European Journal of Operational Research 223 (2012) 595–604

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Path relinking for unconstrained binary quadratic programming

Yang Wang [a], Zhipeng Lü [b], Fred Glover [c], Jin-Kao Hao [a,*]

[a] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
[b] School of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, China
[c] OptTek Systems, Inc., 2241 17th Street Boulder, CO 80302, USA

ARTICLE INFO

Article history:
Received 8 June 2011
Accepted 6 July 2012
Available online 20 July 2012

Keywords:
UBQP
Path relinking
Tabu search
MaxCut

ABSTRACT

This paper presents two path relinking algorithms to solve the unconstrained binary quadratic programming (UBQP) problem. One is based on a greedy strategy to generate the relinking path from the initial solution to the guiding solution and the other operates in a random way. We show extensive computational results on five sets of benchmarks, including 31 large random UBQP instances and 103 structured instances derived from the MaxCut problem. Comparisons with several state-of-the-art algorithms demonstrate the efficacy of our proposed algorithms in terms of both solution quality and computational efficiency. It is noteworthy that both algorithms are able to improve the previous best known results for almost 40 percent of the 103 MaxCut instances.

1. Introduction

The objective of the unconstrained binary quadratic programming (UBQP) problem is to maximize the function:

$$f(x) = x'Qx = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_ix_j \qquad (1)$$

where $Q = (q_{ij})$ is an n by n matrix of constants and x is an n-vector of binary (zero-one) variables, i.e., $x_i \in \{0,1\}$, $i = 1, \ldots, n$.

The formulation of UBQP can represent a wide range of important problems, including those from financial analysis [28], social psychology [20], computer aided design [25] and cellular radio channel allocation [9]. Moreover, a quite number of combinatorial optimization problems can be transformed into UBQP, such as graph coloring problem, maxcut problem, set packing problem, set partitioning problem, maximum clique problem, etc. Interested readers can refer to [23] for the general transformation procedures.

Given the interest of UBQP, many solution procedures have been reported in the literature during the past few decades. Exact methods based on branch and bound or branch and cut [6,21,35] are quite useful to obtain optimal solutions to instances of limited sizes. To handle larger instances, a number of heurisric and metaheuristic methods have been developed, including local search [7], Simulated Annealing [4,22], Tabu Search [14,19,32,34,37,38], and Evolutionary and Memetic Algorithms [5,26,27,30,31].

Among the existing heuristics, tabu search (TS) based algorithms are the most successful ones. For example, the first adaptive memory tabu search algorithm for the UBQP [14] has been used to solve applications coming from a wide variety of settings. Also, several multi-start tabu search strategies have been explored in [32] and a sequel using an iterated tabu search algorithm has been investigated in [34], leading to very good results on large and challenging UBQP random instances. More recently, the diversification-driven tabu search method [19], a memetic algorithm [27] using embedded tabu search and a variable fixing tabu search method [37,38] have proved to be especially effective for solving the most challenging UBQP instances.

Although numerous algorithms and approaches have been proposed for this well-known problem, we are not aware of any study on applying path relinking to the UBQP in the literature. 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 [15–17], with applications in a variety of contexts where it has proved to be very effective in solving difficult problems. In this paper, we follow the general scheme described in [17] and propose two path relinking algorithms for the UBQP. These two algorithms differ from each other mainly on the way of generating the path, one employing a greedy strategy and the other employing a random construction. In order to assess the performance of our path relinking algorithms, we provide computational results on five sets of random and structured benchmarks with a total of 134 test instances. These results indicate that our proposed algorithms yield highly competitive outcomes on the tested instances.

The remaining part of the paper is organized as follows. Section 2 briefly reviews some representative approaches for the UBQP.

* Corresponding author. Tel.: +33 2 41 73 50 76.
E-mail addresses: yangw@info.univ-angers.fr (Y. Wang), zhipeng.lui@gmail.com, zhipeng.lv@hust.edu.cn (Z. Lü), glover@opttek.com (F. Glover), hao@info.univ-angers.fr (J.-K. Hao).

0377-2217/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2012.07.012

Section 3 describes the ingredients of our path relinking algorithms. Section 4 presents computational results and detailed comparisons with other state-of-the-art algorithms in the literature. Section 5 discusses the results obtained on two other well-known combinatorial problems. Concluding remarks are given in Section 6.

2. Previous work

This section reviews some representative heuristic approaches for the UBQP, including in particular those that are used as the reference methods for our experimental evaluation.

Glover et al. [14] introduced the first tabu search algorithm for the UBQP (AMTS). AMTS is based on the one-flip move and two types of memory structures to record recency and frequency information. 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. The amplitude of the scillation is adaptively controlled by a span parameter. Computational results for instances with up to 500 variables show AMTS outperforms the best exact and heuristic methods previously reported in the literature.

Katayama and Narihisa [22] designed a simulated annealing algorithm (SA) that is also based on the one-flip move and an incremental neighborhood evaluation technique. To enhance its search ability, the SA algorithm adopts multiple annealing processes starting from different temperatures. Tested on instances with variables ranging from 500 to 2500, the proposed SA heuristic shows very competitive performances, particularly for the largest instances.

Merz and Katayama [31] conducted a landscape analysis and observed that local optima of the UBQP instances are contained in a small fraction of the search space. Based on this, they designed a memetic algorithm (MA) in which a dedicated crossover operator is utilized to generate good starting solutions for a k-opt local search. The proposed approach is remarkably effective in solving a set of problems with up to 2500 variables.

Palubeckis [32] presented several multistart tabu search strategies (MST) dedicated to the construction of the initial solution. An additional set of challenging random instances with up to 7000 variables were generated to evaluate the proposed MST algorithms. Subsequently, Palubeckis [34] developed an iterated tabu search algorithm (ITS) in which the perturbation mechanism operates on a specific set of variables. The experimental results indicated that the ITS consumes less computational effort to find the best solutions than several MSTS algorithms.

Glover et al. [19] presented a diversification-driven tabu search (D^{2}TS) algorithm that alternates between a basic tabu search procedure and a memory-based perturbation strategy guided by a long-term memory. Despite its simplicity, computational results showed that D^{2}TS is capable of matching or improving the previously reported results for the challenging instances introduced in [32].

Lü et al. [27] proposed a hybrid metaheuristic approach (HMA) which combines a basic tabu search procedure and the genetic search framework. HMA is characterized by its diversification-guided recombination operator and quality-and-distance-based population updating strategy. The dedicated recombination operator aims to generate diversified offspring solutions in order to explore new promising search regions while the tabu search procedure is responsible for intensified examination around the offspring solutions. Computational results showed HMA is among the current best performing procedures on the UBQP benchmark instances.

3. Path relinking algorithm

3.1. Main framework

Algorithm 1. Outline of the path relinking procedure

```
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:    $Tag(i) = \text{TRUE}, (i = \{1, \ldots, b\})$
7:    $PairSet \leftarrow \{(i,j): x^i, x^j \in RefSet, x^i \neq x^j, Tag(i) \cup Tag(j) = \text{TRUE}\}$
8:    $Tag(i) = \text{FALSE}, (i = \{1, \ldots, b\})$
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 the improvement method to $x(m)$
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:       $Tag(w) = \text{TRUE}$
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 the improvement method to $y(n)$
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:       $Tag(w) = \text{TRUE}$
29:       Record the new worst solution $x^w$ in $RefSet$
30:     end if
31:     $PairSet \leftarrow PairSet \setminus (i,j)$
32:   end while
33: until the stopping criterion is satisfied
```

Algorithm 1 shows the path relinking procedure for UBQP. It starts with the creation of an initial set of b elite solutions *RefSet* (line 4, see Section 3.2) 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 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, see Section 3.5). Secondly, one solution $x(m)$ on each path is selected to be further improved by the Improvement Method (lines 12, 22, see Section 3.3). The next step tests *Update_RefSet* to decide if the new improved solution is used to update *RefSet* (lines 16, 26, see Section 3.4). 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, see Section 3.4). 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.

3.2. The RefSet initialization method

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 our improvement method to obtain a local optimum (see Section 3.3). 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.

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.

3.3. The improvement method

The improvement method employs a basic tabu search procedure that is implemented in the same way as the tabu search component of the hybrid metaheuristic approach (HMA) [27]. Specifically, it employs a simple *one-flip move* neighborhood, which consists of changing (flipping) the value of a single variable x_i to its complementary value $1 - x_i$. Each time a move is carried out, the reverse move is forbidden for the next *TabuTenure* iterations [13]. In practice, we elected to set the tabu tenure by $TabuTenure(i) = ttc + rand(10)$, where ttc is a selected constant and rand(10) takes a random value from 1 to 10. Once a move is performed, we update a subset of move values affected by the move using a fast incremental evaluation technique introduced in [18]. 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 TS procedure stops when the best solution cannot be improved within a given number μ of moves that called *improvement cutoff*.

3.4. The RefSet update method

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.

3.5. The relinking method

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 [15–17]. 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 UBQP problem) while the other operates in a random manner. Algorithms 2 and 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 2. Pseudo-code of Relinking Method 1

```
1:   Input: A pair of solutions $x^i$ and $x^j$
2:   Output: Path solution $x(1), \ldots, x(r)$ from $x^i$ to $x^j$
3:   Identify the set $NC$ between $x^i$ and $x^j$
4:   Initialize the $\Delta_t$ assignments for $t \in NC$
5:   $PV = \emptyset$, $FI_0 = 0$, $r = |NC| - 1$
6:   for $k = 1$ to $r$ then
7:     Find a $t \in NC$ with the best $\Delta_t$ value
8:     $PV \leftarrow PV \cup \{t\}$
9:     $x(k) = \{x_u : x_u = x_u^j, u \in PV; x_u = x_u^i, u \in N \setminus PV\}$
10:    $FI_k = FI_{k-1} + \Delta_t$
11:    $f(x(k)) = f(x^i) + FI_k$
12:    Update all $\Delta_t$ values ($t \in NC$) affected by the move
13:    $NC \leftarrow NC \setminus \{t\}$
14:  end for
```

Algorithm 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 [18]. 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 3. Pseudo-code of Relinking Method 2

```
1:  **Input**: A pair of solutions $x^i$ and $x^j$
2:  **Output**: Path solution $x(1), \ldots, x(r)$ from $x^i$ to $x^j$
3:  Identify the set $NC$ between $x^i$ and $x^j$
4:  Initialize the $\Delta_t$ assignments for $t \in NC$
5:  $PV = \emptyset$, $FI_0 = 0$, $r = |NC| - 1$
6:  **for** $k = 1$ to $r$ **do**
7:      Select a $t \in NC$ at random
8:      $PV \leftarrow PV \cup \{t\}$
9:      $x(k) = \{x_u : x_u = x_u^j, u \in PV; x_u = x_u^i, u \in N \setminus PV\}$
10:     $FI_k = FI_{k-1} + \Delta_t$
11:     $f(x(k)) = f(x^i) + FI_k$
12:     Update all $\Delta_t$ values ($t \in NC$) affected by the move
13:     $NC \leftarrow NC \setminus \{t\}$
14: **end for**
```

The second relinking method, shown in Algorithm 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 2.

3.6. 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). The solution with the highest quality in CSL is picked for further amelioration by the improvement method.

4. Computational results

In this section, we report extensive computational results of our two path relinking algorithms on a large collection of various benchmark instances and compare our results with those of several state-of-the-art methods in the literatures.

4.1. Test instances

Five sets of test problems are considered in the experiments, amounting to 134 instances. The first set of benchmarks is composed of 10 largest instances of size $n = 2500$ from the ORLIB [3]. They all have a density of 0.1 and are named by b2500.1, …, b2500.10. These instances are frequently used in the literature by many authors, see for instance [4,22,30–32,34,19,27].

The second set of benchmarks consists of 21 randomly generated large problem instances named p3000.1, …, p7000.3 with sizes ranging from $n = 3000$ to 7000 and with densities from 0.5 to 1.0.[1] Experiments reported in [32,34,19,27,37,38] show that these large instances are particularly challenging UBQP problems, especially in the case of instances with more than 5000 variables.

The third set of benchmarks includes 69 instances derived from the MaxCut problem, named G1, …, G72, with variable sizes ranging from $n = 800$ to 10000.[2] These instances are created by using a machine-independent graph generator, composed of toroidal, planar and random weighted graphs with weight values 1, 0 or −1. The first 54 instances have been employed by numerous authors to test their algorithms [8,12,29,33,36] and the results for the remaining 15 larger instances are reported in [10].

The fourth set of benchmarks contains 30 instances with size $n = 128$ (named G54100, …, G541000), $n = 1000$ (named G10100, …, G101000) and $n = 2744$ (named G14100, …, G141000), respectively.[3] These instances are created from cubic lattices modeling Ising spin glasses with weight values 1, 0 or −1. Computational results on these instances were reported in [8,12,29,33,36].

The last set is composed of 4 DIMACS instances containing from 512 to 3375 vertices and 1536 to 10,125 edges.[4]

4.2. Experimental protocol

Our path relinking (PR) algorithms are programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.83 GHz CPU and 8 GB RAM. The computational results reported in this section were obtained with the parameter values shown in Table 1, where the last two columns respectively denote the settings for the 31 random UBQP instances and the 103 MaxCut instances. Given the stochastic nature of our PR algorithms, each instance is independently solved 20 times by each algorithm.

4.3. Computational results on the random UBQP instances

Our first experiment undertakes to evaluate the PR algorithms on the 31 random instances with 2500–7000 variables (the first two sets of benchmarks). The results are summarized in Tables 2 and 3. Our algorithms use CPU clock time to give the stopping condition subject to having completed at least one round of the PR procedure. 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. This time cutoff is the same as in [27,32,34].

Tables 2 and 3 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 f_{prev}. These best values were first reported in [32,34] and recently improved in [19]. The columns under heading PR1 and PR2 list: the best objective value f_{best}, the average objective gap to the previous best objective values g_{avr} (i.e., $f_{prev} - f_{avr}$) (where f_{avr} represents the average objective value over 20 runs) and the average CPU time in seconds denoted by *time* for reaching the best objective values f_{best} over 20 runs. Furthermore, the last row "Average" indicates the summary of our algorithm's average performance.

Table 2 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_{avr} and *time* to the previous best result f_{prev}. Table 3 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

[1] 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.

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

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

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

Table 1
Settings of important parameters.

Parameters	Section	Description	Values	
			UBQP	MaxCut
b	3.2	RefSet size	10	10
ttc	3.3	Tabu tenure constant	$n/100$	$n/10$
μ	3.3	Improvement cutoff of TS	$5n$	10,000
γ	3.6	Distance scale	1/3	1/3

vs. 690.4) although the CPU time to obtain the best solution is slightly longer, (749.2 vs. 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 [34], MST2 [32], SA [22], D^2TS [19] and HMA [27]. The results for the first 3 of these reference algorithms are directly extracted from [34] and those for D^2TS and HMA come from [19,27].

Tables 4 and 5 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. Note that the results of all these algorithms are obtained almost under the same time limit. Since best known values can be easily reached for the small size instances by all these state-of-the art algorithms, we only list larger instances, consisting of 11 instances in Table 4 and 21 instances in Table 5.

Table 4 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 presents 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.

4.4. Computational results on the MaxCut instances

In this section, we test our PR algorithms on 3 sets of benchmarks with a total of 103 instances derived from MaxCut problem. In Tables 6–9, columns 1 and 2 respectively give the instance name and the previous best solution value f_{prev} from references [8,29,33,36] which are dedicated MaxCut algorithms. The columns under the headings PR1 and PR2 list the best objective value f_{best}, the average objective value f_{avr} and the CPU time in seconds denoted by *time* for reaching the best results f_{best}. The columns under the headings SS and CirCut report the best objective value f_{best}

Table 2
Computational results on Beasley instances.

Instance	f_{prev}	PR1			PR2		
		f_{best}	g_{avr}	*time*	f_{best}	g_{avr}	*time*
b2500.1	1,515,944	1,515,944	0.0	11	1,515,944	0.0	14
b2500.2	1,471,392	1,471,392	0.0	101	1,471,392	58.4	102
b2500.3	1,414,192	1,414,192	13.4	49	1,414,192	0.0	36
b2500.4	1,507,701	1,507,701	0.0	6	1,507,701	0.0	7
b2500.5	1,491,816	1,491,816	0.0	14	1,491,816	0.0	18
b2500.6	1,469,162	1,469,162	0.0	25	1,469,162	0.0	23
b2500.7	1,479,040	1,479,040	0.0	48	1,479,040	0.0	50
b2500.8	1,484,199	1,484,199	0.0	20	1,484,199	0.0	16
b2500.9	1,482,413	1,482,413	0.0	51	1,482,413	0.0	103
b2500.10	1,483,355	1,483,355	0.0	55	1,483,355	0.0	75
Average			1.34	38		5.84	44.4

Table 3
Computational results on Palubeckis instances.

Instance	f_{prev}	PR1			PR2		
		f_{best}	g_{avr}	*time*	f_{best}	g_{avr}	*time*
p3000.1	3,931,583	3,931,583	0.0	85	3,931,583	80.4	81
p3000.2	5,193,073	5,193,073	0.0	68	5,193,073	0.0	64
p3000.3	5,111,533	5,111,533	35.8	115	5,111,533	71.7	155
p3000.4	5,761,822	5,761,822	0.0	56	5,761,822	0.0	97
p3000.5	5,675,625	5,675,625	90.2	162	5,675,625	278.5	226
p4000.1	6,181,830	6,181,830	0.0	125	6,181,830	0.0	159
p4000.2	7,801,355	7,801,355	71.2	456	7,801,355	313.5	302
p4000.3	7,741,685	7,741,685	0.0	295	7,741,685	63.9	436
p4000.4	8,711,822	8,711,822	0.0	277	8,711,822	0.0	392
p4000.5	8,908,979	8,908,979	490.8	272	8,908,979	385.1	327
p5000.1	8,559,680	8,559,680	611.8	623	8,559,680	918.0	387
p5000.2	10,836,019	10,836,019	620.3	821	10,836,019	498.7	609
p5000.3	10,489,137	10,489,137	995.4	1285	10,489,137	317.5	967
p5000.4	12,252,318	12,252,318	1257.7	760	12,252,318	1168.4	767
p5000.5	12,731,803	12,731,803	51.3	676	12,731,803	166.3	726
p6000.1	11,384,976	11,384,976	201.0	1820	11,384,976	822.4	1136
p6000.2	14,333,855	14,333,855	221.1	1391	14,333,855	576.8	1076
p6000.3	16,132,915	16,132,915	1743.5	1128	16,132,915	2017.3	1053
p7000.1	14,478,676	14,478,676	935.4	2275	14,478,676	1523.1	1917
p7000.2	18,249,948	18,249,948	1942.4	1793	18,249,948	2986.1	1591
p7000.3	20,446,407	20,446,407	331.9	1251	20,446,407	2310.5	1503
Average			457.1	749.2		690.4	665.3

Table 4
Best results comparison on Palubeckis instances.

Instance	f_{prev}	Best solution gap (i.e., $f_{prev} - f_{best}$)						
		PR1	PR2	ITS [34]	MST2 [32]	SA [22]	D^{2}TS [19]	HMA [27]
p5000.1	8,559,680	0	0	700	325	1432	325	0
p5000.2	10,836,019	0	0	0	582	582	0	0
p5000.3	10,489,137	0	0	0	0	354	0	0
p5000.4	12,252,318	0	0	934	1643	444	0	0
p5000.5	12,731,803	0	0	0	0	1025	0	0
p6000.1	11,384,976	0	0	0	0	430	0	0
p6000.2	14,333,855	0	0	88	0	675	0	0
p6000.3	16,132,915	0	0	2729	0	0	0	0
p7000.1	14,478,676	0	0	340	1607	2579	0	0
p7000.2	18,249,948	0	0	1651	2330	5552	104	0
p7000.3	20,446,407	0	0	0	0	2264	0	0
Average		0	0	585.6	589.7	1394.3	39	0

Table 5
Average Results Comparison on Palubeckis Instances.

Instance	f_{prev}	Average solution gap (i.e., $f_{prev} - f_{avr}$)						
		PR1	PR2	ITS [34]	MST2 [32]	SA [22]	D^{2}TS [19]	HMA [27]
p3000.1	3,931,583	0	80	0	0	0	0	0
p3000.2	5,193,073	0	0	97	97	97	0	0
p3000.3	5,111,533	36	72	344	287	535	0	33
p3000.4	5,761,822	0	0	154	77	308	0	0
p3000.5	5,675,625	90	279	501	382	459	0	145
p4000.1	6,181,830	0	0	0	0	734	0	0
p4000.2	7,801,355	71	314	1285	804	1887	0	142
p4000.3	7,741,685	0	64	471	1284	79	0	6
p4000.4	8,711,822	0	0	438	667	536	0	38
p4000.5	8,908,979	491	385	572	717	984	0	546
p5000.1	8,559,680	612	918	971	581	2455	656	507
p5000.2	10,836,019	620	499	1068	978	2101	12,533	512
p5000.3	10,489,137	995	318	1266	1874	2451	12,876	332
p5000.4	12,252,318	1258	1168	1952	2570	1134	1962	1228
p5000.5	12,731,803	51	166	835	1233	1172	239	284
p6000.1	11,384,976	201	822	57	34	2248	0	140
p6000.2	14,333,855	221	577	1709	1269	2067	1286	526
p6000.3	16,132,915	1744	2017	3064	2673	3845	787	2311
p7000.1	14,478,676	935	1523	1139	2515	5504	2138	819
p7000.2	18,249,948	1942	2986	4301	3814	7837	8712	1323
p7000.3	20,446,407	332	2311	3078	7868	8978	2551	1386
Average		457.1	690.4	1109.6	1415.4	2162.4	2082.9	489.4

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 [29]. 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 6 reports the results on 54 instances of the third set of benchmarks 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 7 reports the results of 15 largest instances from the same set of benchmark as above with variables ranging from 5000 to 10000. For instances with 5000, 7000, 8000, 9000 and 10,000 variables, we report the results for a time limit of 1, 2, 4, 4 and 4 hours, respectively. The previous best objective values f_{prev} are cited from [10], which is the only paper, to the best of our knowledge, that reports the results on these instances. As can be seen from Table 7, 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 from the fourth set of benchmarks are shown in Table 8. For the instances with variables numbering 128, 1000 and 2744, the results are reported with a time limit of 1 second, 10 minutes and 30 minutes. Table 8 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.

Table 6
Computational results on small and medium MaxCut instances of Set1.

Instance	f_{prev}	PR1			PR2			SS [29]		CirCut [8]	
		f_{best}	f_{avr}	*time*	f_{best}	f_{avr}	*time*	f_{best}	*time*	f_{best}	*time*
G1	11,624	11,624	11624.0	2	11,624	11624.0	1	11,624	139	11,624	352
G2	11,620	11,620	11620.0	6	11,620	11620.0	9	11,620	167	11,617	283
G3	11,622	11,620	11620.0	17	11,620	11620.0	2	11,622	180	11,622	330
G4	11,646	11,646	11646.0	3	11,646	11646.0	2	11,646	194	11,641	524
G5	11,631	11,631	11631.0	3	11,631	11631.0	4	11,631	205	11,627	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	13,359	13,359	13353.5	1652	13,359	13354.5	943	13,346	1336	13,346	493
G23	13,342	13,342	13333.0	517	13,342	13331.6	879	13,317	1022	13,317	457
G24	13,337	13,337	13327.3	1257	13,333	13325.3	1876	13,303	1191	13,314	521
G25	13,326	13,338	13328.0	957	**13,339**	13328.2	1078	13,320	1299	13,326	1600
G26	13,314	13,324	13313.7	710	**13,326**	13312.3	333	13,294	1415	13,314	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
Average				469.3			490.6		621.0		616.7
Better		24			25			0		0	
Equal		22			24			22		20	
Worse		8			5			32		34	

Improved results are indicated in bold.

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 the fifth set of benchmarks using a time limit of 30 minutes are shown in Table 9. 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 refered 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,

Table 7
Computational results on large MaxCut instances of Set1.

Instance	f_{prev}	PR1			PR2		
		f_{best}	f_{avr}	time	f_{best}	f_{avr}	time
G55	9960	10,253	10233.7	3996	**10,265**	10234.0	3231
G56	3649	3975	3958.0	3991	**3981**	3959.2	3842
G57	3220	3448	3436.0	3656	**3472**	3462.0	4403
G58	–	19,183	19159.3	3979	**19,205**	19182.0	3715
G59	–	**6027**	5989.2	3876	**6027**	6006.2	5194
G60	13,658	14,109	14077.5	7738	**14,112**	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	26,876	26845.8	4826	**26,916**	26879.3	5867
G64	7861	8623	8569.5	8790	**8641**	8594.1	6974
G65	13,286	5482	5468.7	16,248	5526	5515.9	15,004
G66	–	6272	6257.8	16,031	**6314**	6302.4	15,191
G67	–	6856	6832.0	17,213	**6902**	6884.6	12,372
G70	9499	9405	9378.6	15,202	9463	9434.0	14,531
G72	6644	6892	6876.2	14,422	**6946**	6933.8	15,898
Better		13			13		
Equal		0			0		
Worse		2			2		

Improved results are indicated in bold.

we re-ran PR1 and PR2 on the MaxCut instances using 10 times longer time than before, as shown in Table 10. 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 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 set 1 (except the instance G31).

4.5. Additional comparisons

In order to further compare the proposed path relinking algorithms and the HMA algorithm in [27], we apply the time-to-target (TTT) analysis to show the empirical probability distribution of the needed time to attain a given target value [1]. For this experiment, we also include a multistart tabu search algorithm (MSTS) which is the tabu search procedure used in the path relinking algorithms reinforced with a random restart procedure.

We carry out the TTT experiment on a random UBQP instance (p5000.5) and a structured MaxCut instance (G25) with the PR1, PR2, HMA and MSTS algorithms. We perform 200 independent runs for each algorithm and each graph and record the time needed to attain an objective value at least as good as a given target value for each run. Then we sort the recorded times in an increasing order so that t_i represents the ith lowest time and a probability $p_i = (i - 1/2)/200$ is associated to each time t_i. Finally, the points (t_i, p_i) are plotted. Fig. 1 shows the results of the TTT experiment for PR1, PR2, HMA and MSTS on the two tested instance p5000.5 (Left) and G25 (Right). From the left part of Fig. 1, we first observe that for the first 400 seconds, PR1, PR2, HMA and MSTS almost perform the same with a low probability of 17% to reach the target value. Afterwards, PR1 and PR2 are obviously superior to HMA and MSTS. Specifically, at the moment of 2000 seconds, both PR1 and PR2 reach the target value with a probability of 100% against a probability of 60% for HMA and a probability of 38% for MSTS.

From the right part of Fig. 1, we notice that MSTS performs much worse than the other algorithms with a probability less than 5% to reach the target value during the overall time span of 500 seconds while PR1, PR2 and HMA only need 50 seconds to yield an equal or a better performance. In addition, after 50 seconds PR2 always

Table 8
Computational results on MaxCut instances of Set2.

Instance	f_{prev}	PR1			PR2			SS [29]		CirCut [8]	
		f_{best}	f_{avr}	time	f_{best}	f_{avr}	time	f_{best}	time	f_{best}	time
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
Average				473.2			377.5		537.3		163.2
Better		0			0			0		0	
Equal		21			20			10		10	
Worse		9			10			20		20	

Table 9
Computational results on MaxCut instances of Set3.

Instance	f_{prev}	PR1			PR2			SS [29]		CirCut [8]	
		f_{best}	f_{avr}	*time*	f_{best}	f_{avr}	*time*	f_{best}	*time*	f_{best}	*time*
g3-15	283,206,561	279,830,931	277345801.1	3000	276,903,146	273564256.6	1272	281,029,888	1023	268,519,648	788
g3-8	41,684,814	41,684,814	41508934.7	292	41,684,814	41521529.9	258	40,314,704	66	41,684,814	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
Average				1224.0			855.5		367.7		326.9
Better		1			1			0		0	
Equal		2			2			0		1	
Worse		1			1			4		3	

Improved results are indicated in bold.

Table 10
Computational results on MaxCut with longer CPU time.

Instance	f_{best}	*time*	Instance	f_{best}	*time*	Instance	f_{best}	*time*
G25	13,340	3539	G27	3341	3040	G28	3298	17,482
G30	3413	4795	G31	3310	10,801	G37	7686	3903
G38	7688	17,230	G39	2408	3087	G40	2400	11,947
G41	2405	945	G42	2481	5580	G55	10,274	31,764
G56	3993	11,727	G57	3484	4968	G58	19,225	20,499
G59	6039	28,790	G60	14,131	62,466	G61	5748	29,056
G62	4854	59,568	G63	26,941	45,136	G64	8693	66,851
G65	5544	94,934	G66	6340	74,375	G67	6928	114,438
G70	9529	135,572	G72	6978	141,167	G14100	2446	2105
G14200	2458	1657	G14600	2450	1476	G14700	2442	2824
G14800	2446	3543	G14900	2426	7165	G141000	2458	8929

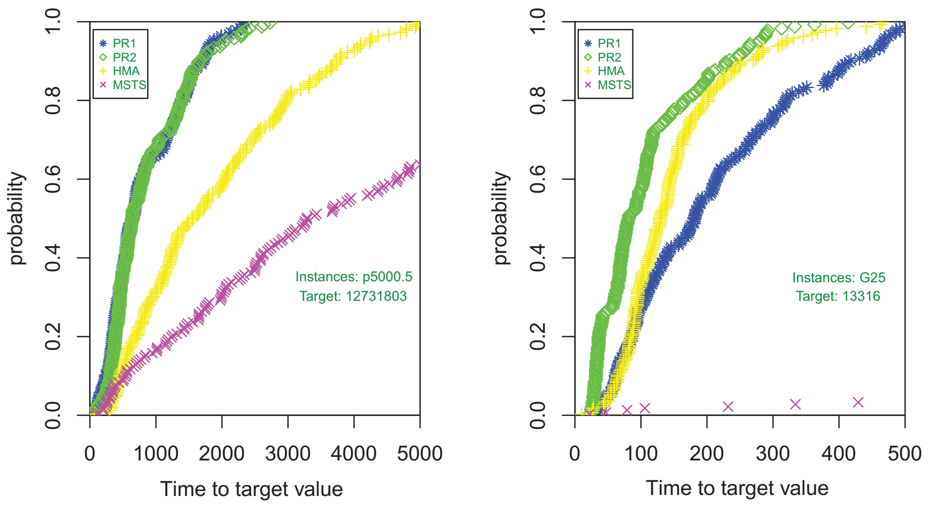

Fig. 1. Empirical probability distribution for the time to achieve a target value.

has a higher probability to achieve the target value than HMA and PR1. However, HMA is superior to PR1 from the moment of 100 seconds, which reverses the observation on instance p5000.5 where HMA is generally inferior to PR1. Therefore, this experiment shows that the path relinking procedure, as one of the important components of the proposed PR1 and PR2 algorithms, does play a key role for the good performance of our algorithms, especially in comparison with the MSTS algorithm.

5. Discussion

In the previous section, we showed that the proposed path relinking algorithms are able to achieve very competitive results on the UBQP and MaxCut benchmark instances. In this section, we discuss the results obtained on two other well-known combinatorial problems: set packing and graph k-coloring.

For the set packing problem, we first recast the problem into the UBQP model as shown in [2]. This experiment is based on a set of 16 large random benchmark instances with up to 2000 variables used in [2,11]. The experimental results (within a time limit of 30 minutes) show that our path relinking algorithms can match the best known results on 10 of the 16 instances. Remarkably, PR2 is able to improve the best known results ever reported in the literature for 2 instances. This performance can be considered to be very competitive in comparison with the state of the art methods like the GRASP algorithm of [11] which is specially designed for the set packing problem.

For the graph k-coloring problem, we recast the problem to the UBQP model according to the transformation shown in [24]. For each graph, we set k to be equal to the smallest known number ever reported in the literature and ran PR2 to check whether PR2 can find a feasible coloring. On the one hand, for the 21 small graph instances considered in [24] with up to 450 vertices and 1000 edges, PR2 can find a feasible coloring for each tested instance. On the other hand, tests on 20 challenging DIMACS graphs indicate that it is very difficult for PR2 to find feasible colorings with k set to be the smallest color number reported in the literature. Indeed, PR2 only found the feasible coloring on 2 out of 20 instances. This experiment indicates that though our path relinking algorithms are able to find good approximate solutions for the k-coloring problem, they cannot compete with the current best coloring algorithms.

6. Conclusion

In this paper, we proposed two effective path relinking algorithms for the unconstrained binary quadratic programming problem. The proposed algorithms are composed of a reference set initialization method, an improvement method by tabu search, 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). The experiments suggest that PR1 is more appropriate for random instances while PR2 is preferable for structured instances.

Computational experiments on five sets of 134 well-known random and structured 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 three sets of benchmarks with a total of 103 instances derived from the MaxCut problem, our algorithms can improve the previous best known results for almost 40 percent of these instances whose optimum solutions are still unknown. We also indicated that the path relinking algorithms perform quite well on 16 large set packing benchmark instances, but their performance on k-coloring is more moderate. It would be interesting to verify the performance of the proposed algorithms in solving other combinational problems that can be reformulated into the UBQP model.

There are several issues for future consideration. First, more elaborate methods can be used to better manage the reference set by considering both the the quality of solution and its distance to the previously found solutions, given the fact that a good diversity of the reference set is important for the path generation. Second, it would be interesting to verify if selecting more than one solution from a path for improvement is a good strategy. Third, by replacing the basic tabu search based improvement method with a more advanced tabu search method, still better outcomes could be expected.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments and suggestions. The work is partially supported by the Regional RaDaPop (2009–2013) and LigeRo Projects (2009–2013).

References

[1] R.M. Aiex, M.G.C. Resende, C.C. Ribeiro, TTT plots: a perl program to create time-to-target plots, Optimization Letters 1 (2007) i355–366.
[2] B. Alidaee, G.A. Kochenberger, K. Lewisa, M. Lewisc, H.B. Wang, A new approach for modeling and solving set packing problems, European Journal of Operational Research 186 (2) (2008) 504–512.
[3] J.E. Beasley, Obtaining test problems via internet, Journal of Global Optimization 8 (1996) 429–433.
[4] J.E. Beasley, Heuristic algorithms for the unconstrained binary quadratic programming problem, Working paper, The Management School, Imperial College, London, England, 1998.
[5] I. Borgulya, An evolutionary algorithm for the binary quadratic problems, Advances in Soft Computing 2 (2005) 3–6.
[6] E. Boros, P.L. Hammer, R. Sun, G. Tavares, A max-flow approach to improved lower bounds for quadratic 0-1 minimization, Discrete Optimization 5 (2) (2008) 501–529.
[7] E. Boros, P.L. Hammer, G. Tavares, Local search heuristics for Quadratic Unconstrained Binary Optimization (QUBO), Journal of Heuristics 13 (2007) 99–132.
[8] S. Burer, R.D.C. Monteiro, Y. Zhang, Rank-two relaxation heuristics for max-cut and other binary quadratic programs, SIAM Journal on Optimization 12 (2001) 503–521.
[9] P. Chardaire, A. Sutter, A decomposition method for quadratic zero-one programming, Management Science 41 (4) (1994) 704–712.
[10] C. Choi, Y. Ye, Solving sparse semidefinite programs using the dual scaling algorithm with an iterative solver, Working paper, Department of Management Sciences, The University of Iowa, 2000.
[11] X. Delorme, X. Gandibleau, J. Rodriques, GRASP for set packing, European Journal of Operational Research 153 (2004) 564–580.
[12] P. Festa, P.M. Pardalos, M.G.C. Resende, C.C. Ribeiro, Randomized heuristics for the max-cut problem, Optimization Methods and Software 7 (2002) 1033–1058.
[13] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.
[14] F. Glover, G.A. Kochenberger, B. Alidaee, Adaptive memory tabu search for binary quadratic programs, Management Science 44 (1998) 336–345.
[15] F. Glover, M. Laguna, R. Marti, Fundamentals of scatter search and path-relinking, Control and Cybernetics 39 (2000) 654–684.
[16] F. Glover, M. Laguna, R. Marti, Scatter search and path relinking: advances and applications, Handbook of Metaheuristics 57 (2003) 1–35.
[17] F. Glover, M. Laguna, R. Marti, Scatter search and path relinking: foundations and advanced designs, in: G.C. Onwubolu, B.V. Babu (Eds.), New Optimization Technologies in Engineering, Studies in Fuzziness and Soft Computing, vol. 141, 2004, pp. 87–100.
[18] F. Glover, J.K. Hao, Efficient evaluation for solving 0-1 unconstrained quadratic optimization problems, International Journal of Metaheuristics 1 (1) (2010) 3–10.
[19] F. Glover, Z. Lü, J.K. Hao, Diversification-driven tabu search for unconstrained binary quadratic problems, 4OR: A Quarterly Journal of Operations Research 8 (3) (2010) 239–253.
[20] F. Harary, On the notion of balanced of a signed graph, Michigan Mathematical Journal 2 (1953) 143–146.
[21] C. Helmberg, F. Rendl, Solving quadratic (0,1)-problem by semidefinite programs and cutting planes, Mathematical Programming 82 (1998) 388–399.
[22] K. Katayama, H. Narihisa, Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem, European Journal of Operational Research 134 (2001) 103–119.
[23] G.A. Kochenberger, F. Glover, B. Alidaee, C. Rego, A unified modeling and solution framework for combinatorial optimization problems, OR Spectrum 26 (2004) 237–250.
[24] G.A. Kochenberger, F. Glover, B. Alidaee, C. Rego, An unconstrained quadratic binary programming approach to the vertex coloring problem, Annals OR 139 (1) (2005) 229–241.
[25] J. Krarup, A. Pruzan, Computer aided layout design, Mathematical Programming Study 9 (1978) 75–84.
[26] A. Lodi, K. Allemand, T.M. Liebling, An evolutionary heuristic for quadratic 0-1 programming, European Journal of Operational Research 119 (3) (1999) 662–670.
[27] Z. Lü, F. Glover, J.K. Hao, A hybrid metaheuristic approach to solving the UBQP problem, European Journal of Operational Research 207 (3) (2010) 1254–1262.
[28] R.D. McBride, J.S. Yormark, An implicit enumeration algorithm for quadratic integer programming, Management Science 26 (1980) 282–296.
[29] R. Marti, A. Duarte, M. Laguna, Advanced scatter search for the max-cut problem, INFORMS Journal on Computing 21 (1) (2009) 26–38.
[30] P. Merz, B. Freisleben, Genetic algorithms for binary quadratic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), Morgan Kaufmann, 1999, pp. p 417–424.
[31] P. Merz, K. Katayama, Memetic algorithms for the unconstrained binary quadratic programming problem, BioSystems 78 (2004) 99–118.
[32] G. Palubeckis, Multistart tabu search strategies for the unconstrained binary quadratic optimization problem, Annals of Operations Research 131 (2004) 259–282.
[33] G. Palubeckis, Application of multistart tabu search to the MaxCut problem, Information Technology and Control 2 (31) (2004) 29–35.
[34] G. Palubeckis, Iterated tabu search for the unconstrained binary quadratic optimization problem, Informatica 17 (2) (2006) 279–296.
[35] P. Pardalos, G.P. Rodgers, Computational aspects of a branch and bound algorithm for quadratic zero-one programming, Computing 45 (1990) 131–144.
[36] V.P. Shylo, O.V. Shylo, Solving the maxcut problem by the global equilibrium search, Cybernetics and Systems Analysis 46 (5) (2010) 744–754.
[37] Y. Wang, Z. Lü, F. Glover, J.K. Hao, Backbone guided tabu search for solving the UBQP problem, Journal of Heuristics, 2011. http://dx.doi.org/10.1007/s10732-011-9164-4.
[38] Y. Wang, Z. Lü, F. Glover, J.K. Hao, Effective variable fixing and scoring strategies for binary quadratic programming, in: P. Merz, J.K. Hao (Eds.), EvoCOP 2011, Lecture Notes in Computer Science, vol. 6622, 2011, pp. 72–83.