furthermore, the last row "Av." indicates the summary of average performances of our algorithms.

Table 4.1 discloses that generally GRASP-TS/PM performs better than GRASP-TS on these BQO benchmarks. First, we notice that both GRASP-TS and GRASP-TS/PM can reach all the previous best objective values for the 31 BQO instances within the given time limit, demonstrating their very good performance in finding the best solution. However, GRASP-TS/PM is superior to GRASP-TS when it comes to the average gap to the previous best objective values g_{avg} on these instances, 316.9 versus 509.6, although the CPU time to obtain the best solution is slightly longer. Moreover, the average variance of GRASP-TS/PM is 252.0, which is much smaller than 386.4 of GRASP-TS.

In order to further evaluate our GRASP-TS and GRASP-TS/PM algorithms, we compare our results with some best performing algorithms in the literature. Notice that a completely fair comparison is impossible since the reference algorithms are implemented by different authors and run under different conditions. Our comparison here on the BQO instances as well as that on the MaxCut problem are thus presented only for indicative

Table 4.1: Computational results of GRASP-TS and GRASP-TS/PM on BQO instances

Instance	BKR	GRASP-TS				GRASP-TS/PM			
		f_{best}	f_{avg}	σ	$time$	f_{best}	f_{avg}	σ	$time$
b2500.1	1515944	1515944	1515944	0	12	1515944	1515944	0	12
b2500.2	1471392	1471392	1471138	218	38	1471392	1471257	154	52
b2500.3	1414192	1414192	1414179	58	34	1414192	1414192	0	33
b2500.4	1507701	1507701	1507701	0	11	1507701	1507701	0	10
b2500.5	1491816	1491816	1491816	0	13	1491816	1491816	0	17
b2500.6	1469162	1469162	1469162	0	24	1469162	1469162	0	20
b2500.7	1479040	1479040	1479014	63	34	1479040	1479039	3	60
b2500.8	1484199	1484199	1484198	4	27	1484199	1484199	0	25
b2500.9	1482413	1482413	1482407	6	30	1482413	1482412	4	42
b2500.10	1483355	1483355	1483308	142	31	1483355	1483355	0	56
p3000.1	3931583	3931583	3931573	44	103	3931583	3931583	0	113
p3000.2	5193073	5193073	5193073	0	47	5193073	5193073	0	63
p3000.3	5111533	5111533	5111501	86	103	5111533	5111533	0	153
p3000.4	5761822	5761822	5761822	0	78	5761822	5761822	0	53
p3000.5	5675625	5675625	5675514	162	160	5675625	5675573	180	172
p4000.1	6181830	6181830	6181830	0	128	6181830	6181830	0	141
p4000.2	7801355	7801355	7801098	709	316	7801355	7801332	47	363
p4000.3	7741685	7741685	7741679	19	232	7741685	7741685	0	253
p4000.4	8711822	8711822	8711783	72	357	8711822	8711812	30	321
p4000.5	8908979	8908979	8908376	985	206	8908979	8908643	726	385
p5000.1	8559680	8559680	8558628	554	893	8559680	8558895	422	530
p5000.2	10836019	10836019	10835517	469	553	10836019	10835858	288	760
p5000.3	10489137	10489137	10488369	722	86	10489137	10488780	321	570
p5000.4	12252318	12252318	12250975	635	662	12252318	12251098	641	960
p5000.5	12731803	12731803	12731151	509	478	12731803	12731710	221	804
p6000.1	11384976	11384976	11384218	476	1314	11384976	11384613	205	1415
p6000.2	14333855	14333855	14332637	786	1255	14333855	14333119	843	229
p6000.3	16132915	16132915	16130966	1254	371	16132915	16131166	1224	1350
p7000.1	14478676	14478676	14476478	1128	2798	14478676	14477110	881	2540
p7000.2	18249948	18249948	18247495	1566	2178	18249948	18248499	901	1938
p7000.3	20446407	20446407	20444906	1310	1704	20446407	20445621	720	2809
Av.		0*	509.6*	386.4	460.5	0*	316.9*	252.0	524.2

*: The gaps to the best known result ($BKR - f_{best}$,$BKR - f_{avg}$) are calculated.

Table 4.2: Best results comparison among GRASP-TS, GRASP-TS/PM and other state-of-the-art algorithms on larger BQO instances

Instance	BKR	best solution gap (i.e., $BKR - f_{best}$)							
		ITS	MST1	MST2	SA	D^2TS	HMA	GRASP-TS	GRASP-TS/PM
p5000.1	8559680	700	3016	325	1432	325	0	0	0
p5000.2	10836019	0	0	582	582	0	0	0	0
p5000.3	10489137	0	3277	0	354	0	0	0	0
p5000.4	12252318	934	3785	1643	444	0	0	0	0
p5000.5	12731803	0	5150	0	1025	0	0	0	0
p6000.1	11384976	0	3198	0	430	0	0	0	0
p6000.2	14333855	88	10001	0	675	0	0	0	0
p6000.3	16132915	2729	11658	0	0	0	0	0	0
p7000.1	14478676	340	7118	1607	2579	0	0	0	0
p7000.2	18249948	1651	8902	2330	5552	104	0	0	0
p7000.3	20446407	0	17652	0	2264	0	0	0	0
Av.		585.6	6705.2	589.7	1394.3	39	0	0	0

purposes and should be interpreted with caution. Nevertheless, our experiments provide an indication of the performance of the proposed algorithms relative to the state-of-the-art algorithms.

For this purpose, we restrict our attention to comparisons in terms of quality with six methods that have reported the best results for the most challenging problems. These methods are respectively named ITS [Palubeckis, 2006], MST1 [Palubeckis, 2004b], MST2 [Palubeckis, 2004b], SA [Katayama and Narihisa, 2001], D^2TS [Glover *et al.*, 2010] and HMA [Lü *et al.*, 2010a]. Moreover, we focus only on the 11 largest and most difficult instances with variables from 5000 to 7000 since the best results for instances with size smaller than 5000 can be easily reached by all these state-of-the art algorithms.

Table 4.2 shows the gap to the best known objective value of our GRASP-TS and GRASP-TS/PM algorithms compared with the reference algorithms. The last row presents the averaged results over the 11 instances. The results of the first 4 reference algorithms are directly extracted from [Palubeckis, 2006], the results of D^2TS are from [Glover *et al.*, 2010] and the results of HMA come from [Lü *et al.*, 2010a]. Note that the results of all these algorithms are obtained almost under the same time limit.

From Table 4.2 it is observed that both GRASP-TS and GRASP-TS/PM outperform the 5 reference algorithms (ITS, MST1, MST2, SA and D^2TS) and are also competitive with our HMA algorithm in terms of the quality of the best solution, demonstrating the efficacy of the two GRASP-Tabu Search algorithms in finding the best objective values. In order to further discriminate between GRASP-TS, GRASP-TS/PM and HMA, we compare the average solution gaps (20 independent runs) to the best known objective values over 31 instances. We find that GRASP-TS/PM is slightly better than HMA with a gap of 316.9 against 332.2. Also GRASP-TS is inferior to both GRASP-TS/PM and HMA with a gap of 509.6.

We also apply the Friedman non-parametric statistical test followed by the Post-hoc test to the results in Table 4.2 to see whether there exists significant performance differences between our proposed algorithms and the reference methods. Firstly, we observe from the Friedman test that there is a significant difference among the compared algorithms

(with a *p-value* of 3.737e-06). Furthermore, the Post-hoc analysis shows that GRASP-TS is significantly better than MST1 and SA (with *p-values* of 5.330108e-06 and 3.622423e-03, respectively) but is not significantly better than ITS, MST2 and D2TS (with *p-values* of 5.347580e-01, 5.347227e-01 and 9.995954e-01, respectively).

Since the best solution values obtained by GRASP-TS, GRASP-TS/PM and HMA are the same, we carry out the above statistical tests with regard to the average solution values. Notice that 31 BQO instances are considered in this experiment. Firstly, from the Friedman test, we confirm that there exists a significant performance difference between GRASP-TS, GRASP-TS/PM and HMA (with a *p-value* of 4.267e-06). Furthermore, the Post-hoc analysis shows that both GRASP-TS/PM and HMA are significantly better than GRASP (with *p-values* of 4.089688e-06 and 3.296903e-04, respectively). However, we cannot conclude whether GRASP-TS/PM or HMA performs significantly better than the other (with a *p-value* of 5.999315e-01).

4.5.2 Results on MaxCut benchmark

In this section, we directly solve the MaxCut problem with the proposed GRASP-TS and GRSASP-TS/PM algorithms. The formulation of MaxCut as well as the transformation of MaxCut into BQO can be found in Section 1.1.2.2. The experiment is conducted on 54 instances and uses the following parameter settings: (1) time limit=30 minutes, comparable with the time reported in [Marti *et al.*, 2009] (2) $tt = [n/10, n/10 + 10]$ (3) $\mu = 10000$. The experimental results are summarized in Table 4.3, using the same statistics as in Table 4.1. The previous best results are from references [Burer *et al.*, 2001; Festa *et al.*, 2002; Marti *et al.*, 2009; Palubeckis, 2004a; Shylo and Shylo, 2010].

From Table 4.3, we observe that GRASP-TS/PM outperforms GRASP-TS with respect to the best and average objective values. Specifically, GRASP-TS/PM has the best gap relative to the previous best result of 0.78 on average over 54 instances while GRASP-TS has a gap of 5.76. Moreover, GRASP-TS/PM has an average objective gap over 20 runs relative to the previous best known value of 4.50, which is two times smaller than obtained by GRASP-TS with a gap of 9.68. However, GRASP-TS/PM needs slightly more CPU time to reach its best solutions and its objective value variance is slightly larger than GRASP-TS. It is noteworthy that both methods achieve exceedingly high quality outcomes, although GRASP-TS/PM emerges the clear winner. In particular, GRASP-TS/PM improves the previous best known results on 19 instances (in bold), while GRASP-TS improves the previous best known results for 9 instances.

For comparative purposes, Table 4.4 also includes the results of three state-of-the-art algorithms. These reference methods are Scatter Search [Marti *et al.*, 2009] (column 3), CirCut heuristic [Burer *et al.*, 2001] (column 4) and VNSPR [Festa *et al.*, 2002] (column 5). The last three rows of Table 4.4 show the summary of the comparison between each algorithm including ours and the previous best known results. The rows *better*, *equal*, *worse* respectively denote the number of instances for which each algorithm gets better, equal and worse results than the previous best known results. The results of these reference algorithms are directly extracted from [Marti *et al.*, 2009] (results obtained on a personal computer with a 3.2GHz Intel Xenon processor and 2.0 GB of RAM which is comparable to

Table 4.3: Computational results GRASP-TS and GRASP-TS/PM on MaxCut instances

Instance	BKR	GRASP-TS				GRASP-TS/PM			
		f_{best}	f_{avg}	σ	$time$	f_{best}	f_{avg}	σ	$time$
G1	11624	11624	11624.0	0.0	100	11624	11624.0	0.0	47
G2	11620	11620	11619.6	0.7	677	11620	11620.0	0.0	210
G3	11622	11620	11619.9	0.5	854	11620	11620.0	0.0	297
G4	11646	11646	11646.0	0.0	155	11646	11646.0	0.0	49
G5	11631	11631	11631.0	0.0	235	11631	11631.0	0.0	232
G6	2178	2178	2177.4	0.6	453	2178	2177.9	0.2	518
G7	2003	**2006**	2005.9	0.3	304	**2006**	2006.0	0.0	203
G8	2003	**2005**	2004.7	0.5	565	**2005**	2004.9	0.3	596
G9	2048	**2054**	2053.4	0.7	581	**2054**	2053.6	0.7	559
G10	1994	**2000**	1999.3	0.6	845	**2000**	1999.3	0.7	709
G11	564	564	564.0	0.0	18	564	564.0	0.0	10
G12	556	556	555.5	0.9	723	556	556.0	0.0	233
G13	582	582	581.1	1.0	842	582	581.8	0.6	516
G14	3064	3062	3061.6	0.5	812	3063	3062.1	0.4	1465
G15	3050	3040	3037.7	1.0	419	3050	3049.1	0.2	1245
G16	3052	3049	3044.4	1.2	1763	3052	3050.9	0.7	335
G17	3043	3043	3040.6	0.8	1670	**3047**	3045.8	1.1	776
G18	988	**992**	989.3	1.3	977	**992**	992.0	0.0	81
G19	903	**906**	904.4	1.0	490	**906**	906.0	0.2	144
G20	941	941	941.0	0.0	578	941	941.0	0.0	80
G21	931	927	925.7	0.8	484	931	930.6	0.5	667
G22	13359	13346	13336.1	4.9	983	13349	13342.4	3.0	1276
G23	13342	13318	13311.7	3.7	1668	13332	13322.4	4.4	326
G24	13337	13313	13306.0	4.5	643	13324	13317.3	3.7	1592
G25	13326	13315	13306.9	3.8	767	13326	13318.1	3.3	979
G26	13314	13306	13294.8	4.9	1483	13313	13303.3	4.2	1684
G27	3318	3316	3304.2	4.5	256	**3325**	3318.1	3.7	832
G28	3285	3275	3267.8	3.5	82	**3287**	3277.4	3.8	1033
G29	3389	3386	3370.9	7.1	21	**3394**	3384.5	6.0	993
G30	3403	3395	3383.3	4.4	1375	3402	3393.4	4.1	1733
G31	3288	3286	3279.4	3.7	904	**3299**	3287.7	4.2	888
G32	1410	1394	1391.8	1.4	903	1406	1397.3	3.1	1232
G33	1382	1368	1365.6	1.0	1501	1374	1369.1	2.1	506
G34	1384	1376	1371.3	1.7	1724	1376	1372.5	2.2	1315
G35	7684	7653	7648.6	2.6	1124	7661	7657.4	2.7	1403
G36	7677	7646	7641.1	2.4	543	7660	7652.1	5.1	1292
G37	7689	7664	7657.1	2.4	983	7670	7662.0	4.1	1847
G38	7681	7653	7644.3	4.0	667	7670	7659.8	4.8	1296
G39	2395	2388	2381.9	2.5	911	**2397**	2387.1	5.0	742
G40	2387	2378	2359.6	5.8	134	**2392**	2384.3	5.8	1206
G41	2398	2367	2355.3	6.9	612	2398	2383.7	8.2	1490
G42	2469	2453	2447.5	2.9	1300	**2474**	2461.7	5.6	1438
G43	6660	6660	6658.3	1.0	969	6660	6659.4	0.7	931
G44	6650	6649	6647.1	1.1	929	6649	6647.7	0.8	917
G45	6654	6654	6652.5	0.8	1244	6654	6652.6	0.7	1791
G46	6645	**6648**	6645.4	1.4	702	**6649**	6646.0	1.7	405
G47	6656	6656	6654.5	1.0	1071	6656	6655.4	0.7	725
G48	6000	6000	6000.0	0.0	13	6000	6000.0	0.0	4
G49	6000	6000	6000.0	0.0	27	6000	6000.0	0.0	6
G50	5880	5880	5880.0	0.0	80	5880	5880.0	0.0	14
G51	3846	3843	3839.3	1.9	628	**3847**	3843.8	1.5	701
G52	3849	3844	3840.6	1.5	1274	**3850**	3846.8	1.9	1228
G53	3846	**3847**	3844.3	1.3	1317	**3848**	3845.8	1.0	1419
G54	3846	**3848**	3845.6	1.2	1231	**3850**	3847.8	1.9	1215
Av.		5.76*	9.68*	1.89	770.6	0.78*	4.50*	1.96	804.3

*: The gaps to the best known result ($BKR - f_{best}$,$BKR - f_{avg}$) are calculated.

Table 4.4: Best results comparison among GRASP-TS, GRASP-TS/PM and other state-of-the-art algorithms on MaxCut instances

Instance	BKR	best solution value				
		SS	CirCut	VNSPR	GRASP-TS	GRASP-TS/PM
G1	11624	11624	11624	11621	11624	11624
G2	11620	11620	11617	11615	11620	11620
G3	11622	11622	11622	11622	11620	11620
G4	11646	11646	11641	11600	11646	11646
G5	11631	11631	11627	11598	11631	11631
G6	2178	2165	2178	2102	2178	2178
G7	2003	1982	2003	1906	**2006**	**2006**
G8	2003	1986	2003	1908	**2005**	**2005**
G9	2048	2040	2048	1998	**2054**	**2054**
G10	2000	1993	1994	1910	**2000**	**2000**
G11	564	562	560	564	564	564
G12	556	552	552	556	556	556
G13	582	578	574	580	582	582
G14	3064	3060	3058	3055	3062	3063
G15	3050	3049	3049	3043	3040	3050
G16	3052	3045	3045	3043	3049	3052
G17	3043	3043	3037	3030	3043	**3047**
G18	988	988	978	916	**992**	**992**
G19	903	903	888	836	**906**	**906**
G20	941	941	941	900	941	941
G21	931	930	931	902	931	931
G22	13359	13346	13346	13295	13346	13349
G23	13342	13317	13317	13290	13318	13332
G24	13337	13303	1314	13276	13313	13324
G25	13326	13320	13326	12298	13315	13326
G26	13314	13294	13314	12290	13306	13313
G27	3318	3318	3306	3296	3316	**3325**
G28	3285	3285	3260	3220	3275	**3287**
G29	3389	3389	3376	3303	3389	**3394**
G30	3403	3403	3385	3320	3395	3402
G31	3288	3288	3285	3202	3286	**3299**
G32	1410	1398	1390	1396	1394	1406
G33	1382	1362	1360	1376	1368	1374
G34	1384	1364	1368	1372	1376	1376
G35	7684	7668	7670	7635	7653	7661
G36	7677	7660	7660	7632	7646	7660
G37	7689	7664	7666	7643	7664	7670
G38	7681	7681	7646	7602	7653	7670
G39	2395	2393	2395	2303	2388	**2397**
G40	2387	2374	2387	2302	2378	**2392**
G41	2398	2386	2398	2298	2367	2398
G42	2469	2457	2469	2390	2453	**2474**
G43	6660	6656	6656	6659	6660	6660
G44	6650	6648	6643	6642	6649	6649
G45	6654	6642	6652	6646	6654	6654
G46	6645	6634	6645	6630	**6648**	**6649**
G47	6656	6649	6656	6640	6656	6656
G48	6000	6000	6000	6000	6000	6000
G49	6000	6000	6000	6000	6000	6000
G50	5880	5880	5880	5880	5880	5880
G51	3846	3846	3837	3808	3843	**3847**
G52	3849	3849	3833	3816	3844	**3850**
G53	3846	3846	3842	3802	**3847**	**3848**
G54	3846	3846	3842	3820	**3848**	**3850**
Better	—	0	0	0	9	19
Matched	—	22	20	6	18	20
Worse	—	32	34	48	27	15

our computer with a Pentium 2.83GHz and 8 GB RAM). However, not all the algorithms are run under the same conditions and hence, this comparison should be interpreted with caution. Notice also that while some reference algorithms are MaxCut specific heuristics, our algorithm is designed for the more general BQO problem.

Table 4.4 discloses that GRASP-TS/PM and GRASP-TS can find new best results on 19 and 9 instances, respectively among the 54 instances and both match the previous best known results on 20 and 18 instances. For the tested instances, both GRASP-TS/PM and GRASP-TS perform better than the reference algorithms. In particular, GRASP-TS/PM (GRASP-TS respect.) fails to reach the best known results for 15 (27 respect.) instances while the reference algorithms SS, CirCut and VNSPR fail on 32, 34 and 48 instances, respectively. The computing times (in seconds) to reach the best solution of GRASP-TS (770) and GRASP-TS/PM (804) are larger than SS (621) and CirCut (616) but much smaller than VNSPR (64505).

As for Table 4.2, we apply the Friedman test and the Post-hoc test to the results in Table 4.4 to see whether there are significant performance differences between the proposed methods and other competitors on the 54 MaxCut instances. Firstly, we discover from the Friedman test that SS, CirCut, VNSPR, GRASP-TS and GRASP-TS/PM demonstrate significant differences (with a p-value of 2.2e-16). Secondly, when comparing GRASP-TS with SS, CirCut and VNSPR, the Post-hoc analysis indicates that GRASP-TS is significantly better than VNSPR (with a p-value of 3.788002e-10) but is not significantly better than SS and CirCut (with *p-values* of 4.534268e-01 and 9.358923e-02, respectively). Thirdly, when comparing GRASP-TS/PM with SS, CirCut and VNSPR, the Post-hoc analysis indicates that GRASP-TS/PM is significantly better than SS, CirCut and VNSPR (with *p-values* of 4.059707e-06, 2.433377e-08, 0.000000e+00, respectively). Finally, we observe that GRASP-TS/PM is significantly better than GRASP-TS (with a *p-value* of 6.795472e-03).

In summary, the computational results on the 85 random and structured instances demonstrate the efficacy of our proposed GRASP-Tabu Search algorithms for solving the BQO problems, with GRASP-TS/PM emerging as superior to the other methods studied in our comparative tests.

4.5.3 Results on MCP benchmark

This experiment evaluates the performance of the GRASP-TS/MCPs algorithm on 80 DIMACS maximum clique instances. For this set of benchmark, we use the following parameter settings: (1) time limit: 1 minute for instances of dsjc, keller except keller6, mann except mann_a45 and mann_a81, hamming, gen, c-fat, johnson, p_hat, san except san1000, and sanr; 10 minutes for instances of brock except those with 800 variables, C except C2000.9 and C4000.5, and keller6; 60 minutes for instances brock800_1, brock800_2, brock800_3, brock800_4 and mann_a45; 600 minutes for instances C2000.9, C4000.5, mann_a81 and san1000 (2) $tt = [5, 12]$ (3) $\mu = 1000$ for instances of brock, dsjc and mann; $\mu = 50$ for c-fat and san; $\mu = 10000$ for other instances.

Table 4.5 shows computational statistics of GRASP-TS/MCPs for the set of 80 DIMACS benchmark instances. Columns 1 to 3 give the instances names (*Instance*), num-

ber of vertices ($Order$) and the best known results (BKR) ever reported in the literature [Katayama *et al.*, 2005; Singh and Gupta, 2006; Pullan, 2006; Pullan and Hoos, 2006; Pullan, 2008; Wu and Hao, 2011]. The columns under heading GRASP-TS/MCPs report the best solution values (f_{best}), the average solution values (f_{avg}), the standard deviations (σ), the number of times of reaching (f_{best}) over 100 runs ($Succ.$), and the average CPU time in seconds (t_{b_avg}) of $Succ.$ runs (f_{best}). Results marked in bold in the fourth column and seventh column respectively indicate that GRASP-TS/BQO is able to reach the best known results BKR on these instances and reach BKR for each run out of 100 runs. An entry with $< \epsilon$ signifies that the average CPU time was less than 0.01 second.

From Table 4.5, we observe that GRASP-TS/MCPs is able to reach the best known results for 77 out of 80 instances. For the 3 remaining instances GRASP-TS/MCPs gets a value of 79 for C2000.9, 344 for mann_a45 and 1098 for mann_a81, with small gaps 1, 1, and 2 to the best known results. Moreover, GRASP-TS/MCPs has a success rate of 100% for 67 instances. Only a few of the best algorithms that are specifically tailored for solving the maximum clique problem can compete with this result (see also Table 4.7). The above results demonstrate thus the effectiveness of our proposed method to tackle the maximum clique problem via the binary quadratic optimization framework.

Table 4.5: Computational results of GRASP-TS/MCPs on 80 MCP instances

Instance	Order	BKR	GRASP-TS/MCPs				
			f_{best}	f_{avg}	σ	Succ.	t_{b_avg}
brock200_1	200	21	**21**	21.0	0.0	**100**	0.11
brock200_2	200	12	**12**	12.0	0.0	**100**	0.33
brock200_3	200	15	**15**	15.0	0.0	**100**	2.20
brock200_4	200	17	**17**	17.0	0.0	**100**	0.73
brock400_1	400	27	**27**	25.8	1.0	42	265.81
brock400_2	400	29	**29**	29.0	0.4	99	139.10
brock400_3	400	31	**31**	31.0	0.0	**100**	20.87
brock400_4	400	33	**33**	33.0	0.0	**100**	11.92
brock800_1	800	23	**23**	21.1	0.3	3	1600.37
brock800_2	800	24	**24**	21.2	0.7	6	2165.91
brock800_3	800	25	**25**	22.5	1.1	17	2067.96
brock800_4	800	26	**26**	23.0	2.4	40	1719.04
C125.9	125	34	**34**	34.0	0.0	**100**	$< \epsilon$
C250.9	250	44	**44**	44.0	0.0	**100**	0.01
C500.9	500	57	**57**	57.0	0.0	**100**	0.95
C1000.9	1000	68	**68**	68.0	0.0	**100**	12.01
C2000.5	2000	16	**16**	16.0	0.1	98	59.43
C2000.9	2000	80	79	78.4	0.5	41	17591.58
C4000.5	4000	18	**18**	17.9	0.2	95	8953.89
DSJC500.5	500	13	**13**	13.0	0.0	**100**	0.16
DSJC1000.5	1000	15	**15**	15.0	0.0	**100**	13.87
keller4	171	11	**11**	11.0	0.0	**100**	$< \epsilon$
keller5	776	27	**27**	27.0	0.0	**100**	0.12
keller6	3361	59	**59**	58.6	0.8	79	238.55
MANN_a9	45	16	**16**	16.0	0.0	**100**	$< \epsilon$
MANN_a27	378	126	**126**	126.0	0.0	**100**	17.30
MANN_a45	1035	345	344	343.0	0.3	5	2073.98
MANN_a81	3321	1100	1098	1097.0	0.3	5	18847.50
hamming6-2	64	32	**32**	32.0	0.0	**100**	$< \epsilon$
hamming6-4	64	4	**4**	4.0	0.0	**100**	$< \epsilon$
hamming8-2	256	128	**128**	128.0	0.0	**100**	$< \epsilon$
hamming8-4	256	16	**16**	16.0	0.0	**100**	$< \epsilon$
hamming10-2	1024	512	**512**	512.0	0.0	**100**	0.23

(Continued...)

Instance	Order	BKR	GRASP-TS/MCPs				
			f_{best}	f_{avg}	σ	Succ.	t_{b_avg}
hamming10-4	1024	40	**40**	40.0	0.0	**100**	0.06
gen200_p0.9_44	200	44	**44**	44.0	0.0	**100**	0.01
gen200_p0.9_55	200	55	**55**	55.0	0.0	**100**	$< \epsilon$
gen400_p0.9_55	400	55	**55**	55.0	0.0	**100**	0.21
gen400_p0.9_65	400	65	**65**	65.0	0.0	**100**	0.01
gen400_p0.9_75	400	75	**75**	75.0	0.0	**100**	0.01
c-fat200-1	200	12	**12**	12.0	0.0	**100**	0.01
c-fat200-2	200	24	**24**	24.0	0.0	**100**	0.02
c-fat200-5	200	58	**58**	58.0	0.0	**100**	0.01
c-fat500-1	500	14	**14**	14.0	0.0	**100**	0.05
c-fat500-2	500	26	**26**	26.0	0.0	**100**	0.02
c-fat500-5	500	64	**64**	64.0	0.0	**100**	0.13
c-fat500-10	500	126	**126**	126.0	0.0	**100**	0.05
johnson8-2-4	28	4	**4**	4.0	0.0	**100**	$< \epsilon$
johnson8-4-4	70	14	**14**	14.0	0.0	**100**	$< \epsilon$
johnson16-2-4	120	8	**8**	8.0	0.0	**100**	$< \epsilon$
johnson32-2-4	496	16	**16**	16.0	0.0	**100**	0.01
p_hat300-1	300	8	**8**	8.0	0.0	**100**	$< \epsilon$
p_hat300-2	300	25	**25**	25.0	0.0	**100**	$< \epsilon$
p_hat300-3	300	36	**36**	36.0	0.0	**100**	0.01
p_hat500-1	500	9	**9**	9.0	0.0	**100**	0.01
p_hat500-2	500	36	**36**	36.0	0.0	**100**	0.01
p_hat500-3	500	50	**50**	50.0	0.0	**100**	0.02
p_hat700-1	700	11	**11**	11.0	0.0	**100**	0.21
p_hat700-2	700	44	**44**	44.0	0.0	**100**	0.01
p_hat700-3	700	62	**62**	62.0	0.0	**100**	0.02
p_hat1000-1	1000	10	**10**	10.0	0.0	**100**	0.05
p_hat1000-2	1000	46	**46**	46.0	0.0	**100**	0.04
p_hat1000-3	1000	68	**68**	68.0	0.0	**100**	0.17
p_hat1500-1	1500	12	**12**	11.8	0.4	78	24.24
p_hat1500-2	1500	65	**65**	65.0	0.0	**100**	0.11
p_hat1500-3	1500	94	**94**	94.0	0.0	**100**	0.27
san200_0.7_1	200	30	**30**	30.0	0.0	**100**	0.29
san200_0.7_2	200	18	**18**	18.0	0.0	**100**	0.75
san200_0.9_1	200	70	**70**	70.0	0.0	**100**	$< \epsilon$
san200_0.9_2	200	60	**60**	60.0	0.0	**100**	0.03
san200_0.9_3	200	44	**44**	44.0	0.0	**100**	0.01
san400_0.5_1	400	13	**13**	13.0	0.0	**100**	45.39
san400_0.7_1	400	40	**40**	40.0	0.0	**100**	14.26
san400_0.7_2	400	30	**30**	30.0	0.0	**100**	5.66
san400_0.7_3	400	22	**22**	22.0	0.0	**100**	3.95
san400_0.9_1	400	100	**100**	100.0	0.0	**100**	0.96
san1000	1000	15	**15**	15.0	0.4	99	8882.43
sanr200-0.7	200	18	**18**	18.0	0.0	**100**	0.01
sanr200-0.9	200	42	**42**	42.0	0.0	**100**	0.01
sanr400-0.5	400	13	**13**	13.0	0.0	**100**	0.40
sanr400-0.7	400	21	**21**	21.0	0.0	**100**	0.11

In order to further evaluate our BQO approach for the MCP, we show a comparison of GRASP-TS/MCPs with the very recent GLS-H2 algorithm [Pajouh *et al.*, 2011]. To the best of our knowledge, only GLS-H1 and GLS-H2 use a BQO formulation to solve the equivalent maximum stable problem. Given that GLS-H2 generally performs much better than GLS-H1, we compare our algorithm with GLS-H2. The GLS-H2 algorithm is a local search based approach utilizing the local maxima properties of a box-constrained quadratic optimization formulation of the equivalent maximum independent set problem. Notice that experiments of GLS-H2 are conducted on a HP workstation with Intel 2.67

Table 4.6: Comparison between GRASP-TS/MCPs and GLS-H2 on 32 MCP instances

Instance	BKR	GRASP-TS/MCPs			GLS-H2		
		f_{best}	f_{avg}	t_{b_avg}	f_{best}	f_{avg}	t_{b_avg}
brock200_1	21	**21**	**21.0**	**0.11**	19	16.07	45.61
brock200_2	12	**12**	**12.0**	**0.33**	11	7.93	21.95
brock200_4	17	**17**	**17.0**	**0.73**	15	11.87	35.71
brock400_2	29	**29**	**29.0**	**139.10**	23	18.79	482.74
brock400_4	33	**33**	**33.0**	**11.92**	22	19.07	479.59
brock800_2	24	**24**	**21.2**	**2165.91**	18	15.05	5291.88
brock800_4	26	**26**	**23.0**	**1719.04**	18	14.91	5343.73
keller4	11	**11**	**11.0**	$< \epsilon$	9	7.51	22.31
keller5	27	**27**	**27.0**	**0.12**	18	15.75	6156.88
MANN_a9	16	**16**	**16.0**	$< \epsilon$	14	13	1.74
MANN_a27	126	**126**	**126.0**	**17.30**	118	117.97	278.57
MANN_a45	345	**344**	**343.1**	**2073.98**	331	330.98	15830.69
hamming6-2	32	32	**32.0**	$< \epsilon$	32	31.18	2.16
hamming6-4	4	4	**4.0**	$< \epsilon$	4	3.38	1.43
hamming8-2	128	128	**128.0**	$< \epsilon$	128	127.64	103.51
hamming8-4	16	16	**16.0**	$< \epsilon$	16	14.84	78.27
c-fat200-1	12	12	**12.0**	**0.01**	12	11.06	2.01
c-fat200-2	24	24	**24.0**	**0.02**	24	22.31	4.15
c-fat200-5	58	58	**58.0**	**0.01**	58	57.14	17.23
johnson8-2-4	4	4	**4.0**	$< \epsilon$	4	3.42	0.53
johnson8-4-4	14	14	**14.0**	$< \epsilon$	14	13.62	2.62
johnson16-2-4	8	8	**8.0**	$< \epsilon$	8	6.87	10.02
p_hat300-1	8	8	**8.0**	$< \epsilon$	8	5.45	22.83
p_hat300-2	25	**25**	**25.**0	$< \epsilon$	23	19.22	77.42
p_hat300-3	36	**36**	**36.0**	**0.01**	33	30.02	164.96
p_hat700-1	11	**11**	**11.0**	**0.21**	8	6.49	527.55
p_hat700-2	44	**44**	**44.0**	**0.01**	41	34.99	1850.03
p_hat700-3	62	**62**	**62.0**	**0.02**	59	53.94	3962.09
san200_0.7_2	18	**18**	**18.0**	**0.75**	12	12	39.75
san200_0.9_1	70	**70**	**70.0**	$< \epsilon$	45	45	43.53
san200_0.9_2	60	**60**	**60.0**	**0.03**	35	34.9	50.89
san200_0.9_3	44	**44**	**44.0**	**0.01**	32	24.21	58.75

GHz processor and 3GB RAM while our experiments are conducted on a PC with 2.83GHz processor and 8GB memory. Table 4.6 shows the results of GRASP-TS/MCPs compared with GLS-H2, where results marked in bold indicate a better performance with respect to the corresponding statistical criteria. Given that GLS-H2 only reported the results on a selected subset of the DIMACS benchmarks, the comparison between GRASP-TS/MCPs and GLS-H2 is based on these instances.

From Table 4.6, we find that our GRASP-TS/MCPs algorithm dominates GLS-H2 on all three measures of best solution values, average solution values and computing time. Specifically, GRASP-TS/MCPs obtains better solution values than GLS-H2 for 21 instances and equal solution values for other 11 instances. The table shows not only that GRASP-TS/MCPs is superior, but is much faster to reach the best solution values than GLS-H2.

Furthermore, we compare our GRASP-TS/MCPs algorithm with several recent algorithms that are specially designed for the max clique problem. These algorithms include an adaptive tabu search algorithm AMTS [Wu and Hao, 2011], a simulated annealing algorithm SAA [Geng *et al.*, 2007] and a phased local search that interleaves sub-algorithms and a plateau search PLS [Pullan, 2006]. Table 4.7 presents the best solution values f_{best}

and the average time t_{b_avg} to reach f_{best} for each reference algorithm (GRASP-TS/MCPs, AMTS, SAA and PLS, respectively) where the best solution values among them are marked in bold.

As can be seen from Table 4.7, GRASP-TS/MCPs, AMTS and PLS outperform SAA in terms of the best solution values. Specifically, the adaptive tabu search method AMTS performs the best since it attains the largest clique for all the graphs except $MANN\_a81$. Both GRASP-TS/MCPs and PLS attain the best known results for 77 instances but GRASP-TS/MCPs performs slightly better than PLS on C2000.9, with a value 79 versus 78. SAA only finds the best known results for 56 instances. When referring to the computing time, it is difficult to make a fair comparison since SAA, PLS and GRASP-TS/MCPs are run on different computers (Pentium IV 1.4 GHz CPU for SAA, Pentium IV 2.4GHz CPU for PLS and Pentium IV 2.83GHz CPU for GRASP-TS/MCPs and AMTS). Nevertheless, to our best estimation it appears that GRASP-TS/MCPs is much faster to reach the best solution values than SAA, but is slower than AMTS and PLS.

Table 4.7: Comparison among GRASP-TS/MCPs, AMTS, SAA and PLS on 80 MVWCP instances

Instance	BKR	GRASP-TS/MCPs		AMTS		SAA		PLS	
		f_{best}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}
brock200_1	21	**21**	0.11	**21**	0.01	**21**	5	**21**	$< \epsilon$
brock200_2	12	**12**	0.33	**12**	0.36	**12**	8	**12**	0.03
brock200_3	15	**15**	2.20	**15**	0.01	14	2	**15**	0.03
brock200_4	17	**17**	0.73	**17**	1.76	16	< 1	**17**	0.08
brock400_1	27	**27**	265.81	**27**	37.77	25	22	**27**	1.08
brock400_2	29	**29**	139.10	**29**	1.18	25	29	**29**	0.38
brock400_3	31	**31**	20.87	**31**	1.79	25	26	**31**	0.18
brock400_4	33	**33**	11.92	**33**	0.60	25	26	**33**	0.10
brock800_1	23	**23**	1600.37	**23**	234.63	21	131	**23**	30.09
brock800_2	24	**24**	2165.91	**24**	33.14	21	124	**24**	24.41
brock800_3	25	**25**	2067.96	**25**	52.40	21	122	**25**	15.08
brock800_4	26	**26**	1719.04	**26**	15.23	21	125	**26**	6.54
C125.9	34	**34**	$< \epsilon$	**34**	$< \epsilon$	**34**	< 1	**34**	$< \epsilon$
C250.9	44	**44**	0.01	**44**	$< \epsilon$	**44**	4	**44**	$< \epsilon$
C500.9	57	**57**	0.95	**57**	0.13	**57**	59	**57**	0.19
C1000.9	68	**68**	12.01	**68**	1.15	**68**	222	**68**	1.88
C2000.5	16	**16**	59.43	**16**	0.66	**16**	877	**16**	0.73
C2000.9	80	79	17591.58	**80**	450.10	74	776	78	112.82
C4000.5	18	**18**	8953.89	**18**	126.63	17	903	**18**	149.65
DSJC500.5	13	**13**	0.16	**13**	$< \epsilon$	**13**	17	**13**	0.01
DSJC1000.5	15	**15**	13.87	**15**	0.31	**15**	363	**15**	0.47
keller4	11	**11**	$< \epsilon$	**11**	$< \epsilon$	**11**	< 1	**11**	$< \epsilon$
keller5	27	**27**	0.12	**27**	0.06	**27**	143	**27**	0.05
keller6	59	**59**	238.55	**59**	10.81	51	644	**59**	550.95
MANN_a9	16	**16**	$< \epsilon$	**16**	0.02	**16**	< 1	**16**	$< \epsilon$
MANN_a27	126	**126**	17.30	**126**	0.07	**126**	49	**126**	0.03
MANN_a45	345	344	2073.98	**345**	112.85	334	393	344	28.76
MANN_a81	1100	**1098**	18847.50	**1098**	27.55	1080	1879	**1098**	269.66
hamming6-2	32	**32**	$< \epsilon$	**32**	$< \epsilon$	**32**	< 1	**32**	$< \epsilon$
hamming6-4	4	**4**	$< \epsilon$	**4**	$< \epsilon$	**4**	< 1	**4**	$< \epsilon$
hamming8-2	128	**128**	$< \epsilon$	**128**	$< \epsilon$	**128**	3	**128**	$< \epsilon$
hamming8-4	16	**16**	$< \epsilon$	**16**	$< \epsilon$	**16**	< 1	**16**	$< \epsilon$
hamming10-2	512	**512**	0.23	**512**	0.31	**512**	427	**512**	$< \epsilon$
hamming10-4	40	**40**	0.06	**40**	0.92	**40**	144	**40**	$< \epsilon$
gen200_p0.9_44	44	**44**	0.01	**44**	$< \epsilon$	**44**	21	**44**	$< \epsilon$
gen200_p0.9_55	55	**55**	$< \epsilon$	**55**	$< \epsilon$	**55**	1	**55**	$< \epsilon$
gen400_p0.9_55	55	**55**	0.21	**55**	0.55	**55**	31	**55**	0.25

(Continued...)

Instance	BKR	GRASP-TS/MCPs		AMTS		SAA		PLS	
		f_{best}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}
gen400_p0.9_65	65	**65**	0.01	**65**	0.01	**65**	28	**65**	$< \epsilon$
gen400_p0.9_75	75	**75**	0.01	**75**	0.04	**75**	75	**75**	$< \epsilon$
c-fat200-1	12	**12**	0.01	**12**	$< \epsilon$	**12**	< 1	**12**	$< \epsilon$
c-fat200-2	24	**24**	0.02	**24**	0.17	**24**	< 1	**24**	$< \epsilon$
c-fat200-5	58	**58**	0.01	**58**	0.11	**58**	< 1	**58**	$< \epsilon$
c-fat500-1	14	**14**	0.05	**14**	0.14	**14**	4	**14**	$< \epsilon$
c-fat500-2	26	**26**	0.02	**26**	0.23	**26**	< 1	**26**	$< \epsilon$
c-fat500-5	64	**64**	0.13	**64**	0.10	**64**	< 1	**64**	$< \epsilon$
c-fat500-10	126	**126**	0.05	**126**	2.66	**126**	< 1	**126**	$< \epsilon$
johnson8-2-4	4	**4**	$< \epsilon$	**4**	$< \epsilon$	**4**	< 1	**4**	$< \epsilon$
johnson8-4-4	14	**14**	$< \epsilon$	**14**	$< \epsilon$	**14**	< 1	**14**	$< \epsilon$
johnson16-2-4	8	**8**	$< \epsilon$	**8**	$< \epsilon$	**8**	< 1	**8**	$< \epsilon$
johnson32-2-4	16	**16**	0.01	**16**	$< \epsilon$	**16**	< 1	**16**	$< \epsilon$
p_hat300-1	8	**8**	$< \epsilon$	**8**	$< \epsilon$	**8**	< 1	**8**	$< \epsilon$
p_hat300-2	25	**25**	$< \epsilon$	**25**	$< \epsilon$	**25**	< 1	**25**	$< \epsilon$
p_hat300-3	36	**36**	0.01	**36**	$< \epsilon$	**36**	2	**36**	$< \epsilon$
p_hat500-1	9	**9**	0.01	**9**	$< \epsilon$	**9**	< 1	**9**	$< \epsilon$
p_hat500-2	36	**36**	0.01	**36**	$< \epsilon$	**36**	1	**36**	$< \epsilon$
p_hat500-3	50	**50**	0.02	**50**	$< \epsilon$	**50**	32	**50**	$< \epsilon$
p_hat700-1	11	**11**	0.21	**11**	$< \epsilon$	**11**	18	**11**	0.01
p_hat700-2	44	**44**	0.01	**44**	$< \epsilon$	**44**	3	**44**	$< \epsilon$
p_hat700-3	62	**62**	0.02	**62**	$< \epsilon$	**62**	12	**62**	$< \epsilon$
p_hat1000-1	10	**10**	0.05	**10**	$< \epsilon$	**10**	6	**10**	$< \epsilon$
p_hat1000-2	46	**46**	0.04	**46**	$< \epsilon$	**46**	16	**46**	$< \epsilon$
p_hat1000-3	68	**68**	0.17	**68**	0.08	**68**	100	**68**	0.02
p_hat1500-1	12	**12**	24.24	**12**	2.18	**12**	490	**12**	3.28
p_hat1500-2	65	**65**	0.11	**65**	0.33	**65**	40	**65**	0.01
p_hat1500-3	94	**94**	0.27	**94**	0.32	**94**	215	**94**	0.03
san200_0.7_1	30	**30**	0.29	**30**	0.21	17	9	**30**	$< \epsilon$
san200_0.7_2	18	**18**	0.75	**18**	0.24	15	9	**18**	0.02
san200_0.9_1	70	**70**	$< \epsilon$	**70**	0.17	61	12	**70**	$< \epsilon$
san200_0.9_2	60	**60**	0.03	**60**	0.13	**60**	12	**60**	$< \epsilon$
san200_0.9_3	44	**44**	0.01	**44**	0.08	**44**	6	**44**	$< \epsilon$
san400_0.5_1	13	**13**	45.39	**13**	11.46	7	< 1	**13**	0.06
san400_0.7_1	40	**40**	14.26	**40**	8.76	21	36	**40**	0.06
san400_0.7_2	30	**30**	5.66	**30**	29.98	16	25	**30**	0.10
san400_0.7_3	22	**22**	3.95	**22**	56.29	17	30	**22**	0.19
san400_0.9_1	100	**100**	0.96	**100**	1.87	57	38	**100**	$< \epsilon$
san1000	15	**15**	8882.43	**15**	315.17	8	< 1	**15**	4.72
sanr200-0.7	18	**18**	0.01	**18**	$< \epsilon$	**18**	< 1	**18**	$< \epsilon$
sanr200-0.9	42	**42**	0.01	**42**	$< \epsilon$	**42**	5	**42**	0.01
sanr400-0.5	13	**13**	0.40	**13**	0.01	**13**	17	**13**	0.01
sanr400-0.7	21	**21**	0.11	**21**	$< \epsilon$	**21**	13	**21**	0.01

4.5.4 Results on MVWCP benchmark

In this section, we verify the effectiveness of the GRASP-TS/MCPs algorithm on the set of 80 DIMACS-VW benchmark instances of the more complex MVWCP problem. Our parameters have the following settings: (1) time limit: 1 minute for instances of keller except keller6, hamming, gen, c-fat, johnson, p_hat, sanr and mann_a9; 5 minutes for instances of brock, C except C2000.5, C2000.9, C4000.5, dsjc and san; 60 minutes for C2000.5, C2000.9 and keller6; 600 minutes for C4000.5, mann_a27, mann_a45, mann_a81 (2) $tt = [5, 12]$ (3) $\mu = 5000$ for all the instances except instances of san with $\mu = 10$. Table 4.8 presents the results of this experiment in which columns 1 to 8 give the same statistical

characteristics as in Table 4.5. The best known results BKR (column 3) are taken from [Pullan, 2008]. The last column newly added gives the clique cardinality achieved by our approach for information.

From Table 4.8, we observe that our GRASP-TS/MCPs algorithm is able to find new best solution values for 13 instances (marked in bold) and match the previous best known results for 66 instances. Only for one instance p_hat500-2, GRASP-TS/MCPs is slightly worse with a value of 3920 vs. 3925. In addition, GRASP-TS/MCPs consumes a very short time to reach these values for most instances, indicating its effectiveness for solving the MVWCP in terms of both solution quality and computing time.

Table 4.8: Computational results of GRASP-TS/MCPs on 80 MVWCP instances

Instance	Order	BKR	GRASP-TS/MCPs					
			f_{best}	f_{avg}	σ	Succ.	t_{b_avg}	Clique
brock200_1	200	2821	2821	2821.0	0.0	100	0.02	19
brock200_2	200	1428	1428	1428.0	0.0	100	0.08	9
brock200_3	200	2062	2062	2062.0	0.0	100	0.09	13
brock200_4	200	2107	2107	2107.0	0.0	100	0.22	13
brock400_1	400	3422	3422	3422.0	0.0	100	0.72	21
brock400_2	400	3350	3350	3350.0	0.0	100	1.00	21
brock400_3	400	3471	3471	3471.0	0.0	100	0.57	23
brock400_4	400	3626	3626	3626.0	0.0	100	4.01	33
brock800_1	800	3121	3121	3121.0	0.0	100	3.95	20
brock800_2	800	3043	3043	3043.0	0.0	100	42.29	18
brock800_3	800	3076	3076	3076.0	0.0	100	8.22	20
brock800_4	800	2971	2971	2970.1	0.3	8	105.53	26
C125.9	125	2529	2529	2529.0	0.0	100	0.02	30
C250.9	250	5092	5092	5092.0	0.0	100	0.05	40
C500.9	500	6822	**6955**	6955.0	0.0	100	0.21	48
C1000.9	1000	8965	**9254**	9254.0	0.0	100	37.50	61
C2000.5	2000	2466	2466	2460.7	11.1	71	1366.51	14
C2000.9	2000	10028	**10999**	10987.5	18.7	72	2711.97	72
C4000.5	4000	2792	2792	2753.2	34.2	19	19902.77	16
DSJC500.5	500	1725	1725	1725.0	0.0	100	3.82	12
DSJC1000.5	1000	2186	2186	2180.9	10.6	81	115.42	13
keller4	171	1153	1153	1153.0	0.0	100	0.05	11
keller5	776	3317	3317	3317.0	0.0	100	5.34	27
keller6	3361	7382	**8062**	7741.3	104.3	2	3418.36	56
MANN_a9	45	372	372	372.0	0.0	100	0.01	16
MANN_a27	378	12264	**12277**	12271.5	1.9	4	22864.81	126
MANN_a45	1035	34129	**34194**	34183.4	4.9	2	17524.05	343
MANN_a81	3321	110564	**111137**	111117.4	6.8	1	6167.28	1096
hamming6-2	64	1072	1072	1072.0	0.0	100	$< \epsilon$	32
hamming6-4	64	134	134	134.0	0.0	100	$< \epsilon$	4
hamming8-2	256	10976	10976	10976.0	0.0	100	0.80	128
hamming8-4	256	1472	1472	1472.0	0.0	100	$< \epsilon$	16
hamming10-2	1024	50512	50512	50193.2	770.3	67	24.47	512
hamming10-4	1024	5086	**5129**	5125.2	2.3	8	32.49	34
gen200_p0.9_44	200	5043	5043	5043.0	0.0	100	0.02	37
gen200_p0.9_55	200	5416	5416	5416.0	0.0	100	0.43	52
gen400_p0.9_55	400	6718	6718	6718.0	0.0	100	0.28	47
gen400_p0.9_65	400	6935	**6940**	6940.0	0.0	100	0.11	48
gen400_p0.9_75	400	8006	8006	8006.0	0.0	100	0.67	75
c-fat200-1	200	1284	1284	1284.0	0.0	100	0.01	12
c-fat200-2	200	2411	2411	2411.0	0.0	100	0.34	23
c-fat200-5	200	5887	5887	5887.0	0.0	100	0.20	58
c-fat500-1	500	1354	1354	1354.0	0.0	100	0.20	12
c-fat500-2	500	2628	2628	2628.0	0.0	100	3.10	24

(Continued...)

Instance	Order	BKR	GRASP-TS/MCPs					
			f_{best}	f_{avg}	σ	Succ.	t_{b_avg}	Clique
c-fat500-5	500	5841	5841	5841.0	0.0	100	1.15	62
c-fat500-10	500	11586	11586	11586.0	0.0	100	1.29	124
johnson8-2-4	28	66	66	66.0	0.0	100	$< \epsilon$	4
johnson8-4-4	70	511	511	511.0	0.0	100	$< \epsilon$	14
johnson16-2-4	120	548	548	548.0	0.0	100	$< \epsilon$	8
johnson32-2-4	496	2033	2033	2022.4	10.1	40	26.71	16
p_hat300-1	300	1057	1057	1057.0	0.0	100	0.03	7
p_hat300-2	300	2487	2487	2487.0	0.0	100	0.02	20
p_hat300-3	300	3774	3774	3774.0	0.0	100	0.04	29
p_hat500-1	500	1231	1231	1231.0	0.0	100	0.17	8
p_hat500-2	500	3925	3920	3920.0	0.0	100	$< \epsilon$	31
p_hat500-3	500	5361	**5375**	5375.0	0.0	100	0.36	42
p_hat700-1	700	1441	1441	1441.0	0.0	100	0.30	9
p_hat700-2	700	5290	5290	5290.0	0.0	100	0.03	40
p_hat700-3	700	7565	7565	7565.0	0.0	100	2.07	58
p_hat1000-1	1000	1514	1514	1514.0	0.0	100	3.78	9
p_hat1000-2	1000	5777	5777	5777.0	0.0	100	0.09	42
p_hat1000-3	1000	7986	**8111**	8111.0	0.0	100	0.65	58
p_hat1500-1	1500	1619	1619	1618.8	0.9	95	17.25	10
p_hat1500-2	1500	7328	**7360**	7360.0	0.0	100	3.61	58
p_hat1500-3	1500	10014	**10321**	10267.4	35.8	9	34.14	84
san200_0.7_1	200	3370	3370	3370.0	0.0	100	0.06	30
san200_0.7_2	200	2422	2422	2422.0	0.0	100	0.41	14
san200_0.9_1	200	6825	6825	6825.0	0.0	100	0.02	70
san200_0.9_2	200	6082	6082	6082.0	0.0	100	0.02	60
san200_0.9_3	200	4748	4748	4748.0	0.0	100	0.64	34
san400_0.5_1	400	1455	1455	1455.0	0.0	100	5.74	8
san400_0.7_1	400	3941	3941	3941.0	0.0	100	2.64	40
san400_0.7_2	400	3110	3110	3110.0	0.0	100	6.81	30
san400_0.7_3	400	2771	2771	2770.8	2.2	99	42.54	18
san400_0.9_1	400	9776	9776	9776.0	0.0	100	0.31	100
san1000	1000	1716	1716	1716.0	0.0	100	40.93	9
sanr200-0.7	200	2325	2325	2325.0	0.0	100	0.08	15
sanr200-0.9	400	5126	5126	5126.0	0.0	100	$< \epsilon$	36
sanr400-0.5	400	1835	1835	1835.0	0.0	100	1.41	11
sanr400-0.7	400	2992	2992	2992.0	0.0	100	0.47	18

Table 4.9 compares the results obtained by our GRASP-TS/MCPs algorithm and the well-known PLS$_W$ algorithm [Pullan, 2008]. The columns under headings GRASP-TS/MCPs and PLS$_W$ list the best solution values f_{best} obtained by each algorithm, number of times to reach f_{best} over 100 runs *Succ.*, and the average CPU time t_{b_avg} (in seconds) to reach f_{best}. From Table 4.9, we observe that GRASP-TS/MCPs obtains solutions of the same or better quality for 79 out of 80 instances in comparison with PLS$_W$. In addition, GRASP-TS/MCPs has a success rate of 100% for 64 instances while PLS$_W$ has a 100% success rate for only 52 instances. Finally, the computing time of GRASP-TS/MCPs is globally competitive to that of PLS$_W$.

In sum, the BQO model with GRASP-TS/MCPs proves to be an effective approach to provide competitive results not only for the maximum clique problem but also for the more complex maximum vertex weight clique problem.

Table 4.9: Comparison between GRASP-TS/MCPs and PLS on 80 MVWCP instances

Instance	GRASP-TS/MCPs			PLS_W [Pullan, 2008]		
	f_{best}	Succ.	t_{b_avg}	f_{best}	Succ.	t_{b_avg}
brock200_1	**2821**	**100**	0.02	**2821**	**100**	0.19
brock200_2	**1428**	**100**	0.08	**1428**	**100**	0.02
brock200_3	**2062**	**100**	0.09	**2062**	**100**	0.01
brock200_4	**2107**	**100**	0.22	**2107**	**100**	0.70
brock400_1	**3422**	**100**	0.72	**3422**	32	437.19
brock400_2	**3350**	**100**	1.00	**3350**	61	415.95
brock400_3	**3471**	**100**	0.57	**3471**	**100**	12.04
brock400_4	**3626**	**100**	4.01	**3626**	**100**	0.05
brock800_1	**3121**	**100**	3.95	**3121**	**100**	31.46
brock800_2	**3043**	**100**	42.29	**3043**	69	893.42
brock800_3	**3076**	**100**	8.22	**3076**	**100**	3.35
brock800_4	**2971**	8	105.53	**2971**	**100**	3.77
C125.9	**2529**	**100**	0.02	**2529**	**100**	8.08
C250.9	**5092**	**100**	0.05	**5092**	17	247.69
C500.9	**6955**	**100**	0.21	6822	–	–
C1000.9	**9254**	**100**	37.50	8965	5	344.74
C2000.5	**2466**	71	1366.51	**2466**	18	711.27
C2000.9	**10999**	72	2711.97	10028	–	–
C4000.5	**2792**	19	19902.77	**2792**	–	–
DSJC500.5	**1725**	**100**	3.82	**1725**	**100**	0.95
DSJC1000.5	**2186**	81	115.42	**2186**	**100**	47.76
keller4	**1153**	**100**	0.05	**1153**	**100**	0.02
keller5	**3317**	**100**	5.34	**3317**	**100**	119.24
keller6	**8062**	2	3418.36	7382	–	–
MANN_a9	**372**	**100**	0.01	**372**	**100**	$< \epsilon$
MANN_a27	**12277**	4	22864.81	12264	–	–
MANN_a45	**34194**	2	17524.05	34129	–	–
MANN_a81	**111137**	1	6167.28	110564	–	–
hamming6-2	**1072**	**100**	$< \epsilon$	**1072**	**100**	$< \epsilon$
hamming6-4	**134**	**100**	$< \epsilon$	**134**	**100**	$< \epsilon$
hamming8-2	**10976**	**100**	0.80	**10976**	**100**	$< \epsilon$
hamming8-4	**1472**	**100**	$< \epsilon$	**1472**	**100**	$< \epsilon$
hamming10-2	**50512**	67	24.47	**50512**	**100**	$< \epsilon$
hamming10-4	**5129**	8	32.49	5086	1	1433.07
gen200_p0.9_44	**5043**	**100**	0.02	**5043**	**100**	4.44
gen200_p0.9_55	**5416**	**100**	0.43	**5416**	**100**	0.05
gen400_p0.9_55	**6718**	**100**	0.28	**6718**	2	340.11
gen400_p0.9_65	**6940**	**100**	0.11	6935	4	200.79
gen400_p0.9_75	**8006**	**100**	0.67	**8006**	**100**	$< \epsilon$
c-fat200-1	**1284**	**100**	0.01	**1284**	**100**	$< \epsilon$
c-fat200-2	**2411**	**100**	0.34	**2411**	**100**	$< \epsilon$
c-fat200-5	**5887**	**100**	0.20	**5887**	**100**	$< \epsilon$
c-fat500-1	**1354**	**100**	0.20	**1354**	**100**	$< \epsilon$
c-fat500-2	**2628**	**100**	3.10	**2628**	**100**	0.01
c-fat500-5	**5841**	**100**	1.15	**5841**	**100**	$< \epsilon$
c-fat500-10	**11586**	**100**	1.29	**11586**	**100**	$< \epsilon$
johnson8-2-4	**66**	**100**	$< \epsilon$	**66**	**100**	$< \epsilon$
johnson8-4-4	**511**	**100**	$< \epsilon$	**511**	**100**	$< \epsilon$
johnson16-2-4	**548**	**100**	$< \epsilon$	**548**	**100**	$< \epsilon$
johnson32-2-4	**2033**	40	26.71	**2033**	**100**	44.68
p_hat300-1	**1057**	**100**	0.03	**1057**	**100**	0.01
p_hat300-2	**2487**	**100**	0.02	**2487**	**100**	19.36
p_hat300-3	**3774**	**100**	0.04	**3774**	47	418.11
p_hat500-1	**1231**	**100**	0.17	**1231**	**100**	0.42
p_hat500-2	3920	100	$< \epsilon$	**3925**	–	–
p_hat500-3	**5375**	**100**	0.36	5361	–	–
p_hat700-1	**1441**	**100**	0.30	**1441**	**100**	0.20

(Continued...)

Instance	GRASP-TS/MCPs			PLS_W		
	f_{best}	Succ.	t_{b_avg}	f_{best}	Succ.	t_{b_avg}
p_hat700-2	**5290**	**100**	0.03	**5290**	**100**	78.51
p_hat700-3	**7565**	**100**	2.07	**7565**	12	718.40
p_hat1000-1	**1514**	**100**	3.78	**1514**	**100**	7.61
p_hat1000-2	**5777**	**100**	0.09	**5777**	87	940.62
p_hat1000-3	**8111**	**100**	0.65	7986	–	–
p_hat1500-1	**1619**	95	17.25	**1619**	**100**	48.91
p_hat1500-2	**7360**	**100**	3.61	7328	4	1056.19
p_hat1500-3	**10321**	9	34.14	10014	–	–
san200_0.7_1	**3370**	**100**	0.06	**3370**	**100**	$< \epsilon$
san200_0.7_2	**2422**	**100**	0.41	**2422**	66	397.38
san200_0.9_1	**6825**	**100**	0.02	**6825**	**100**	$< \epsilon$
san200_0.9_2	**6082**	**100**	0.02	**6082**	**100**	$< \epsilon$
san200_0.9_3	**4748**	**100**	0.64	**4748**	72	219.68
san400_0.5_1	**1455**	**100**	5.74	**1455**	**100**	200.44
san400_0.7_1	**3941**	**100**	2.64	**3941**	**100**	0.03
san400_0.7_2	**3110**	**100**	6.81	**3110**	**100**	0.05
san400_0.7_3	**2771**	99	42.54	**2771**	**100**	4.41
san400_0.9_1	**9776**	**100**	0.31	**9776**	**100**	$< \epsilon$
san1000	**1716**	**100**	40.93	**1716**	–	–
sanr200-0.7	**2325**	**100**	0.08	**2325**	**100**	0.62
sanr200-0.9	**5126**	**100**	$< \epsilon$	**5126**	5	182.54
sanr400-0.5	**1835**	**100**	1.41	**1835**	**100**	0.67
sanr400-0.7	**2992**	**100**	0.47	**2992**	**100**	141.50

4.6 Conclusions

In this chapter, we studied a simple and a population-based GRASP-Tabu Search algorithms for solving the BQO problem. Both algorithms are based on a dedicated randomized greedy construction heuristic, enhanced by reference to the ideas of "strongly determined variables" and "elite solution recovery" of probabilistic Tabu Search, and using a tabu search local optimization procedure. Additionally, the algorithm with population management (GRASP-TS/PM) integrates a population management strategy for maintaining a pool of diversified elite solutions.

Furthermore, we have reformulated the maximum cut problem, maximum clique problem and maximum vertex weight clique problem into the BQO formulation and applied the general BQO approach rather than a tailored algorithm to tackle these problems. Specifically, we directly solved the maximum cut problem with the two proposed GRASP-Tabu Search algorithms and solved the maximum clique problems with the GRASP-TS/MCPs algorithm that adapts and extends the simple GRASP-Tabu Search algorithm.

Experiments conducted with both GRASP-TS and GRASP-TS/PM algorithms on 31 BQO instances and 54 MaxCut instances have demonstrated that both GRASP-Tabu Search algorithms obtain highly competitive results in comparison with the previous best known results from the literature. In particular, for the 54 MaxCut instances, GRASP-TS/PM can improve the best known results for 19 instances whose optimum solution values are still unknown. Experiments conducted with the GRASP-TS/MCPs algorithm on a total of 160 maximum clique and maximum vertex weight clique instances have shown that the proposed GRASP-TS/MCPs algorithm not only proves competitive with the leading methods that are specifically tailored for the MCP problem, but outperforms the leading

methods for the more complex MVWCP problem. Out of 80 benchmark instances of MVWCP our method matches the best known results on 66 instances, and finds new best known results on 13 instances, while accomplishing this in a very short time span.

In the next chapter, we will resort to a highly effective path relinking metaheuristic approach to produce initial solutions, whose idea is to build a path (a sequence of solutions) connecting two elite solutions and select out based on certain measures one or several solutions from the constructed path. In addition to the solving of BQO and MaxCut problems, we will also investigate the application of BQO on the minimum sum coloring problem with the proposed path relinking algorithms.

Chapter 5

Path Relinking

This chapter presents two path relinking (PR) algorithms for BQO; one is based on a greedy strategy (PR1) to generate the relinking path from the initial solution to the guiding solution and the other operates in a random way (PR2). In addition, we directly apply the proposed Path Relinking algorithms to solve the minimum cut problem (MaxCut) and the minimum sum coloring problem (MSCP) after transforming them into the formulation of BQO. Extensive computational results with both PR algorithms on 31 BQO problem instances and 103 MaxCut problem instances indicate that both PR1 and PR2 are very competitive with several state-of-the-art algorithms. Moreover, experiments on 23 most often used MSCP instances indicate that our PR2 algorithm is able to reach competitive results compared with several special purpose MSCP algorithms but requires considerable computing time to find solutions of good quality for large instances. The content of this chapter is based on the paper [Wang *et al.*, 2012f] accepted to European Journal of Operational Research and the paper [Wang *et al.*, 2012c] submitted to Optimization.

Contents

5.1 Introduction

Path relinking is a general search strategy closely associated with tabu search and its underlying ideas share a significant intersection with the tabu search perspective [Glover *et al.*, 2000; Glover *et al.*, 2003; Glover *et al.*, 2004], with applications in a variety of contexts where it has proved to be very effective in solving difficult problems [Glover *et al.*, 2003]. The path relinking approach is mainly composed of a diversification generation method, a path generation method, a solution selection method and an improvement method, where the common purpose of the path generation method and solution selection method lies in the generation of potential solutions with good quality and diversity.

Consider that no study has been reported on applying path relinking to BQO, this chapter proposes two path relinking algorithms for the BQO, by following the general scheme described in [Glover *et al.*, 2004]. These two algorithms differ from each other mainly on the way of generating the path, PR1 employing a greedy strategy and PR2 employing a random construction. In addition, given that BQO has emerged during the past decade as a unified model for a wide range of combinatorial optimization problems and the BQO approach has the advantage of directly applying an algorithm designed for BQO to solve other classes of problems rather than resorting to a specialized solution method, we investigate the performance of the proposed Path Relinking algorithms for the maximum cut problem. In particular, we investigate for the first time the application of this BQO approach to solve the minimum sum coloring problem.

Experiments with PR1 and PR2 on a total of 134 BQO and MaxCut test instances indicate that our Path Relinking algorithms yield highly competitive outcomes when comparing with other best performing algorithms. In addition, we carry out the experiment on 23 MSCP instances with PR2 and contrast results with those of several reference algorithms specifically dedicated to the MSCP and those of an IP model solved with the latest CPLEX version (CPLEX V12.2).

5.2 Path relinking algorithms

5.2.1 General framework

Algorithm 5.1 shows the path relinking procedure for BQO. It starts with the creation of an initial set of b elite solutions $RefSet$ (line 4, Section 5.2.2) and identifies the best and worst solutions in $RefSet$ in terms of the objective function value for the purpose of updating $RefSet$ (line 5). For each elite solution $x_i \in RefSet$, a binary value $Tag(i)$ indicates whether x_i can take part in a relinking process. Initially, assigning each solution in $RefSet$ a TRUE Tag which becomes FALSE when it is selected as the initiating solution or the guiding solution. The set $PairSet$ contains the index pairs (i, j) designating the initiating and guiding solution from $RefSet$ used for the relinking process. $PairSet$ is initially composed of all the index pairs (i, j) such that at least one corresponding Tag has the value TRUE (line 7). As soon as $PairSet$ is constructed, all the Tag are marked FALSE (line 8).

The inner while loop (lines 9-32) generates new solutions by building paths for each

Algorithm 5.1: Outline of the Path Relinking algorithms

1: **Input**: matrix Q
2: **Output**: the best binary n-vector x^* found so far and its objective value f^*
3: **repeat**
4: Initialize $RefSet = \{x^1, \ldots, x^b\}$
5: Identify the best solution x^* and the worst solution x^w in $RefSet$ and record the objective value f^* of solution x^*
6: $Tag(i) =$ TRUE, $(i = \{1, \ldots, b\})$
7: $PairSet \longleftarrow \{(i,j) : x^i, x^j \in RefSet, x^i \neq x^j, Tag(i) \cup Tag(j) =$ TRUE$\}$
8: $Tag(i) =$ FALSE, $(i = \{1, \ldots, b\})$
9: **while** $(PairSet \neq \emptyset)$ **do**
10: Pick solution pair $(x^i, x^j) \in RefSet$ with index pair (i,j) in $PairSet$
11: Apply the Relinking Method to produce the sequence $x^i = x(1), \ldots, x(r) = x^j$
12: Select $x(m)$ from the sequence and apply tabu search to improve $x(m)$
13: **if** $f(x(m)) > f^*$ **then**
14: $x^* = x(m), \ \ f^* = f(x(m))$
15: **end if**
16: **if** (Update_RefSet$(RefSet, x(m))$ **then**
17: $RefSet \leftarrow RefSet \cup \{x(m)\} \setminus \{x^w\}$
18: $Tag(w) =$ TRUE
19: Record the new worst solution x^w in $RefSet$
20: **end if**
21: Apply the Relinking Method to produce the sequence $x^j = y(1), \ldots, y(r) = x^i$
22: Select $y(n)$ from the sequence and apply tabu search to improve $y(n)$
23: **if** $(f(y(n)) > f^*)$ **then**
24: $x^* = y(n), \ \ f^* = f(y(n))$
25: **end if**
26: **if** (Update_RefSet$(RefSet, y(n))$ **then**
27: $RefSet \leftarrow RefSet \cup \{y(n)\} \setminus \{x^w\}$
28: $Tag(w) =$ TRUE
29: Record the new worst solution x^w in $RefSet$
30: **end if**
31: $PairSet \longleftarrow PairSet \setminus (i,j)$
32: **end while**
33: **until** the stopping criterion is satisfied

pair of solutions of $PairSet$ and updates $RefSet$ with specific new solutions. First, one index pair (i,j) is selected from $PairSet$ according to lexicographical order (line 10) to designate two solutions $x^i, x^j \in RefSet$. The Relinking Method is then applied to these two solutions to generate two paths connecting x^i and x^j (lines 11, 21, Section 5.2.3). Secondly, one solution $x(m)$ on each path is selected to be further improved by the tabu search algorithm (lines 12, 22, Section 2.2.2). The next step tests $Update\_RefSet$ to decide if the new improved solution is used to update $RefSet$ (lines 16, 26, Section 5.2.2). If the update is confirmed, the new solution is inserted in $RefSet$ to replace the worst solution x^w with its Tag set to be TRUE (lines 16-18, 26-28, Section 5.2.2). The current selected pair (i,j) is then deleted from the set $PairSet$ (line 31). This while-loop procedure continues until all the pairs in $PairSet$ are examined, i.e., $PairSet$ becomes empty.

Our path relinking algorithm has the following characteristics. First, considering the path generation procedure, each solution pair originating from $RefSet$ undergoes a relinking phase and two paths are considered for each pair (x^i, x^j): one from x^i to x^j and the

other from x^j to x^i. Secondly, each new high-quality solution derived by path relinking is a candidate to take part in a subsequent relinking process as an initiating or guiding solution, using a probabilistic selection process that assures the solution will eventually get selected. Thirdly, upon the completion of the path relinking phase that ultimately examines all pairs of solutions in $RefSet$, we rebuild $RefSet$ to restart the path-relinking procedure, and repeat this restarting process until the stopping criterion is satisfied.

5.2.2 RefSet initialization, rebuilding and updating

The initial RefSet contains b different locally optimal solutions and is constructed as follows. Starting from scratch, we randomly assign a value of 0 or 1 to each variable to produce an initial solution, and then subject this solution to tabu search to obtain a local optimum (see Sect. 2.2.2). The resulting improved solution is added to $RefSet$ if it does not duplicate any solution currently in $RefSet$. This procedure is repeated until the size of $RefSet$ reaches the cardinality b (b is a parameter and set as 10 in the experiment).

When $PairSet$ becomes empty, RefSet is recreated. The best solution x^* previously found becomes a member of the new $RefSet$ and the remaining solutions are generated in the same way as in constructing $RefSet$ in the first round.

The updating procedure of $RefSet$ is invoked each time a newly constructed solution is improved by tabu search. The improved solution is permitted to be added into $RefSet$ if it is distinct from any solution in $RefSet$ and better than the worst solution x^w in $RefSet$. Once this condition is satisfied, the worst solution x^w is replaced by the improved solution and the position w is indicated as referring to a new solution.

5.2.3 Path relinking

The relinking method is used to generate new solutions by exploring trajectories (strictly confined to the neighborhood space) that connect high-quality solutions. The solution that begins the path is called the initiating solution while the solution that the path leads to is called the guiding solution [Glover *et al.*, 2000; Glover *et al.*, 2003; Glover *et al.*, 2004]. We propose two ways to generate such a path: one is based on a dedicated greedy function (whose evaluations are given by the objective function of UBQO problem) while the other operates in a random manner. Algorithms 5.2 and 5.3 describe these two methods in details.

In order to describe our relinking procedure, we first give some primary definitions, denoting the initiating solution by x^i and the guiding solution by x^j:

- NC: the set of variable indices for which x^i and x^j have different values.
- Δ_t: a vector that stores the objective value deviation of the current solution from the resulting solution after flipping the tth variable.
- PV: the path vector that stores the selected flip variable at each step throughout the transiting from x^i to x^j (Consequently, by knowing either the initiating solution or the current terminal solution, each solution generated on the path can be recovered by referring to PV).

- FI: a vector that records the difference $f(x) - f(x^i)$ for each solution x generated when transiting from x^i to x^j.

Algorithm 5.2: Pseudo-code of Relinking Method 1

Input: A pair of solutions x^i and x^j
Output: Path solution $x(1), \ldots, x(r)$ from x^i to x^j
Identify the set NC between x^i and x^j
Initialize the Δ_t assignments for $t \in NC$
$PV = \emptyset$, $FI_0 = 0$, $r = |NC| - 1$
for $k = 1$ to r **do**
    Find a $t \in NC$ with the best Δ_t value
    $PV \leftarrow PV \cup \{t\}$
    $x(k) = \{x_u : x_u = x^j_u, u \in PV; x_u = x^i_u, u \in N \setminus PV\}$
    $FI_k = FI_{k-1} + \Delta_t$
    $f(x(k)) = f(x^i) + FI_k$
    Update all Δ_t values ($t \in NC$) affected by the move
    $NC \leftarrow NC \backslash \{t\}$
end for

Algorithm 5.2 shows the first relinking method. Initially, we identify the set NC of variables whose values differ between the initiating solution and the guiding solution. The Δ value of each element in NC is also precalculated. At each step toward the guiding solution, we select the variable with the best Δ value and then add it into the path vector PV. Moreover, we record the current increment FI value and the objective value $f(x)$ of the current generated solution x. Finally, the vector Δ is updated using the fast incremental evaluation technique of [Glover *et al.*, 2010]. Since two adjacent solutions on the path differ from each other in the assignment of only one variable, this relinking procedure accomplishes the path construction from the initiating solution to the guiding solution after exactly $|NC| - 1$ steps.

Algorithm 5.3: Pseudo-code of Relinking Method 2

Input: A pair of solutions x^i and x^j
Output: Path solution $x(1), \ldots, x(r)$ from x^i to x^j
Identify the set NC between x^i and x^j
Initialize the Δ_t assignments for $t \in NC$
$PV = \emptyset$, $FI_0 = 0$, $r = |NC| - 1$
for $k = 1$ to r **do**
    Select a $t \in NC$ at random
    $PV \leftarrow PV \cup \{t\}$
    $x(k) = \{x_u : x_u = x^j_u, u \in PV; x_u = x^i_u, u \in N \setminus PV\}$
    $FI_k = FI_{k-1} + \Delta_t$
    $f(x(k)) = f(x^i) + FI_k$
    Update all Δ_t values ($t \in NC$) affected by the move
    $NC \leftarrow NC \backslash \{t\}$
end for

The second relinking method, shown in Algorithm 5.3, is based on the rule of selecting an element in NC randomly at each step (line 7). The remained components of the method are the same as in Algorithm 5.2.

5.2.4 Path solution selection

Since two consecutive solutions on a relinking path differ only by flipping a single variable, it is not productive to apply an improvement method to each solution on the path since many of these solutions would lead to the same local optimum. In addition, the improvement method is a time-consuming process, so we restrict its use to being applied to only a single solution on the path, which we select by reference both to its solution quality and to the hamming distance of this solution to the initiating and guiding solutions. Specifically, we set up a candidate solution list (CSL), consisting of the path solutions having a distance of at least $\gamma \cdot |NC|$ from both the initiating and guiding solutions (where $\gamma \in (0, 1]$ is a parameter and set as 1/3 in the experiments). The solution with the highest quality in CSL is picked for further amelioration by the improvement method.

5.3 Experimental results

We carry out three sets of experiments to evaluate the proposed PR algorithms. The first two experiments apply both PR1 and PR2 on BQO and MaxCut benchmarks and the last experiment employs PR2 on MSCP benchmark. Our algorithms use CPU clock time to give the stopping condition subject to having completed at least one round of the PR procedure.

5.3.1 Experiments on BQO benchmark

Our first experiment undertakes to evaluate the PR algorithms on the 31 BQO instances with 2500 to 7000 variables. The results are summarized in Tables 5.1 and 5.2. The time limit for 10 ORLIB instances for a single run is set to be 1 minute and for the 21 larger random instances with 3000, 4000, 5000, 6000 and 7000 variables is set at 5, 10, 20, 30 and 50 minutes. For BQO instances, the tabu tenure (tt) and the improvement cutoff (μ) are set as: $tt = [n/100, n/100 + 10]$ and $\mu = 5n$.

Tables 5.1 and 5.2 respectively show the computational statistics of applying our PR1 and PR2 algorithms to the 10 ORLIB instances and the 21 large random instances. In both tables, columns 1 and 2 respectively give the instance names and the previous best objective values BKR. These best values were first reported in [Palubeckis, 2004b; Palubeckis, 2006] and recently improved in [Glover *et al.*, 2010]. The columns under headings PR1 and PR2 list: the best objective value f_{best}, the average objective gap to the previous best objective values g_{avg} (i.e., $BKR - f_{avg}$) (where f_{avg} represents the average objective value over 20 runs) and the average CPU time in seconds denoted by t_{b_avg} for reaching the best objective values f_{best} over 20 runs. Furthermore, the last row "Av." indicates the summary of our algorithm's average performance.

Table 5.1: Computational results of PR1 and PR2 on ORLIB instances

Instance	BKR	PR1			PR2		
		f_{best}	g_{avg}	t_{b_avg}	f_{best}	g_{avg}	t_{b_avg}
b2500.1	1515944	1515944	0.0	11	1515944	0.0	14
b2500.2	1471392	1471392	0.0	101	1471392	58.4	102
b2500.3	1414192	1414192	13.4	49	1414192	0.0	36
b2500.4	1507701	1507701	0.0	6	1507701	0.0	7
b2500.5	1491816	1491816	0.0	14	1491816	0.0	18
b2500.6	1469162	1469162	0.0	25	1469162	0.0	23
b2500.7	1479040	1479040	0.0	48	1479040	0.0	50
b2500.8	1484199	1484199	0.0	20	1484199	0.0	16
b2500.9	1482413	1482413	0.0	51	1482413	0.0	103
b2500.10	1483355	1483355	0.0	55	1483355	0.0	75
Av.			1.34	38		5.84	44.4

Table 5.2: Computational results of PR1 and PR2 on Palubeckis instances

Instance	BKR	PR1			PR2		
		f_{best}	g_{avg}	t_{b_avg}	f_{best}	g_{avg}	t_{b_avg}
p3000.1	3931583	3931583	0.0	85	3931583	80.4	81
p3000.2	5193073	5193073	0.0	68	5193073	0.0	64
p3000.3	5111533	5111533	35.8	115	5111533	71.7	155
p3000.4	5761822	5761822	0.0	56	5761822	0.0	97
p3000.5	5675625	5675625	90.2	162	5675625	278.5	226
p4000.1	6181830	6181830	0.0	125	6181830	0.0	159
p4000.2	7801355	7801355	71.2	456	7801355	313.5	302
p4000.3	7741685	7741685	0.0	295	7741685	63.9	436
p4000.4	8711822	8711822	0.0	277	8711822	0.0	392
p4000.5	8908979	8908979	490.8	272	8908979	385.1	327
p5000.1	8559680	8559680	611.8	623	8559680	918.0	387
p5000.2	10836019	10836019	620.3	821	10836019	498.7	609
p5000.3	10489137	10489137	995.4	1285	10489137	317.5	967
p5000.4	12252318	12252318	1257.7	760	12252318	1168.4	767
p5000.5	12731803	12731803	51.3	676	12731803	166.3	726
p6000.1	11384976	11384976	201.0	1820	11384976	822.4	1136
p6000.2	14333855	14333855	221.1	1391	14333855	576.8	1076
p6000.3	16132915	16132915	1743.5	1128	16132915	2017.3	1053
p7000.1	14478676	14478676	935.4	2275	14478676	1523.1	1917
p7000.2	18249948	18249948	1942.4	1793	18249948	2986.1	1591
p7000.3	20446407	20446407	331.9	1251	20446407	2310.5	1503
Av.			457.1	749.2		690.4	665.3

Table 5.3: Best results comparison among PR1, PR2 and other state-of-the-art algorithms on Palubeckis instances

Instance	BKR	best solution gap (i.e., $BKR - f_{best}$)						
		PR1	PR2	ITS	MST2	SA	D^2TS	HMA
p5000.1	8559680	0	0	700	325	1432	325	0
p5000.2	10836019	0	0	0	582	582	0	0
p5000.3	10489137	0	0	0	0	354	0	0
p5000.4	12252318	0	0	934	1643	444	0	0
p5000.5	12731803	0	0	0	0	1025	0	0
p6000.1	11384976	0	0	0	0	430	0	0
p6000.2	14333855	0	0	88	0	675	0	0
p6000.3	16132915	0	0	2729	0	0	0	0
p7000.1	14478676	0	0	340	1607	2579	0	0
p7000.2	18249948	0	0	1651	2330	5552	104	0
p7000.3	20446407	0	0	0	0	2264	0	0
Av.		0	0	585.6	589.7	1394.3	39	0

Table 5.1 discloses that both PR1 and PR2 can stably reach all the previous best objective values for the 10 largest Beasley instances. Moreover, PR1 performs slightly better than PR2 when it comes to the criteria of g_{avg} and t_{b_avg} to the previous best result BKR. Table 5.2 indicates that on the 21 large and difficult random instances, PR1 produced the same results as PR2 given that both can reach the previous best known objective values for all of the tested instances. However, PR1 is superior to PR2 in terms of the average gap (457.1 versus 690.4) although the CPU time to obtain the best solution is slightly longer (749.2 versus 665.3 seconds).

In order to further evaluate our PR1 and PR2 algorithms, we compare our results with those obtained from some of best performing algorithms in the literature. For this purpose, we restrict our attention to comparisons with 5 methods that have reported the best results for the most challenging problems. These methods are respectively named ITS [Palubeckis, 2006], MST2 [Palubeckis, 2004b], SA [Katayama and Narihisa, 2001], D^2TS [Glover *et al.*, 2010] and HMA [Lü *et al.*, 2010a]. The results for the first 3 of these reference algorithms are directly extracted from [Palubeckis, 2006] and those for D^2TS and HMA come from [Glover *et al.*, 2010; Lü *et al.*, 2010a].

Tables 5.3 and 5.4 show the best solution gap and average solution gap to the best known objective value of the 7 algorithms used for comparison, including PR1 and PR2. In these two tables, the last row presents the averaged results over the listed instances. Notice that the results of all these algorithms are obtained almost under the same time limit. Since best known results can be easily reached for the small size instances by all these state-of-the art algorithms, we only list the results comparison for larger instances where Table 5.3 contains 11 instances and Table 5.4 contains 21 instances.

Table 5.3 indicates that both PR1 and PR2 outperform ITS, MST2 and SA in terms of the best solution values. PR1 and PR2 achieve the best known results for the 11 most challenging instances while ITS, MST2, SA fail for 5, 5, 10 out of 11 instances. In addition, D^2TS performs slightly worse since it fails to reach the best known result for one instance p7000.2. However, it is difficult to conclude which algorithm among PR1, PR2 and HMA performs the best based on the evaluation criterion of the best solution found.

In order to further discriminate among the compared algorithms, Table 5.4 presents

Table 5.4: Average results comparison among PR1, PR2 and other state-of-the-art algorithms on Palubeckis instances

Instance	BKR	average solution gap (i.e., $BKR - f_{avg}$)						
		PR1	PR2	ITS	MST2	SA	D^2TS	HMA
p3000.1	3931583	0	80	0	0	0	0	0
p3000.2	5193073	0	0	97	97	97	0	0
p3000.3	5111533	36	72	344	287	535	0	33
p3000.4	5761822	0	0	154	77	308	0	0
p3000.5	5675625	90	279	501	382	459	0	145
p4000.1	6181830	0	0	0	0	734	0	0
p4000.2	7801355	71	314	1285	804	1887	0	142
p4000.3	7741685	0	64	471	1284	79	0	6
p4000.4	8711822	0	0	438	667	536	0	38
p4000.5	8908979	491	385	572	717	984	0	546
p5000.1	8559680	612	918	971	581	2455	656	507
p5000.2	10836019	620	499	1068	978	2101	12533	512
p5000.3	10489137	995	318	1266	1874	2451	12876	332
p5000.4	12252318	1258	1168	1952	2570	1134	1962	1228
p5000.5	12731803	51	166	835	1233	1172	239	284
p6000.1	11384976	201	822	57	34	2248	0	140
p6000.2	14333855	221	577	1709	1269	2067	1286	526
p6000.3	16132915	1744	2017	3064	2673	3845	787	2311
p7000.1	14478676	935	1523	1139	2515	5504	2138	819
p7000.2	18249948	1942	2986	4301	3814	7837	8712	1323
p7000.3	20446407	332	2311	3078	7868	8978	2551	1386
Av.		457.1	690.4	1109.6	1415.4	2162.4	2082.9	489.4

the average solution gap to the best known value of each algorithm. Firstly, we notice that over the first 10 instances with 3000 and 4000 variables, D^2TS outperforms all the other 6 compared algorithms with an average gap of 0 to the best known values, meaning that D^2TS is quite robust over 20 runs for these 10 instances. PR1 and PR fail to reach the gap of 0 for 4 and 6 instances, respectively. Secondly, considering the overall set of 21 instances, we find that PR1 performs the best with a gap of 457.1. HMA performs slightly worse than PR1 with a gap of 489.4. PR2 takes the third place with a gap of 690.4. In conclusion, this experiment demonstrates that both PR1 and PR2 also perform quite well with regard to the average solution quality.

5.3.2 Experiments on MaxCut benchmark

Our second experiment undertakes to test our PR algorithms without any alternation on 3 sets of MaxCut benchmarks with a total of 103 instances, which are named Set1, Set2 and Set3. The transformation method of MaxCut into BQO formulation can be found in Section 1.1.2.2. The benchmark Set1 includes 69 instances, named G1,...,G72, with variable sizes ranging from n=800 to 10000.[1]. The first 54 instances have been employed by numerous authors to test their algorithms [Burer *et al.*, 2001; Festa *et al.*, 2002; Marti *et al.*, 2009; Palubeckis, 2004a; Shylo and Shylo, 2010] and the results for the remaining 15 larger instances are reported in [Choi and Ye, 2000]. The benchmark Set2 contains 30 instances with size n=128 (named G54100,...,G541000), n=1000 (named

[1]http://www.stanford.edu/~yyye/yyye/Gset

G10100,...,G101000) and n=2744 (named G14100,...,G141000), respectively.[2] Computational results on these instances were reported in [Burer *et al.*, 2001; Festa *et al.*, 2002; Marti *et al.*, 2009; Palubeckis, 2004a; Shylo and Shylo, 2010]. The benchmark Set3 is composed of 4 DIMACS instances containing from 512 to 3375 vertices and 1536 to 10125 edges.[3]. For MaxCut instances, the tabu tenure (tt) and the improvement cutoff (μ) are set as: $tt = [n/10, n/10 + 10]$ and $\mu = 10000$.

In Tables 5.5-5.8, columns 1 and 2 respectively give the instance names and the best known results BKR from references [Burer *et al.*, 2001; Marti *et al.*, 2009; Palubeckis, 2004a; Shylo and Shylo, 2010] which are all tailored MaxCut algorithms unlike our algorithms aiming at the solving of BQO. The columns under the headings PR1 and PR2 list the best objective value f_{best}, the average objective value f_{avg} and the CPU time in seconds denoted by t_{b_avg} for reaching the best results f_{best}. The columns under the headings SS and CirCut report the best objective value f_{best} and the required CPU time to reach f_{best}. We focus on comparing our algorithms with the SS and CirCut algorithms, which yield best results in the literature on many test instances. The results of SS and CirCut algorithms are directly extracted from [Marti *et al.*, 2009]. The last three rows summarize the comparison between these algorithms and ours. The rows *better*, *equal* and *worse* respectively denote the number of instances for which each algorithm gets results that are better, equal and worse than the previous best known results. We mark in bold those results that are the updated best known values obtained by PR1 and PR2.

Table 5.5 reports the results on 54 instances from Set1 within a time limit of 30 minutes. From this table, we first notice that our algorithms are able to find better objective values than the best known values in the literature. Meanwhile, PR2 slightly outperforms PR1 in terms of the best objective values. Specifically, PR1 can improve the previous best known objective values for 24 instances and match the previous best for 22 instances, while PR2 can improve the previous best known objective values for 25 instances and match the previous best for 24 instances. Moreover, PR1 and PR2 fail to reach the best known results for 8 and 5 instances respectively, while SS and CirCut fail on 32 and 34 instances, respectively. Additionally, PR1 and PR2 reaches its best results in a shorter CPU time than the time taken by SS and CirCut to reach their best results. These outcomes provide evidence of the efficacy of our path relinking approach.

Table 5.6 reports the results of 15 largest instances in Set1 with variables ranging from 5000 to 10000. For instances with 5000, 7000, 8000, 9000 and 10000 variables, we report the results for a time limit of 1, 2, 4, 4 and 4 hours, respectively. The previous best objective values BKR are cited from [Choi and Ye, 2000], which is the only paper, to the best of our knowledge, that reports the results on these instances. As can be seen from Table 5.6, both PR1 and PR2 obtain new best known results on 13 out of these 15 large instances and obtains results inferior to the best known results only on 2 instances. Moreover, PR2 outperforms PR1 by obtaining better solutions for 14 instances.

The results of the 30 instances in Set2 are shown in Table 5.7. For the instances with variables numbering 128, 1000 and 2744, the results are reported with a time limit of 1

[2]http://www.optsicom.es/maxcut/#instances

[3]http://dimacs.rutgers.edu/Challenges/Seventh/Instances/

Table 5.5: Computational results of PR1 and PR2 and comparison with other state-of-the-art algorithms on small and medium MaxCut instances of Set1

Instance	BKR	PR1			PR2			SS		CirCut	
		f_{best}	f_{avg}	t_{b_avg}	f_{best}	f_{avg}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}
G1	11624	11624	11624.0	2	11624	11624.0	1	11624	139	11624	352
G2	11620	11620	11620.0	6	11620	11620.0	9	11620	167	11617	283
G3	11622	11620	11620.0	17	11620	11620.0	2	11622	180	11622	330
G4	11646	11646	11646.0	3	11646	11646.0	2	11646	194	11641	524
G5	11631	11631	11631.0	3	11631	11631.0	4	11631	205	11627	1128
G6	2178	2178	2178.0	9	2178	2178.0	6	2165	176	2178	947
G7	2003	**2006**	2006.0	2	**2006**	2006.0	7	1982	176	2003	867
G8	2003	**2005**	2005.0	8	**2005**	2005.0	6	1986	195	2003	931
G9	2048	**2054**	2054.0	16	**2054**	2054.0	10	2040	158	2048	943
G10	1994	**2000**	2000.0	22	**2000**	1999.8	29	1993	210	1994	881
G11	564	564	564.0	4	564	564.0	1	562	172	560	74
G12	556	556	556.0	17	556	556.0	15	552	242	552	58
G13	582	582	582.0	28	582	582.0	22	578	228	574	62
G14	3064	3063	3062.1	44	3064	3062.6	1188	3060	187	3058	128
G15	3050	3050	3049.3	49	3050	3049.3	51	3049	143	3049	155
G16	3052	3052	3051.3	27	3052	3051.4	47	3045	162	3045	142
G17	3043	**3047**	3045.5	235	**3047**	3046.4	110	3043	313	3037	366
G18	988	**992**	992.0	16	**992**	992.0	12	988	174	978	497
G19	903	**906**	906.0	11	**906**	906.0	14	903	128	888	507
G20	941	941	941.0	13	941	941.0	9	941	191	941	503
G21	931	931	931.0	11	931	931.0	19	930	233	931	524
G22	13359	13359	13353.5	1652	13359	13354.5	943	13346	1336	13346	493
G23	13342	13342	13333.0	517	13342	13331.6	879	13317	1022	13317	457
G24	13337	13337	13327.3	1257	13333	13325.3	1876	13303	1191	13314	521
G25	13326	13338	13328.0	957	**13339**	13328.2	1078	13320	1299	13326	1600
G26	13314	13324	13313.7	710	**13326**	13312.3	333	13294	1415	13314	1569
G27	3318	**3337**	3327.3	851	3336	3326.9	753	3318	1438	3306	1456
G28	3285	**3296**	3286.0	1723	**3296**	3288.9	1512	3285	1314	3260	1543
G29	3389	3404	3395.2	861	**3405**	3391.9	1618	3389	1266	3376	1512
G30	3403	**3412**	3404.6	1655	3411	3404.8	843	3403	1196	3385	1463
G31	3288	**3306**	3299.7	624	**3306**	3299.5	752	3288	1336	3285	1448
G32	1410	1408	1400.9	893	1410	1404.6	450	1398	901	1390	221
G33	1382	1382	1373.9	1019	1382	1376.1	986	1362	926	1360	198
G34	1384	1382	1375.4	1608	1384	1378.2	1747	1364	950	1368	237
G35	7684	7674	7663.3	1372	7679	7670.8	959	7668	1258	7670	440
G36	7677	7666	7653.1	316	7671	7658.7	1790	7660	1392	7660	400
G37	7689	7673	7663.3	1736	7682	7667.9	965	7664	1387	7666	382
G38	7681	7674	7663.4	614	**7682**	7670.4	1775	7681	1012	7646	1189
G39	2395	2402	2391.3	526	**2407**	2391.1	1588	2393	1311	2395	852
G40	2387	2394	2381.2	1748	**2399**	2383.3	879	2374	1166	2387	901
G41	2398	2402	2380.0	1181	**2404**	2388.9	529	2386	1017	2398	942
G42	2469	2475	2462.3	1177	**2478**	2466.2	1575	2457	1458	2469	875
G43	6660	6660	6660.0	22	6660	6659.9	19	6656	406	6656	213
G44	6650	6650	6649.9	18	6650	6649.9	32	6648	356	6643	192
G45	6654	6654	6653.9	43	6654	6653.9	50	6642	354	6652	210
G46	6645	**6649**	6648.2	18	**6649**	6648.8	36	6634	498	6645	639
G47	6656	**6657**	6656.6	99	**6657**	6656.8	20	6649	359	6656	633
G48	6000	6000	6000.0	3	6000	6000.0	3	6000	20	6000	119
G49	6000	6000	6000.0	3	6000	6000.0	2	6000	35	6000	134
G50	5880	5880	5880.0	2	5880	5880.0	2	5880	27	5880	231
G51	3846	**3848**	3844.6	312	**3848**	3846.4	158	3846	513	3837	497
G52	3849	**3851**	3847.6	610	**3851**	3848.4	373	3849	551	3833	507
G53	3846	3849	3846.9	151	**3850**	3847.7	88	3846	424	3842	503
G54	3846	**3852**	3848.6	522	3851	3847.8	318	3846	429	3842	524
Av.				469.3			490.6		621.0		616.7
Better		24			25			0		0	
Equal		22			24			22		20	
Worse		8			5			32		34	

Table 5.6: Computational results of PR1 and PR2 on large MaxCut instances of Set1

Instance	BKR	PR1			PR2		
		f_{best}	f_{avg}	t_{b_avg}	f_{best}	f_{avg}	t_{b_avg}
G55	9960	10253	10233.7	3996	**10265**	10234.0	3231
G56	3649	3975	3958.0	3991	**3981**	3959.2	3842
G57	3220	3448	3436.0	3656	**3472**	3462.0	4403
G58	—	19183	19159.3	3979	**19205**	19182.0	3715
G59	—	**6027**	5989.2	3876	**6027**	6006.2	5194
G60	13658	14109	14077.5	7738	**14112**	14091.8	6300
G61	5273	5716	5688.8	7782	**5730**	5695.7	5381
G62	4612	4804	4785.7	8110	**4836**	4830.2	6114
G63	8059	26876	26845.8	4826	**26916**	26879.3	5867
G64	7861	8623	8569.5	8790	**8641**	8594.1	6974
G65	13286	5482	5468.7	16248	5526	5515.9	15004
G66	—	6272	6257.8	16031	**6314**	6302.4	15191
G67	—	6856	6832.0	17213	**6902**	6884.6	12372
G70	9499	9405	9378.6	15202	9463	9434.0	14531
G72	6644	6892	6876.2	14422	**6946**	6933.8	15898
Better		13			13		
Equal		0			0		
Worse		2			2		

second, 10 minutes and 30 minutes. Table 5.7 shows that our PR1 and PR2 algorithms once again outperform the two reference algorithms. Both PR1 and PR2 can match the best known results on 21 and 20 out of 30 instances, respectively. By contrast, SS and CirCut can match the previous best results on 10 instances. PR1 and PR2 fail to match the best known results on 9 and 10 out of 30 instances, respectively. By contrast, both SS and CirCut fail to match the previous best results on 20 instances.

Comparing PR1 and PR2 to each other, the PR2 algorithm achieves better results for 4 instances (G14100, G14400, G14800 and G141000) while PR1 obtain better results for 2 instances (G14300 and G14500). In addition, PR2 obtains its best solutions faster than PR1, 377.5 vs 473.2 seconds on average. We note that CirCut consumes less CPU time than ours, though the quality of its solutions does not measure up.

The results of 4 instances from Set3 using a time limit of 30 minutes are shown in Table 5.8. For the instance pm3-15-50, both PR1 and PR2 are able to improve the previous best known result from a value of 3000 to the value of 3010 and 3014, respectively. For the instance pm3-8-50, PR1 and PR2 match the previously best known result but the other referred algorithms fail to do so. (We note that an algorithm fail to obtain a number of best known results and still qualify as a top performing algorithm in the literature, given that other algorithms may generally obtain still fewer best known results.) Moreover, both of our algorithms and CirCut can reach the best known result on instance g3-8 with CPU time 292, 258 and 54 seconds, respectively. However, both PR algorithms perform slightly worse than SS on instance g3-15.

To verify whether the proposed PR algorithms are able to further improve the results by allowing longer computational time, we re-ran PR1 and PR2 on the MaxCut instances using 10 times longer time than before, as shown in Table 5.9. Surprisingly, both PR1 and PR2 can further improve its best results on a total of 33 instances. Although we only show the better results without differentiating whether they come from PR1 or PR2, we

Table 5.7: Computational results of PR1 and PR2 and comparison with other state-of-the-art algorithms on MaxCut instances of Set2

Instance	BKR	PR1			PR2			SS		CirCut	
		f_{best}	f_{avg}	t_{b_avg}	f_{best}	f_{avg}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}
G54100	110	110	110.0	0	110	110.0	0	110	1.9	110	16.2
G54200	112	112	112.0	0	112	112.0	0	112	1.9	112	18.6
G54300	106	106	106.0	0	106	106.0	0	106	2.1	106	15.8
G54400	114	114	114.0	0	114	114.0	0	114	2.1	114	16.0
G54500	112	112	112.0	0	112	112.0	0	112	2.3	112	15.8
G54600	110	110	110.0	0	110	110.0	0	110	2.1	110	15.4
G54700	112	112	112.0	0	112	112.0	0	112	2.0	112	14.8
G54800	108	108	108.0	0	108	108.0	0	108	2.1	108	15.4
G54900	110	110	110.0	0	110	110.0	0	110	1.8	110	15.5
G541000	112	112	112.0	0	112	112.0	0	112	1.4	112	16.4
G10100	896	896	894.3	99	896	894.6	24	882	406.1	880	106.0
G10200	900	900	900.0	1	900	900.0	1	894	302.4	892	116.0
G10300	892	892	890.5	342	892	891.3	71	884	410.4	882	112.0
G10400	898	898	898.0	3	898	898.0	1	892	485.9	894	103.0
G10500	886	886	885.4	48	886	885.4	36	880	400.9	882	106.0
G10600	888	888	888.0	1	888	888.0	1	870	461.8	886	119.0
G10700	900	900	898.1	400	900	898.2	414	890	386.2	894	115.0
G10800	882	882	881.3	39	882	881.2	31	880	466.9	874	104.0
G10900	902	902	900.9	143	902	901.5	63	888	493.6	890	121.0
G101000	894	894	893.5	27	894	893.7	8	886	352.8	886	111.0
G14100	2446	2442	2437.1	581	2444	2437.6	1682	2428	1320.6	2410	382.0
G14200	2458	2456	2452.1	985	2456	2452.4	361	2418	1121.1	2416	351.0
G14300	2442	2440	2432.9	491	2438	2435.5	551	2410	1215.8	2408	377.0
G14400	2450	2446	2440.2	1739	2448	2440.0	1036	2422	1237.2	2414	356.0
G14500	2446	2446	2437.9	877	2444	2438.7	1193	2416	1122.5	2406	388.0
G14600	2450	2448	2441.2	1163	2448	2442.3	884	2424	1213.9	2412	331.0
G14700	2444	2440	2431.5	1829	2440	2435.0	1384	2404	1230.6	2410	381.0
G14800	2448	2442	2436.9	1725	2444	2438.9	1055	2416	1132.0	2418	332.0
G14900	2426	2422	2414.7	1605	2422	2417.3	1185	2412	1213.9	2388	333.0
G141000	2458	2452	2445.8	2097	2454	2448.8	1345	2430	1125.8	2420	391.0
Av.				473.2			377.5		537.3		163.2
Better		0			0			0		0	
Equal		21			20			10		10	
Worse		9			10			20		20	

Table 5.8: Computational results of PR1 and PR2 and comparison with other state-of-the-art algorithms on MaxCut instances of Set3

Instance	BKR	PR1			PR2			SS		CirCut	
		f_{best}	f_{avg}	t_{b_avg}	f_{best}	f_{avg}	t_{b_avg}	f_{best}	t_{b_avg}	f_{best}	t_{b_avg}
g3-15	283206561	279830931	277345801.1	3000	276903146	273564256.6	1272	281029888	1023	268519648	788
g3-8	41684814	41684814	41508934.7	292	41684814	41521529.9	258	40314704	66	41684814	54
pm3-15-50	3000	3010	3006.6	1602	**3014**	3007.3	1890	2964	333	2895	427
pm3-8-50	458	458	458.0	2	458	458.0	2	442	49	454	39
Av.				1224.0			855.5		367.7		326.9
Better		1			1			0		0	
Equal		2			2			0		1	
Worse		1			1			4		3	

Table 5.9: Computational results of our PR algorithms on MaxCut instances with longer CPU time

Instance	f_{best}	t_{b_avg}	Instance	f_{best}	t_{b_avg}	Instance	f_{best}	t_{b_avg}
G25	13340	3539	G27	3341	3040	G28	3298	17482
G30	3413	4795	G31	3310	10801	G37	7686	3903
G38	7688	17230	G39	2408	3087	G40	2400	11947
G41	2405	945	G42	2481	5580	G55	10274	31764
G56	3993	11727	G57	3484	4968	G58	19225	20499
G59	6039	28790	G60	14131	62466	G61	5748	29056
G62	4854	59568	G63	26941	45136	G64	8693	66851
G65	5544	94934	G66	6340	74375	G67	6928	114438
G70	9529	135572	G72	6978	141167	G14100	2446	2105
G14200	2458	1657	G14600	2450	1476	G14700	2442	2824
G14800	2446	3543	G14900	2426	7165	G141000	2458	8929

find that PR1 and PR2 obtain the same results on 7 instances of set 2, while better results come from PR2 for the 25 instances of Set1 (except the instance G31).

5.3.3 Experiments on MSCP benchmark

In this section, we recast the minimum sum coloring problem (MSCP) into the BQO formulation and solve it with the proposed PR2 algorithm. Before presenting our experimental results, we first introduce the MSCP and illustrate how to transform it into the form of BQO.

5.3.3.1 Minimum sum coloring problem

Given an undirected graph $G = (V, E)$ with vertex set V and edge set E, a K-coloring of G is a function $c : v \mapsto c(v)$ that assigns to each vertex $v \in V$ a color $c(v)$, where $c(v) \in \{1, 2, \ldots, K\}$. A K-coloring is considered proper if each pair of vertices (u, v) connected by an edge $(u, v) \in E$ receive different colors $c(u) \neq c(v)$. The minimum sum coloring problem is to find a proper K-coloring c such that the total sum of colors over all the vertices $\sum_{v \in V} c(v)$ is minimized. The minimum value of this sum is called the chromatic sum of G and denoted by $\sum(G)$. The number of colors related to the chromatic sum is called the strength of the Graph and denoted by $s(G)$.

The MSCP is NP-hard for general graphs [Kubicka and Schwenk, 1989] and provides applications mainly including VLSI design, scheduling and resource allocation [Bar-Noy *et al.*, 1998; Malafiejski, 2004]. Given the theoretical and practical significance of the MSCP, effective approximation algorithms and polynomial algorithms have been presented for some special cases of graphs, such as trees, interval graphs and bipartite graphs [Bar-Noy and Kortsarz, 1998; Bonomo *et al.*, 2011; Hajiabolhassan *et al.*, 2000; Jansen, 2000; Kroon *et al.*, 1996; Malafiejski, 2004; Salavatipour, 2003; Thomassen *et al.*, 1989]. For the purpose of practical solving of the general MSCP, a variety of heuristics have been proposed in recent years, comprising a parallel genetic algorithm [Kokosinski and Kawarciany, 2007], a greedy algorithm [Li *et al.*, 2009], a tabu search algorithm [Bouziri and Jouini, 2010], a hybrid local search algorithm [Douiri and Elbernoussi, 2011], an independent set extraction

based algorithm [Wu and Hao, 2012] and a local search algorithm [Helmar and Chiarandini, 2011].

5.3.3.2 Transformation of the MSCP to the BQO model

5.3.3.2.1 Linear model for the MSCP Given an undirected graph $G = (V, E)$ with vertex set V ($n = |V|$) and edge set E. Let x_{uk} be 1 if vertex u is assigned color k, and 0 otherwise. The linear programming model for the MSCP can be formulated as follows:

$$\begin{aligned} Min \quad & f(x) = \sum_{u=1}^{n} \sum_{k=1}^{K} k \cdot x_{uk} \\ \text{subject to:} \quad & c1. \ \sum_{k=1}^{K} x_{uk} = 1, \quad u \in \{1, \ldots, n\} \\ & c2. \ x_{uk} + x_{vk} \leq 1, \ \forall (u, v) \in E, \ k \in \{1, \ldots, K\} \\ & c3. \ x_{uk} \in \{0, 1\} \end{aligned} \tag{5.1}$$

5.3.3.2.2 Nonlinear BQO alternative The linear model of the MSCP can be recast into the BQO form according to the following steps:

For the constraints $c1.$, we represent these linear equations by a matrix $Ax = b$ and incorporate the following penalty transformation [Kochenberger *et al.*, 2004]:

$$\begin{aligned} \#1: \quad f1(x) &= P(Ax - b)^t(Ax - b) \\ &= P[x^t(A^tA)x - x^t(A^tb) - (b^tA)x + b^tb] \\ &= P[x^t(A^tA)x - x^t(A^tb) - (b^tA)x] + Pb^tb \\ &= xD_1x + c \end{aligned} \tag{5.2}$$

For the constraints $c2.$, we utilize the quadratic penalty function $g(x) = Px_{uk}x_{vk}$ to replace each inequality $x_{uk} + x_{vk} \leq 1$ in $c2.$ and add them up as follows [Kochenberger *et al.*, 2004]:

$$\begin{aligned} \#2: \quad f2(x) &= \sum_{u=1}^{n} \sum_{v=1, u \neq v}^{n} \sum_{k=1}^{K} w_{uv}x_{uk}x_{vk} \\ &= xD_2x \end{aligned} \tag{5.3}$$

where $w_{uv} = P$ if $(u, v) \in E$ and 0 otherwise.

To construct the nonlinear BQO formulation $h(x)$, we first inverse the minimum objective of the MSCP to be $-f(x)$ in accordance with the general BQO model under a maximum objective, which becomes the first component of $h(x)$. Then we add the penalty function $f1(x)$ into $h(x)$ such that $f1(x) = 0$ if all the linear equations in $c1.$ are satisfied and otherwise $f1(x)$ is a penalty term with large negative values. In the same way, we add the penalty function $f2(x)$ into $h(x)$. Hence, the resulting BQO formulation for the MSCP can be expressed as follows:

$$\begin{aligned} BQO: \quad Max\ h(x) &= -f(x) + f1(x) + f2(x) \\ &= xQ'x + c \end{aligned} \tag{5.4}$$

Once the optimal objective value for this BQO formulation is obtained, the minimum sum coloring value can be readily obtained by taking its inverse value.

Further, a penalty scalar P is considered to be suitable as long as its absolute value $|P|$ is larger than half of the maximum color ($|P| > K/2$). Consider that penalty functions should be negative under the case of a maximal objective, we select $P = -500$ for the benchmark instances experimented in this paper. The optimized solution x obtained by solving the nonlinear BQO formulation indicates that such selection ensures both $f1(x)$ and $f2(x)$ equal to 0. In other words, each variable x_{uk} with the assignment of 1 in the optimized solution x forms a feasible K-coloring in which vertex u gets the color k.

5.3.3.2.3 An example of the transformation To illustrate the transformation from the MSCP to the BQO formulation, we consider the following graph with $|V| = 4$ and expect to find a legal K-coloring with $K = 2$.

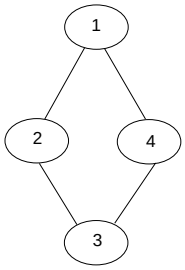


Figure 5.1: A graph sample of illustrating the transformation method of MSCP into BQO

Its linear formulation according to Equation (5.1) is:

$$\begin{aligned} Max \quad & f(x) = -x_{11} - 2x_{12} - x_{21} - 2x_{22} - x_{31} - 2x_{32} - x_{41} - 2x_{42} \\ & \text{c1.} \quad x_{11} + x_{12} = 1; \\ & \qquad x_{21} + x_{22} = 1; \\ & \qquad x_{31} + x_{32} = 1; \\ & \qquad x_{41} + x_{42} = 1; \\ & \text{c2.} \quad x_{11} + x_{21} \leq 1; \quad x_{12} + x_{22} \leq 1; \\ & \qquad x_{11} + x_{41} \leq 1; \quad x_{12} + x_{42} \leq 1; \\ & \qquad x_{21} + x_{31} \leq 1; \quad x_{22} + x_{32} \leq 1; \\ & \qquad x_{31} + x_{41} \leq 1; \quad x_{32} + x_{42} \leq 1; \\ & \text{c3.} \quad x_{11}, x_{12}, \ldots, x_{42} \in \{0, 1\} \end{aligned}$$

Choosing the scalar penalty $P = -5$, we obtain the following BQO model:

$$Max \quad f(x) = xQx - 20$$

where xQx is written as:

$$\begin{pmatrix} x_{11} & x_{12} & \dots & x_{41} & x_{42} \end{pmatrix} \times \begin{pmatrix} 4 & -5 & -5 & 0 & 0 & 0 & -5 & 0 \\ -5 & 3 & 0 & -5 & 0 & 0 & 0 & -5 \\ -5 & 0 & 4 & -5 & -5 & 0 & 0 & 0 \\ 0 & -5 & -5 & 3 & 0 & -5 & 0 & 0 \\ 0 & 0 & -5 & 0 & 4 & -5 & -5 & 0 \\ 0 & 0 & 0 & -5 & -5 & 3 & 0 & -5 \\ -5 & 0 & 0 & 0 & -5 & 0 & 4 & -5 \\ 0 & -5 & 0 & 0 & 0 & -5 & -5 & 3 \end{pmatrix} \times \begin{pmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{41} \\ x_{42} \end{pmatrix}$$

Solving this BQO model yields $x_{11} = x_{22} = x_{31} = x_{42} = 1$ (all other variables equal zero) and the optimal objective function value $f(x) = -6$. Reversing this objective function value leads to the optimum (the minimum sum coloring) of 6 for this graph.

5.3.3.3 Results of the BQO-PR approach for the MSCP

Our final experiment aims at the evaluation of our proposed BQO-PR approach for the minimum sum coloring problem. We use a set of 23 graphs[4], which are the most often used benchmark instances in the literature. The time limit for a single run of our BQO-PR approach is set as follows: 1 hour for the first 16 instances in Table 5.10; 10 hours for dsjc125.1, dsjc125.5, dsjc125.9, dsjc250.1 and dsjc250.5; 20 hours for dsjc250.9 and dsjc500.1. Given the stochastic nature of our approach, each problem instance is independently solved 20 times.

According to preliminary experiments, we set tt as the maximal value between 40 and a random integer from the interval [N/100, N/100+50] (where N denotes the number of variables in the resulting BQO model). In addition, we set $\mu = 2N$ for the improvement of the initial solutions in $RefSet$ and $\mu = 500$ for the improvement of the solutions on the path, respectively.

Table 5.10 presents the computational statistics of the BQO model. Columns 1 to 3 give the instance names *Instances* along with the vertex number V and edge number E of the graphs. Columns 4 and 5 show the number of colors K to be used and the number of variables N in the BQO formulation. Column 6 summarizes the best known results BKR from the previous literature [Bouziri and Jouini, 2010; Douiri and Elbernoussi, 2011; Helmar and Chiarandini, 2011; Li *et al.*, 2009; Kokosinski and Kawarciany, 2007; Wu and Hao, 2012]. The columns under the heading of BQO-PR report our results of the BQO model solved by the PR algorithm: the best objective values $Best$, the average objective values Avr, the standard deviation σ, the average time t_{b_avg} (in seconds) to reach the best objective value f_{best} over 20 runs, and the average time (in seconds) t_{avg} consumed to reach the best objective value obtained in each run. The last two columns of Table 5.10 show the results (the best solution and the time to reach the best solution) of the CPLEX V12.2 MIP solver using the linear model of Section 5.3.3.2.1. Results marked

[4] http://mat.gsia.cmu.edu/COLOR/instances.html

Table 5.10: Computational results of BQO-PR and CPLEX on MSCP instances

Instances	V	E	K	N	*BKR*	BQO-PR					CPLEX	
						f_{best}	f_{avg}	σ	t_{b_avg}	t_{avg}	f_{best}	t_{b_avg}
myciel3	11	20	6	66	21*	**21**	21.0	0.0	< 1	< 1	**21**	< 1
myciel4	23	71	7	161	45*	**45**	45.0	0.0	< 1	< 1	**45**	< 1
myciel5	47	236	8	376	93*	**93**	93.0	0.0	2	2	**93**	1
myciel6	95	755	10	950	189*	**189**	189.0	0.0	497	497	**189**	123
myciel7	191	2360	10	1910	381	**381**	384.9	3.7	70	384	383	609
anna	138	493	13	1794	276*	**276**	276.3	0.4	870	721	**276**	< 1
david	87	406	13	1131	237*	**237**	237.0	0.0	524	524	**237**	< 1
huck	74	301	13	962	243*	**243**	243.0	0.0	3	3	**243**	< 1
jean	80	254	12	960	217*	**217**	217.0	0.0	79	79	**217**	< 1
queen5.5	25	160	7	175	75*	**75**	75.0	0.0	< 1	< 1	**75**	< 1
queen6.6	36	290	9	324	138*	**138**	138.0	0.0	6	6	**138**	284
queen7.7	49	476	9	441	196*	**196**	196.0	0.0	6	6	**196**	1
queen8.8	64	728	10	640	291*	**291**	298.1	5.1	1064	780	**291**	27595
games120	120	638	10	1200	443*	**443**	446.5	3.2	755	880	**443**	4
miles250	128	387	10	1280	325*	**325**	328.6	2.3	777	1704	**325**	< 1
miles500	128	1170	22	2816	705*	713	722.5	6.4	653	1942	**705**	15
DSJC125.1	125	736	8	1000	326	329	338.5	6.3	1684	7115	333	36000
DSJC125.5	125	3891	22	2750	1015	1050	1082.6	20.2	32924	15186	1127	36000
DSJC125.9	125	6961	50	6250	2511	2529	2573.8	26.2	34801	24650	NF	–
DSJC250.1	250	3218	12	3000	977	1027	1062.9	16.8	8893	17206	1064	36000
DSJC250.5	250	15668	35	8750	3246	3604	3724.9	59.1	27009	26065	NF	–
DSJC250.9	250	27897	80	20000	8286	8604	8869.4	122.2	70737	65673	NF	–
DSJC500.1	500	12458	16	8000	2850	3152	3234.1	41.7	42447	59241	3874	72000

in bold indicate BQO-PR or CPLEX matches the *BKR* on these instances and results marked as "NF" suggest no feasible solution are found.

From Table 5.10, we observe that our BQO-PR approach is able to reach the best known results for 15 out of 23 instances, among which 14 results are proved by CPLEX to be optimal values. Only very few best heuristics specifically tailored to the MSCP can compete with this performance for these instances (see Table 5.11, Section 5.3.3.4). To attain these solutions, BQO-PR needs an average time ranging from less than one second to 30 minutes. However, BQO-PR fails to match the best known results for the remaining instances after 0.6 to 18.5 hours.

On the other hand, CPLEX V12.2 solves 15 instances among which a new upper bound (which is also the optimal solution) for the instance miles500 is discovered. The running time needed to solve these instances is in most cases short (from less than one second to 10 minutes) with the exception for the problem queen8.8 (requiring about 7.6 hours). For the remaining 8 instances, CPLEX V12.2 either terminates with an sub-optimal solution (5 cases) which is worse than the best upper bound (and worse than the bound of BQO-PR) or fails to find a feasible integer solution (3 cases indicated by 'NF') even after 10 to 20 hours.

5.3.3.4 Comparison with other special purpose algorithms for the MSCP

In order to further evaluate our BQO-PR approach, we show a comparison of the proposed approach with several special purpose algorithms for the MSCP. These algorithms include a hybrid local search algorithm HLS [Douiri and Elbernoussi, 2011], an advanced recursive

Table 5.11: Comparison between BQO-PR and other specific MSCP algorithms

Instances	*BKR*	BQO-PR	HLS	MRLF	PGA	TS	EXSCOL	MDS(5)
myciel3	21*	**21**	**21**	**21**	**21**	–	**21**	**21**
myciel4	45*	**45**	**45**	**45**	**45**	–	**45**	**45**
myciel5	93*	**93**	**93**	**93**	**93**	–	**93**	**93**
myciel6	189*	**189**	**189**	**189**	**189**	–	**189**	**189**
myciel7	381	**381**	**381**	**381**	382	–	**381**	**381**
anna	276*	**276**	–	277	281	–	283	**276**
david	237*	**237**	–	241	243	–	**237**	**237**
huck	243*	**243**	**243**	244	**243**	–	**243**	**243**
jean	217*	**217**	–	**217**	218	–	**217**	**217**
queen5.5	75*	**75**	–	**75**	**75**	–	**75**	**75**
queen6.6	138*	**138**	**138**	**138**	**138**	–	150	**138**
queen7.7	196*	**196**	–	**196**	**196**	–	**196**	**196**
queen8.8	291*	**291**	–	303	302	–	**291**	**291**
games120	443*	**443**	446	446	460	–	**443**	**443**
miles250	325*	**325**	343	334	347	–	328	**325**
miles500	705*	713	755	715	762	–	**709**	712
DSJC125.1	326	329	–	352	–	344	**326**	**326**
DSJC125.5	1015	1050	–	1141	–	1103	1017	**1015**
DSJC125.9	2511	2529	–	2653	–	2631	2512	**2511**
DSJC250.1	977	1027	–	1068	–	1046	985	**977**
DSJC250.5	3246	3604	–	3658	–	3779	**3246**	3281
DSJC250.9	8286	8604	–	8942	–	9198	**8286**	8412
DSJC500.1	2850	3152	–	3229	–	3205	**2850**	2951

largest first algorithm MRLF [Li *et al.*, 2009], a parallel genetic algorithm PGA [Kokosinski and Kawarciany, 2007], a tabu search algorithm TS [Bouziri and Jouini, 2010], a very recent independent set extraction based heuristic EXSCOL [Wu and Hao, 2012] and a recent local search heuristic MSD(5) [Helmar and Chiarandini, 2011]. Given that no computing time is reported in [Bouziri and Jouini, 2010; Douiri and Elbernoussi, 2011; Kokosinski and Kawarciany, 2007; Li *et al.*, 2009] and different termination conditions are used in these studies, we would prefer to base the comparison on solution quality. Table 5.11 presents the best objective values obtained by each algorithm (BQO-PR, HLS, MRLF, PGA, TS, EXSCOL and MSD(5), respectively) where the best solution values among them are marked in bold. The results for HLS, PGA and TS which are unavailable are marked with "–".

As we can observe in Table 5.11, the proposed BQO-PR approach outperforms HLS, MRLF, PGA and TS in terms of the best solution values. Specifically, BQO-PR finds better solutions than HLS, MRLF, PGA and TS for 3, 14, 8 and 7 instances, respectively. However, BQO-PR performs less well compared with the most effective MSCP heuristics EXSCOL and MDS(5) for most instances.

5.3.3.5 A short discussion

From Tables 5.10 and 5.11, we observe that our BQO-PR approach is quite robust to reach optimal or best known solution values within a short period of time for the instances with $N < 2000$ variables in comparison with a slow convergence for instances with many more BQO variables (see column T_{AVR} in Table 5.10). This can be partially explained by the fact that the number of the BQO variables (equaling $V \cdot K$ where V is the number

of vertices and K is the number of colors) sharply increases with the growth of V and K. Additionally, at present our approach is not able to solve graph instances with BQO variables surpassing the threshold value of 20,000 because of the memory limitation. These obstacles could be overcome by designing more effective data structures used by the BQO algorithms.

Generally, to improve the efficiency of the BQO approach, it would be useful to integrate in the BQO implementation dedicated techniques and solution strategies. For instance, pre-processing techniques could be applied to simplify the input matrix. Specific techniques can be envisaged to take advantage of particular matrix characteristics (induced by the targeted problem such as the MSCP). Indeed, based on these characteristics (which could be found in different types of problems), special purpose search mechanisms and operators could be designed. These possibilities (and certainly many other ones) constitute interesting research directions.

5.4 Conclusions

In this chapter, we proposed two effective path relinking algorithms for the BQO problem. The proposed algorithms are composed of a reference set construction method, a tabu search based improvement method, a reference set update method, a relinking method and a path solution selection method. The proposed algorithms differ from each other mainly on the way they generate the path, one employing a greedy strategy (PR1) and the other employing a random strategy (PR2). Moreover, we investigated the performance of the proposed PR1 and PR2 algorithms for the minimum cut problem (MaxCut) and investigated for the first time the possibility of solving the NP-hard minimum sum coloring problem (MSCP) via binary quadratic optimization (BQO).

Computational experiments with PR1 and PR2 on 134 well-known BQO and MaxCut benchmark instances have demonstrated that both algorithms are capable of attaining highly competitive results in comparison with the previous best known results from the literature. In particular, for 103 MaxCut instances, our algorithms can improve the best known results for almost 40 percent of these instances whose optimum solutions are still unknown.

Moreover, computational experiment on 23 MSCP instances have shown that the proposed BQO-PR approach is able to reach competitive solutions when compared with several special purpose MSCP algorithms for a number of instances. However, due to the limitation of the current implementation, the BQO approach for the MSCP requires considerable computing time to find solutions of good quality for large instances. This study also shows that even if the state of the art CPLEX V12.2 MIP solver can solve some MSCP instances, the MSCP would remain a very challenging problem for CPLEX V12.2 (and any exact solution approach).

Up to now, we have successfully dealt with the general binary quadratic optimization problem without any constraints included. In the next chapter, we are interested in solving cardinality constrained binary quadratic optimization, also known as the maximum diversity problem and will develop a tabu search based memetic algorithm for this

problem.

Chapter 6

A tabu search based memetic algorithm for the maximum diversity problem

This chapter presents a highly effective memetic algorithm based on tabu search for handling a cardinality constrained binary quadratic optimization problem (also known as the maximum diversity problem (MDP)). The tabu search component uses a successive filter candidate list strategy and the solution combination component employs a combination operator based on identifying strongly determined and consistent variables. Computational experiments on three sets of 40 popular benchmark instances indicate that our tabu search/memetic algorithm (TS/MA) easily obtains the best known results for all the tested instances (which no previous algorithm has achieved) and obtains improved results for 6 instances. Analysis of comparisons with state-of-the-art algorithms demonstrate statistically that our TS/MA algorithm competes very favorably with the best performing algorithms. Key elements and properties of TS/MA are also analyzed to disclose the source of its success. The content of this chapter is based on the paper [Wang *et al.*, 2012d] submitted to Computers & Operations Research.

Contents

6.1 Introduction

The maximum diversity problem (MDP) consists in identifying a subset M of a given cardinality m from a set of elements N, such that the sum of the pairwise distance between the elements in M is maximized. More precisely, let $N = \{e_1, \ldots, e_n\}$ be a set of elements and d_{ij} be the distance between elements e_i and e_j. The objective of MDP can be formulated as follows [Kuo *et al.*, 1993]:

$$\begin{aligned} Maximize \quad & f(x) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij} \cdot x_i \cdot x_j \\ \text{subject to} \quad & \sum_{i=1}^{n} x_i = m, \quad x_i \in \{0,1\}, \quad i = 1, \ldots, n \end{aligned} \tag{6.1}$$

where each x_i is a binary (zero-one) variable indicating whether an element $e_i \in N$ is selected to be a member of the subset M.

MDP is an NP-hard problem and provides practical applications mainly including location, ecological systems, medical treatment, genetics, ethnicity, product design, immigration and admissions policies, committee formation, curriculum design, and so on [Katayama *et al.*, 2005; Martí *et al.*, 2011].

Due to its theoretical significance and many potential applications, various solution procedures have been devised for the MDP problem. Exact algorithms are able to solve instances with less than 100 variables in reasonable computing time [Martí *et al.*, 2010]. However, because of the high computational complexity, heuristic and metaheuristic algorithms are commonly used to produce approximate solutions to larger problem instances. Examples of these methods include various GRASP variants [Andrade *et al.*, 2003; Andrade *et al.*, 2005; Duarte and Marti, 2007; Silva *et al.*, 2004; Silva *et al.*, 2007], tabu search based algorithms [Aringhieri *et al.*, 2008; Aringhieri and Cordone, 2011; Palubeckis, 2007; Wang *et al.*, 2012a], variable neighborhood search [Brimberg J, 2009], iterated greedy algorithm [Lozano *et al.*, 2011] and hybrid evolutionary algorithm [Gallego *et al.*, 2009; Katayama *et al.*, 2005]. A review concerning MDP can be found in [Martí *et al.*, 2011].

Our proposed TS/MA falls within the memetic algorithm classification as laid out in [Neri *et al.*, 2011] (and in particular adopts the scatter search template described in [Glover and Laguna, 1997]). First, we use tabu search to improve each solution generated initially

or created by combining members of a current population. The TS moves are simple swaps that flip (or add and drop) solution elements, drawing on the successive filter candidate list strategy [Glover and Laguna, 1997] to accelerate the move evaluations. Second, we design a solution combination operator to take advantage of solution properties by reference to the analysis of strongly determined and consistent variables [Glover, 1977]. Finally, we introduce a population rebuilding strategy that effectively maintains population diversity.

In order to evaluate the performance of TS/MA, we conduct experimental tests on 3 sets of challenging benchmarks with a total of 40 instances. The test results indicate that TS/MA yields highly competitive outcomes on these instances by finding improved best known solutions for 6 instances and matching the best known results for the other instances. Furthermore, we analyze the influence of some critical components and demonstrate their key roles to the performance of the proposed TS/MA algorithm.

6.2 Tabu Search/Memetic Algorithm

Algorithms that combine improvement methods with population-based solution combination algorithms, and hence that can be classified as memetic algorithms [Neri *et al.*, 2011], often prove effective for discrete optimization [Hao, 2011]. By linking the global character of recombinant search with the more intensive focus typically provided by local search, the memetic framework offers interesting possibilities to create a balance between intensification and diversification within a search procedure. Our TS/MA algorithm follows the general memetic framework and is mainly composed of four components: a population initialization and rebuilding procedure, a tabu search procedure, a specific solution combination operator and a population updating rule. As previously noted, our procedure more specifically adopts the form of a scatter search procedure, and utilizes combinations from the structured class proposed for scatter search in [Glover, 1994].

6.2.1 Main scheme

The general architecture of our TS/MA algorithm is described in Algorithm 6.1. It starts with the creation of an initial population P (line 3, Section 6.2.3). Then, the solution combination is employed to generate new offspring solution (line 8, Section 6.2.5), whereupon a TS procedure (line 9, Section 6.2.4) is launched to optimize each newly generated solution. Subsequently, the population updating rule decides whether such an improved solution should be inserted into the population and which existing individual should be replaced (line 13, Section 6.2.3). Finally, if the population is not updated for a certain number of generations, the population rebuilding procedure is triggered to build a new population (line 20, Section 6.2.3). In the following subsections, the main components of our TS/MA algorithm are described in detail.

6.2.2 Search space and evaluation function

Given an n element set $N = \{e_1, \ldots, e_n\}$, the search space Ψ of MDP consists of all the m-elements subsets of N; i.e., $\Psi = \{S | S \subset N, |S| = m\}$. Thus the search space size

Algorithm 6.1: Outline of the TS/MA algorithm for MDP

```
1: Input: an n × n matrix (d_ij), a given cardinality m ≤ n
2: Output: the best solution x* found
3: P = {x^1, ..., x^p} ← Population_Initialization( )          /* Section 6.2.3 */
4: x* = arg max{f(x^i)|i = 1, ..., p}
5: while a stop criterion is not satisfied do
6:    repeat
7:       randomly choose two solutions x^i and x^j from P
8:       x^0 ← Combination_Operator(x^i, x^j)                  /* Section 6.2.5 */
9:       x^0 ← Tabu_Search(x^0)                                /* Section 6.2.4 */
10:      if f(x^0) > f(x*) then
11:         x* = x^0
12:      end if
13:      {x^1, ..., x^p} ← Population_Updating(x^0, x^1, ..., x^p)   /* Section 6.2.3 */
14:      if P does not change then
15:         UpdatingNonSuccess = UpdatingNonSuccess + 1
16:      else
17:         UpdatingNonSuccess = 0
18:      end if
19:   until UpdatingNonSuccess > θ
20:   P = {x^1, ..., x^p} ← Population_Rebuliding()            /* Section 6.2.3 */
21: end while
```

equals $\binom{n}{m}$. A feasible solution of MDP can be conveniently represented as an n-vector of binary variables x such that exactly m variables receive the value of 1 and the other $n-m$ variables receive the value of 0. Given a solution $x \in \Psi$, its quality or fitness is directly measured by the objective function $f(x)$ of Eq. 6.1.

6.2.3 Population initialization, rebuilding and updating

The initial population contains p different local optimal solutions and is constructed as follows. First, we randomly generate an initial feasible solution, i.e., any n-vector with exactly m elements assigned the value of 1. Then this solution is submitted to the tabu search procedure to obtain an improved solution which is also a local optimum but not a first local optimum encountered (see Section 6.2.4). Then, the solution improved by tabu search is added in the population if it does not duplicate any solution in the population. This procedure is repeated until the population size reaches the specified value p.

This procedure is also used by the TS/MA algorithm when the population is not updated for θ consecutive generations (θ is a parameter and called the *population rebuilding threshold*). In this case, the population is recreated as follows. First, the best solution x^* from the old population becomes the first member of the new population. Second, for each of the remaining solutions in the old population, we carry out the following steps: (1) randomly interchange $\rho \cdot m$ variables with the value of 1 and $\rho \cdot m$ variables with the value of 0 where $0 < \rho < 1$ (ρ is a parameter and called the *perturbation fraction*); (2) this perturbed solution is submitted to tabu search to obtain an improved solution; (3) if this

refined solution is not a duplication of any solution in the new population, it is added in the new population; otherwise, the method returns to step (1).

The population updating procedure is invoked each time a new offspring solution is generated by the combination operator and then improved by tabu search. As in a simple version of the scatter search template of [Glover and Laguna, 1997], the improved offspring solution is added into the population if it is distinct from any solution in the population and better than the worst solution, while the worst solution is removed from the population.

6.2.4 Tabu search procedure

To improve the quality of a solution, we use a tabu search procedure which applies a constrained *swap* operator to exchange a variable having the value of 1 with a variable having the value of 0. More formally, given a feasible solution $x = \{x_1, \ldots, x_n\}$, let U and Z respectively denote the set of variables with the value of 1 and 0 in x. Then, the neighborhood $N(x)$ of x consists of all the solutions obtained by swapping two variables $x_i \in U$ and $x_j \in Z$. Since this swap operator keeps the m cardinality constraint satisfied, the neighborhood contains only feasible solutions. Clearly, for a given solution x, its neighborhood $N(x)$ has a size of $m \cdot (n - m)$.

To rapidly determine the move gain (the objective change on passing from the current solution to its neighboring solution), we apply the following technique:

First, we employ a vector Δ to record the objective variation of moving a variable x_i from its current subset U/Z into the other subset Z/U. This vector can be initialized as follows:

$$\Delta_i = \begin{cases} \sum_{j \in U} -d_{ij} & (x_i \in U) \\ \sum_{j \in U} d_{ij} & (x_i \in Z) \end{cases} \tag{6.2}$$

Then, the move gain of interchanging two variables $x_i \in U$ and $x_j \in Z$ can be calculated using the following formula:

$$\delta_{ij} = \Delta_i + \Delta_j - d_{ij} \tag{6.3}$$

Finally, once a move is performed, we just need to update a subset of move gains affected by the move. Specifically, the following abbreviated calculation can be performed to update Δ upon swapping variables x_i and x_j [Lü *et al.*, 2012]:

$$\Delta_k = \begin{cases} -\Delta_i + d_{ij} & (k = i) \\ -\Delta_j + d_{ij} & (k = j) \\ \Delta_k + d_{ik} - d_{jk} & (k \neq \{i, j\}, x_k \in U) \\ \Delta_k - d_{ik} + d_{jk} & (k \neq \{i, j\}, x_k \in Z) \end{cases} \tag{6.4}$$

Given the size of the swap neighborhood which is equal to $m \cdot (n - m)$, it could be computationally costly to identify the best move at each iteration of tabu search. To overcome this obstacle, we employ the successive filter candidate list strategy of [Glover and Laguna, 1997] that breaks a compound move (like a swap) into component operations and reduces the set of moves examined by restricting consideration to those that produce high quality outcomes for each separate operation.

For the swap move, we first subdivide it into two successive component operations: (1) move the variable x_i from U to Z; (2) move the variable x_j from Z to U. Since the resulting objective difference of each foregoing operation can be easily obtained from the vector Δ, we then pick for each component operation top cls variables in terms of their Δ values recorded in a non-increasing order to construct the candidate lists UCL and ZCL. Finally, we restrict consideration to swap moves involving variables from UCL and ZCL. The benefits of this strategy will be verified in Section 6.4.

To ensure solutions visited within a certain span of iterations will not be revisited, tabu search typically incorporates a short-term memory, known as *tabu list* [Glover and Laguna, 1997]. In our implementation, each time two variables x_i and x_j are swapped, two random integers are taken from an interval $tt = [a, b]$ (where a and b are chosen integers) as the tabu tenure of variables x_i and x_j to prevent any move involving either x_i or x_j from being selected for a specified number of iterations. (The integers defining the range of tt are parameters of our procedure, identified later.) Specifically, our tabu list is defined by a n-element vector T. When x_i and x_j are swapped, we assign the sum of a random integer from tt and the current iteration count $Iter$ to the i_{th} element $T[i]$ of T and the sum of another random integer from tt and $Iter$ to $T[j]$. Subsequently, for any iteration $Iter$, a variable x_k is forbidden to take part in a swap move if $T[k] > Iter$.

Tabu search then restricts consideration to variables not currently tabu, and at each iteration performs a swap move that produces the best (largest) move gain according to Eq. (6.3). In the case that two or more swap moves have the same best move gain, one of them is chosen at random.

To accompany this rule, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the best solution found so far. The tabu search procedure terminates when the best solution cannot be improved within a given number α of iterations which we call the *improvement cutoff*.

The pseudo-code of the tabu search procedure is shown in Algorithm 6.1.

6.2.5 Solution combination by reference to critical variables

Our memetic algorithm uses a dedicated solution combination operator to generate promising offspring solutions. The combination operator is based on the idea of critical variables which are given the name *strongly determined and consistent variables* in [Glover, 1977]. In the context of MDP, the notion of strongly determined and consistent variables can be defined as follows.

Definition 1 (Strongly determined variables). Relative to a given solution $x = \{x_1, x_2, ..., x_n\}$, let U denote the set of variables with the value of 1 in x. Then, for a specific variable $x_i \in U$, the (objective function) *contribution* of x_i in relation to x is defined as:

$$VC_i(x) = \sum_{x_j \in U} d_{ij} \tag{6.5}$$

Obviously, the objective function of MDP can be computed with regard to VC as follows:

Algorithm 6.2: Pseudo-code of the tabu search procedure for MDP

Input: a given solution x and its objective function value $f(x)$
Output: an improved solution x^* and its objective function value $f(x^*)$
Initialize vector Δ according to Eq. 6.2, initialize tabu list vector T by assigning each element with value 0, initialize U and Z composed of variables with value of 1 and 0 in x, respectively, $Iter = 0$, $NonImpIter = 0$
while $NonImpIter < \alpha$ **do**
  Identify top cls variables from U and top cls variables from Z in terms of the Δ value to construct UCL and ZCL
  Identify the index i^*_{nt} and j^*_{nt} of non-tabu variables from UCL and ZCL that leads to the maximum δ value (computed according to Eq. 6.3) by swapping $x_{i^*_{nt}}$ and $x_{j^*_{nt}}$ (break ties randomly); Similarly identify i^*_t and j^*_t for tabu variables
  if $\delta_{i^*_t j^*_t} > \delta_{i^*_{nt} j^*_{nt}}$ and $f(x^*) + \delta_{i^*_t j^*_t} > f(x^*)$ **then**
    $i^* = i^*_t, j^* = j^*_t$
  else
    $i^* = i^*_{nt}, j^* = j^*_{nt}$
  end if
  $x_{i^*} = 0, x_{j^*} = 1$, $f(x) = f(x) + \delta_{i^* j^*}$, $U = U \backslash \{x_{i^*}\} \cup \{x_{j^*}\}$, $Z = Z \cup \{x_{i^*}\} \backslash \{x_{j^*}\}$
  Update Δ according to Eq. 6.4
  Update T by assigning $T[i] = Iter + rand(tt), T[j] = Iter + rand(tt)$
  if $f(x) > f(x^*)$ **then**
    $x^* = x, f(x^*) = f(x)$
    $NonImpIter = 0$
  else
    $NonImpIter = NonImpIter + 1$
  end if
  $Iter = Iter + 1$
end while

$$f(x) = \frac{1}{2} \cdot \sum_{x_i \in U} VC_i(x) \tag{6.6}$$

We sort all the variables in a non-increasing order according to their objective function contribution and select top β variables (β is a parameter of our algorithm) as strongly determined variables SD.

Definition 2 (Consistent variables). Relative to two local optimal (high quality) solutions x^i and x^j, let U_i and U_j respectively denote the set of variables with the value of 1 in x^i and x^j. Then, the consistent variables are defined as:

$$C = \{x_k | x_k \in U_i \cap U_j\} \tag{6.7}$$

Given two local optimal solutions x^i and x^j and a set of variables N, our critical variable combination operator constructs one offspring solution according to the following steps:

1. Identify strongly determined variables SD_i and SD_j with regard to x^i and x^j, respectively;

2. Select consistent variables that simultaneously emerges in SD_i and SD_j; i.e., $C = SD_i \cap SD_j$;

3. Randomly pick m-$|C|$ variables from the set N-C to satisfy the cardinality constraint (maintaining the number of variables with the value of 1 equal to m);

4. Construct a feasible offspring solution by assigning the value 1 to the variables selected in steps (2) and(3) and assigning the value 0 to the remaining variables.

6.3 Experimental results

6.3.1 Benchmark instances

Three sets of benchmarks with a total of 40 large instances (with at least 2000 variables) are utilized to evaluate the performance of the proposed approach. Small and medium scale benchmarks are excluded in our experimentation because these problem instances can be easily solved by many heuristics in a very short time and present no challenge for our TS/MA algorithm.

1. Random Type 1 instances (Type1_22): 20 instances with $n = 2000, m = 200$, where d_{ij} are integers generated from a [0,10] uniform distribution. These instances are first introduced in [Duarte and Marti, 2007] and can be downloaded from: http://www.uv.es/∼rmarti/paper/mdp.html.

2. ORLIB instances (b2500): 10 instances with $n = 2500, m = 1000$, where d_{ij} are integers randomly generated from [-100,100]. They all have a density of 0.1. These instances are derived from the BQO problem by ignoring the diagonal elements and are available from ORLIB.

3. Palubeckis instances (p3000 and p5000): 5 instances with $n = 3000, m = 0.5n$ and 5 instances with $n = 5000, m = 0.5n$, where d_{ij} are integers generated from a [0,100] uniform distribution. The density of the distance matrix is 10%, 30%, 50%, 80% and 100%. The sources of the generator and input files to replicate these problem instances can be found at: http://www.soften.ktu.lt/∼gintaras/max_div.html.

6.3.2 Experimental protocol

Our TS/MA algorithm is programmed in C and compiled using GNU g++ on a Xeon E5440 with 2.83GHz CPU and 8GB RAM. All computational results were obtained with the parameter values shown in Table 6.1. Given the stochastic nature of our algorithm, we solve each instance in the Type1_22 and ORLIB benchmarks 30 times, and solve each instance in the Palubeckis benchmark 15 times. For comparative purposes, TS/MA has the same time limit as other reference algorithms, which is set to 20, 300, 600 and 1800 seconds for each instance of Type1_22, b2500, p3000 and p5000, respectively.

Table 6.1: Settings of important parameters of the TS/MA algorithm

Parameters	Section	Description	Value
p	6.2.3	population size	10
θ	6.2.3	population rebuilding threshold	30
ρ	6.2.3	perturbation fraction	0.3
tt	6.2.4	tabu tenure interval	[15,25]
α	6.2.4	tabu search improvement cutoff	$6 \cdot m$
cls	6.2.4	candidate list size of each component operation	$\min(\sqrt{m}, \sqrt{n-m})$
β	6.2.5	number of strongly determined variables	$0.7 \cdot m$

Table 6.2: Computational results of TS/MA for Type1_22 instances

Instance	*BKR*	TS/MA					
		Best	*Succ.*	*Avg.*	σ	T_{best}	$T_{avg.}$
Type1_22.1	114271	**114271(0)**	17/30	114265.03(5.97)	12.18	14.69	14.95
Type1_22.2	114327	**114327(0)**	28/30	114318.60(8.40)	31.43	6.06	6.47
Type1_22.3	114195	**114195(0)**	14/30	114183.77(11.23)	14.77	11.36	11.33
Type1_22.4	114093	**114093(0)**	2/30	114076.13(16.87)	14.31	18.13	11.97
Type1_22.5	114196	**114196(0)**	7/30	114164.63(31.37)	28.58	12.51	12.67
Type1_22.6	114265	**114265(0)**	9/30	114248.57(16.43)	12.67	8.12	10.54
Type1_22.7	114361	**114361(0)**	30/30	114361.00(0.00)	0.00	5.12	5.12
Type1_22.8	114327	**114327(0)**	21/30	114313.87(13.13)	29.01	7.33	8.43
Type1_22.9	114199	**114199(0)**	8/30	114191.00(8.00)	12.20	8.24	10.87
Type1_22.10	114229	**114229(0)**	25/30	114227.27(1.73)	4.89	9.76	9.64
Type1_22.11	114214	**114214(0)**	6/30	114191.57(22.43)	18.43	13.76	12.76
Type1_22.12	114214	**114214(0)**	6/30	114189.50(24.50)	21.12	10.95	10.74
Type1_22.13	114233	**114233(0)**	22/30	114230.33(2.67)	6.95	11.74	11.60
Type1_22.14	114216	**114216(0)**	28/30	114214.70(1.30)	4.95	8.21	8.23
Type1_22.15	114240	**114240(0)**	9/30	114239.27(0.73)	0.51	15.70	11.57
Type1_22.16	114335	**114335(0)**	22/30	114331.40(3.60)	7.48	9.11	9.52
Type1_22.17	114255	**114255(0)**	12/30	114245.20(9.80)	9.78	8.42	10.97
Type1_22.18	114408	**114408(0)**	15/30	114406.93(1.07)	1.12	6.13	8.24
Type1_22.19	114201	**114201(0)**	24/30	114196.47(4.53)	9.45	6.52	6.73
Type1_22.20	114349	**114349(0)**	25/30	114341.37(7.63)	23.33	11.56	11.88
Av.		**(0)**	16.5/30	(9.57)	13.158	10.171	10.212

Table 6.3: Computational results of TS/MA for ORLIB instances

Instance	*BKR*	TS/MA					
		Best	*Succ.*	*Avg.*	σ	T_{best}	$T_{avg.}$
b2500-1	1153068	**1153068(0)**	28/30	1152983.20(84.80)	317.29	78.33	82.01
b2500-2	1129310	**1129310(0)**	24/30	1129224.87(85.13)	195.90	124.79	139.29
b2500-3	1115538	**1115538(0)**	24/30	1115399.60(138.40)	276.80	78.13	81.75
b2500-4	1147840	**1147840(0)**	19/30	1147701.33(138.67)	216.29	108.33	109.71
b2500-5	1144756	**1144756(0)**	23/30	1144702.87(53.13)	126.81	49.65	45.52
b2500-6	1133572	**1133572(0)**	24/30	1133517.60(54.40)	108.80	63.74	62.74
b2500-7	1149064	**1149064(0)**	17/30	1148977.13(86.87)	116.40	87.16	87.04
b2500-8	1142762	**1142762(0)**	23/30	1142731.33(30.67)	160.70	70.33	30.67
b2500-9	1138866	**1138866(0)**	30/30	1138866.00(0.00)	0.00	77.63	77.63
b2500-10	1153936	**1153936(0)**	30/30	1153936.00(0.00)	0.00	71.22	71.22
Av.		**(0)**	24.2/30	(67.21)	151.899	80.937	83.071

Table 6.4: Computational results of TS/MA for Palubeckis instances

Instance	*BKR*	TS/MA					
		Best	*Succ.*	*Avg.*	σ	T_{best}	$T_{avg.}$
p3000-1	6502308	***6502330(-22)***	6/15	6502282.80(25.20)	39.98	247.92	332.45
p3000-2	18272568	**18272568(0)**	14/15	18272560.87(7.13)	26.69	114.72	129.23
p3000-3	29867138	**29867138(0)**	15/15	29867138.00(0.00)	0.00	78.64	78.64
p3000-4	46915044	**46915044(0)**	15/15	46915044.00(0.00)	0.00	214.77	214.77
p3000-5	58095467	**58095467(0)**	14/15	58095448.47(18.53)	69.35	136.47	131.38
p5000-1	17509215	***17509369(-154)***	13/15	17509325.80(-110.8)	116.74	1034.80	1012.41
p5000-2	50102729	***50103071(-342)***	7/15	50103060.07(-331.07)	12.49	840.94	1003.51
p5000-3	82039686	***82040316(-630)***	3/15	82040150.13(-464.13)	85.02	1239.04	1228.27
p5000-4	129413112	***129413710(-598)***	3/15	129413543.13(-431.13)	138.10	1164.76	1002.71
p5000-5	160597781	***160598156(-375)***	3/15	160598009.87(-228.87)	103.71	1434.65	1045.47
Av.		***(-212.1)***	9.6/15	(151.51)	59.208	650.671	617.884

6.3.3 Computational results for TS/MA

Tables 6.2, 6.3 and 6.4 respectively show the computational statistics of the TS/MA algorithm on the 20 Type1_22 instances, 10 ORLIB instances and 10 Palubeckis instances. In each table, columns 1 and 2 give the instance names (*Instance*) and the best known results (*BKR*) reported in the literature [Brimberg J, 2009; Lozano *et al.*, 2011; Palubeckis, 2007; Wang *et al.*, 2012a]. The columns under the heading TS/MA report the best solution values (*Best*) along with the gap of *Best* to *BKR* shown in parenthesis (*BKR-Best*), the success rate (*Succ.*) for reaching *Best*, the average solution values (*Avg.*) along with the gap of *Avg.* to *BKR* shown in parenthesis (*BKR-Avg.*), the standard deviation (σ), the average time (T_{best}) to reach *Best* and the average time ($T_{avg.}$) to reach the solution value at the end of each run (in seconds). Results marked in bold indicate that TS/MA matches *BKR* and if also marked in italic indicate that TS/MA improves *BKR*. Furthermore, the last row *Av.* summarizes TS/MA's average performance over the whole set of benchmark instances.

From Tables 6.2, 6.3 and 6.4, we observe that TS/MA can easily reach the best known results for all the tested instances within the given time limit, which none of current state-of-the-art algorithms can compete with. In particular, TS/MA improves the best known results for 6 Palubeckis instances and even its average quality is better than the best known results previously reported in the literature.

6.3.4 Comparison with state-of-the-art algorithms

In order to further evaluate our TS/MA algorithm, we compare it with four best performing algorithms recently proposed in the literature. These reference methods are Iterated Tabu Search (ITS) [Palubeckis, 2007], Variable Neighborhood Search (VNS) [Brimberg J, 2009], Tuned Iterated Greedy (TIG) [Lozano *et al.*, 2011] and Learnable Tabu Search with Estimation of Distribution Algorithm (LTS-EDA) [Wang *et al.*, 2012a]. The results of these reference algorithms are directly extracted from [Wang *et al.*, 2012a].

Tables 6.5, 6.6 and 6.7 display the best and average solution values obtained by ITS, VNS, TIG, LTS-EDA and our TS/MA algorithm. Since the absolute solution values are very large, we report the gap of best and average solution values to the best known results.

Table 6.5: Comparison among TS/MA and other state-of-the-art algorithms for Type1_22 instances

Instance	*BKR*	ITS[2007]		VNS[2009]		TIG[2011]		LTS-EDA[2012]		TS/MA	
		Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
Type1_22.1	114271	65	209.87	48	150.60	48	101.57	5	60.73	**0**	**5.97**
Type1_22.2	114327	29	262.27	**0**	168.87	**0**	69.90	**0**	89.87	**0**	**8.40**
Type1_22.3	114195	69	201.40	19	110.83	5	117.77	**0**	98.97	**0**	**11.23**
Type1_22.4	114093	22	200.53	70	188.13	58	141.93	**0**	79.87	**0**	**16.87**
Type1_22.5	114196	95	273.27	87	184.10	99	194.70	51	134.47	**0**	**31.37**
Type1_22.6	114265	41	168.17	30	99.30	9	96.20	**0**	40.17	**0**	**16.43**
Type1_22.7	114361	12	167.47	**0**	56.30	**0**	71.27	**0**	18.20	**0**	**0.00**
Type1_22.8	114327	25	256.40	**0**	163.33	**0**	193.60	**0**	159.10	**0**	**13.13**
Type1_22.9	114199	9	139.83	16	78.47	16	80.37	**0**	70.97	**0**	**8.00**
Type1_22.10	114229	24	204.93	7	139.33	35	121.43	**0**	56.20	**0**	**1.73**
Type1_22.11	114214	74	237.77	42	145.13	59	139.57	3	69.87	**0**	**22.43**
Type1_22.12	114214	55	249.53	95	143.30	88	156.00	15	84.93	**0**	**24.50**
Type1_22.13	114233	93	279.87	22	168.07	42	167.40	6	85.30	**0**	**2.67**
Type1_22.14	114216	92	248.50	117	194.30	64	202.80	**0**	81.00	**0**	**1.30**
Type1_22.15	114240	11	117.50	1	62.87	6	80.53	**0**	22.03	**0**	**0.73**
Type1_22.16	114335	11	225.40	42	215.43	35	67.90	**0**	36.47	**0**	**3.60**
Type1_22.17	114255	56	217.53	**0**	170.00	18	144.53	6	57.07	**0**	**9.80**
Type1_22.18	114408	46	169.97	**0**	57.10	2	117.37	2	22.83	**0**	**1.07**
Type1_22.19	114201	34	243.20	**0**	124.60	**0**	144.37	**0**	35.87	**0**	**4.53**
Type1_22.20	114349	151	270.67	65	159.43	45	187.23	**0**	95.40	**0**	**7.63**
Av.		50.7	217.204	33.05	138.97	31.45	129.82	4.4	69.97	**0**	**9.57**

Table 6.6: Comparison among TS/MA and other state-of-the-art algorithms for ORLIB instances

Instance	*BKR*	ITS[2007]		VNS[2009]		TIG[2011]		LTS-EDA[2012]		TS/MA	
		Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
b2500-1	1153068	624	3677.33	96	1911.93	42	1960.33	**0**	369.20	**0**	**84.80**
b2500-2	1129310	128	3677.33	88	1034.33	1096	1958.47	154	453.53	**0**	**85.13**
b2500-3	1115538	316	3281.93	332	1503.67	34	2647.87	**0**	290.40	**0**	**138.40**
b2500-4	1147840	870	2547.93	436	1521.07	910	1937.13	**0**	461.73	**0**	**138.67**
b2500-5	1144756	356	1800.27	**0**	749.40	674	1655.87	**0**	286.07	**0**	**53.13**
b2500-6	1133572	250	2173.47	**0**	1283.53	964	1807.60	80	218.00	**0**	**54.40**
b2500-7	1149064	306	1512.60	116	775.47	76	1338.73	44	264.60	**0**	**86.87**
b2500-8	1142762	**0**	247.73	96	862.47	588	1421.53	22	146.47	**0**	**30.67**
b2500-9	1138866	642	2944.67	54	837.07	658	1020.60	6	206.33	**0**	**0.00**
b2500-10	1153936	598	2024.60	278	1069.40	448	1808.73	94	305.27	**0**	**0.00**
Av.		409	2388.79	149.6	1154.83	549	1755.69	40	300.16	**0**	**67.21**

Table 6.7: Comparison among TS/MA and other state-of-the-art algorithms for Palubeckis instances

Instance	*BKR*	ITS[2007]		VNS[2009]		TIG[2011]		LTS-EDA[2012]		TS/MA	
		Best	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*	*Best*	*Avg.*
p3000-1	6502308	466	1487.53	273	909.80	136	714.67	96	294.07	**-22**	**25.20**
p3000-2	18272568	**0**	1321.60	**0**	924.20	**0**	991.07	140	387.00	**0**	**7.13**
p3000-3	29867138	1442	2214.73	328	963.53	820	1166.13	**0**	304.33	**0**	**0.00**
p3000-4	46915044	1311	2243.93	254	1068.47	426	2482.20	130	317.07	**0**	**0.00**
p3000-5	58095467	423	1521.60	**0**	663.00	278	1353.27	**0**	370.40	**0**	**18.53**
p5000-1	17509215	2200	3564.93	1002	1971.27	1154	2545.80	191	571.00	**-154**	**-110.8**
p5000-2	50102729	2910	4786.80	1478	2619.00	528	2511.73	526	892.80	**-342**	**-331.07**
p5000-3	82039686	5452	8242.33	1914	3694.40	2156	6007.13	704	1458.53	**-630**	**-464.13**
p5000-4	129413112	1630	5076.90	1513	2965.90	1696	3874.80	858	1275.20	**-598**	**-431.13**
p5000-5	160597781	2057	4433.90	1191	2278.30	1289	2128.90	579	1017.90	**-375**	**-228.87**
Av.		1789.1	3489.43	795.3	1805.79	848.3	2377.57	322.4	688.83	**-212.1**	**-151.51**

Table 6.8: TS/MA versus ITS, VNS, TIG and LTS-EDA (Wilcoxon's test at the 0.05 level)

	Type1_22		ORLIB		Palubeckis	
	p-value	Diff.?	p-value	Diff.?	p-value	Diff.?
ITS	1.91e-06	Yes	0.002	Yes	0.002	Yes
VNS	1.91e-06	Yes	0.002	Yes	0.002	Yes
TIG	1.91e-06	Yes	0.002	Yes	0.002	Yes
LTS-EDA	1.91e-06	Yes	0.002	Yes	0.002	Yes

Table 6.9: Post-hoc test for solution sets obtained by varying p

$p =$	5	10	20	30	40
10	0.00057				
20	0.00665	0.98856			
30	0.00954	0.97713	1.00000		
40	0.67881	0.08822	0.33919	0.40233	
50	0.99509	0.00447	0.03686	0.05000	0.93341

Smaller gaps indicate better performances. Negative gaps represent improved results. The best performances among the 5 compared algorithms are highlighted in bold. In addition, the averaged results over the whole set of instances are presented in the last row.

As we can observe from Tables 6.5, 6.6 and 6.7, our TS/MA algorithm outperforms the four reference algorithms in terms of both the best and average solution values. Specifically, TS/MA is able to match or surpass the best known results for all the 40 instances, while ITS, VNS TIG and LTS-EDA can only match for 2, 10, 5 and 19 out of 40 instances, respectively. Furthermore, the average gap to the best known results of TS/MA is much smaller than that of each reference algorithm.

We also conduct nonparametric statistical tests to verify the observed differences between TS/MA and the reference algorithms in terms of solution quality are statistically significant. Table 6.8 summarizes the results by means of the Wilcoxon signed-ranked test, where p-value$<$0.05 indicates that there is significant difference between our TS/MA algorithm and a reference algorithm. We observe that TS/MA is significantly better than all these reference algorithms for each set of benchmark.

In sum, this comparison demonstrates the efficacy of our TS/MA algorithm in attaining the best and average solution values.

Table 6.10: Post-hoc test for solution sets obtained by varying θ

$\theta =$	10	20	30	40
20	0.94381			
30	0.97070	0.99994		
40	0.06421	0.32684	0.26205	
50	0.00410	0.04573	0.03171	0.90493

Table 6.11: Post-hoc test for solution sets obtained by varying tt

$tt =$	[1,15]	[15,25]	[15,50]	[25,50]	[25,100]
[15,25]	0.00039				
[15,50]	0.03906	0.80788			
[25,50]	0.03907	0.80786	1.00000		
[25,100]	0.98891	0.00492	0.19158	0.19122	
[50,100]	0.99959	0.00009	0.01470	0.01478	0.93517

Table 6.12: Post-hoc test for solution sets obtained by varying α

$\alpha =$	m	$2 \cdot$ m	$3 \cdot$ m	$4 \cdot$ m	$5 \cdot$ m	$6 \cdot$ m	$7 \cdot$ m	$8 \cdot$ m	$9 \cdot m$
$2 \cdot m$	0.85330								
$3 \cdot m$	0.40765	0.99961							
$4 \cdot m$	0.18981	0.98742	0.99999						
$5 \cdot m$	0.07000	0.90758	0.99887	1.00000					
$6 \cdot m$	0.00000	0.00091	0.01459	0.05167	0.15025				
$7 \cdot m$	0.09187	0.93831	0.99961	1.00000	1.00000	0.11842			
$8 \cdot m$	0.56656	0.99999	1.00000	0.99983	0.99200	0.00640	0.99624		
$9 \cdot m$	0.00238	0.30550	0.76509	0.93839	0.99371	0.74514	0.98743	0.61304	
$10 \cdot m$	0.00033	0.10856	0.45123	0.72441	0.91867	0.94677	0.88215	0.30514	0.99999

Table 6.13: Post-hoc test for solution sets obtained by varying cls

$cls =$	$m^{0.1}$	$m^{0.2}$	$m^{0.3}$	$m^{0.4}$	$m^{0.5}$	$m^{0.6}$	$m^{0.7}$	$m^{0.8}$	$m^{0.9}$
$m^{0.2}$	0.54620								
$m^{0.3}$	0.00009	0.20682							
$m^{0.4}$	0.00000	0.00069	0.80485						
$m^{0.5}$	0.00000	0.00001	0.27148	0.99846					
$m^{0.6}$	0.00000	0.00400	0.96156	0.99999	0.96747				
$m^{0.7}$	0.00006	0.16582	1.00000	0.85461	0.32774	0.97735			
$m^{0.8}$	0.02148	0.93895	0.96158	0.09351	0.00663	0.25356	0.93893		
$m^{0.9}$	0.36772	1.00000	0.34752	0.00210	0.00003	0.01062	0.28899	0.98470	
$m^{1.0}$	0.99005	0.99004	0.00952	0.00000	0.00000	0.00003	0.00676	0.32694	0.95488

Table 6.14: Post-hoc test for solution sets obtained by varying β

$\beta =$	$0.1 \cdot m$	$0.2 \cdot m$	$0.3 \cdot m$	$0.4 \cdot m$	$0.5 \cdot m$	$0.6 \cdot m$	$0.7 \cdot m$	$0.8 \cdot m$	$0.9 \cdot m$
$0.2 \cdot m$	1.00000								
$0.3 \cdot m$	0.89398	0.91762							
$0.4 \cdot m$	0.40401	0.44709	0.99885						
$0.5 \cdot m$	0.08163	0.09732	0.89377	0.99942					
$0.6 \cdot m$	0.02051	0.02487	0.63285	0.97663	0.99999				
$0.7 \cdot m$	0.00309	0.00376	0.28305	0.80029	0.99359	1.00000			
$0.8 \cdot m$	0.93734	0.95360	1.00000	0.99617	0.83526	0.54017	0.21556		
$0.9 \cdot m$	0.99999	1.00000	0.95358	0.53992	0.13699	0.03792	0.00602	0.97662	
$1.0 \cdot m$	1.00000	1.00000	0.93739	0.49383	0.11605	0.03077	0.00497	0.96654	1.00000

6.4 Analysis

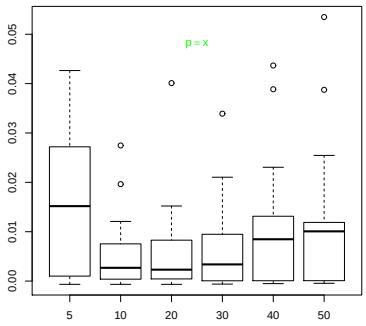
6.4.1 Parameter sensitivity analysis

We first show a parameter sensitivity analysis based on a subset of 11 instances. For each TS/MA parameter, we test a number of possible values while fixing the other parameters to their default values from Table 6.1. We test p (population size) in the range [5, 50], θ (population rebuilding threshold) in the range [10, 50], ρ (tabu search perturbation fraction) in the range [0.1, 1.0], α (tabu search improvement cutoff) in the range $[m, 10 \cdot m]$, cls (candidate list size) in the range $[m^{0.1}, m^{1.0}]$ and β (number of strongly determined variables) in the range $[0.1 \cdot m, m]$. Similarly, for the tabu tenure tt, we try several intervals in the range [1,100]. For each instance and each parameter setting, we conduct experiments under exactly the same conditions as before.

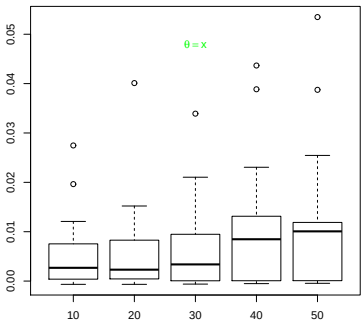
We use the Friedman test to see whether the performance of TS/MA varies significantly in terms of its average solution values when we vary the value of a single parameter as mentioned above. The Friedman test indicates that the values of ρ do not significantly affect the performance of TS/MA (with p-value=0.2983). This means that TS/MA is not very sensitive to the perturbation fraction when rebuilding the population. However, the Friedman test reveals a statistical difference in performance to the different settings of parameters p, θ, tt, α, cls and β (with p-values of 0.000509, 0.004088, 0.0001017, 1.281e-07, 1.735e-11 and 0.002715, respectively). Hence, we perform the Post-hoc test to examine the statistical difference between each pair of settings of these parameters and show the results in Tables 6.9 to 6.14. We observe that although certain pairs of settings present significant differences (with p-value$<$0.05), there does not exist a determined setting for each parameter that is significantly better than all the other settings.

To further investigate the performance of TS/MA with different settings for each parameter, we show in Figure 6.1 the box and whisker plots which depict the smallest result, lower quartile, median, upper quartile, and the largest result obtained with each parameter value. For the sake of clarity, these results are displayed as the percentage deviation of the average results from the best-known results reported in the literature, computed as $\frac{BKR - Avg.}{BKR} \cdot 100\%$.

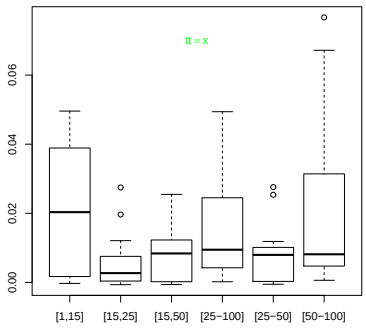
From the box and whisker plots in Figure 6.1, we obtain the following observations. First, setting $p \in \{10, 20, 30\}$, $\theta \in \{10, 20, 30\}$, $tt \in [15, 25]$, $\alpha \in \{6 \cdot m, 10 \cdot m\}$, $cls \in \{m^{0.4}, m^{0.5}\}$, $\beta \in \{0.6 \cdot m, 0.7 \cdot m\}$ seems preferable in terms of both the solution quality and the variation of solution values. This observation also demonstrates the appropriateness of the settings of parameters in Table 6.1. Second, varying values of the parameter cls, i.e., candidate list size of the swap-based neighborhood mostly affects the performance of the TS/MA algorithm, with deviations ranging from [0,0.5%] against deviations ranging from [0,0.05%] with other parameters. Third, the performance of TS/MA is less sensitive to the population rebuilding threshold (θ) than to other parameters with deviations less than 0.03% for each setting.



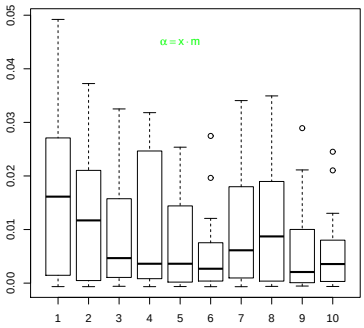
(a) varying population size p



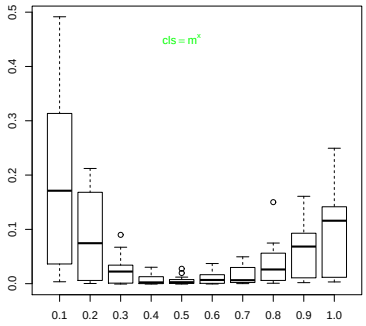
(b) varying population rebuilding threshold θ



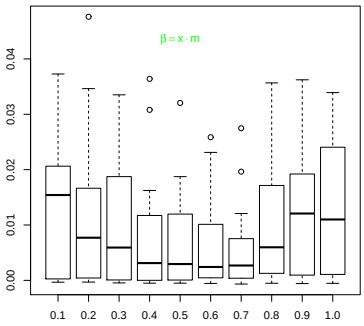
(c) varying tabu tenure tt



(d) varying tabu search improvement cutoff α



(e) varying candidate list size of each component operation cls



(f) varying number of strongly determined variables β

Figure 6.1: Box and whisker plot of the results obtained with different settings for each sensitive parameter

6.4.2 Tabu search analysis

In this section, we provide experiments to demonstrate the successive filter candidate list strategy implemented in our tabu search procedure, denoted as *FastBestImp*, plays an important role to the performance of the TS/MA algorithm. For this purpose, we test the following three other tabu search procedures within our TS/MA algorithm.

Successive 1-flip based tabu search (*1-flip*): This approach starts from an initial feasible solution x and at each iteration first picks a variable x_i from Z such that flipping x_i to the value of 1 would increase the objective function value of the current solution x by the greatest amount. Next, given the selected first flip, we pick a variable x_j from U such that flipping x_j to the value of 0 creates the least loss in the objective function value of x. These two successive 1-flip moves assure the resulting solution is always feasible with $|U| = m$. In addition, each time a variable is flipped, a tabu tenure is assigned to the variable to prevent it from being flipped again for the next A iterations (where A is drawn randomly from the interval tt; see Table 6.1). Finally, a move leading to a new solution better than the best solution found so far is always selected even if it is classified tabu. The above procedure repeats until the solution cannot be improved for consecutive $m/4$ iterations. Additional details can be found in [Lü *et al.*, 2010a; Wang *et al.*, 2012a].

First Improvement based tabu search (*FirstImp*): Starting from an initial feasible solution, each iteration sequentially fetches a variable x_i from U and then scans each variable x_j from Z. If swapping x_i and x_j improves the current solution, then we perform this move to obtain a new solution. If there is no improved move by interchanging the unit-value of x_i with the zero-value of any variable from Z, we fetch the next variable from U and so on. If no improved move is found by interchanging each variable from U and each variable from Z, the best move among them (which does not improve the current solution) is then performed. The selected variables x_i and x_j become tabu active and thus neither can be involved in a new move during the next B iterations (where B is drawn randomly from tt; see Table 6.1). However, if a move improves the best solution found so far, it is always performed even if it is tabu active. The method continues until the best solution found so far cannot be improved for α consecutive iterations (see Table 6.1).

Best Improvement based tabu search (*BestImp*): The only difference between *BestImp* and our *FastBestImp* approach is that *BestImp* identifies a best neighborhood solution within the complete swap neighborhood, without employing the successive filter candidate list strategy described in Section 6.2.4. Several algorithms in the literature (e.g.,[Aringhieri and Cordone, 2011; Ghosh, 1996; Palubeckis, 2007]) are based on *BestImp*.

We carry out experiments for the TS/MA algorithm with *FastBestImp* replaced by *1-flip*, *FirstImp* and *BestImp* under the same experimental conditions (see Section 6.3.2) and test all the 40 instances (see Section 6.3.1). The experimental results are shown in Figure 6.2, in which the left portion and the right portion respectively present the best gap and the average gap, for each tested instance, to the best known result.

As shown in the left portion of Figure 6.2, *FastBestImp* achieves the best performance with a smaller gap between the best solution value and the best known result than *1-flip*, *FirstImp* and *BestImp* for each instance, except for several Type1_22 instances where

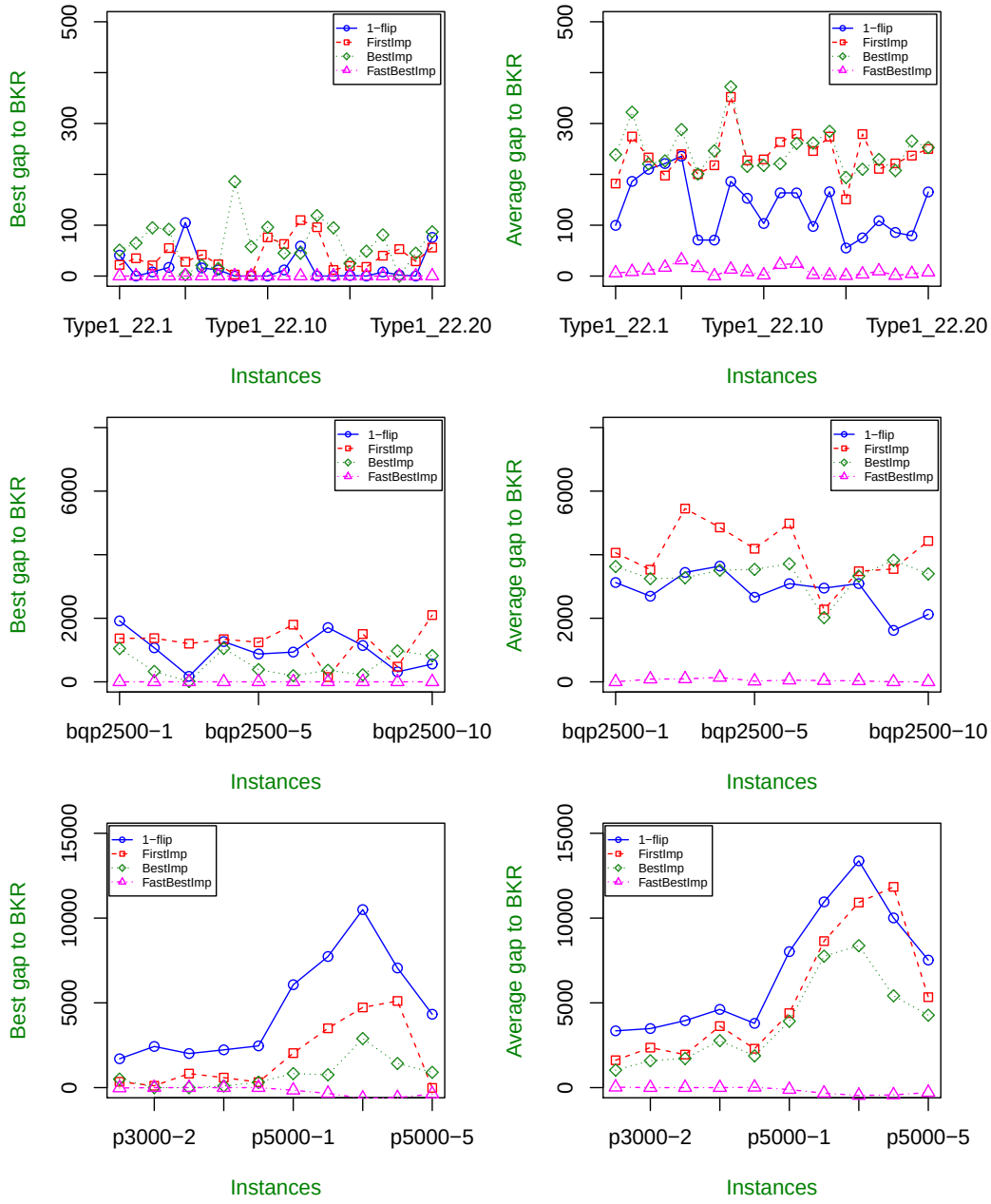


Figure 6.2: Best and average solution gaps to the best known result for 3 sets of benchmark instances

Table 6.15: TS/MA$_{cx}$ versus TS/MA$_{ux}$ using Wilcoxon's test (at the 0.05 level)

Problem	R+	R-	p-value	Diff.?	TS/MA$_{cx}$		TS/MA$_{ux}$	
					AD-B	AD-Av	AD-B	AD-Av
	TS/MA$_{cx}$	TS/MA$_{ux}$						
Type1_22	190	0	0.000143	Yes	0	9.57	0.40	27.38
ORLIB	55	0	0.001953	Yes	0	67.21	0	267.39
Palubeckis	55	0	0.001953	Yes	-212.10	-151.51	-194.50	38.48

both *FastBestImp* and *1-flip* can reach the best known results. In addition, *1-flip* basically outperforms *FirstImp* and *BestImp* for the Type1_22 instances while *BestImp* outperforms *1-flip* and *FirstImp* for the ORLIB and Palubeckis instances.

When it comes to the average gap to the best known result, the right portion of Figure 6.2 clearly shows that once again *FastBestImp* achieves the best performance among the compared strategies for all the tested instances. In addition, the comparison among *1-flip*, *FirstImp* and *BestImp* indicates that *1-flip* generally performs better for the Type1_22 and ORLIB instances while *BestImp* performs better for the Palubeckis instances.

6.4.3 Solution combination operator analysis

In order to assess the role of the operator described in Section in 6.2.5 for combining solutions, we conduct additional experiments to compare it with a traditional uniform crossover operator for combining solutions [Syswerda, 1989]. For MDP, uniform crossover consists in identifying variables that have the value of 1 in both parents and keeping this value unchanged for these variables in the offspring solution. Then the remaining variables are randomly assigned the value 0 or 1 subject to the cardinality constraint, i.e., the total number of variables with the value of 1 equals m in the offspring solution.

We compare this modified TS/MA algorithm with the uniform crossover, denoted by TS/MA$_{ux}$, and the original TS/MA with the critical variable solution combination operator, denoted by TS/MA$_{cx}$ under the same experimental conditions (see Section 6.3.2). In order to detect the difference between TS/MA$_{ux}$ and TS/MA$_{cx}$, we also conduct the Wilcoxon nonparametric statistical test and summarize the results in Table 6.15. In this table, columns 2 to 5 report the results from the Wilcoxon test in terms of the average quality. Column AD-B reports the average gap over each set of benchmark instances of the best solution value to the best known result. Column AD-AV reports the average gap over each set of benchmark instances of the average solution values to the best known results.

The following observations can be made from Table 6.15. First, the results from the Wilcoxon test indicate that TS/MA$_{cx}$ is significantly better than TS/MA$_{ux}$ for each set of benchmark instances. Second, in terms of AD-B, TS/MA$_{cx}$ performs better than TS/MA$_{ux}$ for both Type1_22 (0 for TS/MA$_{cx}$ versus 0.40 for TS/MA$_{ux}$) and Palubeckis benchmarks (-212.1 for TS/MA$_{cx}$ versus -194.50 for TS/MA$_{ux}$). TS/MA$_{cx}$ performs the same as TS/MA$_{ux}$ for the ORLIB benchmark considering that both can reach the best known results for each instance. Notice that although inferior to TS/MA$_{cx}$, TS/MA$_{ux}$ is still able to improve the best known results over the Palubeckis benchmark. Finally, in terms of AD-Av, TS/MA$_{cx}$ always outperforms TS/MA$_{ux}$.

6.5 Conclusion

In this chapter, we have proposed an effective memetic algorithm for the maximum diversity problem (a special case of BQO with cardinality constraint) based on tabu search. The tabu search component utilizes successive filter candidate list strategy and is joined with a solution combination strategy based on identifying strongly determined and consistent variables.

Computational experiments on three sets of 40 popular benchmark instances have demonstrated that the proposed TS/MA algorithm is capable of easily attaining all the previous best known results and improving the best known results for 6 instances. Moreover, statistical tests have confirmed that our proposed algorithm performs significantly better than several recently proposed state-of-the-art algorithms.

Finally, in addition to a parameter sensitivity analysis, we have studied the effects of the dedicated tabu search procedure based on the swap move combined with the successive filter candidate list strategy and the specific combination operator based on the concept of strongly determined and consistent variables. These studies have confirmed the importance of these two key components for the high performance of the proposed algorithm.

General Conclusion

Conclusions

In this thesis, we developed several effective algorithms for solving the binary quadratic optimization problem. In addition, we tackled several other combinatorial optimization problems either by directly applying or with a slight adaptation of one or multiple algorithms proposed for BQO, with the premise that these problems are transformed into the form of BQO. Finally, we proposed a highly effective algorithm for a cardinality constrained binary quadratic optimization problem.

The backbone guided tabu search algorithms and the backbone multilevel memetic algorithm have been proposed for dealing with large problem instances, based on the idea of decreasing the scale of initial problem so as to carry out the exploitation in a small but promising search area. Specifically, BGTS relies on the variable fixation technique to fix backbone variables at specific values, which are forbidded being changed during the successive optimization phase. The BMMA applies a multilevel framework to solve the original problem level by level. We evaluated BGTS on 21 challenging instances from the Palubeckis benchmark and BMMA on 11 largest instances from the same benchmark. Tests on BGTS algorithms indicated that one of the proposed BGTS algorithms obtained highly competitive outcomes in comparison with the best known results and boosted the performance of the basic TS algorithm. Tests on the BMMA algorithm showed that BMMA matched the best known results for all the examined instances while using much less computing time than BGTS as well as state-of-the-art algorithms from the literature.

The GRASP-Tabu Search and GRASP-Tabu Search with Population Management algorithms have been proposed, placing an emphasis on constructing initial solutions in order to locate at a hopeful search area. GRASP-TS constructs an initial solution by reference to a random greedy adaptive function. GRASP-TS/PM employs a destructive/constructive process that dismantles only part of a previously visited elite solution and rebuilds the remaining portion as GRASP-TS does. In addition, we have developed a GRASP-TS/MCPs algorithm to solve the maximum clique and maximum vertex weight clique problems in the form of BQO, which extended GRASP-TS by making use of the neighborhood union (*1-flip* and *2-flip*) instead of *1-flip* in the tabu search procedure. Although both GRASP-TS and GRASP-TS/PM are used for solving BQO, we also directly applied them without particular adaptation to address the maximum cut problem that is transformed into the BQO formulation. Experiments conducted with both GRASP-TS and GRASP-TS/PM algorithms on two sets of 31 BQO instances and one set of 54 MaxCut instances have demonstrated that both GRASP-TS and GRASP-TS/PM algorithms obtained highly competitive results in comparison with the best known results from the literature. In particular, for the 54 MaxCut instances, GRASP-TS/PM improved the previously best known results for 19 instances. Experiments conducted with GRASP-TS/MCPs on a total of 160 MCP and MVWCP benchmark instances have shown that

GRASP-TS/MCPs are competitive with the leading methods that are specifically tailored for the MCP and MVWCP problems. In particular, out of the 80 MVWCP benchmark instances our method matches the best known results on 66 instances, and finds new best known results on 13 instances, while accomplishing this with very short solution times.

We have also devised two path relinking algorithms that generate initial solutions by exploiting within the neighborhood space a path that connects an initiating solution with a guiding solution, where each step of the path (representing a solution) has the distance to the guiding solution reduced by 1. The proposed algorithms differ from each other mainly on the way they generate the path, one employing a greedy strategy (PR1) and the other employing a random strategy (PR2). Besides, we transformed the minimum sum coloring problem and tackled for the first time the possibility of solving MSCP via BQO, which we denote as BQO-PR approach. Extensive experiments with PR1 and PR2 on five sets of BQO and MaxCut benchmarks with a total of 134 instances have demonstrated that both algorithms are capable of attaining highly competitive results. In particular, for 103 MaxCut instances, our algorithms improved the best known results for almost 40 percent of these instances. Experiments on 23 MSCP instances have shown that the proposed BQO-PR approach is able to reach competitive solutions when compared with several special purpose MSCP algorithms for a number of instances. However, due to the limitation of the current implementation, the BQO approach for the MSCP requires considerable computing time to find solutions of good quality for large instances.

Finally, we have developed a highly effective memetic algorithm based on tabu search for the cardinality constrained binary quadratic optimization problem, in which the tabu search component utilizes successive filter candidate list strategy and the solution combination component is based on identifying strongly determined and consistent variables. Experiments on 40 popular benchmark instances have shown that the proposed TS/MA algorithm have achieved all the best known results (which no previous algorithm has achieved) and improved the best known results for 6 instances. Besides a parameter sensitivity analysis, we have studied the effects of the dedicated tabu search procedure and the specific combination operator and confirmed the importance of these two key components for the high performance of the proposed algorithm.

Perspectives

The following aspects can be considered in the future study. For the BGTS algorithm, the fixing phase identifies backbone variables by drawing on the idea of exploiting strongly determined variables. We can also include consideration of consistent variables by reference to the frequency that variables receive assigned values in high quality solutions. In addition, our current BGTS implementation forbids a backbone variable changing its assigned value in the follow-up tabu search phase unless a freeing phase enables it to become a non-backbone variable. An interesting alternative strategy is to allow a backbone variable to change its assigned value when some specific condition is verified. For example, such a condition could be that the change of the assigned value of a certain backbone variable leads to a solution that is better than the best solution found so far.

The preservation of population diversification in our population based algorithms (e.g., GRASP-TS/PM, PR and TS/MA) depends on the rebuilding of the population when the search stagnates. This is a quite rudimentary strategy, thus more advanced strategies are worthy of investigation. The first consideration consists in incorporating a distance threshold that designates a minimum distance of one solution to the population, requiring that a solution is qualified to be a member of the population only if it passes the specified distance threshold. Another alternative is to define a fitness function in terms of both solution quality and distance to evaluate solutions. Besides, one can elaborate two populations where one features good solution quality and the other features good solution diversity, in order that a couple of solutions used to produce offspring solutions can be chosen either both from the high-quality population or from the diverse population or selected with one from each population.

The solution selection method that determines how to select among solutions generated on the path is a critical component in our path relinking algorithms. The current method chooses the one with the best objective function value subject to a predetermined distance of the chosen solution from both the initiating solution and the guiding solution. For the future study, we can select a solution by reference to the quality of its best neighbor solution or we could also take into account the quality of its second and third best neighbor solutions. (A weighted quality measure involving these three solutions would include the rule of picking only the best neighbor solution by setting the weights of the other two solutions equal to a "small epsilon" value to break ties.)

Inspired by the fact that the search space of the cadinality constrained BQO is smaller than that of BQO as a result of requiring that the number of "1" equals to the specified cardinality K, we intend to address BQO through solving a sequence of cardinality canstrained BQO problems with different K values, taking advantage of conducting extensive exploitation in a constrained search space. Since it is not practical to solve cardinality constrained BQO with K fetching from 0 to the maximum number of variables of BQO, a related issue is how to pick out some effective K values. A possible choice would refer to the number of "1" in a set of local optimal solutions obtained by using an algorithm for BQO.

List of figures

List of tables

List of algorithms

List of publications

Published/accepted journal papers

- Yang Wang, Zhipeng Lü, Fred Glover and Jin-Kao Hao, Path relinking for unconstrained binary quadratic programming. European Journal of Operational Research 223:595-604, 2012.
- Yang Wang, Zhipeng Lü, Fred Glover and Jin-Kao Hao, Probabilistic GRASP-Tabu Search algorithms for the UBQP problem. Computers & Operations Research, Doi:10.1016/j.cor.2011.12.006.
- Yang Wang, Zhipeng Lü, Fred Glover and Jin-Kao Hao, Backbone guided tabu search for solving the UBQP problem problem. Journal of Heuristics, Doi: 10.1007/s10732-011-9164-4.

Published conference papers

- Yang Wang, Zhipeng Lü, Fred Glover and Jin-Kao Hao, A Multilevel Algorithm for Large Unconstrained Binary Quadratic Optimization. In N. Beldiceanu, N. Jussien and E. Pinson (Eds.): CPAIOR 2012, Lecture Notes in Computer Science 7298: 395-408, 2012.
- Yang Wang, Zhipeng Lü, Fred Glover and Jin-Kao Hao, Effective Variable Fixing and Scoring Strategies for Binary Quadratic Programming. In P. Merz and J.K. Hao. (Eds.): EvoCOP 2011, Lecture Notes in Computer Science 6622: 72-83, 2011
- Yang Wang, Zhipeng Lü and Jin-Kao Hao, A Study of Multi-Parent Crossover Operator in a Memetic Algorithm. In R. Schaerfer et al. (Eds.): PPSN 2010, Lecture Notes in Computer Science 6238: 556-565, 2010

Submitted journal papers

- Yang Wang, Jin-Kao Hao, Fred Glover and Zhipeng Lü, Solving the maximum clique and maximum vertex weight clique problems via binary quadratic programming. Submitted to Discrete Optimization (March 2012).

- Yang Wang, Jin-Kao Hao, Fred Glover and Zhipeng Lü, A tabu search based memetic algorithm for the maximum diversity problem. Submitted to Computers & Operations Research (Sept. 2012).
- Yang Wang, Jin-Kao Hao, Fred Glover and Zhipeng Lü, Solving the minimum sum coloring problem via binary quadratic programming. Submitted to Optimization (Dec. 2012).

References

[Aarts *et al.*, 1988] cited page 13
E.H.L. Aarts, J. Korst, and P.V. Laarhoven. A quantitive analysis of the simulated annealing algorithm: A case study for the travelling salesman problem. *Journal of Statistical Physics*, 50:187–206, 1988.

[Alidaee *et al.*, 1994] cited page 7
B. Alidaee, G. Kochenberger, and A. Ahmadian. 0-1 quadratic programming approach for optimum solutions of two scheduling problems. *International Journal of Systems Science*, 25(2):401–408, 1994.

[Alidaee *et al.*, 2007] cited page 7
B. Alidaee, F. Glover, G. Kochenberger, and H. Wang. Solving the maximum edge weight clique problem via unconstrained quadratic programming. *European Journal of Operational Research*, 181:592–597, 2007.

[Alidaee *et al.*, 2008] cited page 7
B. Alidaee, G. Kochenberger, K. Lewis, M. Lewis, and H. Wang. A new approach for modeling and solving set packing problem. *European Journal of Operational Research*, 86(2):504–512, 2008.

[Alkhamis *et al.*, 1998] cited page 13
T.M. Alkhamis, M. Hasan, and M.A. Ahmed. Simulated annealing for the unconstrained quadratic pseudo-boolean function. *European Journal of Operational Research*, 108:641–652, 1998.

[Amini *et al.*, 1999] cited page 17
M.M. Amini, B. Alidaee, and G. Kochenberger. A scatter search approach to unconstrained quadratic binary programs. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 317–329. McGraw-Hill, 1999.

[Andrade *et al.*, 2003] cited page 98
P.M.F. Andrade, A. Plastino, L.S. Ochi, and S.L. Martins. Grasp for the maximum diversity problem. In *Proceedings of the Fifth Metaheuristics International Conference (MIC2003)*, 2003.

[Andrade *et al.*, 2005] cited page 98
M.R.Q. Andrade, P.M.F. Andrade, S.L. Martins, and A. Plastino. Grasp with path-relinking for the maximum diversity problem. In S. Nikoletseas, editor, *Experimental and Efficient Algorithms*, volume 3503, pages 558–569, 2005.

[Aringhieri and Cordone, 2011] cited page 98, 112
R. Aringhieri and R. Cordone. Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society*, 62(2):266–280, 2011.

[Aringhieri *et al.*, 2008] cited page 98
R. Aringhieri, R. Cordone, and Y. Melzani. Tabu search versus grasp for the maximum diversity problem. *4OR*, 6(1):45–60, 2008.

[Babel, 1994] cited page 53
L. Babel. A fast algorithm for the maximum weight clique problem. *Computing*, 52(1):31–38, 1994.

[Ballard and Brown, 1983] cited page 53
D. Ballard and C. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, 1983.

[Balus and Yu, 1986] cited page 53
E. Balus and C. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal of Computing*, 15(4):1054–1068, 1986.

[Bar-Noy and Kortsarz, 1998] cited page 88
A. Bar-Noy and G. Kortsarz. Minimum color sum of bipartite graphs. *Journal of Algorithm*, 28(2):339–365, 1998.

[Bar-Noy *et al.*, 1998] cited page 88
A. Bar-Noy, M. Bellareb, M.M. Halldorsson, H. Shachnai, and T. Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

[Barahona *et al.*, 1988] cited page 7
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.

[Barahona, 1986] cited page 6
F. Barahona. A solvable case of quadratic 0-1 programming. *Discrete Applied Mathematics*, 13:23–26, 1986.

[Battiti and Mascia, 2010] cited page 53
R. Battiti and F. Mascia. Reactive and dynamic local search for the max-clique problem:engineering effective building blocks. *Computers & Operations Research*, 37:534–542, 2010.

[Battiti and Protasi, 2001] cited page 53
R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. *Algorithmica*, 29(4):610–637, 2001.

[Beasley, 1996] cited page 18
J.E. Beasley. Obtaining test problems via internet. *Journal of Global Optimization*, 8:429–433, 1996.

[Beasley, 1998] cited page 5, 13, 14, 18
J.E. Beasley. *Heuristic algorithms for the unconstrained binary quadratic programming problem*. PhD thesis, Imperial College, England, December 1998.

[Benlic and Hao, 2011] cited page 36
U. Benlic and J.K. Hao. A multilevel memetic approach for improving graph k-partitions. *IEEE Transactions on Evolutionary Computation*, 15(5):624–642, 2011.

[Billionnet and Elloumi, 2007] cited page 8
Alain Billionnet and Sourour Elloumi. Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Mathematical Programming*, 109:55–68, 2007.

[Billionnet and Sutter, 1994] cited page 8
Alain Billionnet and A. Sutter. Minimization of a quadratic pseudo-boolean function. *European Journal of Operational Research*, 78:106–115, 1994.

[Bomze *et al.*, 2000] cited page 53
I.M. Bomze, M. Pelillo, and V. Stix. Approximating the maximum weight clique using replicator dynamics. *IEEE Transactions on Neural Networks*, 11:1228–1241, 2000.

[Bonomo *et al.*, 2011] cited page 88
F. Bonomo, G. Duran, J. Marenco, and M.V. Pabon. Minimum sum set coloring of trees and line graphs of trees. *Discrete Applied Mathematics*, 159(5):288–294, 2011.

[Boros *et al.*, 1989] cited page 11, 11
E. Boros, P.L. Hammer, and X. Sun. The ddt method for quadratic 0-1 minimization. Rrr 39-89, RUTCOR Research Center, 1989.

[Boros *et al.*, 2007] cited page 12
E. Boros, P.L. Hammer, and G. Tavares. Local search heuristics for quadratic unconstrained binary optimization (qubo). *Journal of Heuristics*, 13:99–132, 2007.

[Bouziri and Jouini, 2010] cited page 88, 91, 92, 92
H. Bouziri and M. Jouini. A tabu search approach for the sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36(1):915–922, 2010.

[Brimberg J, 2009] cited page 98, 104, 106
Urošević D Ngai E Brimberg J, Mladenović N. Variable neighborhood search for the heaviest-subgraph. *Computers & Operations Research*, 36(11):2885–2891, 2009.

[Burer *et al.*, 2001] cited page 60, 60, 83, 83, 84
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, 2001.

[Busygin, 2006] cited page 53, 53
S. Busygin. A new trust region technique for the maximum weight clique problem. *Discrete Applied Mathematics*, 154:2080–2096, 2006.

[Cai *et al.*, 2011] cited page 18
Y. Cai, J. Wang, J. Yin, and Y. Zhou. Memetic clonal selection algorithm with eda vaccination for unconstrained binary quadratic programming problem. *Expert Systems with Applications*, 38:7817–7827, 2011.

[Chardaire and Sutter, 1994] cited page 7, 8
P. Chardaire and A. Sutter. A decomposition method for quadratic zero-one programming. *Management Science*, 41(4):704–712, 1994.

[Choi and Ye, 2000] cited page 83, 84
C. Choi and Y. Ye. Solving sparse semidefinite programs using the dual scaling algorithm with an iterative solver. Technical report, Department of Management Sciences; The University of Iowa, 2000.

[Douiri and Elbernoussi, 2011] cited page 88, 91, 92, 92
S.M. Douiri and S. Elbernoussi. New algorithm for the sum coloring problem. *International Journal of Contemporary Mathematical Sciences*, 6(10):453–463, 2011.

[Duarte and Marti, 2007] cited page 98, 104
A. Duarte and R. Marti. Tabu search and grasp for the maximum diversity problem. *European Journal of Operational Research*, 178:71–84, 2007.

[Festa *et al.*, 2002] cited page 60, 60, 83, 83
P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the max-cut problem. *Optimization Methods and Software*, 17:1033–1058, 2002.

[Gallego *et al.*, 2009] cited page 98
M. Gallego, A. Duarte, M. Laguna, and R. Martí. Hybrid heuristics for the maximum diversity problem. *Computational Optimization and Applications*, 200(1):36–44, 2009.

[Gallo *et al.*, 1980] cited page 7
G. Gallo, P.L. Hammer, and B. Simeone. Quadratic knapsack problems. *Mathematical Programming Studies*, 12:132–149, 1980.

[Garey and Johnson, 1979] cited page 53
M.R. Garey and D.S. Johnson. *Computers and intractability: a guide to the theory of NP-Completeness*. W. H. Freeman and Company, USA, 1979.

[Geng *et al.*, 2007] cited page 53, 66
X.T. Geng, J. Xu, J.H. Xiao, and L.Q. Pan. A simple simulated annealing algorithm for the maximum clique problem. *Information Sciences*, 177(22):5064–5071, 2007.

[Ghosh, 1996] cited page 112
J.B. Ghosh. Computational aspects of the maximum diversity problem. *Operation Research Letters*, 19:175–181, 1996.

[Glover and Hao, 2010] cited page 10, 24, 56
F. Glover and J.K. Hao. Efficient evaluation for solving 0-1 unconstrained quadratic optimization problems. *International Journal of Metaheuristics*, 1:3–10, 2010.

[Glover and Laguna, 1997] cited page 22, 24, 56, 98, 98, 101, 101, 102
F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, 1997.

[Glover and Woolsey, 1973] cited page 8
F. Glover and R.E.D. Woolsey. Further reduction of zero-one polynomial programs to zero-one linear programming problems. *Operations Research*, 21:156–161, 1973.

[Glover and Woolsey, 1974] cited page 8
F. Glover and R.E.D. Woolsey. Note on converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22:180–181, 1974.

[Glover *et al.*, 1998] cited page 13, 16, 18, 24, 26
F. Glover, G. Kochenberge, and B. Alidaee. Adaptive memory tabu search for binary quadratic programs. *Management Science*, 44:336–345, 1998.

[Glover *et al.*, 2000] cited page 76, 78
F. Glover, M. Laguna, and R. Marti. Fundamentals of scatter search and path-relinking. *Control and Cybernetics*, 39:654–684, 2000.

[Glover *et al.*, 2002] cited page 11, 11
F. Glover, C. Rego, B. Alidaee, and G. Kochenberge. One-pass heuristic for large-scale unconstrained binary quadratic problems. *European Journal of Operational Research*, 137:272–287, 2002.

[Glover *et al.*, 2003] cited page 76, 76, 78
F. Glover, M. Laguna, and R. Marti. Scatter search and path relinking: advances and applications. *Handbook of Metaheuristics*, 57:1–35, 2003.

[Glover *et al.*, 2004] cited page 76, 76, 78
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.

[Glover *et al.*, 2010] cited page 15, 15, 18, 24, 26, 30, 30, 42, 43, 57, 59, 59, 79, 80, 82, 82
F. Glover, Z. Lü, and J.K. Hao. Diversification-driven tabu search for unconstrained binary quadratic problems. *4OR: A Quarterly Journal of Operations Research*, 8:239–253, 2010.

[Glover, 1977] cited page 22, 22, 25, 52, 52, 98, 102
F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[Glover, 1989] cited page 48, 49, 52
F. Glover. Tabu search - part i. *ORSA Journal on Computing*, 1:190–206, 1989.

[Glover, 1994] cited page 99
F. Glover. Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics*, 49:231–255, 1994.

[Glover, 2005] cited page 22
F. Glover. Adaptive memory projection methods for integer programming. In C. Rego and B. Alidaee, editors, *Metaheuristic Optimization Via Memory and Evolution*, pages 425–440. Kluwer Academic Publishers, 2005.

[Grosso *et al.*, 2008] cited page 53
A. Grosso, M. Locatelli, and W. Pullan. Simple ingredients leading to very efficient heuristics for the maximum clique problem. *Journal of Heuristics*, 14:587–612, 2008.

[Hajiabolhassan *et al.*, 2000] cited page 88
H. Hajiabolhassan, M.L. Mebrabadi, and R. Tusserkani. Minimal coloring and strength of graphs. *Discrete Mathematics*, 215(1):265–270, 2000.

[Hanafi *et al.*, 2011] cited page 11
S. Hanafi, A.R. Rebai, and M. Vasquez. Several versions of the devour digest tidy-up heuristic for unconstrained binary quadratic problems. *Journal of Heuristics*, 2011.

[Hansen and Mladenović, 2003] cited page 15
P. Hansen and N. Mladenović. Variable neighborhood search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research Management Science*, pages 145–184. 2003.

[Hansen *et al.*, 2000] cited page 8
P. Hansen, B. Jaumard, and C. Meyer. A simple enumerative algorithm for unconstrained 0-1 quadratic programming. Technical report g-2000-59, les cahiers du gerad, 2000.

[Hao, 2011] cited page 99
J.K. Hao. Memetic algorithms in discrete optimization. In F. Neri, C. Cotta, and P. Moscato, editors, *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*, pages 73–94. 2011.

[Harary, 1953] cited page 7
F. Harary. On the notion of balanced of a signed graph. *Michigan Mathematical Journal*, 2:143–146, 1953.

[Helmar and Chiarandini, 2011] cited page 88, 91, 92
A. Helmar and M. Chiarandini. A local search heuristic for chromatic sum. In *MIC2011: The IX Metaheuristic International Conference*, 2011.

[Helmberg and Rendl, 1998] cited page 8
C. Helmberg and F. Rendl. Solving quadratic (0,1)-problem by semidefinite programs and cutting planes. *Mathematical Programming*, 82:388–399, 1998.

[Jansen, 2000] cited page 88
K. Jansen. Approximation results for the optimum cost chromatic partition problem. *Jounal of Algorithms*, 34(1):54–89, 2000.

[Jha and Pardalos, 1987] cited page 6
S. Jha and P.M. Pardalos. Graph separation techniques for quadratic zero-one programming. Technical report cs-87-39, Computer Science Department, the Pennsylvania State University, 1987.

[Ji *et al.*, 2004] cited page 53
Y. Ji, X. Xu, and G.D. Stormo. A graph theoretical approach for predicting common rna secondary structure motifs including pesudoknots in unaligned sequences. *Bioinformatics*, 20(10):1591–1602, 2004.

[Jones and Forrest, 1995] cited page 32
T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, 1995.

[Katayama and Narihisa, 2001] cited page 13, 16, 42, 59, 82
K. Katayama and H. Narihisa. Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. *European Journal of Operational Research*, 134:103–119, 2001.

[Katayama and Narihisa, 2004] cited page 10
K. Katayama and H. Narihisa. An evolutionary approach for the maximum diversity problem. In W. Hart, N. Krasnogor, and J.E. Smith, editors, *Recent advances in memetic algorithms*, pages 31–47. Springer Berlin, 2004.

[Katayama *et al.*, 2000] cited page 16, 17
K. Katayama, M. Tani, and H. Narihisa. Solving large binary quadratic programming problems by effective genetic local search algorithm. In *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 643–650, 2000.

[Katayama *et al.*, 2005] cited page 53, 63, 98, 98
K. Katayama, A. Hamamoto, and H. Narihisa. An effective local search for the maximum clique problem. *Information Processing Letters*, 95(5):503–511, 2005.

[Kilby *et al.*, 2005] cited page 22
P. Kilby, J. Slaney, S. Thiébaux, and T. Walsh. Backbones and backdoors in satisfiablity. In *Proceedings of AAAI-2005*, pages 1368–1373, 2005.

[Knot, 2001] cited page 53
S. Knot. Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In *Proceedings of 42nd annual IEEE symposium on fundations of computer science (FOCS)*, pages 600–609, 2001.

[Kochenberger *et al.*, 2004] cited page 54, 89, 89
G. Kochenberger, F. Glover, B. Alidaee, and C. Rego. A unified modeling and solution framework for combinatorial optimization problems. *OR Spectrum*, 26:237–250, 2004.

[Kochenberger *et al.*, 2005] cited page 7
G. Kochenberger, F. Glover, B. Alidaee, and C. Rego. An unconstrained quadratic binary programming approach to the vertex coloring problem. *Annals OR*, 139(1):229–241, 2005.

[Kochenberger *et al.*, 2007] cited page 7
G. Kochenberger, B. Alidaee, F. Glover, and H. Wang. An effective modeling and solution approach for the generalized independent set problem. *Optimization Letters*, 1:111–117, 2007.

[Kochenberger *et al.*, 2011] cited page 7
G. Kochenberger, J.K. Hao, Z. Lü, H. Wang, and F. Glover. Solving large scale max cut problems via tabu search. *Journal of Heuristics*, 2011.

[Kokosinski and Kawarciany, 2007] cited page 88, 91, 92, 92
Z. Kokosinski and K. Kawarciany. On sum coloring of graphs with parallel genetic algorithms. In *Lecture Notes in Computer Science*, volume 4431, pages 211–219, 2007.

[Krarup and Pruzan, 1978] cited page 7
J. Krarup and A. Pruzan. Computer aided layout design. *Mathematical Programming Study*, 9:75–94, 1978.

[Kroon *et al.*, 1996] cited page 88
L.G. Kroon, A. Sen, H. Deng, and A. Roy. The optimal cost chromatic partition problem for trees and interval graphs. *Lecture Notes in Computer Science*, 1197:279–292, 1996.

[Kubicka and Schwenk, 1989] cited page 88
E. Kubicka and A.J. Schwenk. An introduction to chromatic sums. In *Proceedings of the 17th Annual ACM Computer Science Conference*, pages 39–45, 1989.

[Kuo *et al.*, 1993] cited page 6, 98
C.C. Kuo, F. Glover, and K.S. Dhir. Analyzing and modeling the maximum diversity problem by zero-one programming. *Decision Sciences*, 24(6):1171–1185, 1993.

[Laughunn, 1970] cited page 7
D.J. Laughunn. Quadratic binary programming with application to capital-budgeting problems. *Operations Research*, 18(3):454–461, 1970.

[Lewis *et al.*, 2008] cited page 7
M. Lewis, G. Kochenberger, and B. Alidaee. A new modeling and solution approach for the set-partitioning problem. *Computers & Operations Research*, 35(3):807–813, 2008.

[Lewis *et al.*, 2009] cited page 7
M. Lewis, B. Alidaee, F. Glover, and G. Kochenberger. A note on xqx as a modelling and solution framework for the linear ordering problem. *International Journal of Operational Research*, 5(2):152–162, 2009.

[Li *et al.*, 2009] cited page 88, 91, 92, 92
Y. Li, C. Lucet, A. Moukrim, and K. Sghiouer. Greedy algorithms for the minimum sum coloring problem. In *Proceedings LT2009 Conference*, 2009.

[Liu *et al.*, 2006] cited page 15
W. Liu, D. Wilkins, and B. Alidaee. A hybrid multi-exchange local search for unconstrained binary quadratic program. Working papers series, Hearin center for enterprise science, 2006.

[Lodi *et al.*, 1999] cited page 16
A. Lodi, K. Allemand, and T.M. Liebling. An evolutionary heuristic for quadratic 0-1 programming. *European Journal of Operational Research*, 119:662–670, 1999.

[Lozano *et al.*, 2011] cited page 98, 104, 106
M. Lozano, D. Molina, and C. García-Martínez. Iterated greedy for the maximum diversity problem. *European Journal of Operational Research*, 214(1):31–38, 2011.

[Lü *et al.*, 2010a] cited page 17, 24, 30, 36, 38, 38, 42, 42, 56, 57, 59, 59, 82, 82, 112
Z. Lü, F. Glover, and J.K. Hao. A hybrid metaheuristic approach to solving the ubqp problem. *European Journal of Operational Research*, 207:1254–1262, 2010.

[Lü *et al.*, 2010b] cited page 17, 24
Z. Lü, J.K. Hao, and F. Glover. A study of memetic search with multi-parent combination for ubqp. In *Proceedings of the 10th European conference on Evolutionary Computation in Combinatorial Optimization*, volume 1 of *EvoCOP'10*, pages 154–165, 2010.

[Lü *et al.*, 2012] cited page 15, 101
Z. Lü, F. Glover, and J.K. Hao. Neighborhood combination for unconstrained binary quadratic problems. In M. Caserta and S. Voss, editors, *MIC-2009 Post-Conference Book*, pages 49–61. Springer Berlin, 2012.

[Malafiejski, 2004] cited page 88, 88
M. Malafiejski. Sum coloring of graphs. In *Graph Colorings, Contemporary Mathematics*, pages 55–65. 2004.

[Manninno and Stefanutti, 1999] cited page 53
C. Manninno and E. Stefanutti. An augmentation algorithm for the maximum weighted stable set problem. *Computational Optimization and Applications*, 14:367–381, 1999.

[Marti *et al.*, 2009] cited page 60, 60, 60, 60, 83, 83, 84, 84
R. Marti, A. Duarte, and M. Laguna. Advanced scatter search for the max-cut problem. *INFORMS Journal on Computing*, 21:26–38, 2009.

[Martí *et al.*, 2010] cited page 98
R. Martí, M. Gallego, and A. Duarte. A branch and bound algorithm for the maximum diversity problem. *European Journal of Operational Research*, 1:36–44, 2010.

[Martí *et al.*, 2011] cited page 98, 98
R. Martí, G. Micael, D. Abraham, and G.P. Eduardo. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 2011.

[Mauri and Lorena, 2011] cited page 8
G.R. Mauri and L.A.N. Lorena. Lagrangean decompositions for the unconstraied binary quadratic programming problem. *International Transaction in Operational Research*, 18:257–270, 2011.

[Mauri and Lorena, 2012] cited page 8
G. R. Mauri and L.A.N. Lorena. A column generation approach for the unconstrained binary quadratic programming problem. *European Journal of Operational Research*, 217:69–74, 2012.

[McBride and Yormark, 1980] cited page 7
R.D. McBride and J.S. Yormark. An implicit enumeration algorithm for quadratic integer programming. *Management Science*, 26(3):282–296, 1980.

[Merz and Freisleben, 1999] cited page 16
P. Merz and B. Freisleben. Genetic algorithms for binary quadratic programming. In *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, volume 1, pages 417–424, 1999.

[Merz and Freisleben, 2002] cited page 12, 16, 17
P. Merz and B. Freisleben. Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics*, 8:197–213, 2002.

[Merz and Katayama, 2004] cited page 17
P. Merz and K. Katayama. Memetic algorithms for the unconstrained binary quadratic programming problem. *Biosystems*, 78:99–118, 2004.

[Meyerhenke *et al.*, 2009] cited page 36
H. Meyerhenke, B. Monien, and T. Sauerwald. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. *Journal of Parallel and Distributed Computing*, 69(9):750–761, 2009.

[Monasson *et al.*, 1998] cited page 22
R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity for characteristic 'phase transitions'. *Nature*, 400:133–137, 1998.

[Neri *et al.*, 2011] cited page 98, 99
F. Neri, C. Cotta, and P. Moscato. *Handbook of Memetic Algorithms. Studies in Computational Intelligence 379*. Springer, 2011.

[Östergård, 2001] cited page 53
P.R.J. Östergård. A new algorithm for the maximum weight clique problem. *Nordic Journal of Computing*, 8(4):424–436, 2001.

[Pajouh *et al.*, 2011] cited page 7, 53, 65
F.M. Pajouh, B. Balasumdaram, and O. Prokopyev. On characterization of maximal independent sets via quadratic optimization. *Journal of Heuristics*, 2011.

[Palubeckis and Tomkevicius, 2002] cited page 12, 14
G. Palubeckis and A. Tomkevicius. Grasp implementations for the unconstrained binary quadratic optimization problem. *Information Technology and Control*, 24:14–20, 2002.

[Palubeckis, 1995] cited page 8
G. Palubeckis. A heuristic-based branch and bound algorithm for unconstrained quadratic zero-one programming. *Computing*, 54:283–301, 1995.

[Palubeckis, 2004a] cited page 60, 83, 83, 84
G. Palubeckis. Application of multistart tabu search to the maxcut problem. *Information Technology and Control*, 2:29–35, 2004.

[Palubeckis, 2004b] cited page 14, 14, 18, 24, 42, 57, 59, 59, 80, 82
G. Palubeckis. Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research*, 131:259–282, 2004.

[Palubeckis, 2006] cited page 14, 24, 42, 57, 59, 59, 80, 82, 82
G. Palubeckis. Iterated tabu search for the unconstrained binary quadratic optimization problem. *Informatica*, 17:279–296, 2006.

[Palubeckis, 2007] cited page 98, 104, 106, 112
G. Palubeckis. Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, 189(1):371–383, 2007.

[Pardalos and Jha, 1992] cited page 6, 6
P.M. Pardalos and S. Jha. Complexity of uniqueness and local search in quadratic 0-1 programming. *Operations Research Letters*, 11:119–123, 1992.

[Pardalos, 1990] cited page 8
P. M. Pardalos. Computational aspects of a branch and bound algorithm for quadratic zero-one programming. *Computing*, 45:131–144, 1990.

[Phillips and Rosen, 1994] cited page 7
A.T. Phillips and J.B. Rosen. A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization*, 4:229–241, 1994.

[Picard, 1974] cited page 6
J.C. Picard. Minimum cuts and related problems. *Networks*, 5:357–370, 1974.

[Pullan and Hoos, 2006] cited page 53, 63
W. Pullan and H.H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[Pullan, 2006] cited page 53, 63, 66
W. Pullan. Phased local search for the maximum clique problem. *Journal of Combinatorial Optimization*, 12:303–323, 2006.

[Pullan, 2008] cited page 53, 63, 68, 70, 70
W. Pullan. Approximating the maximum vertex/edge weighted clique using local search. *Journal of Heuristics*, 14:117–134, 2008.

[Rendl *et al.*, 2006] cited page 8
F. Rendl, G. Rinaldi, and A. Wiegele. A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations. Technical report, Alpen-Adria-Universität Klagenfurt, 2006.

[Resende and Ribeiro, 2003] cited page 48
M. Resende and C. Ribeiro. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 219–249. 2003.

[Resendel and Ribeiro, 2005] cited page 48
M. Resendel and C. Ribeiro. In *Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces*, pages 29–63. 2005.

[Salavatipour, 2003] cited page 88
M.R. Salavatipour. On sum coloring of graphs. *Discrete Applied Mathematics*, 127(3):477–488, 2003.

[Sengor *et al.*, 1999] cited page 53
N.S. Sengor, Y. Cakir, C. Guzelis, F. Pekergin, and O. Morgul. An analysis of maximum clique formulations and saturated linear dynamical network. *ARI - An International Journal for Physical and Engineering Sciences*, 51:268–276, 1999.

[Shylo and Shylo, 2010] cited page 60, 83, 83, 84
V.P. Shylo and O.V. Shylo. Solving the maxcut problem by the global equilibrium search. *Cybernetics and Systems Analysis*, 46:744–754, 2010.

[Shylo and Shylo, 2011] cited page 18
V.P. Shylo and O.V. Shylo. Solving unconstrained binary quadratic programming problem by global equilibrium search. *Cybernetics and System Analysis*, 47(6):889–897, 2011.

[Silva *et al.*, 2004] cited page 98
G.C. Silva, L.S. Ochi, and S.L. Martins. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, volume 3059, pages 498–512, 2004.

[Silva *et al.*, 2007] cited page 98
G.C. Silva, M.R.Q. Andrade, L.S. Ochi, S.L. Martins, and A. Plastino. New heuristics for the maximum diversity problem. *Journal of Heuristics*, 13(4):315–336, 2007.

[Singh and Gupta, 2006] cited page 53, 63
A. Singh and A.K. Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12:5–22, 2006.

[Solnon and Fenet, 2005] cited page 53
C. Solnon and S. Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3):155–180, 2005.

[Syswerda, 1989] cited page 38, 114
G. Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, 1989.

[Thomassen *et al.*, 1989] cited page 88
C. Thomassen, P. Erdos, Y. Alavi, and P.J. Malde. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13:353–357, 1989.

[Toulouse *et al.*, 1999] cited page 36
M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level cooperative search: a new paradigm for combinatorial optimization and an application to graph partitioning. In P. Amestoy, P. Berger, M. Daydé, D. Ruiz, I. Duff, V. Frayssé, and L. Giraud, editors, *Euro-Par' 99 Parallel Processing*, volume 1685 of *Lecture Notes in Computer Science*, pages 533–542, 1999.

[Walshaw and Cross, 2000] cited page 36
C. Walshaw and M. Cross. Mesh partitioning : a multilevel balancing and refinement algorithm. *SIAM Journal on Scientific Computing*, 22(1):63–80, 2000.

[Walshaw, 2004] cited page 36, 36
C. Walshaw. Multilevel refinement for combinatorial optimisation problems. *Annals of Operations Research*, 131:325–372, 2004.

[Wang *et al.*, 2011a] cited page 18, 42
J. Wang, Y. Zhou, and J. Yin. Combining tabu hopfield network and estimation of distribution for unconstrained binary quadratic programming problem. *Expert System with Applications*, 2011.

[Wang *et al.*, 2011b] cited page 21, 42
Y. Wang, Z. Lü, F. Glover, and J.K. Hao. Effective variable fixing and scoring strategies

for binary quadratic programming. In P. Merz and J.K. Hao, editors, *EvoCOP2011*, volume 6622 of *Lecture Notes in Computer Science*, pages 72–83, 2011.

[Wang *et al.*, 2012a] cited page 98, 104, 106, 106, 112
J. Wang, Y. Zhou, Y. Cai, and J. Yin. Learnable tabu search guided by estimation of distribution for maximum diversity problems. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 16(4):711–728, 2012.

[Wang *et al.*, 2012b] cited page 47
Y. Wang, J.K. Hao, F. Glover, and Z. Lü. Solving the maximum clique and maximum vertex weight clique problems via binary quadratic programming. *Submitted to Discrete Applied Mathematics*, 2012.

[Wang *et al.*, 2012c] cited page 75
Y. Wang, J.K. Hao, F. Glover, and Z. Lü. Solving the minimum sum coloring problem via binary quadratic programming. *Submitted to Optimization Letters*, 2012.

[Wang *et al.*, 2012d] cited page 97
Y. Wang, J.K. Hao, F. Glover, and Z. Lü. A tabu search based memetic algorithm for the maximum diversity problem. *Submitted to Computers & Operations Research*, 2012.

[Wang *et al.*, 2012e] cited page 35
Y. Wang, Z. Lü, F. Glover, and J.K. Hao. A multilevel algorithm for large unconstrained binary quadratic optimization. In N. Beldiceanu, N. Jussien, and E. Pinson, editors, *CPAIOR 2012, Lecture Notes in Computer Science*, pages 395–408, 2012.

[Wang *et al.*, 2012f] cited page 75
Y. Wang, Z. Lü, F. Glover, and J.K. Hao. Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, 223:595–604, 2012.

[Wang *et al.*, 2012g] cited page 47
Y. Wang, Z. Lü, F. Glover, and J.K. Hao. Probabilistic grasp-tabu search algorithms for the ubqp problem. *Computers & Operations Research*, 2012.

[Wilbaut *et al.*, 2009] cited page 22
C. Wilbaut, S. Salhi, and S. Hanafi. An iterative variable-based fixation heuristic for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 199(2):339–348, 2009.

[Witzgall, 1975] cited page 7
C. Witzgall. Mathematical methods of site selection for electronic message systems (ems). Final report national bureau of standards, 1975.

[Wu and Hao, 2011] cited page 53, 63, 66
Q. Wu and J.K. Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, 2011.

[Wu and Hao, 2012] cited page 88, 91, 92
Q. Wu and J.K. Hao. An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600, 2012.

[Xu *et al.*, 1996] cited page 49
J.F. Xu, S.Y. Chiu, and F. Glover. Probabilistic tabu search for telecommunications network design. *Combinational Optimization: Theory and Practice*, 1:69–94, 1996.

[Xu *et al.*, 1998] cited page 48, 49, 52
J.F. Xu, S.Y. Chiu, and F. Glover. Fine-tuning a tabu search algorithm with statistical tests. *Internatioanl Transactions in Operational Research*, 5:233–244, 1998.

[Zhang, 2004] cited page 22
W. Zhang. Configuartion landscape analysis and backbone guided local search: Part i: Satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1):1–26, 2004.

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

université angers

Thèse de Doctorat

Yang WANG

Metaheuristics for large binary quadratic optimization and its applications

Métaheuristiques pour l'optimization quadratique en 0/1 à grande échelle et ses applications

Résumé

Cette thése étudie le problème NP-difficile de optimization quadratique en variables binaires (BQO), à savoir le problème de la maximisation d'une fonction quadratique en variables binaires. BQO peut représenter de nombreux problèmes importants de différents domaines et servir de modèle unifié pour un grand nombre de problèmes d'optimisation combinatoire portant sur les graphes. Cette thèse est consacrée au développement d'algorithmes métaheuristiques efficaces pour résoudre le BQO et ses applications. Premièrement, nous proposons algorithmes de "backbone guided" recherche tabou et d'un algorithme mémétique multi-niveaux sur la base de la technique de la fixation de variables. Ces techniques sont toutes deux basées sur l'idée de la réduction du problème afin de mener à bien une exploitation exhaustive d'une petite région de recherche. Ensuite, nous nous concentrons sur des procédés avancés de génération des solutions initiales préférables et développons des algorithmes combinant GRASP avec la recherche tabou et les algorithmes de path-relinking. En outre, nous résolvons des problèmes, y compris le problème de coupe maximum, de clique maximum, de clique maximale de sommets pondérés et la somme coloration minimum, soit en appliquant directement ou avec une légère adaptation de nos algorithmes développés pour BQO, avec l'hypothèse que ces problèmes sont reformulés en BQO. Enfin, nous présentons un algorithme mémétique basé sur la recherche tabou qui s'attaque efficacement au BQO avec contrainte de cardinalité.

Mots clés

Optimization quadratique en variables binaires, recherche tabou, fixation de variables, GRASP, path relinking, algorithme mémétique.

Abstract

This thesis investigates the NP-hard binary quadratic optimization (BQO) problem, i.e. the problem of maximizing a quadratic function in binary variables. BQO can represent numerous important problems from a variety of domains and serve as a unified model for many combinatorial optimization problems pertaining to graphs. This thesis is devoted to developing effective metaheuristic algorithms for solving BQO and its applications. First, we propose backbone guided tabu search algorithms on the basis of variable fixation technique and a backbone multilevel memetic algorithm following the general multilevel framework, both of which are based on the idea of decreasing the problem scale so as to carry out extensive exploitation in a small search area. Then we focus on advanced methods of generating preferable initial solutions and develop GRASP combined with tabu search algorithms and path relinking algorithms. In addition, we undertake to tackle problems including maximum cut, maximum clique, maximum vertex weight clique and minimum sum coloring either by directly applying or with a trivial adaptation of our developed algorithms for BQO, with the premise that these problems are recast into the form of BQO. Finally, we present a memetic algorithm based on tabu search that effectively tackles the cardinality constrained BQO.

Key Words

Binary quadratic optimization, tabu search, variable fixation, GRASP, path relinking, memetic algorithm.