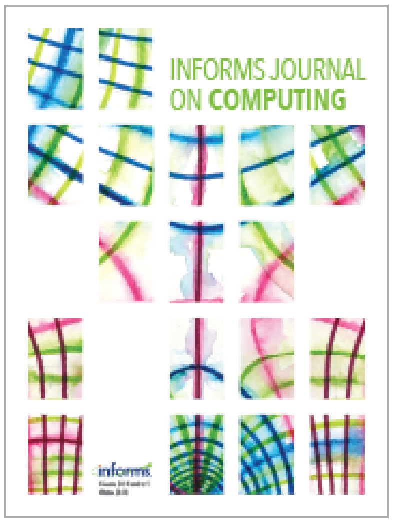


INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Advanced Tabu Search Algorithms for Bipartite Boolean Quadratic Programs Guided by Strategic Oscillation and Path Relinking

Qinghua Wu, Yang Wang*, Fred Glover

informs
http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 32, No. 1, Winter 2020, pp. 74–89
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Advanced Tabu Search Algorithms for Bipartite Boolean Quadratic Programs Guided by Strategic Oscillation and Path Relinking

Qinghua Wu,[a] **Yang Wang,**[b,*] **Fred Glover**[c]

[a] School of Management, Huazhong University of Science and Technology, 430074 Wuhan, China; [b] School of Management, Northwestern Polytechnical University, 710072 Xi'an, China; [c] Electrical, Computer, and Energy Engineering, School of Engineering and Science, University of Colorado Boulder, Boulder, Colorado 80309

*Corresponding author

Contact: qinghuawu1005@gmail.com, http://orcid.org/0000-0003-4015-0305 (QW); sparkle.wy@gmail.com, http://orcid.org/0000-0002-3864-0834 (YW); fredwglover@yahoo.com (FG)

Received: December 15, 2017
Revised: July 14, 2018
Accepted: October 5, 2018
Published Online in Articles in Advance: July 19, 2019

https://doi.org/10.1287/ijoc.2018.0871

Abstract. The bipartite Boolean quadratic programming problem (BBQP) is a generalization of the well-studied NP-hard Boolean quadratic programming problem and can be regarded as a unified model for many graph theoretic optimization problems, including maximum weight-induced subgraph problems, maximum weight biclique problems, matrix factorization problems, and maximum cut problems on bipartite graphs. This paper introduces three main algorithms for solving the BBQP, based on three variants of tabu search, the first two consisting of strategic oscillation–tabu search (SO-TS) algorithms, which use destructive and constructive procedures to guide the search into unexplored and promising areas. The third algorithm, whichDoes also incorporates the SO-TS algorithms as solution improvement methods, uses a path relinking (PR) algorithm that is capable of further enhancing search performance. Experimental results demonstrate that all three algorithms perform very effectively compared with the best methods in the literature, and the PR algorithm joined with tabu search is able to discover new best solutions for two-thirds of the large problem instances and match the previous best known solutions for the other instances. Additional analysis discloses the contributions of the key ingredients of each of the proposed algorithms.

History: Accepted by Alice Smith, Area Editor for Heuristic Search.
Funding: Financial support was received from the National Natural Science Foundation of China [Grants 71501157, 71401059, 71771099, and 71531009], Huazhong University of Science and Technology [Grant 5001300001], and Fundamental Research Funds for the Central Universities [Grant 3102017zy059].
Supplemental Material: The online supplement is available at https://doi.org/10.1287/ijoc.2018.0871.

Keywords: bipartite Boolean quadratic programs • tabu search • strategic oscillation • path relinking

1. Introduction

Let $Q = (q_{ij})$ be an $m \times n$ matrix, $c = (c_1, c_2, \ldots, c_m)$ be a row vector in R^m, and $d = (d_1, d_2, \ldots, d_n)$ be a row vector in R^n. The bipartite Boolean quadratic programming problem (BBQP) is defined as

$$\text{BBQP}: \quad \text{Maximize} \quad f(x,y) = x^T Q y + cx + dy \tag{1}$$
$$\text{Subject to} \quad x \in \{0,1\}^m, y \in \{0,1\}^n, \tag{2}$$

where x and y are required to be column vectors.

The BBQP is a generalization of the well-studied Boolean quadratic programming problem (BQP).

$$\text{BQP}: \quad \text{Maximize} \quad f(x) = x^T Q' x + c' x \tag{3}$$
$$\text{Subject to} \quad x \in \{0,1\}^n, \tag{4}$$

where Q' is an $n \times n$ matrix and c' is a row vector in R^n (Glover et al. 1998, Lü et al. 2010, Wang et al. 2012). As demonstrated in Punnen et al. (2015b), the BQP can be formulated as the BBQP by choosing $Q = Q' + 2MI$, $c = \frac{1}{2}c' - Me$, and $d = \frac{1}{2}c' - Me$, where I is an $n \times n$ identity matrix, $e \in R^n$ is an all-one vector, and M is a large positive constant.

A graph theoretic interpretation of the BBQP is as follows (Punnen et al. 2015b). Let $G = (I, J, E)$ be a bipartite graph, where $I = \{1, 2, \ldots, m\}$ and $J = \{1, 2, \ldots, n\}$ are sets of nodes for the two parts of G, and c_i and d_j are associated with the nodes $i \in I$ and $j \in J$, respectively. Furthermore, an interaction profit w_{ij} is associated with each edge $(i,j) \in E$. Then the maximum weight-induced subgraph problem (MWISP) on G is to find a subgraph $G' = (I', J', E')$ to maximize $\sum_{i \in I'} c_i + \sum_{j \in J'} d_j + \sum_{(i,j) \in E'} w_{ij}$, where $I' \subseteq I$, $J' \subseteq J$, and G' is induced by $I' \cup J'$.

The BBQP is a unified model that encompasses a wide range of graph theoretic optimization problems. By choosing $q_{ij} = w_{ij}$ if $(i,j) \in E$ and $q_{ij} = 0$ if $(i,j) \notin E$, the BBQP gives rise to the MWISP on a bipartite graph. If the selected subgraph is required to be a clique, then the resulting problem is called a maximum weight biclique problem (MWBP) (Ambühl et al. 2011). In addition, the BBQP model can also be used to formulate the maximum weight cut problem in a bipartite graph (Punnen et al. 2015b), the binary matrix factorization problem (Koyutürk et al. 2006, Shen et al. 2009, Gillis and Glineur 2011), and the cut-norm approximation of a matrix (Alon and Naor 2006).

Theoretical results for the BBQP include domination analysis of approximation algorithms, the demonstration that BBQP is NP-hard, and the identification of polynomially solvable special cases (Punnen et al. 2015a, b). Important applications of BBQP arise in bioinformatics (Tanay et al. 2002, Chang et al. 2012) and data mining (Shen et al. 2009). For instance, discovering a bicluster of genes that exhibit similar expression patterns is a core step in the analysis of gene expression data, and mining discrete patterns in binary data by use of a rank-one binary approximation matrix is important for subsampling, compression, and clustering. Because of its NP-hard nature and the great difficulty of solving it exactly, several heuristic and meta-heuristic algorithms have been proposed for the BBQP.

Karapetyan and Punnen (2013) designed multiple constructive heuristics, local search heuristics, and their combinations for the BBQP, identifying worst case performance ratios for the greedy algorithm and providing a complexity analysis of all algorithms. The authors generated five classes of benchmarks from various applications, which have become standard benchmarks used for performance evaluation in the BBQP literature. Experimental analysis of these proposed methods shows that no single algorithm dominates the others for all the instances studied.

Glover et al. (2015) developed a tabu search (TS) approach, a very large-scale neighborhood (VLSN) search approach and a hybrid method (HM) that integrates TS and VLSN search. An experimental study on a set of 85 benchmark instances discloses that the hybrid method performs better in terms of solution quality than each component algorithm in isolation.

Duarte et al. (2014) proposed a branch-and-bound algorithm (B&B) and multiple iterated local search (ILS) algorithms for solving the BBQP. Different strategies for each search element are developed to obtain ILS variants, including semigreedy versus greedy constructive procedures for initial solution generation, and local search versus intensification–diversification search procedures for solution improvement. Computational results on solving small instances with $m = 50$ and $n = 50$ disclose the B&B implementation generally produces tighter lower bounds but inferior upper bounds than the general CPLEX solver. For large instances, the best ILS variant is able to significantly outperform algorithms in Karapetyan and Punnen (2013) but presents no significant difference when compared with HM in Glover et al. (2015).

Karapetyan et al. (2017) go a step farther to present a high-performance conditional Markov chain search (CMCS) approach for the BBQP, which significantly improves the algorithms developed in the preliminary version. Hill climbing for search intensification and mutation for search diversification are automatically selected according to a CMCS scheme. Extensive experimental results reveal the proposed approach yields excellent results and competes favorably with HM.

In this work, we propose two effective strategic oscillation guided tabu search (SO-TS) algorithms and an especially effective path relinking (PR) algorithm that incorporates the SO-TS algorithms within it for the BBQP. The proposed algorithms integrate several new and general-purpose search ingredients, which are not only demonstrated to obtain exceptional results for the BBQP but can also be applied in other search approaches to improve their performance. The main contributions of this paper are summarized as follows.

First, VLSN search has been recently shown to be a useful tool for solving combinatorial optimization problems (Ahuja et al. 2002, Glover et al. 2015, Karapetyan et al. 2017). In this work, we investigate how VLSN can be beneficially integrated into TS. Our designed VLSN-TS method uses a solution-based tabu strategy to effectively determine tabu status of VLSN moves. Experimental analysis discloses that incorporating VLSN in solution-based TS can improve upon the popular VLSN integration in hill climbing search.

Second, we propose a frequency-driven strategic oscillation strategy based on ideas from tabu search to diversify search trajectories, where adaptive frequency memory is introduced to generate starting solutions in promising search areas. This simple diversification mechanism plays an important role in enhancing the performance of tabu search and can be used as a general diversification component in various heuristic search approaches.

Third, we explore for the first time the usefulness of multiple solution improvement methods to enhance the PR algorithm by a procedure that selects a solution improvement method adaptively. More importantly, this strategy can also be useful as an intensification search component in other population-based search approaches, such as memetic search, genetic algorithms, and scatter search, among others.

The computational assessment on five classes of benchmarks with a total of 50 instances discloses that our proposed algorithms perform well compared with the best methods in the literature, and our composite PR algorithm finds new best solutions for 17 instances (including 16 out of the 25 larger and more challenging problem instances), while matching the previous best known solutions for all other instances.

The rest of the paper is organized as follows. Section 2 describes the tabu search algorithms guided by strategic oscillation, and Section 3 describes the tabu search algorithm guided by path relinking. Section 4 analyzes the influence of key ingredients of our algorithms on their performance. Experimental results and comparisons with state-of-the-art algorithms in the literature are presented in Section 5, and concluding remarks are given in Section 6.

2. Strategic Oscillation–Tabu Search Algorithms

Approaches based on the general tabu search framework have been demonstrated to be highly effective for solving the BBQP (Glover et al. 2015). Strategic oscillation is closely linked to tabu search as one of the earliest proposed strategies for diversifying the search (Glover 1977, Glover and Laguna 1997, Glover 2000). Motivated by the capacity of strategic oscillation to play a diversification role in tabu search, we propose two new strategic oscillation guided tabu search algorithms to further enhance the basic tabu search approach for the BBQP. The two algorithms share the same strategic oscillation scheme but use different variants of tabu search.

2.1. Main Scheme

Algorithm 1 shows the structure of our SO-TS approach, which alternates between a tabu search phase (Section 2.2) to find a locally optimal solution and a strategic oscillation phase (Section 2.3) to direct the search into unexplored and promising search areas. Starting from a given initial solution (x, y), where $x = (x_1, x_2, \ldots, x_m)$ and $y = (y_1, y_2, \ldots, y_n)$ are two vectors of binary (zero–one) variables, the tabu search phase iteratively selects and executes moves until the solution cannot be further improved for a specified consecutive number of iterations, identified by the iteration cutoff value ρ. During this TS phase, two frequency memory *FreqX* and *FreqY* are maintained to record the frequency that each variable receives the value 1 in preparation for the follow-up strategic oscillation phase. We save the best solution found during the tabu search phase as (x', y') and the solution obtained in the last iteration as the critical solution (x^c, y^c). If the solution (x', y') improves the best solution (x^*, y^*) found so far, then (x^*, y^*) is updated to be (x', y').

Once the TS phase terminates, the strategic oscillation phase begins. A destructive procedure is first launched to drop a certain number (identified below by a value $\lfloor (m+n) \times span \rfloor$, where $\lfloor (m+n) \times span \rfloor$ is the integer part of $(m+n) \times span$) of variables in the critical solution by changing their values from 1 to 0 to produce a partial solution (x^p, y^p). Afterward, the constructive procedure adds back the same number of variables by changing their values from 0 to 1 in (x^p, y^p) to produce a new trial solution (x, y) for the next phase of tabu search. Both the selection of the dropped and added variables depends on the frequency information provided by *FreqX* and *FreqY*. The SO-TS method repeats the tabu search phase and the strategic oscillation phase until the specified stopping condition is satisfied.

Algorithm 1 (Outline of the SO-TS Approach)

```
 1. Input: a given solution $(x, y)$
 2. Output: the improved solution $(x^*, y^*)$
 3. Set $x^* = x$, $y^* = y$
 4. **repeat**
 5.     $(x', y', x^c, y^c) \leftarrow$ TabuSearch($x, y, FreqX, FreqY, \rho$) (see Section 2.2)
 6.     **if** $f(x', y') > f(x^*, y^*)$ **then**
 7.         $x^* = x'$, $y^* = y'$
 8.     **end if**
 9.     $(x^p, y^p) \leftarrow$ Destructive($x^c, y^c, FreqX, FreqY, span$) (see Section 2.3.1)
10.     $(x, y) \leftarrow$ Constructive($x^p, y^p, FreqX, FreqY, span$) (see Section 2.3.2)
11. **until** the stopping condition is satisfied
```

2.2. Tabu Search

We design two tabu search phases, one using a simple neighborhood along with an attribute-based determination of tabu status (SN-TS) and the other using a very large-scale neighborhood accompanied by a solution-based determination of tabu status (VLSN-TS). In the following sections, we first describe the move operators and fast evaluation of move gains that provide the starting point of our search procedures and then describe the details of our complete SN-TS and VLSN-TS phases.

2.2.1. Move Operators. Move operators in neighborhood search are used to generate a sequence of solutions by modifying each member of the sequence to generate the next solution from a collection of neighbor candidates solutions. Typically the neighbors of a given solution result by very simple local changes, such as modifying the value of a single variable or the values of a small number of variables. For a given solution (x, y), let $I = \{1, 2, \ldots, m\}$ and $J = \{1, 2, \ldots, n\}$ be the index sets of variables in x and y, respectively. The proposed two move operators are defined as follows.

One-flip move: the simple move of flipping a variable $x_{i \in I}$ or $y_{j \in J}$ to change its current value to the complementary value $1 - x_i$ or $1 - y_j$.

Flip-x-optimize-y/flip-y-optimize-x move: this move is a very large-scale neighborhood move that flips multiple variables at the same time. The idea of the *flip-x-optimize-y/flip-y-optimize-x* move originates from the specific problem structure of the BBQP. More precisely, given a fixed vector x, we can efficiently compute an optimal vector $y^*(x)$ using Equation (5). Similarly, we can easily determine an optimal $x^*(y)$ using Equation (6) for a fixed y:

$$y_j^*(x) = \begin{cases} 1, & \text{if } d_j + \sum_{i=1}^{m} q_{ij}x_i > 0 \\ 0, & \text{otherwise;} \end{cases} \quad (5)$$

$$x_i^*(y) = \begin{cases} 1, & \text{if } c_i + \sum_{j=1}^{n} q_{ij}y_j > 0 \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

Based on this property, the *flip-x-optimize-y* move consists in flipping a variable x_i and then replacing y with the optimized $y^*(x)$. Similarly, the *flip-y-optimize-x move* consists in flipping a variable y_j and then replacing x with the optimized $x^*(y)$. Compared with the simple and cheap *one-flip* move, the *flip-x-optimize-y/flip-y-optimize-x* move is much more computationally expensive but able to reach a better candidate solution. Equation (5) and Equation (6) were first introduced in Punnen et al. (2015b).

2.2.2. Fast Evaluation of Move Gains.

To evaluate the effect of a move on the objective function as fast as possible, we adopt the use of a move gain analogous to that calculated in Glover and Hao (2010) and Benlic and Hao (2011). For a given move denoted by mv, the move gain Δ_{mv} indicates the variation in the objective value when mv is applied to transform the incumbent solution (x, y) into a neighboring solution (x', y'); that is, $\Delta_{mv} = f(x', y') - f(x, y)$, where f is the objective function defined in Equation (1). Specifically, we use two vectors ρ and γ, where ρ_i and γ_j respectively record the potentials of each variable x_i and y_j with respect to the objective function f:

$$\rho_i = c_i + \sum_{j=1}^{n} q_{ij}y_j, i \in I; \quad (7)$$

$$\gamma_j = d_j + \sum_{i=1}^{m} q_{ij}x_i, j \in J. \quad (8)$$

The move gain of flipping a variable x_i can be quickly calculated as ρ_i if x_i is changed from 0 to 1 and $-\rho_i$ otherwise. After the variable x_i is flipped, γ is accordingly updated in Equation (9) to propagate the effect of flipping x_i (in Equation (9), we use the convention that x_i represents the value of the variable x_i before being flipped):

$$\gamma_j = \gamma_j + (1 - 2x_i)q_{ij}, j \in J. \quad (9)$$

The move gain of flipping a variable y_i can be calculated in a like fashion. By using ρ and γ, the complexity to identify the best *one-flip* move for each tabu search iteration is reduced from $O(mn)$ to $O(m + n)$.

In this case, the move gain of flipping a variable x_i and reoptimizing y is calculated as

$$\Delta_{mv} = (1 - 2x_i)c_i + \sum_{j=1}^{n} \max(\gamma_j + (1 - 2x_i)q_{ij}, 0) - \sum_{j=1}^{n} \gamma_j y_j. \quad (10)$$

Once the *flip-x-optimize-y* move is performed, γ_j, $y_j^*(x)$, and ρ_i are updated by

$$\gamma_j = \gamma_j + (1 - 2x_i)q_{ij}, j \in J; \quad (11)$$

$$y_j^*(x) = \begin{cases} 1, & \text{if } \gamma_j > 0 \\ 0, & \text{otherwise;} \end{cases} \quad (12)$$

$$\rho_i = \rho_i + \sum_{j=1, y_j^*(x) \neq y_j}^{n} 2q_{ij}y_j^*(x) - q_{ij}, i \in I. \quad (13)$$

The move gain of performing a *flip-y-optimize-x* move can be calculated analogously. Compared with the *one-flip* move, the identification of the best *flip-x-optimize-y* or *flip-y-optimize-x* move is computationally expensive with complexity of $O(mn)$.

2.2.3. Simple Neighborhood–Based Tabu Search (SN-TS).

Based on the neighborhood induced by the *one-flip* move, our proposed SN-TS phase works as follows. Starting from an initial solution, each SN-TS iteration performs the best admissible move (which is non-tabu or satisfies the aspiration rule) that yields the largest move gain value to generate the next solution. This process is repeated until no improved solution is found for ρ_1 consecutive iterations. To prevent the search from short-term cycling, SN-TS applies a popular attribute-based tabu strategy to stipulate that if a variable is flipped at the current iteration, it is prohibited to be flipped during the following tl iterations, where tl is called the tabu tenure. A classic aspiration rule is used to allow a tabu variable to be flipped if this leads to a better solution than the best solution found so far.

2.2.4. Very Large-Scale Neighborhood–Based Tabu Search (VLSN-TS).

Incorporating neighborhoods based on VLSN in the hill climbing search framework has recently shown impressive results for solving the BBQP (Glover et al. 2015, Karapetyan et al. 2017). However, hill climbing search is confronted with prematurely getting trapped in a local optimum. To overcome this problem, we integrate VLSN in the tabu search framework to develop a VLSN-based tabu search phase. VLSN-TS uses *flip-x-optimize-y/flip-y-optimize-x* moves to generate its neighborhood and a solution-based tabu strategy to determine tabu status of moves. The solution-based tabu strategy records all solutions encountered during the search and judges a move as tabu only if the solution induced by this move is marked as previously visited.

We are motivated to use the solution-based tabu strategy that undertakes to identify an entire visited solution as tabu, rather than just one of its attributes, because of the following two reasons. First, it is very likely that a *flip-x-optimize-y*/*flip-y-optimize-x* move requires some variables lying in the tabu status to be flipped to achieve an optimized x or y vector, therefore making it counterproductive to refer to the attribute-based tabu settings of these variables. Second (and closely related to the first reason), an attribute-based tabu strategy will set a *flip-x-optimize-y*/*flip-y-optimize-x* move to be tabu if one of the variables involved in this move would receive a tabu status, which will exclude numerous such moves from consideration and thus adversely restrict the search.

To efficiently judge whether a solution was already visited, we use a hash function to map each visited solution into a 32-bit unsigned integer and store the integer in a binary hashing vector as suggested in Woodruff and Zemel (1993) and Carlton and Barnes (1996). The hash function we employ has the following form:

$$h(x,y) = \left[\sum_{i=1}^{m} w_i \times x_i + \sum_{j=1}^{n} w_{j+m} \times y_j\right] \bmod 2^{32}, \quad (14)$$

where w is an array containing integers drawn from the discrete uniform distribution on the interval $[0, 2^{32} - 1]$.

The method to determine whether a solution was previously visited operates as follows. First, we associate the hash function $h(x,y)$ with a binary hashing vector H of length 2^{32} bits. Initially, all components of the hashing vector H are filled with 0's. We record each visited solution (x,y) in the hashing vector by setting $H_i = 1$, where $i = h(x,y)$. Then to determine whether a solution (x^0, y^0) was previously visited, we test its presence in the hashing vector by checking the *ith* component in H, where $i = h(x^0, y^0)$. If (x^0, y^0) is absent in the hashing vector (i.e., $H_i = 0$), we confirm that the candidate solution has not been visited. Otherwise, we assume that the candidate solution was previously visited.

Starting from an initial solution, the VLSN-TS phase first repeatedly performs the best *flip-x-optimize-y* move (yielding the maximum move gain) excluding moves that are tabu. When the best solution cannot be improved using the *flip-x-optimize-y* moves, VLSN-TS then tries to improve it by performing *flip-y-optimize-x* moves in a similar manner. If neither of these types of moves can create an improvement, we select the best admissible move from them. The above procedure is repeated until the best solution cannot be improved for ρ_2 consecutive iterations. Experimental analysis in Section 4.2 reveals that VLSN-TS can improve upon the popular VLSN integration in hill climbing search.

2.3. Strategic Oscillation

Strategic oscillation is a tabu search strategy that orients moves in relation to a *critical event* as a means to create diversified trial solutions (Glover 2000, Gallego et al. 2013). We define the critical event of a BBQP instance to occur when the tabu search phase gets trapped in local "tabu optimum" (i.e., when the best solution obtained by the tabu search phase is not improved for a specified consecutive number of iterations) and call the solution obtained in the last iteration the critical solution.

The strategic oscillation scheme is achieved by applying a destructive procedure to drop variables (changing their values from 1 to 0) and a constructive procedure to add variables (changing their values from 0 to 1) with respect to the critical solutions. To achieve better search diversification, we propose to use frequency information collected during the search history to guide the selection of variables to be dropped and to be added. Specifically, we maintain frequency memories $FreqX_i, i \in I$ and $FreqY_j, j \in J$ to record the number of iterations that each variable x_i and y_j receives the value 1 during the tabu search phase. Our proposed destructive and constructive procedures are described as follows.

2.3.1. Destructive Procedure. Let (x^c, y^c) be the critical solution. The destructive procedure repeats steps that drop a variable x_i with the highest frequency $FreqX_i$ from (x^c, y^c) until $\lfloor m \times span \rfloor$ variables x_i are dropped. In the same manner, $\lfloor n \times span \rfloor$ variables y_j are dropped from (x^c, y^c). In this way, the critical solution (x^c, y^c) is transformed to a partial solution (x^p, y^p). The destructive procedure terminates by returning the partial solution (x^p, y^p).

2.3.2. Constructive Procedure. The selection of added variables in the constructive procedure depends on a combined measure of frequency and a "potential value" derived from the objective function. The procedure first calculates the average frequency afx of all variables x_i by $afx = \sum_{i=1}^{m} FreqX_i/m$. Then, it adds a variable x_i with the maximum potential to the partial solution (x^p, y^p), subject to requiring its frequency $FreqX_i$ to be less than the average frequency afx. This process is repeated until $\lfloor m \times span \rfloor$ variables x_i are added. Finally, $\lfloor n \times span \rfloor$ variables y_j are added in the same manner. The constructive procedure completes by returning the resulting trial solution $(x,y) = (x^p, y^p)$.

By combining the strategic oscillation phase with the simple neighborhood–based tabu search phase described in Section 2.2.3 and the very large-scale neighborhood–based tabu search phase described in Section 2.2.4, we obtain the two SO-TS algorithms, denoted by SO-SN-TS and SO-VLSN-TS. The critical role of the strategic oscillation scheme to the overall performance of the SO-TS algorithms is discussed in Section 4.3.

3. Path Relinking Algorithm

Path relinking is a population-based metaheuristic search approach whose underlying ideas share a significant intersection with TS (Glover et al. 2004). This approach generates new solutions by exploring trajectories that connect elite solutions and has been shown to provide a good balance between intensification and diversification (Laguna and Martí 1999, Yagiura et al. 2006, Wang et al. 2012, Chen et al. 2015, Peng et al. 2015). As the first adaptation of a population-based method tailored to the BBQP, our proposed algorithm follows the general PR framework and additionally integrates a set of original features responsible for its effectiveness, including a greedy relinking procedure to build high-quality solution paths, an adaptive selection of a solution improvement method, a quality-and-distance–based reference set updating strategy, and a distance based solution selection rule.

Algorithm 2 describes the general scheme of the PR algorithm used in this study. At the beginning, a reference set *RefSet* of elite solutions is constructed, where each solution in *RefSet* is generated by a randomized initialization procedure and further improved by a solution improvement method (see Sections 3.1 and 3.2). At the same time, *PairSet* is initialized to contain all the pairs of distinct solutions in *RefSet*, identified by their indexes. Both the pairs in *PairSet* and the two indexes in each pair are unordered. Then, an index pair is randomly selected from *PairSet* to get an initiating solution and a guiding solution, submitting to a relinking method to generate a path that connects the initiating solution where the path starts and the guiding solution where the path ends (see Section 3.3). For the created sequence of intermediate solutions on the path, a solution selection method is applied to select one or more solutions for further quality improvement (see Section 3.4). Each time a selected solution is refined by the solution improvement method, the *RefSet* update method is triggered and the set *PairSet* is updated (see Section 3.5). When *PairSet* becomes empty, the algorithm reinitializes *RefSet* to repeat the above-mentioned procedure until the elapsed time surpasses the given time limit.

Algorithm 2 (Outline of the PR Algorithm for the BBQP)

```
1.  Input: an instance of the BBQP
2.  Output: the best solution $(x^*, y^*)$ found so far
3.  repeat
4.    Initialize $RefSet = \{(x^1, y^1), (x^2, y^2), \ldots, (x^r, y^r)\}$
      (see Sections 3.1 and 3.2)
5.    Initialize $PairSet = \{(p,q) | p \neq q, (x^p, y^p), (x^q, y^q) \in RefSet\}$
6.    Record the best solution $(x^*, y^*)$ in RefSet and
      the objective value $f(x^*, y^*)$
7.    while ($PairSet \neq \emptyset$) do
8.      Pick randomly an index pair $(p,q) \in PairSet$
        to get solutions $(x^p, y^p)$ and $(x^q, y^q)$ from
        RefSet
9.      Set $x^{init} = x^p$, $y^{init} = y^p$, $x^{guid} = x^q$, $y^{guid} = y^q$
10.     $PSS = \{(x^1, y^1), (x^2, y^2), \ldots, (x^t, y^t)\} \leftarrow$ Path
        Relinking($x^{init}, y^{init}, x^{guid}, y^{guid}$) (see Section 3.3)
11.     $(x^s, y^s) \leftarrow$ SolutionSelection($PSS$)
        (see Section 3.4)
12.     $(x^{s'}, y^{s'}) \leftarrow$ SolutionImprovement($x^s, y^s$)
        (see Section 3.2)
13.     if $f(x^{s'}, y^{s'}) > f(x^*, y^*)$ then
14.       $x^* = x^{s'}$, $y^* = y^{s'}$
15.     end if
16.     Update RefSet and PairSet (see Section 3.5)
17.   end while
18. until the given time limit is reached
```

3.1. *RefSet* Initialization

The reference set *RefSet* consists of a set of diversified solutions with good solution quality, each of which is initially generated by the following two steps. The first step applies a randomized initialization procedure to obtain a diversified solution, which enables each variable in x and y to have an equal probability to receive the value 0 or 1. The second step applies the solution improvement method to improve the quality of the initial solution. The improved solution is added to *RefSet* if it is distinct from any solution in *RefSet*; otherwise, it is discarded. The above procedure repeats until *RefSet* reaches the reference set size r.

3.2. Adaptive Selection of Solution Improvement Methods

In contrast to traditional population-based algorithms, which typically rely on a single solution improvement method, our PR algorithm jointly applies the two solution improvement methods SO-SN-TS and SO-VLSN-TS to take advantage of useful features of each component method. Motivation for this approach comes from experimental observations that SO-SN-TS and SO-VLSN-TS exhibit complementary strengths in solving different classes of problem instances. We design an adaptive selection strategy inspired by Martí et al. (2009) to favor the use of the particular component method that produces solutions of better quality. Two counters σ_1 and σ_2 respectively record the number of times SO-SN-TS and SO-VLSN-TS find improved best solutions. At the beginning of the search, the probability $\sigma_1/(\sigma_1+\sigma_2)$ of selecting SO-SN-TS and the probability $\sigma_2/(\sigma_1+\sigma_2)$ of selecting SO-VLSN-TS are made equal by setting $\sigma_1 = 1$ and $\sigma_2 = 1$. As long as a component method obtains an updated best solution, the probability values are updated by increasing the corresponding σ_1 or σ_2 value by 1. In this way, the chance to use the component improvement method that produces better solution quality is increased during the search process. The stopping conditions of the SO-SN-TS and SO-VLSN-TS components in the PR algorithm are respectively set equal to the search depths λ_1 and λ_2, representing the maximum number of iterations of the corresponding approach. The merit of the adaptive selection of solution improvement methods is verified in Section 4.4.

3.3. Relinking Method

The relinking method is used to generate new promising solutions by creating a solution path connecting an initiating solution and a guiding solution selected from *RefSet*. The "combination effect" results from the fact that each intermediate solution on the path incorporates additional attributes from the guiding solution until finally reaching the guiding solution (Glover 1997). Algorithm 3 shows the pseudo-code of our relinking method. We first construct the difference set DS to include variables whose value assignments are different in the initiating solution and the guiding solution. Then a sequence of intermediate solutions $(x^1, y^1), (x^2, y^2), \ldots, (x^{|DS|-1}, y^{|DS|-1})$ is generated step by step to build the path starting with $(x^0, y^0) = (x^{init}, y^{init})$ and ending at $(x^{|DS|}, y^{|DS|}) = (x^{guid}, y^{guid})$. More precisely, we evaluate the move gain of flipping each variable in DS and identify the variable x_{bst} or y_{bst} yielding the maximum move gain. The first solution (x^1, y^1) built on the path is obtained from (x^0, y^0) by changing the value of x_{bst} or y_{bst} (according to whether an x variable or a y variable is the winner) from its current value assignment in (x^0, y^0) to its complementary value. Meantime, the variable x_{bst} or y_{bst} is removed from the set DS. For each follow-up path construction step, a path solution (x^t, y^t) is generated from its predecessor (x^{t-1}, y^{t-1}) in the same way. When the set DS becomes empty, the relinking method arrives at the guiding solution.

Algorithm 3 (Outline of the Relinking Method)

```
1.  Input: an initiating solution $(x^{init}, y^{init})$ and a
    guiding solution $(x^{guid}, y^{guid})$
2.  Output: path solutions $(x^1, y^1), (x^2, y^2), \ldots, (x^{|DS|-1},$
    $y^{|DS|-1})$
3.  $DS = \{x_i | x_i^{init} \neq x_i^{guid}, i \in I\} \cup \{y_j | y_j^{init} \neq y_j^{guid}, j \in J\}$
4.  Set $x^0 = x^{init}$, $y^0 = y^{init}$, $t = 1$
5.  while $|DS| > 0$ do
6.      Set $x^t = x^{t-1}$ and $y^t = y^{t-1}$
7.      Evaluate the move gain of flipping each
        variable in $DS$
8.      if $x_{bst}$ gets the best move gain then
9.          $x_{bst}^t = 1 - x_{bst}^{t-1}$, $DS = DS \setminus \{x_{bst}\}$
10.     end if
11.     if $y_{bst}$ gets the best move gain then
12.         $y_{bst}^t = 1 - y_{bst}^{t-1}$, $DS = DS \setminus \{y_{bst}\}$
13.     end if
14.     $t = t + 1$
15. end while
```

3.4. Path Solution Selection

The solution selection method is used to select solutions from the sequence of intermediate solutions produced by the relinking method for further improvement by the solution improvement method. Several popular path solution selection rules can be found in the literature, which typically select several solutions on the path for improvement. For our PR algorithm, the path solutions are generated in the neighborhood space where two consecutive solutions differ in the value assigned to a single variable. Hence, it is not fruitful to apply the improvement method to each solution on the path because many of these solutions would produce the same local optimum. In addition, the solution improvement method is the most time-consuming part of the PR algorithm, and the relinking method already guarantees to some extent the quality of path solutions by performing the best move at each path construction step. For these reasons, we decided to select a solution on the path for improvement that is midway between the initiating solution and the guiding solution, picking the second such solution if two solutions tie for "midway."

3.5. *RefSet* and *PairSet* Updating

The *RefSet* updating procedure is invoked after a path solution is refined by the solution improvement method. Some common *RefSet* updating approaches have been proposed, which for example may accept any solution that is better than the worst solution but then replaces the solution in *RefSet* that is closest to the new solution (Glover 1997), or which may maintain a two-piece *RefSet* consisting of high-quality and diversified solutions (Resende et al. 2010, Aringhieri and Cordone 2011).

In this work, we use a simple but effective *RefSet* updating method, which takes both quality and diversity criteria into consideration when deciding whether a new solution should be inserted into *RefSet* and, if so, which solution should be replaced. The diversity of a solution is measured as the minimum Hamming distance between this solution and each solution in

RefSet, where the Hamming distance between any two solutions is defined as the number of variables whose value assignments are different in the two solutions. If the minimum Hamming distance d between the new solution and each solution in *RefSet* satisfies $d > \eta \times (m + n)$ (η is called the distance threshold coefficient), we consider this solution sufficiently diverse, and then examine whether its solution quality is better than the worst solution quality in *RefSet*. If both the diversity and quality criteria are satisfied, this solution is inserted into *RefSet* to replace the worst solution.

When relinking occurs, the index pair for the initiating and guiding solutions is removed from *PairSet*, and the size of *PairSet* decreases. When *RefSet* is updated, all the index pairs associated with the replaced solution in *RefSet* are removed from *PairSet*. Meanwhile, new index pairs composed of the newly inserted solution and each other solution in *RefSet* are added to *PairSet*. This use of *PairSet* avoids duplications in examining pairs in *RefSet* that are sometimes permitted in other procedures.

4. Analysis of Important Search Components

Our experimental analysis to verify the effectiveness of each component in our proposed algorithms first determines parameter settings and performs parameter sensitivity analysis. Then we analyze the important ingredients of the proposed algorithms, including the VLSN integration in the solution-based tabu search framework, the strategic oscillation scheme, and the adaptive selection of solution improvement methods. All the experiments use the five classes of problem instances (Random, Max Biclique, Max Induced Graph, Max-Cut, and Matrix Factorization instances), which were first generated in Karapetyan and Punnen (2013) and have become the standard BBQP benchmarks in the literature (Duarte et al. 2014, Glover et al. 2015, Karapetyan et al. 2017). Each class includes five medium instances with $n = 1000$, $m = 200, 400, 600, 800, 1000$ and five large instances with $n = 5000$, $m = 1000, 2000, 3000, 4000, 5000$.

4.1. Parameter Settings and Sensitivity Analysis

Table 1 gives the descriptions and settings of the parameters used in our proposed algorithms. We set $r = 10$ as recommended in many population-based search algorithms (Wang et al. 2012, Chen et al. 2015) and tune the other parameters using the general IRACE automatic parameter configuration tool (López-Ibánez et al. 2011). The tuning was performed on a sample of 20 representative instances of varying sizes, selected from each of the five classes. For each parameter, IRACE requires a limited set of values as input to choose from the column "Considered values" in Table 1. Given that PR is the capstone algorithm of our paper and includes all the parameters, we use the PR algorithm in this parameter-tuning experiment and specify the total time budget for IRACE to be 2,000 PR executions. Each execution is set to be 100 seconds for a medium instance and 1,000 seconds for a large instance. The recommended parameter values from this calibration experiment are shown in the fifth column of Table 1.

Furthermore, we analyze the parameter sensitivity by conducting Friedman statistical tests to determine whether there is a significant difference in PR performance. To evaluate the sensitivity of each parameter, we test the considered values for each parameter while fixing the other parameters to their final value. For each instance and each parameter value, we run the PR algorithm and record the best objective value. The results of the Friedman tests shown in the last column of Table 1 indicate that varying parameters do not present significant differences with all p-values no less than 0.05. Hence, we conclude that the parameters in the PR algorithm present no particular sensitivity.

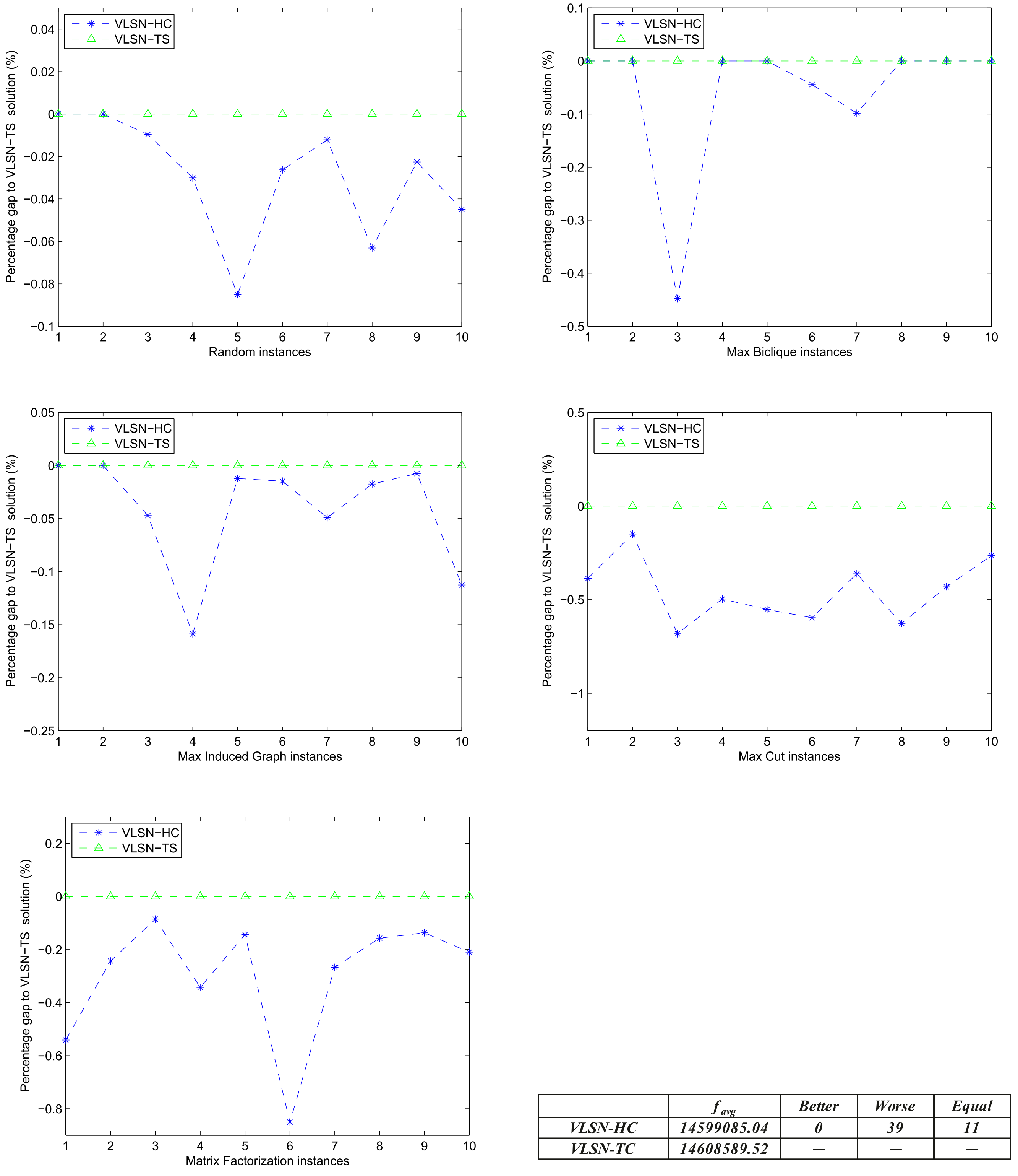
4.2. Impact of Integrating VLSN in the Solution-Based Tabu Search Framework

We assess the impact of integrating VLSN with TS by comparing our VLSN-TS with a VLSN-based hill climbing search VLSN-HC (the same as the Flip-float coordinate method in Glover et al. 2015). In VLSN-HC, we restart the algorithm each time the search is judged to be trapped in a local optimum.

Table 1. Parameter Settings of the PR Algorithm

Parameter	Section	Description	Considered values	Final value	p-value
tt	2.2.3	Tabu tenure	$\{0.01n, 0.02n, 0.03n, 0.04n, 0.05n\}$	$0.04n$	0.143
ρ_1	2.2.3	Iteration cutoff for SN-TS	$\{500, 1000, 1500, 2000, 2500\}$	1,500	0.328
ρ_2	2.2.4	Iteration cutoff for VLSN-TS	$\{50, 100, 150, 200, 250\}$	150	0.133
$span$	2.3	Oscillation amplitude	$\{0.01, 0.02, 0.03, 0.04, 0.05\}$	0.03	0.263
λ_1	3.2	Search depth of SO-SN-TS	$\{250000, 500000, 750000, 1000000, 1250000\}$	500,000	0.687
λ_2	3.2	Search depth of SO-VLSN-TS	$\{500, 1000, 1500, 2000, 2500\}$	1,000	0.300
η	3.5	Distance threshold coefficient	$\{0.01, 0.02, 0.03, 0.04, 0.05\}$	0.04	0.401
r	3.1	*RefSet* size	—	10	—

Figure 1. (Color online) Impact of Integrating VLSN in the Solution-Based Tabu Search Framework

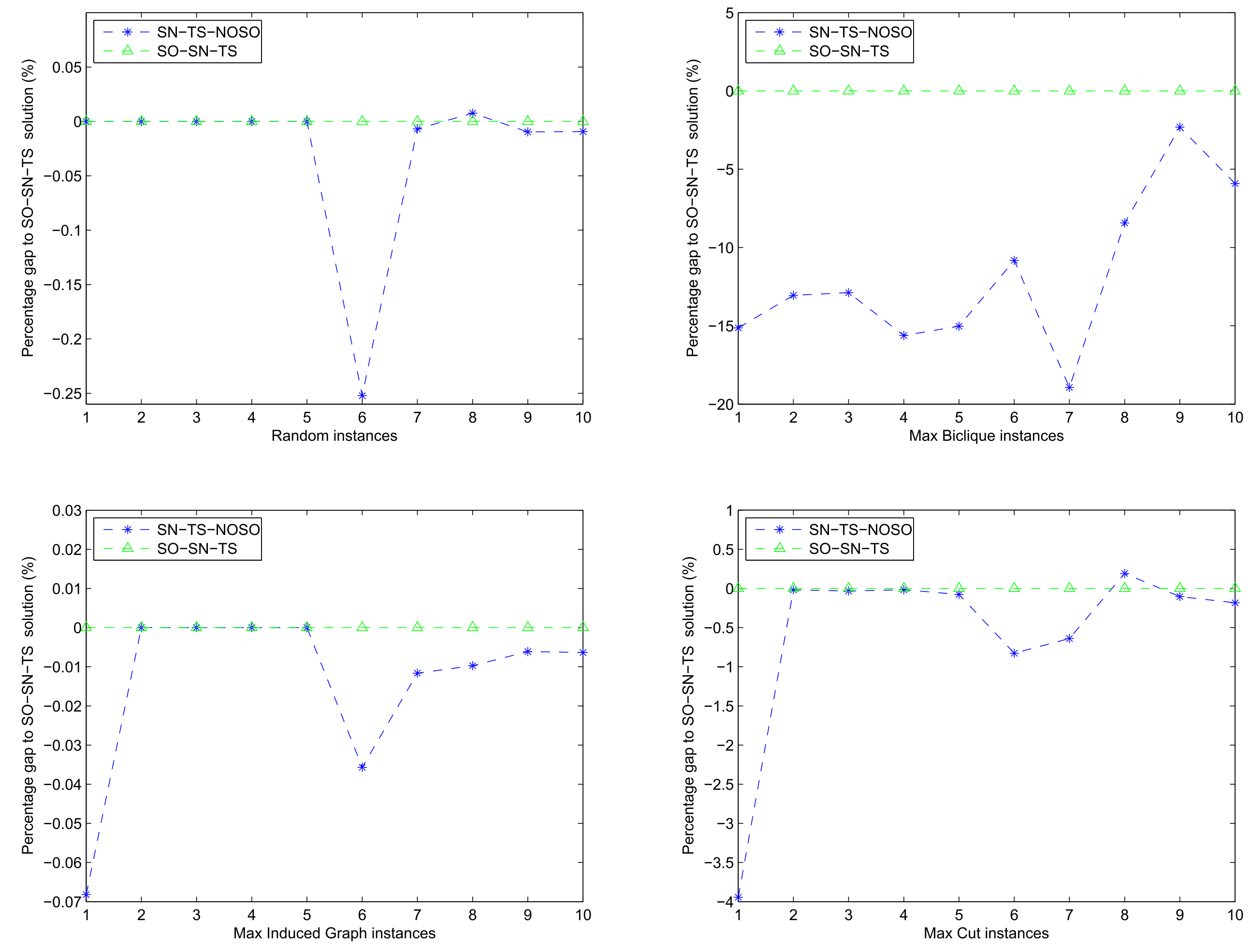


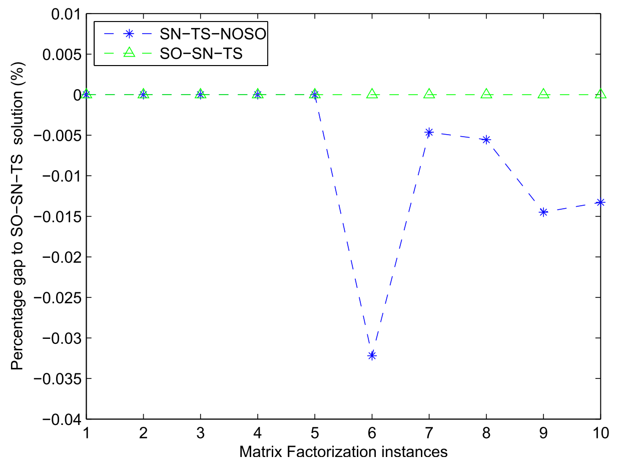
	f_{avg}	*Better*	*Worse*	*Equal*
VLSN-HC	*14599085.04*	*0*	*39*	*11*
VLSN-TC	*14608589.52*	—	—	—

Figure 1 summarizes the comparative results between VLSN-TS and VLSN-HC. Each of the first five subfigures shows the behavior of VLSN-HC relative to VLSN-TS on a specific class of problem instances. The horizontal axis indicates the instances in each class, which appear in the increasing order of the problem size. The vertical axis indicates the percentage gap of VLSN-HC relative to the solution of VLSN-TS. The last subfigure summarizes the average objective values of VLSN-TS and VLSN-HC over the 50 instances and the number of instances on which VLSN-HC obtains better, equal, and worse objective values than VLSN-TS.

From Figure 1, we observe that the percentage gap of VLSN-HC to VLSN-TS is nonpositive for each instance, which means that VLSN-HC cannot find better objective values than VLSN-TS. To be specific, VLSN-HC finds worse and equal objective values compared with VLSN-TS for 39 instances and 11 instances, respectively. Hence, this experiment discloses that our

Figure 2. (Color online) Effectiveness of the SO Mechanism in the SO-SN-TS Algorithm





	f_{avg}	*Better*	*Worse*	*Equal*
SN-TS-NOSO	*12832975.5*	*2*	*32*	*16*
SO-SN-TS	*13595704.6*	—	—	—

proposed tabu search framework is superior to the common hill climbing framework when the very large-scale neighborhood is used in local search.

4.3. Effectiveness of the Strategic Oscillation Mechanism

Our SO-TS algorithms include an informed strategic oscillation mechanism to escape from local tabu optima and locate promising regions. To verify its effectiveness, we remove the strategic oscillation component from the SO-SN-TS algorithm while keeping other components unchanged to produce the variant SN-TS-NOSO. We run experiments for SN-TS-NOSO and SO-SN-TS under the same experimental settings and show the results in Figure 2.

As can be seen from Figure 2, SO-SN-TS performs much better than the variant SN-TS-NOSO by finding better solutions for 32 instances and worse solutions for 2 instances. Moreover, the average objective value obtained by SO-SN-TS is 13,595,704.6, which is an improvement of 5.94% over the average objective value 12,832,975.5 obtained by SN-TS-NOSO. We also perform this experiment for the SO-VLSN-TS algorithm and obtain similar outcomes. Hence, we conclude that strategic oscillation plays an essential role to ensure high-quality solutions in our algorithms.

4.4. Effectiveness of the Adaptive Selection of Solution Improvement Methods

To illustrate the merit of adaptively selecting SO-SN-TS or SO-VLSN-TS to refine solutions, we test three additional PR versions of using different solution improvement methods. The first two us only SO-SN-TS (denoted by PR-SN) or SO-VLSN-TS (denoted by PR-VLSN), and the third one (denoted by PR-EqSel) selects SO-SN-TS and SO-VLSN-TS with an equal probability of 0.5.

Figure 3 shows experimental statistics of the four different PR versions. As shown in Figure 3, the average objective value over the 50 instances obtained by our PR algorithm is 14,667,587.52, which is the best compared with 14,666,442.94 of PR-EqSel, 14,659,385.8 of PR-VLSN, and 14,332,660.78 of PR-SN. Furthermore, PR finds better and worse objective values than PR-EqSel, PR-VLSN, and PR-SN for 11 against 2 instances, 14 against 2 instances, and 12 against 8 instances, respectively. These findings highlight the benefits of the proposed adaptive selection of solution improvement methods.

5. Experimental Results

In this section, we evaluate our proposed SO-SN-TS, SO-VLSN-TS, and PR algorithms and provide detailed comparisons between our best algorithm and state-of-the-art algorithms in the literature using different running time.

5.1. Comparisons Among Our Proposed Algorithms

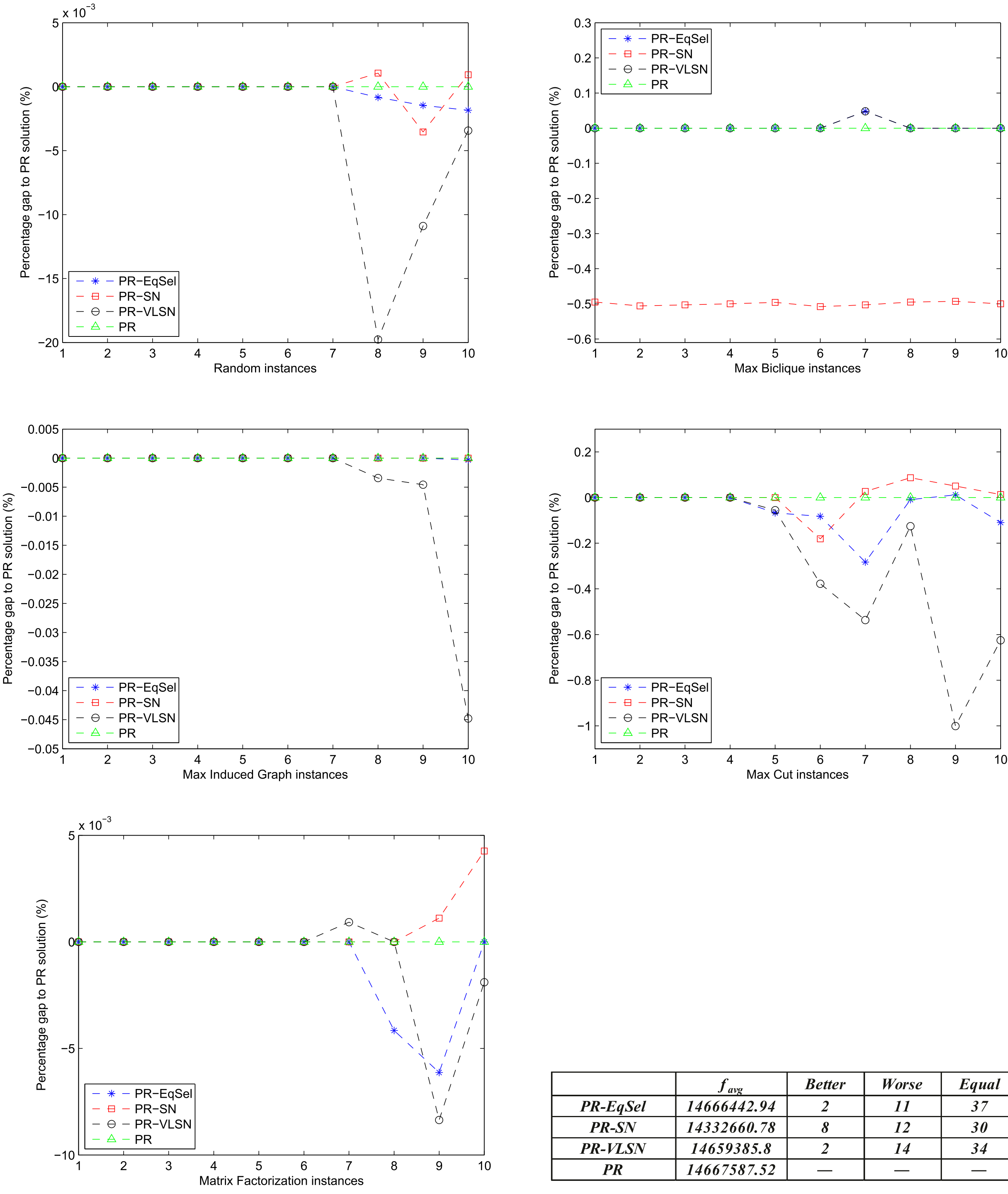
We first test our SN-TS, VLSN-TS, and PR algorithms using a long running time (LT), which is set to be 1,000 seconds for each medium instance and 10,000 seconds for each large instance. These stopping criteria are adopted in the reference algorithms HM (Glover et al. 2015) and CMCS (Karapetyan et al. 2017). Table 10 in the online supplement shows the computational results of our proposed algorithms, where columns 1 and 2, respectively, give the instance names and the previous best known results *BKR* (Karapetyan and Punnen 2013, Duarte et al. 2014, Glover et al. 2015, Karapetyan et al. 2017). The consecutive three columns under the headings SO-SN-TS, SO-VLSN-TS, and PR list the best solution value *Bst* found by each algorithm, the gap between *Bst* and *BKR*, and the time (in seconds) to reach *Bst*. To observe differences among the algorithms, we summarize in Table 2 the statistics over the medium set of benchmarks and the large set of benchmarks, respectively, including the number of instances for which each algorithm gets results that are better than, equal to, and worse than *BKR*, the average objective value f_{avg}, and the average time.

For the medium set of benchmarks, we observe that our PR algorithm is able to discover a new best solution for the instance BMaxCut 1,000 × 1,000 and matches the previous best known results for the other instances. In contrast to PR, which successfully reaches the best solutions for all the instances, SO-SN-TS and SO-VLSN-TS fail to reach the previous best known results for 9 instances and 5 instances, respectively. In addition, PR obtains the best average objective value of 1,782,132.52 and is slightly faster than SO-SN-TS and SO-VLSN-TS.

For the large set of benchmarks, PR once again performs better than SO-SN-TS and SO-VLSN-TS, in this case dramatically so. PR obtains new best solutions for 16 of the 25 large instances and matches the previous best for all the remaining 9 instances. SO-SN-TS finds new best solutions for 6 instances, although its new best results are not as good as those obtained by PR. SO-VLSN-TS performs particularly well for solving the Max Biclique instances. Finally, PR consumes the roughly same computational time as SO-SN-TS and SO-VLSN-TS to obtain the best average objective value of 27553042.52.

We also test the SO-SN-TS, SO-VLSN-TS, and PR algorithms using a shorter running time and present results in Table 11 in the online supplement. These experimental results reveal the effectiveness of our proposed algorithms in finding high-quality solutions and disclose in particular the efficacy of our PR algorithm.

Figure 3. (Color online) Effectiveness of the Adaptive Selection of Solution Improvement Methods



	f_{avg}	*Better*	*Worse*	*Equal*
PR-EqSel	*14666442.94*	*2*	*11*	*37*
PR-SN	*14332660.78*	*8*	*12*	*30*
PR-VLSN	*14659385.8*	*2*	*14*	*34*
PR	*14667587.52*	—	—	—

5.2. Comparisons Between Our PR Algorithm and State-of-the-Art Algorithms Using the Long Running Time

In this section, we compare our PR algorithm with the best performing algorithms in the literature, including the hybrid method (HM) in Glover et al. (2015), conditional Markov chain search (CMCS) in Karapetyan et al. (2017), and the best iterated local search (ILS) variant in Duarte et al. (2014). Our PR algorithm is coded in C++ and run on a Xeon E5440 (2.83-GHz

Table 2. Comparisons Among SO-SN-TS, SO-VLSN-TS, and PR Using a Long Running Time

	Medium benchmarks			Large benchmarks		
Measures	SO-SN-TS	SO-VLSN-TS	PR	SO-SN-TS	SO-VLSN-TS	PR
Better	0	0	1	6	0	16
Equal	16	20	24	0	6	9
Worse	9	5	0	19	19	0
f_{avg}	1,716,226.56	1,779,421.20	1,782,132.52	25,475,182.68	27,468,587.88	27,553,042.52
Time	194.57	189.08	145.15	4,446.55	4,588.84	4,645.31

Table 3. Comparions Between PR and HM Using a Long Running Time

	Medium benchmarks		Large benchmarks	
Measures	PR	HM	PR	HM
Better	1	0	16	1
Equal	24	17	9	0
Worse	0	8	0	24
f_{avg}	1,782,132.52	1,775,815.08	27,553,042.52	27,278,641.68
Time	145.15	198.57	4,645.31	5,868.19

Table 4. Comparions Between PR and CMCS Using a Long Running Time

	Medium benchmarks		Large benchmarks	
Measures	PR	CMCS	PR	CMCS
Better	0	0	11	0
Equal	24	16	10	1
Worse	1	9	4	24
f_{avg}	1,782,080.44	1,765,401.36	27,550,708.72	26,515,447.76
Time	116.89	312.51	4,902.96	4,479.24

central processing unit [CPU] and 8 GB random access memory [RAM]), whereas HM is coded in C++ and run on a Xeon E5-2687W (3.1-GHz CPU and 128 GB RAM), CMCS is coded in C# and run on a Xeon E5-2630 v2 (2.6-GHz CPU and 32 GB RAM), and ILS is coded in Java and run on an Intel Core i7 2600 (3.4-GHz CPU and 4 GB RAM). To make fair comparisons between our PR algorithm and each reference algorithm, we account for the differences of programming languages and computing platforms.

We use the source code of HM and have rerun HM for the long running time (LT) on our computing platform. We summarize the results in Table 3 and show detailed results in Table 10 in the online supplement. The results disclose that PR surpasses or matches the previous best known results for all the 25 medium instances, performing better than, HM which fails to match the previous best results for 8 instances. Moreover, PR yields better results than HM for all the 25 large instances. Statistical testing indicates that PR performs significantly better than HM by obtaining a p-value of 5.645e-07. Finally, the computational time taken by PR to reach better solutions is slightly shorter than HM.

We have reimplemented our PR algorithm in C# and compared it with CMCS[1] by running both for the long running time (LT) on our computing platform. We summarize the comparative results in Table 4 and list detailed results in Table 12 in the online supplement. We observe that PR performs better than CMCS with respect to the best objective value. For the medium set of benchmarks, PR fails to reach the previous best known results for 1 instance but CMCS fails for 9 instances. For the large set of benchmarks, PR fails to reach the previous best known results for 4 instances, much better than CMCS, which fails for 24 instances. Statistical testing indicates that PR is significantly better than CMCS by obtaining a p-value of 5.645e-07. Finally, the computational time consumed by PR to reach better results is shorter than CMCS for

Table 5. Comparisons Between PR and ILS Using a Running Time of 1,000 Seconds

	Medium benchmarks		Large benchmarks	
Measures	PR	ILS	PR	ILS
Better	1	0	6	0
Equal	24	0	4	0
Worse	0	25	15	25
f_{avg}	1,782,132.52	1,755,263.32	27,531,287.88	26,919,959.04
Time	113.11	9.60	725.04	310.44

Table 6. Comparisons Between PR and HM Using a Short Running Time: Number of Instances Where the Algorithm Yields Matched or Better Results Than the Best Known Results

	PR				HM			
Benchmarks	U-ST	ST	MT	LT	U-ST	ST	MT	LT
Rand	1	5	7	10	1	0	2	5
Biclique	0	0	5	10	0	0	0	2
MaxInduced	2	4	6	10	0	4	4	5
BMaxCut	0	0	1	10	0	0	0	1
MatrixFactor	3	5	7	10	0	3	4	5
Total	6	14	26	50	1	7	10	18

Table 7. Comparisons Between PR and HM Using a Short Running Time: Average Objective Values over the 10 Instances in Each Problem Class

	PR				HM			
Benchmarks	U-ST	ST	MT	LT	U-ST	ST	MT	LT
Rand	7,804,316.5	7,812,095.0	7,812,698.5	7,813,423.7	7,799,257.2	7,807,912.2	7,809,313.8	7,810,808.6
Biclique	38,027,456.4	51,789,799.4	52,031,395.8	52,067,255.4	38,054,121.6	51,618,874.9	51,787,468.8	51,834,157.3
MaxInduced	5,914,618.8	5,919,908.2	5,921,822.8	592,2592.2	5,909,782.4	5,919,478.1	5,920,654.0	5,921,820.2
BMaxCut	7,239,406.0	7,378,535.6	7,415,300.8	7,456,896.4	7,208,208.4	7,322,279.2	7,369,307.4	7,390,571.0
MatrixFactor	77,690.0	77,750.9	77,765.8	77,769.9	77,645.7	77,717.8	77,734.8	77,746.9
Average	11,812,697.54	14,595,617.82	14,651,796.74	14,667,587.52	11,809,803.06	14,549,252.44	14,592,895.76	14,607,020.80

medium instances and comparable to CMCS for large instances.

Owing to the unavailability of the ILS source codes, we reimplemented our PR algorithm in Java and have run it on our computing platform using the same time limit of 1,000 seconds used by ILS for all the instances. Although Duarte et al. (2014) did not report the best objective value and computational time for each instance, the authors sent us these results via email for computational comparisons. We report the summary of the results in Table 5 and report detailed results in Table 13 in the online supplement. From the comparisons, we find that ILS fails on all 50 instances to match the previous best known results. In contrast, PR finds better results for 1 medium instance and 6 large instances, and matches previous best results for 24 medium instances and 4 large instances. Statistical testing indicates that PR is significantly better than ILS by obtaining a p-value of 7.789e-10. However, the computational time required by PR to reach the high-quality solutions is longer than ILS for most instances.

5.3. Comparisons Between Our PR Algorithm and State-of-the-Art Algorithms Using a Short Running Time

To evaluate the performance of different algorithms under the imposition of shorter running time, we test PR, HM, and CMCS in an ultra-short running time (U-ST), a short running time (ST), and a medium running time (MT), which are set to be 1 second, 10 seconds, and 100 seconds for each medium instance and 10 seconds, 100 seconds, and 1,000 seconds for each large instance. We also extracted the results of the long running time (LT) from Tables 10 and 12 for comparison. We were not able to run ILS because the source code is unavailable. To avoid an excess of tabulated results, we summarize in Tables 6–9 the following two measures in terms of each problem class (1) the number of instances where the algorithm yields results that match or are superior to the best known results and (2) the average objective values over the 10 instances.

Tables 6 and 7 show the comparisons between PR and HM in U-ST, ST, MT, and LT. We first observe that the performances of both algorithms gradually drop as the running time becomes shorter. Over the 50 tested instances from five problem classes, PR matches or surpasses the best known results for 50 against 18 instances in LT, 26 against 10 instances in MT, 14 against 7 instances in ST, and 6 against 1 instances in U-ST compared with HM. The average objective values obtained by PR and HM are 16,447,587.2 against 14,607,020.80 in LT, 14,651,796.74 against 14,592,895.76 in MT, 14,595,617.82 against 14,549,252.44 in ST, and 11,812,697.54 against 11,809,803.06 in U-ST. Although PR is generally superior to HM under varying running time limits, HM performs better than PR for the Biclique problem instances in the ultra-short running time in terms of the average objective values.

Tables 8 and 9 show the comparisons between PR and CMCS in U-ST, ST, MT, and LT. Similar observations can be found from the two tables, which show that the performances of both PR and CMCS gradually drop as the running time becomes shorter. For example, CMCS matches or surpasses the best known results for 17 instances, 14 instances, 10 instances, and 3 instances, whereas PR obtains the average objective values of 14,666,394.58, 14,654,283.92, 14,566,786.72, and 11,809,683.98 in LT, MT, ST, and U-ST, respectively. Moreover, PR is superior to CMCS in ST, MT, and LT and is as good as CMCS in U-ST in terms of the first measure. In terms of the second measure, PR

Table 8. Comparisons Between PR and CMCS Using a Short Running Time: Number of Instances Where the Algorithm Yields Matched or Better Results than the Best Known Results

	PR				CMCS			
Benchmarks	U-ST	ST	MT	LT	U-ST	ST	MT	LT
Rand	1	4	6	10	1	3	3	6
Biclique	0	2	3	10	0	1	1	1
MaxInduced	0	4	6	9	0	3	5	3
BMaxCut	0	0	1	7	0	0	1	2
MatrixFactor	2	4	6	9	2	3	4	5
Total	3	14	22	45	3	10	14	17

Table 9. Comparisons Between PR and CMCS Using a Short Running Time: Average Objective Values Over the 10 Instances in Each Problem Class

	PR				CMCS			
Benchmarks	U-ST	ST	MT	LT	U-ST	ST	MT	LT
Rand	7,806,438.1	7,810,631.8	7,812,630.3	7,813,430.3	7,798,750.1	7,804,786.4	7,807,020.0	7,810,346.1
Biclique	38,025,553.9	51,668,898.5	52,051,500.7	52,067,255.4	46,276,097.1	47,784,151.9	49,450,561.1	49,589,917.3
MaxInduced	5,913,109.2	5,917,949.7	5,921,745.5	5,922,542.8	5,908,356.9	5,913,803.5	5,920,271.8	5,919,725.4
BMaxCut	7,225,645.8	7,358,705.8	7,407,783.6	7,450,975.2	7,101,788.4	7,238,968.0	7,248,887.2	7,304,404.8
MatrixFactor	77,672.9	77,747.8	77,759.5	77,769.2	77,566.6	77,672.3	77,718.4	77,729.2
Average	11,809,683.98	14,566,786.72	14,654,283.92	14,666,394.58	13,432,511.82	13,763,876.42	14,100,891.70	14,140,424.56

dominates CMCS except for the Biclique problem instances in the ultra-short running time.

To conclude, when short running time limits are permitted, our PR algorithm remains competitive and outperforms the reference algorithms HM and CMCS.

6. Conclusions

We introduce and test several new strategies that integrate strategic oscillation and path relinking under the guidance of tabu search for solving the bipartite Boolean quadratic programming problem. First, the strategic oscillation mechanism based on the combination of frequency and problem-specific knowledge is demonstrated to be highly effective to improve performance of the proposed algorithms. Second, the solution-based tabu strategy is able to identify the tabu status of very-large scale neighborhood moves to improve the outcomes obtained by integrating these moves in the common hill climbing search. Third, an adaptive selection mechanism designed to favor the use of a solution improvement method enhances the quality of solutions produced.

Comprehensive experimental studies on five classes of benchmarks with a total of 50 instances disclose that our proposed PR algorithm dominates the recent state-of-the-art algorithms in the literature by discovering improved best solutions for 17 instances and matching the previous best solutions for all other instances.

Our findings invite future studies that investigate the benefits of using tabu search algorithms incorporating analogous strategic oscillation and path relinking components for solving other combinatorial optimization problems. Open questions include the automatic determination of specific parameter values and the timing for examining points in the relinked paths other than the midway point. Other areas for future exploration to augment our current approach include strategic oscillation procedures that interrupt the constructive or destructive phases at various junctures to seek improved solutions by using fewer iterations than devoted to modifying the final solution of the current oscillation process, where improvement at these junctures may be defined by a diversification objective as well as by the original objective function.

Acknowledgments

The authors thank the editors and reviewers, whose comments have helped to improve their paper.

Endnote

[1] See http://csee.essex.ac.uk/staff/dkarap/?page=publications&key=CMCS-BBQP.

References

Ahuja RK, Ergun O, Orlin JB, Punnen AP (2002) A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* 123(1–3):75–102.

Alon N, Naor A (2006) Approximating the cut-norm via Grothendieck's inequality. *SIAM J. Comput.* 35(4):787–803.

Ambühl C, Mastrolilli M, Svensson O (2011) Inapproximability results for maximum edge biclique, minimum linear arrangement, and sparsest cut. *SIAM J. Comput.* 40(2):567–596.

Aringhieri R, Cordone R (2011) Comparing local search metaheuristics for the maximum diversity problem. *J. Oper. Res. Soc.* 62(2):266–280.

Benlic U, Hao J (2011) An effective multilevel tabu search approach for balanced graph partitioning. *Comput. Oper. Res.* 38(7):123–137.

Carlton WB, Barnes JW (1996) A note on hashing functions and tabu search algorithms. *Eur. J. Oper. Res.* 95(1):237–239.

Chang WC, Vakati S, Krause R, Eulenstein O (2012) Exploring biological interaction networks with tailored weighted quasi-bicliques. *BMC Bioinformatics* 13(Suppl 10):S16.

Chen Y, Hao JK, Glover F (2015) An evolutionary path relinking approach for the quadratic multiple knapsack problem. *Knowledge-Based Systems* 92:23–34.

Duarte A, Laguna M, Martí R, Sánchez-Oro J (2014) Optimization procedures for the bipartite unconstrained 0-1 quadratic programming problem. *Comput. Oper. Res.* 51:123–129.

Gallego M, Laguna M, Martí R, Duarte A (2013) Tabu search with strategic oscillation for the maximally diverse grouping problem. *J. Oper. Res. Soc.* 64(5):724–734.

Gillis N, Glineur F (2011) Low-rank matrix approximation with weights or missing data is np-hard. *SIAM J. Matrix Anal. Appl.* 32(4):1149–1165.

Glover F (1977) Heuristics for integer programming using surrogate constraints. *Decision Sci.* 8(1):156–166.

Glover F (1997) A template for scatter search and path relinking. Hao JK, Lutton E, Ronald E, Schoenauer M, Snyers D, eds. *Artificial Evolution,* Lecture Notes in Computer Science, vol. 1363 (Springer, Berlin, Heidelberg), 1–51.

Glover F (2000) Multi-start and strategic oscillation methods—Principles to exploit adaptive memory. *Comput. Tools Model. Optim. Simulation* 12:1–23.

Glover F, Hao J (2010) Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems. *Internat. J. Metaheuristics* 1(1):3–10.

Glover F, Laguna M (1997) *Tabu Search* (Kluwer, Alphen aan den Rign, Netherlands).

Glover F, Kochenberger G, Alidaee B (1998) Adaptive memory tabu search for binary quadratic programs. *Management Sci.* 44(3): 336–345.

Glover F, Laguna M, Martí R (2004) Scatter search and path relinking: Foundations and advanced designs. Onwubolu GC, Babu BV, eds. *New Optimization Techniques in Engineering*, Studies in Fuzziness and Soft Computing, vol. 141, (Springer, Berlin, Heidelberg), 87–99.

Glover F, Ye T, Punnen AP, Kochenberger G (2015) Integrating tabu search and vlsn search to develop enhanced algorithms: A case study using bipartite boolean quadratic programs. *Eur. J. Oper. Res.* 241(3):697–707.

Karapetyan D, Punnen AP (2013) Heuristic algorithms for the bipartite unconstrained 0-1 quadratic programming problem. Working paper, University of Essex, Colchester, UK.

Karapetyan D, Punnen AP, Parkes AJ (2017) Markov chain methods for the bipartite boolean quadratic programming problem. *Eur. J. Oper. Res.* 260(2):494–506.

Koyutürk M, Grama A, Ramakrishnan N (2006) Nonorthogonal decomposition of binary matrices for bounded-error data compression and analysis. *ACM Trans. Math. Software* 32(1):33–69.

Laguna M, Martí R (1999) Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS J. Comput.* 11(1): 44–52.

López-Ibánez M, Dubois-Lacoste J, Stützle T, Birattari M (2011) The irace package: Iterated racing for automatic algorithm configuration. Technical Report, TR/IRIDIA/2011-004, Université Libre de Bruxelles, IRIDIA, Bruxelles, Belgium.

Lü Z, Glover F, Hao JK (2010) A hybrid metaheuristic approach to solving the ubqp problem. *Eur. J. Oper. Res.* 207(3):1254–1262.

Martí R, Duarte A, Laguna M (2009) Advanced scatter search for the max-cut problem. *INFORMS J. Comput.* 21(1):26–38.

Peng B, Lü Z, Cheng T (2015) A tabu search/path pelinking algorithm to solve the job shop scheduling problem. *Comput. Oper. Res.* 53:154–164.

Punnen AP, Sripratak P, Karapetyan D (2015a) Average value of solutions for the bipartite boolean quadratic programs and rounding algorithms. *Theoret. Comput. Sci.* 565:77–89.

Punnen AP, Sripratak P, Karapetyan D (2015b) The bipartite unconstrained 0-1 quadratic programming problem: Polynomially solvable cases. *Discrete Appl. Math.* 193:1–10.

Resende MGC, Ribeiro CC, Glover F, Martí R (2010) Scatter search and path relinking: Fundamentals, advances and applications. *Internat. Ser. Oper. Res. Management Sci.* 146:87–107.

Shen BH, Ji S, Ye J (2009) Mining discrete patterns via binary matrix factorization. *Proc. 15th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 757–766.

Tanay A, Sharan R, Shamir R (2002) Discovering statistically significant biclusters in gene expression data. *Bioinformatics* 18(Suppl 1): S136–S144.

Wang Y, Lü Z, Glover F, Hao JK (2012) Path relinking for unconstrained binary quadratic programming. *Eur. J. Oper. Res.* 223(3):595–604.

Woodruff D, Zemel E (1993) Hashing vectors for tabu search. *Ann. Oper. Res.* 41(2):123–137.

Yagiura M, Toshihide I, Glover F (2006) A path relinking approach with ejection chains for the generalized assignment problem. *Eur. J. Oper. Res.* 169(2):548–569.