4OR-Q J Oper Res
DOI 10.1007/s10288-009-0115-y

RESEARCH PAPER

Diversification-driven tabu search for unconstrained binary quadratic problems

Fred Glover · Zhipeng Lü · Jin-Kao Hao

Received: 24 April 2009 / Revised: 3 December 2009

Abstract This paper describes a Diversification-Driven Tabu Search (D^2TS) algorithm for solving unconstrained binary quadratic problems. D^2TS is distinguished by the introduction of a perturbation-based diversification strategy guided by long-term memory. The performance of the proposed algorithm is assessed on the largest instances from the ORLIB library (up to 2500 variables) as well as still larger instances from the literature (up to 7000 variables). The computational results show that D^2TS is highly competitive in terms of both solution quality and computational efficiency relative to some of the best performing heuristics in the literature.

Keywords UBQP · Tabu search · Diversification-driven · Long-term memory

Mathematics Subject Classification (2000) 90C27 · 80M50 · 65K10

F. Glover (✉)
OptTek Systems, Inc., 2241 17th Street,
Boulder, CO 80302, USA
e-mail: glover@opttek.com

Z. Lü · J.-K. Hao
LERIA, Université d'Angers,
2 Boulevard Lavoisier, 49045 Angers Cedex 01,
France
e-mail: lu@info.univ-angers.fr; zhipeng.lui@gmail.com

J.-K. Hao
e-mail: hao@info.univ-angers.fr
URL: http://www.info.univ-angers.fr/pub/hao/

Published online: 05 January 2010

1 Introduction

The unconstrained binary quadratic programming problem may be written

$$\text{UBQP}: \quad \begin{array}{l} \text{Maximize } x_o = xQx' \\ x \text{ binary} \end{array}$$

where Q is an n by n matrix of constants and x is an n-vector of binary (zero-one) variables.

The formulation UBQP is notable for its ability to represent a wide range of important problems, including those from social psychology (Harary 1953), financial analysis (Laughunn 1970; McBride and Yormark 1980), computer aided design (Krarup and Pruzan 1978), traffic management (Gallo et al. 1980; Witsgall 1975), machine scheduling (Alidaee et al. 1994), cellular radio channel allocation (Chardaire and Sutter 1994) and molecular conformation (Phillips and Rosen 1994). Moreover, many combinatorial optimization problems pertaining to graphs such as determining maximum cliques, maximum cuts, maximum vertex packing, minimum coverings, maximum independent sets, maximum independent weighted sets are known to be capable of being formulated by the UBQP problem as documented in papers of Pardalos and Rodgers (1990), Pardalos and Xue (1994). A review of additional applications and formulations can be found in Kochenberger et al. (2004, 2005), Alidaee et al. (2008), Lewis et al. (2008).

Given the interest of the UBQP, a large number of solution procedures have been reported in the literature. Some representative examples include local search based approaches such as those of Boros et al. (2007), Simulated Annealing (Alkhamis et al. 1998; Beasley 1998; Katayama and Narihisa 2001) and Tabu Search (Glover et al. 1998; Beasley 1998; Palubeckis 2004, 2006), population-based approaches such as Evolutionary Algorithms (Lodi et al. 1999; Merz and Freisleben 1999; Katayama et al. 2000; Borgulya 2005), Scatter Search (Amini et al. 1999) and Memetic Algorithms (Merz and Katayama 2004).

Among these procedures, TS represents one of the most popular and successful approaches. One of the first adaptive memory TS algorithms for the UBQP (Glover et al. 1998), for instance, has since been used to solve applications arising in a wide variety of settings, as a demonstration of the value of the UBQP model and the ability to solve such applications successfully. More recently, (Palubeckis 2004) has explored several multistart TS strategies and has achieved very good results on large problem instances. A sequel further improves these results by an Iterated Tabu Search algorithm (Palubeckis 2006).

In the current paper, we introduce a new TS algorithm which employs a guided diversification strategy utilizing an information-based perturbation operator. We show that this Diversification-Driven Tabu Search (D^2TS) algorithm is highly effective in solving a large range of benchmark instances from the literature. For example, for the well-known UBQP instances containing up to 2500 variables (Beasley 1998) that has been used in many published papers, D^2TS attains the best known objective values in less than one minute. Moreover, for the set of 21 large instances containing 3000 to

7000 variables introduced in Palubeckis (2004, 2006), our algorithm is able to match or even improve the best previous results.

2 Diversification-driven TS (D^{2}TS) for UBQP

2.1 Main idea of D^{2}TS

D^{2}TS repeatedly alternates between a simple version of Tabu Search that we denote by TS^o and a diversification phase founded on memory-based perturbation operator. Starting from an initial random solution, D^{2}TS uses the TS^o procedure to reach a local optimum. Then, the perturbation operator is applied to displace the solution to a new region, whereupon a new round of TS^o is launched. To achieve a more effective diversification, the perturbation operator is guided by information from a special memory structure for obtaining improved results in this context. The next two sub-sections give a detailed explanation of the neighborhood and the tabu list management of the TS^o procedure, as well as the memory-based perturbation operator.

2.2 Neighborhood and tabu list

2.2.1 Neighborhood using 1-Flip moves

Our TS^o procedure uses a neighborhood defined by the well-known 1-flip move, which consists of changing (flipping) the value of a single variable x_i to its complementary value $1 - x_i$. It is clear that the size of this neighborhood is bounded by $O(n)$, i.e., at most n moves are required to go from any solution to any other solution.

For large problem instances, it is imperative to be able to rapidly determine the effect of a move on the objective function x_o. For this purpose, we employ a fast incremental evaluation technique first introduced by Glover et al. (1998) and enhanced by Glover and Hao (2009a) to exploit an improved representation and to take advantage of sparse data—a characteristic of many real world problems. The procedure maintains a data structure that stores the move value (change in x_o) for each possible move, and employs a streamlined calculation for updating this data structure after each iteration.

The key elements of this procedure may be summarized as follows. Let $N = \{1, \ldots, n\}$ denote the index set for components of the x vector. We preprocess the matrix Q to put it in lower triangular form by redefining (if necessary) $q_{ij} = q_{ij} + q_{ji}$ for $i > j$, which is implicitly accompanied by setting $q_{ji} = 0$ (though these 0 entries above the main diagonal are not stored or accessed). Let Δx_i be the move value of flipping the variable x_i, and let $q_{(i,j)}$ be a shorthand for denoting q_{ij} if $i > j$ and q_{ji} if $j > i$. Then each can be calculated in linear time using the formula:

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

Significantly, it is possible to update the move values upon flipping a variable x_i by performing the following abbreviated calculation, using the convention that x_i represents x_i's value before being flipped.

1. $\Delta x_i = -\Delta x_i$
2. For each $j \in N - \{i\}$,
   $\Delta x_j = \Delta x_j + \sigma_{i,j} \cdot q_{(i,j)}$
   where $\sigma_{i,j} = 1$ if $x_i = x_j$ and $\sigma_{i,j} = -1$ otherwise.

These updates can be implemented highly efficiently in the presence of sparse data using the procedures in Glover and Hao (2009a).

2.2.2 Tabu list management

TS typically incorporates a tabu list as a "recency-based" memory structure to assure that solutions visited within a certain span of iterations, called the tabu tenure, will not be revisited (Glover and Laguna 1997). The approach is designed to introduce vigor into the search by also forbidding moves leading to related solutions that share certain attributes (values of variables) in common with the visited solutions. In our present implementation we use a simple tabu list consisting of an n-vector $TabuTenure(i)$, $i \in N$. When the variable x_i is flipped, we have elected to set

$$TabuTenure(i) = c + rand(10) \tag{2}$$

where c is a constant and rand(10) denotes a randomly generated number from 1 to 10. The constant c is determined according to the size of the problem instance and is experimentally fixed at $n/100$ in our implementation.

This tabu list assignment is used to prevent x_i from being flipped until a number of $TabuTenure(i)$ iterations have elapsed. (To facilitate implementation, $TabuTenure(i)$ is customarily increased by the value of the current iteration at the time when the assignment (1) is made, and this modified value is checked against subsequent values of the iteration counter.) The TS^o algorithm then restricts consideration to variables not forbidden by the tabu list, and selects a variable to flip that produces the largest Δx_i value (thus improving x_o if this value is positive). 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.

This rudimentary TS process stops when the best solution cannot be improved within a given number α of moves that we call the *improvement cutoff*.

2.3 Diversification phase

In order to enhance the diversification capability of the preceding TS^o algorithm, we introduce a strategy which relies on a memory-based perturbation operator composed of three parts: a flip frequency memory ($FlipFreq$), an elite solution memory ($EliteSol$) and an elite value frequency memory ($EliteFreq$). These memory structures are used jointly by the perturbation operator (see Sect. 2.3.2).

2.3.1 Memory management

Our tabu search procedure uses the vector $FlipFreq(i), i \in N$ to record the number of times the variable x_i has been flipped. This information is used to guide the design of the scoring function of the perturbation operator (see Sect. 2.3.2).

$EliteSol$ stores a set of elite locally optimal solutions found by TS^o using a design commonly employed to construct reference sets in scatter search methods. We represent this memory as a list $EliteSol = [S_1, \ldots, S_R]$, where R is a maximum allowed dimension of $EliteSol$ and S_i represents the ith local optimum recorded in this memory. In our implementation, R was set to be 8 for all the problems we have tested in this paper. The first solution inserted on $EliteSol$ is the best solution obtained by the first phase of the TS^o procedure. After that, new local optima obtained by successive runs of the TS^o procedure are added to the list provided they do not already exist in the memory, continuing until R different solutions are stored. From this point on, each time a new local optimum is found that has an x_o value superior to that of the worst local optimum on $EliteSol$, the new solution replaces this worst element. The resulting pool of high quality solutions provides a source of candidates for applying the perturbation operator.

Finally, the vector $EliteFreq(i), i \in N$ records the total number of times variable x_i is assigned value 1 in the elite solutions currently stored in $EliteSol$. This memory is used to penalize the use of flips during the perturbation phase for variables that have more consistently received the same value in the elite solutions, thus constituting a form of intensification process that favors retaining the value assignments that occur more often in the best solutions found to date. See Sect. 2.3.2 for more details.

2.3.2 Memory-based perturbation operator

From a general perspective, the perturbation component of the diversification phase has two aims: to jump out of local optima and to lead the search procedure to a new promising region. In our case, the perturbation step first randomly selects an elite solution from the list $EliteSol$ and then applies a perturbation operator to the selected solution. Contrary to a conventional random perturbation strategy, our perturbation operator uses the so-called critical element-guided perturbation strategy (Lü and Hao 2009), which is composed of three steps: (1) *Scoring*: assign a score to each variable; (2) *Selection*: choose a certain number of highly-scored variables (critical elements); (3) *Perturbing*: perturb the solution using the chosen critical elements.

The scoring function ranks each variable by taking into account its flip frequency ($FlipFreq(i)$ and its elite value frequency ($EliteFreq(i)$). Let r ($0 \leq r \leq R$) be the current number of solutions recorded in $EliteSol$, our scoring function takes the following form:

$$Score(x_i) = \frac{EliteFreq(i)(r - EliteFreq(i))}{r^2} + \beta \left(1 - \frac{FlipFreq(i)}{max\_Freq}\right) \quad (3)$$

where β is a constant and max_Freq is the largest of the $FlipFreq(i)$ values, i.e., $max\_Freq = max_{i=1,\ldots,N}\{FlipFreq(i)\}$. In this paper, we set $\beta = 0.3$ for all our experiments.

The first part of the score function is based on the supposition that a variable x_i whose $EliteFreq(i)$ value equals an extreme value 0 or r should be given little opportunity to be flipped since it always receives the same value in the elite solutions in the memory. On the other hand, a variable x_i whose $EliteFreq(i)$ value equals $r/2$ should have complete freedom to change its value. The basic idea behind the second part of the score function (3) is to give a high flip probability to a variable that is seldom flipped. Our supposition is that changing the value of such a variable can help the search to jump out of local optima.

For the selection step, we first sort all the variables in non-increasing order according to their scores and then probabilistically select γ different variables to be randomly assigned a value 0 or 1 (γ is called the perturbation strength). This selection procedure is implemented in an adaptive way, i.e., the higher the score a variable has, the greater the probability it will be chosen. The jth highly-scored variable is selected to be flipped according to the probability:

$$P_j = \frac{j^{-\lambda}}{\sum_{i=1}^{n} i^{-\lambda}} \tag{4}$$

where λ is a positive number. Note that this selection procedure is problem independent.

Finally for the perturbation step, we just flip the values of the selected critical variables. This perturbed solution is then used to initiate a new round of our tabu search procedure by once again launching TS^o. Computational experiments presented in Sect. 3 confirm the value of this special form of perturbation as a diversification strategy for solving large scale UBQP instances.

2.4 D^2TS algorithm description

Our D^2TS algorithm is summarized in Algorithm 1.

Some brief comments are appropriate. At the beginning of the search, the $EliteSol$ list is empty with $r = 0$. The first loop from lines 5 to 14 fills the list one element at a time until the number of elements in $EliteSol$ reaches its given limit R. The $EliteFreq$ vector is also updated at each iteration. The loop in lines 15–24 repeatedly updates $EliteSol$ and the $EliteFreq$ vector until a specified stop condition is met. In this loop, if a new locally optimal solution S^* is better than the worst solution S_w in $EliteSol$ and if S^* does not exist in $EliteSol$, then S^* replaces S_w on this list.

3 Computational results

To assess the efficiency of our proposed D^2TS algorithm, we carry out experiments on 31 medium and large instances in the literature and compare D^2TS with five best performing algorithms. At the end of this section, we provide an experimental analysis demonstrating the importance of the memory-based perturbation operator described in Sect. 2.3.2.

Algorithm 1 Diversification-Driven Tabu Search (D^2TS) for UBQP

Input: Q matrix
Output: S^*: the best solution found so far
Set $EliteSol = \{\}, r = 0, EliteFreq(i) = 0, i = 1, \ldots, n$
Randomly generate an initial solution S_0
while $r < R$ **do**
  S^* = Tabu_Search(S_0)
  if S^* is not in $EliteSol$ **then**
    Insert S^* into $EliteSol$: $EliteSol = EliteSol + \{S^*\}$
    $r = r + 1$
    $EliteFreq = EliteFreq + S^*$
  end if
  Randomly select a solution S' from $EliteSol$
  S_0 = Perturbation_Operator(S')
end while
while Stop condition is not met **do**
  Randomly select a solution S' from $EliteSol$
  S_0 = Perturbation_Operator(S')
  S^* = Tabu_Search(S_0)
  S_w = The worst solution in $EliteSol$ in terms of solution quality
  if S^* is not in $EliteSol$ and $f(S^*) > f(S_w)$ **then**
    $EliteSol = EliteSol + \{S^*\} - \{S_w\}$
    $EliteFreq = EliteFreq + S^* - S_w$
  end if
end while

Table 1 Settings of important parameters

Parameters	Section	Description	Values
c	2.2.1	Tabu tenure constant	$n/100$
α	2.2.2	Improvement cutoff of TS	$20n$
R	2.3.1	Maximum size of the memory $EliteSol$	8
β	2.3.2	Frequency-related weight in perturbation scoring	0.3
λ	2.3.2	Perturbation selection importance factor	1.2
γ	2.3.2	Perturbation strength	$n/4$

3.1 Experimental protocol

Our algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.66 GHz CPU and 512M RAM. All computational results were obtained without special tuning of the parameters, i.e., all the parameters used in our algorithm are fixed (constant) or dynamically and automatically tuned during the problem solving for all instances considered. It is possible that better solutions would be found by using a set of instance-dependent parameters. However, our aim is to design a robust solver that is able to solve a large panel of instances efficiently. Table 1 gives the descriptions and settings of the parameters used in our D^2TS algorithm. These parameters are tuned in two steps. We calibrate first the two parameters of TS^0 (tabu tenure constant and improvement cutoff), followed by fixing the perturbation operator. Based on preliminary testing, we observed that the following parameter

settings give satisfying results: $c \in [n/200, n/80]$, $\alpha \in [10n, 50n]$, $R \in [6, 15]$, $\beta \in [0.2, 0.5]$, $\lambda \in [1.1, 1.4]$ and $\gamma \in [n/5, n/3]$. The calibrated parameter values are kept constant for all the experiments. It is possible that better solutions would be found by using a set of instance-dependent parameters.

3.2 Test instances

Two sets of test problems are considered in the experiments, in total constituting 31 instances. The first set of benchmarks is composed of 10 largest instances of size $n = 2500$ introduced in Beasley (1998) and available in the ORLIB (Beasley 1996). They all have a density of 0.1 and are named by b2500.1,. . .,b2500.10. These instances are used in the literature by many authors, see for instance (Beasley 1998; Katayama and Narihisa 2001; Merz and Freisleben 2002; Merz and Katayama 2004; Palubeckis 2004, 2006). The second set of benchmarks consists of a set 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 (Palubeckis 2004, 2006). Nonzero entries of Q are drawn uniformly from the interval $[-100, 100]$. 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. Experiments reported in Palubeckis (2004, 2006) showed that these large instances are particularly challenging for UBQP algorithms.

The small test instances from the ORLIB whose sizes range from $n = 500$ to 1000 and the similarly small instances from Glover et al. (1998) are not considered here, since they are solved very easily within 30 s by our algorithm and are also solved relatively easily by most recent heuristics.

3.3 Computational results on ORLIB instances

Our first experiment aims to evaluate the D^{2}TS algorithm on the 10 ORLIB instances with 2500 variables. The results of this experiment are summarized in Tables 2 to 4.

Table 2 shows the computational statistics of our D^{2}TS algorithm. Columns 2 and 3 respectively give the density (dens) and the previous best known results (f_{prev}). Columns 4 to 8 give our results: the best objective value (f_{best}), the difference between our best values with the previous best known values ($f_{best} - f_{prev}$), the average objective value (f_{aver}), the success rate (success) and the average CPU time (seconds) for reaching the best result (f_{best}). Table 2 discloses that our D^{2}TS algorithm can stably reach all the previous best known results within 40 s on our computer, demonstrating the high efficiency of our method.

Table 3 shows the average results of our D^{2}TS algorithm compared with the five leading algorithms in the literature, respectively named ITS (Palubeckis 2006), MST1 (Palubeckis 2004), MST2 (Palubeckis 2004), SA (Katayama and Narihisa 2001) and MA (Merz and Katayama 2004). The results of these five algorithms are extracted from (Palubeckis 2006) and have been obtained by Palubeckis by applying each under the same experimental conditions, which we likewise employ for evaluating our algorithm. These five algorithms were run 25 times for each problem instance with a time limit of 600 s on a Pentium III 800 PC. Since our computer is about 3 times faster than

Table 2 Results of D^{2}TS algorithm on the Beasley instances from ORLIB

Instance	Dens	f_{prev}	D^{2}TS Algorithm				
			f_{best}	$f_{best} - f_{prev}$	f_{aver}	Success	Seconds
b2500.1	0.1	1515944	1515944	0	1515944	25/25	6
b2500.2	0.1	1471392	1471392	0	1471392	25/25	38
b2500.3	0.1	1414192	1414192	0	1414192	25/25	35
b2500.4	0.1	1507701	1507701	0	1507701	25/25	4
b2500.5	0.1	1491816	1491816	0	1491816	25/25	5
b2500.6	0.1	1469162	1469162	0	1469162	25/25	10
b2500.7	0.1	1479040	1479040	0	1479040	25/25	20
b2500.8	0.1	1484199	1484199	0	1484199	25/25	12
b2500.9	0.1	1482413	1482413	0	1482413	25/25	6
b2500.10	0.1	1483355	1483355	0	1483355	25/25	7

Table 3 Average performance of D^{2}TS and other algorithms on the Beasley problems

Instance	f_{prev}	Solution difference (i.e., average heuristic value - f_{prev})					
		D^{2}TS	ITS	MST1	MST2	SA	MA
b2500.1	1515944	0	0	0	0	−4	−13
b2500.2	1471392	0	−9	−133	0	−433	−645
b2500.3	1414192	0	−11	0	−11	−117	−173
b2500.4	1507701	0	0	0	0	0	0
b2500.5	1491816	0	0	0	0	−6	−55
b2500.6	1469162	0	0	−1	0	−58	−190
b2500.7	1479040	0	0	−4	0	−208	−416
b2500.8	1484199	0	0	0	0	−35	−3
b2500.9	1482413	0	0	0	0	−33	−321
b2500.10	1483355	0	0	−8	0	−493	−446
Average		0	−2	−15	−1	−139	−226

that used by Palubeckis (2006), we limit the running CPU time of D^{2}TS to 200 s.[1] The overall results, averaged over 10 instances, are presented in the last row. From Table 3, one observes that our D^{2}TS algorithm obtains the previous best known results more stably than these alternative heuristics that are reported to be the most effective in the literature.

Table 4 compares the average time (in seconds) needed by each of the compared algorithms to hit the best objective value in the run. We have converted our CPU

[1] We tested a benchmark program on our computer and a Pentium III 800 PC with 512M memory and found that the exact speed ratio of these two computers is 2.92. This benchmark program is used by the second International Timetabling Competition and available at: http://www.cs.qub.ac.uk/itc2007/benchmarking/benchmark_machine.zip.

Table 4 Average time performance of D^2TS and other algorithms on the Beasley problems: average time to the best solution in the run (in seconds)

Instance	D^2TS	ITS	MST1	MST2	SA	MA
b2500.1	18	18	14	13	225	461
b2500.2	114	205	281	158	334	430
b2500.3	105	196	91	134	319	422
b2500.4	12	6	8	9	120	293
b2500.5	15	12	7	11	305	469
b2500.6	30	22	48	23	283	452
b2500.7	60	75	168	99	387	478
b2500.8	36	46	26	47	293	359
b2500.9	18	54	77	71	340	450
b2500.10	21	104	161	138	351	477
Average	42	74	88	70	296	429

time reported in Table 2 by multiplying it by 3 to compensate for the fact that our computer is about 3 times faster. A corresponding conversion also applies to Tables 7 and 8. From Table 4, we observe that D^2TS can easily obtain the previous best known solutions within 120 s (converted time). From Tables 2 to 4, we conclude that D^2TS is quite competitive compared with these reference algorithms in terms of both solution quality and computational efficiency. However, from the results presented above, it is impossible to conclude that any given algorithm dominates the others since the problem instances in this set are not sufficiently difficult to solve. More significant differences are observed when larger and harder instances are used, as we show next.

3.4 Computational results on larger instances

In the second experiment we tested our D^2TS algorithm on the second set of 21 randomly generated instances.[2] These instances of larger size and higher density are more difficult for the search algorithms. Tables 5 reports the computational results obtained by D^2TS for solving these instances, following the same format as Table 2. The stop condition is set to be the same as in Palubeckis (2006), i.e., the cutoff time for a run is 15, 30, 60, 90 and 150 minutes on a Pentium III 800 PC for an instance with 3000, 4000, 5000, 6000 and 7000 variables, respectively. (The time limit on our computer is set to be 1/3 of these values.) Column 5 shows that under this stop condition our D^2TS algorithm matches the previous best known results for 18 instances and improves the previous best known results for 3 instances, named p5000.4, p7000.1 and p7000.2.

In order to further compare our D^2TS algorithm with the best competing algorithms, we again refer to the algorithms used in Table 3 (ITS, MST1, MST2, SA and MA). As before, the results of the reference algorithms are directly extracted from (Palubeckis 2006). Table 6 displays the solution difference between the best solutions obtained by these 6 algorithms with the best known results overall. The averaged results over the 21 instances are presented in the last row. From Table 6 it may be observed that our

[2] Our best results are available at: http://www.info.univ-angers.fr/pub/hao/UBQP.html.

Table 5 Results of our D^{2}TS algorithm on 21 large random problem instances with size from $n = 3000$ to $n = 7000$

Instance	Dens	f_{prev}	D^{2}TS Algorithm				
			f_{best}	$f_{best} - f_{prev}$	f_{aver}	Success	Seconds
p3000.1	0.5	3931583	3931583	0	3931583	20/20	70
p3000.2	0.8	5193073	5193073	0	5193073	20/20	82
p3000.3	0.8	5111533	5111533	0	5111533	20/20	79
p3000.4	1.0	5761822	5761822	0	5761822	20/20	111
p3000.5	1.0	5675625	5675625	0	5675625	20/20	159
p4000.1	0.5	6181830	6181830	0	6181830	20/20	91
p4000.2	0.8	7801355	7801355	0	7801355	20/20	252
p4000.3	0.8	7741685	7741685	0	7741685	20/20	178
p4000.4	1.0	8711822	8711822	0	8711822	20/20	223
p4000.5	1.0	8908979	8908979	0	8908979	20/20	702
p5000.1	0.5	8559355	8559355	0	8559024	6/10	2855
p5000.2	0.8	10836019	10836019	0	10823486	8/10	1155
p5000.3	0.8	10489137	10489137	0	10476261	7/10	1326
p5000.4	1.0	12251874	12252318	444	12250356	4/10	838
p5000.5	1.0	12731803	12731803	0	12731564	9/10	623
p6000.1	0.5	11384976	11384976	0	11384976	10/10	509
p6000.2	0.8	14333855	14333855	0	1432569	5/10	1543
p6000.3	1.0	16132915	16132915	0	1613128	4/10	2088
p7000.1	0.5	14478336	14478676	340	1446538	4/10	1217
p7000.2	0.8	18248297	18249844	1547	18241236	7/10	849
p7000.3	1.0	20446407	20446407	0	2043856	3/10	3520

D^{2}TS algorithm outperforms these five reference algorithms in terms of the quality of the best solution obtained. Notably, our D^{2}TS algorithm finds better solutions than any of these five references algorithms for at least 4 instances (roughly 20% of the problems) (Table 7).

In order to compare the time performance between different approaches, the average CPU time to reach the given best solution is reported in Table 7. Similarly, the averaged results over the 21 instances are presented in the last row. Our D^{2}TS algorithm needs 14% to 27% more CPU time than three of the reference algorithms (ITS, MTS1 and MTS2) to reach the results reported in Table 5. However, we also obtained solutions to some of the problems more quickly than all of the other methods.

In Palubeckis (2006), the author identifies ITS as the top performing algorithm among the considered algorithms and reports computational results on five instances with 5000, 6000 and 7000 variables with longer runs of ITS. The time limit was then set at 5, 8 and 10 hours, respectively. For these five instances, the ITS algorithm improved its previous best results reported in Table 6, as shown in Table 8.

To check whether our D^{2}TS algorithm is also able to improve its previous best results by allowing greater computational time, we re-ran D^{2}TS on these five instances

Table 6 Best results comparison between D^2TS and other state-of-the-art algorithms for larger problem instances

Instance	Dens	f_{prev}	Solution difference (i.e., heuristic solution value - f_{prev})					
			D^2TS	ITS	MST1	MST2	SA	MA
p3000.1	0.5	3931583	0	0	0	0	0	−3950
p3000.2	0.8	5193073	0	0	0	0	0	−342
p3000.3	0.8	5111533	0	0	−357	0	0	0
p3000.4	1.0	5761822	0	0	0	0	0	−1097
p3000.5	1.0	5675625	0	0	−478	0	0	−478
p4000.1	0.5	6181830	0	0	0	0	0	−2390
p4000.2	0.8	7801355	0	0	−1686	0	−504	−6564
p4000.3	0.8	7741685	0	0	−54	0	0	−5760
p4000.4	1.0	8711822	0	0	0	0	0	−2359
p4000.5	1.0	8908979	0	0	0	0	0	−9028
p5000.1	0.5	8559355	0	−375	−2691	0	−1107	−4647
p5000.2	0.8	10836019	0	0	0	−582	−582	−7519
p5000.3	0.8	10489137	0	0	−3277	0	−354	−11552
p5000.4	1.0	12251874	444	−490	−3341	−1199	0	−15955
p5000.5	1.0	12731803	0	0	−5150	0	−1025	−6644
p6000.1	0.5	11384976	0	0	−3198	0	−430	−9046
p6000.2	0.8	14333855	0	−88	−10001	0	−675	−21732
p6000.3	1.0	16132915	0	−2729	−11658	0	0	−13400
p7000.1	0.5	14478336	340	0	−6778	−1267	−2239	−13365
p7000.2	0.8	18248297	1547	0	−7251	−679	−3901	−18898
p7000.3	1.0	20446407	0	0	−17652	0	−2264	−14684
Average			126	−175	−3503	−177	−623	−8067

using the same timing conditions used by ITS. The new results appear in Table 8. D^2TS likewise improves its results for two out of five instances, matching the results of ITS for four instances and finding a better solution than ITS for the remaining instance p5000.1 with an objective value 8559680, as indicated in bold in Table 8.

3.5 Influence of the adaptive memory mechanism

We turn our attention now to analyzing one of the most important components of the proposed D^2TS algorithm, the memory-based perturbation operator described in Sect. 2.3.2. This strategy involves randomly and adaptively selecting and flipping a given number of *highly-scored* variables. We believe that constraining the choices to the critical variables is essential for our D^2TS algorithm. In order to be sure this mechanism is meaningful, we carried out additional experiments to examine the influence of the proposed memory-based perturbation operator (denoted by *MBP*).

For this purpose, we compare *MBP* with a pure random perturbation operator (denoted by *PRP*) where the variables to be flipped are totally uniformly selected

Table 7 Time comparison between D^2TS and other algorithms on larger problems: average time to the best solution in the run (in seconds)

Instance	D^2TS	ITS	MST1	MST2	SA	MA
p3000.1	209	228	396	106	251	726
p3000.2	245	212	395	97	337	809
p3000.3	237	327	464	271	517	590
p3000.4	334	519	480	559	336	722
p3000.5	476	462	436	255	327	638
p4000.1	274	215	776	436	842	1515
p4000.2	756	1070	785	1082	1680	1063
p4000.3	534	730	1011	359	1094	1106
p4000.4	678	845	656	624	1002	1373
p4000.5	2106	797	862	700	1279	1287
p5000.1	3368	1520	2260	1621	1816	3000
p5000.2	3465	1264	1984	1946	2072	2562
p5000.3	3278	2015	1410	2365	2836	2925
p5000.4	2513	1787	2005	2805	3178	2075
p5000.5	1869	1652	1922	2156	3171	3095
p6000.1	1527	2935	2860	3112	1844	4009
p6000.2	4628	2517	3119	2661	3256	3688
p6000.3	5264	2871	3217	3655	4422	4364
p7000.1	4649	5313	4954	4348	5806	7942
p7000.2	2547	3039	4484	5165	5215	5525
p7000.3	8436	4339	2801	6342	6417	8197
Average	2257	1650	1775	1936	2271	2724

Table 8 Results of longer runs of ITS and D^2TS

Instance	D^2TS	Time(s)	ITS	Time(s)
p5000.1	**8559680**	4531	8559355	3457
p5000.4	12252318	1698	12252318	12605
p6000.3	16132915	3125	16132915	9830
p7000.1	14478676	6214	14478676	30198
p7000.2	18249948	8423	18249948	1877

without using any memory information. In order to observe the difference between these two perturbation strategies, we disable the memory-based perturbation within the D^2TS algorithm and replace it by the random one while keeping other components unchanged. The algorithm stops after performing 100 perturbation operations. All other parameters are set as described in Sect. 3.1. For the purpose of illustration, we choose two large instances with 5000 variables (p5000.1 and p5000.4 with density equal to 0.5 and 1.0, respectively) as our test bed.

Figure 1 shows the running profiles of the two perturbation strategies. Each point represents the best solution cost (averages over 10 independent runs) found at the moment of each perturbation. It is easy to observe that on both instances the *MBP* strategy obtains better results than the *PRP* strategy, especially when the perturbation iterations become large. We found the same results to occur in other instances.

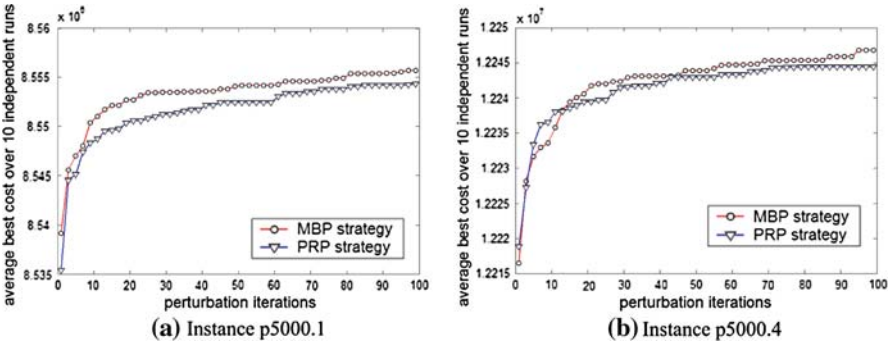

Fig. 1 Comparison between the memory-based perturbation operator with the pure random perturbation operator

4 Discussion and conclusion

Our Diversification-Driven Tabu Search (D^2TS) algorithm for solving unconstrained binary quadratic problems alternates between a rudimentary tabu search procedure (TS^o) and a memory-based perturbation strategy specially designed to achieve diversification. In spite of being quite simple in comparison with most top performing algorithms, D^2TS proves to be highly effective in finding good solutions for two sets of 31 benchmark instances of medium and large sizes, containing from 2500 to 7000 variables. Compared with the five state-of-the-art algorithms from the literature, D^2TS is able to find all the previous best known solutions (which none of the previous methods succeeded in doing) and obtains a new best, previously unknown, solution for one instance of 5000 variables.

There are several directions to extend this work. One immediate possibility is to examine other neighborhoods. D^2TS and most existing algorithms are based on the simple 1-flip neighborhood. Richer neighborhoods using for instance the 2-flip move as described in Glover and Hao (2009b) would be worth examining. Joining such approaches with associated strategies to focus only on a selected subset of neighbors would enhance their effectiveness, given the computational expense of examining all neighbors at each iteration. Similarly, instead of using the objective function as the unique evaluation measure, other evaluation functions using additional information would likewise be worth exploring. Finally, more advanced adaptive memory strategies from tabu search afford opportunities for creating further improvements.

Acknowledgment We are grateful for comments by the referees that have improved the paper. The work is partially supported by a "Chaire d'excellence" from "Pays de la Loire" Region (France) and regional MILES (2007–2009) and RaDaPop projects (2009–2012).

References

Alidaee B, Kochenberger GA, Ahmadian A (1994) 0–1 quadratic programming approach for the optimal solution of two scheduling problems. Int J Syst Sci 25:401–408

Alidaee B, Kochenberger GA, Lewis K, Lewis M, Wang H (2008) A new approach for modeling and solving set packing problems. Eur J Oper Res 86(2):504–512

Alkhamis TM, Hasan M, Ahmed MA (1998) Simulated annealing for the unconstrained binary quadratic pseudo-boolean function. Eur J Oper Res 108:641–652
Amini M, Alidaee B, Kochenberger GA (1999) A scatter search approach to unconstrained quadratic binary programs. McGraw-Hill, New York, pp 317–330. New Methods in Optimization
Beasley JE (1996) Obtaining test problems via internet. J Glob Optim 8:429–433
Beasley JE (1998) Heuristic algorithms for the unconstrained binary quadratic programming problem. Working Paper, The Management School, Imperial College, London, England
Borgulya I (2005) An evolutionary algorithm for the binary quadratic problems. Adv Soft Comput 2:3–16
Boros E, Hammer PL, Tavares G (2007) Local search heuristics for quadratic unconstrained binary optimization (QUBO). J Heuristics 13:99–132
Chardaire P, Sutter A (1994) A decomposition method for quadratic zero-one programming. Manage Sci 41(4):704–712
Gallo G, Hammer P, Simeone B (1980) Quadratic knapsack problems. Math Programm 12:132–149
Glover F, Hao JK (2009a) Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems. To appear in Int J Metaheuristics, 1(1)
Glover F, Hao JK (2009b) Fast 2-flip move evaluations for binary unconstrained quadratic optimization problems. To appear in Int J Metaheuristics
Glover F, Laguna M (1997) Tabu search. Kluwer, Boston
Glover F, Kochenberger GA, Alidaee B (1998) Adaptive memory tabu search for binary quadratic programs. Manag Sci 44:336–345
Harary F (1953) On the notion of balanced of a signed graph. Mich Math J 2:143–146
Katayama K, Narihisa H (2001) Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. Eur J Oper Res 134:103–119
Katayama K, Tani M, Narihisa H (2000) Solving large binary quadratic programming problems by an effective genetic local search algorithm. In: Proceedings of the genetic and evolutionary computation conference (GECCO'00). Morgan Kaufmann, pp 643–650
Kochenberger GA, Glover F, Alidaee B, Rego C (2004) A unified modeling and solution framework for combinatorial optimization problems. OR Spectrum 26:237–250
Kochenberger GA, Glover F, Alidaee B, Rego C (2005) An unconstrained quadratic binary programming approach to the vertex coloring problem. Ann Oper Res 139:229–241
Krarup J, Pruzan A (1978) Computer aided layout design. Math Programm Study 9:75–94
Laughunn DJ (1970) Quadratic binary programming. Oper Res 14:454–461
Lewis M, Kochenberger GA, Alidaee B (2008) A new modeling and solution approach for the set-partitioning problem. Comput Oper Res 35(3):807–813
Lodi A, Allemand K, Liebling TM (1999) An evolutionary heuristic for quadratic 0-1 programming. Eur J Oper Res 119(3):662–670
Lü Z, Hao JK (2009) A critical element-guided perturbation strategy for iterated local search. In: Cotta C, Cowling P (eds) Ninth European conference on evolutionary computation in combinatorial optimization (EvoCop 2009). Springer, LNCS 5482, pp 1–12
McBride RD, Yormark JS (1980) An implicit enumeration algorithm for quadratic integer programming. Manag Sci 26:282–296
Merz P, Freisleben B (1999) Genetic algorithms for binary quadratic programming. In: Proceedings of the genetic and evolutionary computation conference (GECCO'99). Morgan Kaufmann, pp 417–424
Merz P, Freisleben B (2002) Greedy and local search heuristics for unconstrained binary quadratic programming. J Heuristics 8:197–213
Merz P, Katayama K (2004) Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78:99–118
Palubeckis G (2004) Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Ann Oper Res 131:259–282
Palubeckis G (2006) Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2):279–296
Pardalos P, Rodgers GP (1990) Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45:131–144
Pardalos P, Xue J (1994) The maximum clique problem. J Glob Optim 4:301–328
Phillips AT, Rosen JB (1994) A quadratic assignment formulation of the molecular conformation problem. J Glob Optim 4:229–241
Witsgall C (1975) Mathematical methods of site selection for electronic system (ems). NBS Internal Report