European Journal of Operational Research 207 (2010) 1254–1262

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A hybrid metaheuristic approach to solving the UBQP problem

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

[a] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
[b] OptTek Systems, Inc., 2241 17th Street Boulder, CO 80302, USA

ARTICLE INFO

Article history:
Received 10 December 2009
Accepted 23 June 2010
Available online 30 June 2010

Keywords:
UBQP
Memetic algorithm
Tabu search
Pool updating

ABSTRACT

This paper presents a hybrid metaheuristic approach (HMA) for solving the unconstrained binary quadratic programming (UBQP) problem. By incorporating a tabu search procedure into the framework of evolutionary algorithms, the proposed approach exhibits several distinguishing features, including a diversification-based combination operator and a distance-and-quality based replacement criterion for pool updating. The proposed algorithm is able to easily obtain the best known solutions for 31 large random instances up to 7000 variables (which no previous algorithm has done) and find new best solutions for three of nine instances derived from the set-partitioning problem, demonstrating the efficacy of our proposed algorithm in terms of both solution quality and computational efficiency. Furthermore, some key elements and properties of the HMA algorithm are also analyzed.

1. Introduction

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

$$f(x) = x'Qx = \sum_{i=1}^{n}\sum_{j=1}^{n} q_{ij}x_i x_j, \quad (1)$$

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

Besides its theoretical significance as a canonical NP-hard problem [13], the UBQP is notable for its ability to formulate a wide range of important problems, including those from computer aided design [28], social psychology [21], traffic management [12,45], financial analysis [29,34], machine scheduling [1] and cellular radio channel allocation [10]. Moreover, the application potential of UBQP is much greater than might be imagined, due to the ability to incorporate quadratic infeasibility constraints into the objective function in an explicit manner. This re-formulation process enables UBQP to serve as a common model for a wide range of combinatorial optimization problems. A review of additional applications and the re-formulation procedures can be found in [26] demonstrating the utility of UBQP for a variety of applications, such as the vertex coloring problem [27], the set packing problem [2], the set-partitioning problem [30] and the linear ordering problem [31].

Given the relevance of the UBQP across a broad spectrum of problems, a large number of procedures for solving this model have been reported in the literature. Among them are several exact methods using branch and bound or branch and cut (see, e.g., [8,22,23,42]), but the high computational complexity of UBQP and the breadth of applications it embraces has led to the finding that, apart from isolated cases, problems of sizes larger than $n = 100$ cannot be solved by these exact methods in a reasonable time.

For larger instances, the exact methods become prohibitively expensive to apply. By contrast, variants of metaheuristic algorithms have been extensively used to solve UBQP and shown to be effective to find high-quality solutions in an acceptable time. Some representative metaheuristic methods include local search based approaches, both direct [9] and using Simulated Annealing [3,6,24]; adaptive memory approaches based on tabu search [6,16,40,41]; population-based approaches such as evolutionary algorithms [7,25,32,35], scatter search [4] and memetic algorithms [37]; and specially designed one pass heuristics [17].

This paper presents a hybrid metaheuristic algorithm for the UBQP, which integrates a tabu search procedure with an evolutionary approach, hence placing it in the category of a memetic algorithm (see, e.g., [38,39]). Our proposed algorithm is characterized by several features that enhance its effectiveness. First, we introduce a diversification-guided combination operator based on path-relinking, called DG/PR operator to produce a combination scheme that more fully exploits the problem structure within the present context. Second, the proposed DG/PR operator is jointly employed with the conventional uniform crossover operator of genetic algorithms to generate a diversified set of new solutions. Finally, our algorithm relies on a quality-and-distance replacement strategy for population updates to maintain the population diversity. These features distinguish our algorithm from previous population-based algorithms reported in [7,25,35,37].

To assess the performance and the competitiveness of our memetic algorithm in terms of both solution quality and efficiency,

* Corresponding author. Tel.: +33 2 41 73 52 94.
E-mail addresses: zhipeng.lui@gmail.com, lu@info.univ-angers.fr (Z. Lü), glover@opttek.com (F. Glover), hao@info.univ-angers.fr (J.-K. Hao).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.06.039

we provide computational results on the 31 large random benchmark instances with up to 7000 variables as well as nine instances derived from the set partitioning problem, comparing our outcomes with the best results of the literature.

The remaining part of the paper is organized as follows. In Section 2, the ingredients of our algorithm are described, including the tabu search procedure, the diversification-guided combination operator and the pool updating rule. Section 3 is dedicated to the computational results. Section 4 investigates several essential components of the proposed HMA algorithm and concluding remarks are given in Section 5.

2. Hybrid evolutionary algorithm

2.1. Main scheme

Hybrid (memetic) evolutionary algorithms are known to be highly effective for solving a large number of constraint satisfaction and optimization problems. By combining the more global recombinant search and the more intensive local search, the memetic framework offers a useful balance between intensification and diversification as a means of exploiting the search space.

In principle, our hybrid metaheuristic algorithm (HMA) repeatedly alternates between a combination operator that is used to generate new offspring solutions and a tabu search procedure that optimizes the newly generated offspring solutions. As soon as an offspring solution is improved by tabu search, the population is accordingly updated based on two criteria: solution quality and population diversity.

Algorithm 1. Pseudo-code of our memetic algorithm for UBQP

```
1:  Input: matrix Q
2:  Output: the best binary n-vector x* found so far
3:  P = {x^1,...,x^p} ← Population_Initialization()   /* Section 2.3 */
4:  for i = {1,...,p} do
5:     x^i ← Tabu_Search(x^i)                         /* Section 2.4 */
6:  end for
7:  x* = arg max{f(x^i)|i = 1,...,p}
8:  repeat
9:     randomly choose two individuals x^j and x^k from P
10:    x^0 ← Combination_Operator(x^j, x^k)          /* Section 2.5 */
11:    x^0 ← Tabu_Search(x^0)                        /* Section 2.4 */
12:    if f(x^0) > f(x*) then
13:       x* = x^0
14:    end if
15:    {x^1,...,x^p} ← Pool_Updating(x^0,            /* Section 2.6 */
       x^1,...,x^p)
16: until a stop criterion is met
```

The general architecture of the HMA algorithm is described in Algorithm 1. It is composed of four main components: population initialization, a tabu search procedure, a combination operator and a population updating rule. Starting from an initial random population, HMA uses the TS procedure to optimize each individual to reach a local optimum (lines 4–6). Then, the combination operator is employed to generate new offspring solutions (line 10), whereupon a new round of TS is again launched to optimize the objective function. Subsequently, the population updating rule decides whether such an improved solution should be inserted into the population and which existing individual should be replaced (line 15). In the following subsections, the main components of our memetic algorithm are described in details.

2.2. Search space and evaluation function

The search space of our algorithm consists of all the binary n-vectors. Thus, the size of the whole search space equals 2^n. The evaluation function that the HMA algorithm employs is just the objective function $f(x)$ in Eq. (1).

2.3. Initial population

In HMA, the individuals of initial population are generated randomly, i.e., each variable of an individual solution is assigned a value 0 or 1 with equal probability. Then, each individual is further optimized by a TS procedure. Moreover, we take advantage of the initialization step to build a diversified population. A new individual is added to the population only if it is not too *close* to any of the existing solutions of the population. Otherwise, the individual is discarded and another new individual is generated. The *distance threshold* for executing this rule is discussed in Section 2.6. Let us mention that in general the initial solutions have limited influence on the solution quality obtained by the HMA algorithm.

2.4. Tabu search procedure

As demonstrated in [16] and more recently in [20,40,41], TS is one of the most successful approaches for the UBQP. In this paper, we employ a TS algorithm based on a simple *one-flip move* neighborhood. The one-flip move 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 necessary to be able to rapidly determine the effect of a move on the objective function $f(x)$. In our implementation, this neighborhood uses a fast incremental evaluation technique introduced in [16] and enhanced in [18,19] to calculate the cost (move value) of transitioning to each neighboring solution. The procedure maintains a data structure that stores the move value (change in $f(x)$) for each possible move, and employs a streamlined calculation for updating this data structure after each iteration.

More formally, 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 Δ_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 move value can be calculated in linear time using the formula:

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

Once a move is performed, one needs just to update a subset of move values affected by the move. Specifically, the following abbreviated calculation can be performed to update the move values upon flipping a variable x_i:

(1) $\Delta_i = -\Delta_i$.
(2) For each $j \in N - \{i\}$,
if $x_j = x_i$, $\Delta_j = \Delta_j + q_{(i,j)}$.
if $x_j = 1 - x_i$, $\Delta_j = \Delta_j - q_{(i,j)}$.

We employ the convention that x_i represents x_i's value before being flipped.

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 [14]. In

our implementation, each time a variable x_i is flipped, a value is assigned to an associated record *TabuTenure*(i) (identifying the "tabu tenure" of x_i) to prevent x_i from being flipped again for the next *TabuTenure*(i) iterations. For the current study, we elected to set

$$TabuTenure(i) = tt + rand(10), \tag{3}$$

where tt is a given constant and rand(10) takes a random value from 1 to 10.

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

Accompanying this rule, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution. Our TS method stops when the best solution cannot be improved within a given number α of moves and we call this number the *improvement cutoff*.

2.5. Combination operator

In HMA, we jointly use two kinds of combination operators to generate both suitable and diversified offspring: one is the uniform crossover widely used in the literature; the other is a diversification-guided combination operator proposed in this paper. At each iteration, we randomly choose one of these two operators with equal probability to generate new offspring solutions.

2.5.1. Parent selection

The application of both combination operators used in our algorithm is controlled by the Hamming distance h_{ij} between two parent solutions x^i and x^j (i.e., h_{ij} equals the number of variables that receive different values in the parents). We require that two solutions chosen as parents must satisfy $h_{ij} > \overline{h}$, where $\overline{h}$ denotes the average distance between pairs of solutions in the population. Therefore, we have $\overline{h} = \frac{2}{p(p-1)} \sum_{i=1}^{p} \sum_{j=i+1}^{p} h_{ij}$, where p denotes the population size.

2.5.2. Uniform crossover

The main idea of uniform crossover is to assign values to the variables of offspring that represent assignments made in common by both parents, and to randomly assign values to remaining variables of the offspring solution [44]. However, the fitness landscape analysis conducted in [37] has shown that in some cases uniform crossover may not be effective for the UBQP problem, since the probability that the offspring is identical to one of the parents after the crossover *and* local search is high. In order to fix this drawback, we propose the following DG/PR combination procedure to generate offspring whose solution quality and diversity are both reasonably respected.

2.5.3. Diversification-guided path-relinking

Due to the disadvantages of uniform crossover mentioned above, we attempt to design a combination procedure that is able to diversify the search trajectory even after the local search improvement. For this we draw on an instance of the evolutionary path-relinking type of combination (see, e.g., [15]). The design of our procedure is guided by two main objectives inherent in path-relinking: one is to diversify the search trajectory; the other is to explore new search regions that are promising. For this purpose, we construct an offspring solution by considering both the solution quality and its distance to its parent solutions.

On the one hand, if we could control the distance between an offspring solution and its two parents, it would be possible to generate a diversified offspring relative to its parents. On the other hand, it is useful to take into account the objective function during the construction of the offspring such that it might be able to visit search regions which are different from those containing their parent solutions. Path-relinking is a natural mechanism for integrating these two types of control.

Our DG/PR procedure begins by exploiting the connection between path-relinking and scatter search, by first producing a basic scatter search combination $x^0 = \{x_1^0, \ldots, x_n^0\}$ to yield the initiating solution for path-relinking, setting

$$x^0 = .5(x^i + x^j), \tag{4}$$

where $x^i = \{x_1^i, \ldots, x_n^i\}$ and $x^j = \{x_1^j, \ldots, x_n^j\}$ are two selected parent solutions. (This commonly used instance of a scatter search combination is also implicitly used as a starting point in [37].)

Let C and NC denote respectively, the index set of the common and uncommon variables, i.e. $C = \{t | t \in N, x_t^i = x_t^j\}$ and $NC = \{t \mid t \in N, x_t^i \neq x_t^j\}$ and let x^c denote either of the two parent solutions; hence $c = i$ or $c = j$. Then we may equivalently identify x^0 by

$$x_t^0 = \begin{cases} x_t^c, & t \in C; \\ 0.5, & t \in NC. \end{cases} \tag{5}$$

We now take x^i and x^j as the guiding solutions for x^0 using the customary path-relinking design, where at each step we replace a selected component of x^0 by the corresponding component of x^c for $c = i$ or j. By the usual choice rule, we select the component x_t^c for $t \in NC$ that produces a new x^0 with the best objective function evaluation. In addition, we alternate the choice of the guiding solution so that $c = i$ on odd steps and $c = j$ on even steps.

In particular, we first calculate, for each x_t, $t \in NC$, a move value g_t^c of changing x_t^0 from 0.5 to x_t^c. Then we sequentially fix, in k ($1 \leqslant k \leqslant |NC|$) steps, each variable in NC by changing its value from 0.5 to 0 or 1, where $x^c = x^i$ or x^j according to whether k is odd or even. Then, we select x_{t^*} having the best move value g_t^c (i.e., $t^* = arg\ max\{g_t^c | t \in NC\}$) and assign $x_{t^*}^c$ to $x_{t^*}^0$. After this assignment $NC = NC - \{t^*\}$ and the move values g_t^c are updated for the new NC.

The Pseudo-code of this offspring construction procedure is presented in Algorithm 2. One observes that offspring x^0 generated by this procedure has essentially the same Hamming distance $|NC|/2$ to its two parents (referring here to the original NC, and rounding $|NC|/2$ down for one parent and up for the other, as appropriate). Because of the path-relinking choice of a best component to transfer from the chosen x^c to x^0 at each step, the resulting offspring x_0 is of relatively high solution quality, and its final separation of $|NC|/2$ from x^i and x^j reduces the possibility of repeating its parent solutions after the tabu search procedure.

Algorithm 2. Pseudo-code of the DG/PR combination procedure

1: **Input**: Two parents x^i and x^j
2: **Output**: One offspring x^0
   % lines 3–4: create the initial x^0
   % lines 5–13: update x^0 by path-relinking
3: Identify the sets of common and uncommon variables C and NC
4: Initialize the value assignments of x^0 according to Eq. (5)
5: Calculate the move values for changing x_t^0 from 0.5 to $x_t^i(g_t^i)$ or $x_t^j(g_t^j)$
6: **for** $k = 1, \ldots, |NC|$ **do**
7: Select parent x^c to be considered: if k is odd, then $c = i$, else $c = j$
8: Rank all g_t^c values in a decreasing order, where $t \in NC$
9: Select x_{t^*} with the best move value g_t^c
10: Copy the value assignment in x^c to variable x_{t^*}: $x_{t^*}^0 = x_{t^*}^c$
11: Update all g_t^c values ($t \in NC$) affected by the move
12: NC: $\leftarrow NC - \{t^*\}$
13: **end for**

2.6. Population updating

In the HMA algorithm, when an offspring x^0 is obtained by the combination operator, we improve x^0 by the tabu search algorithm and then decide whether the offspring should be inserted into the population, replacing the *worst* solution in the population according to a distance-and-quality goodness score function (see, e.g., [33]). The main idea is to favor the inclusion of x^0 in the population if x^0 is "good enough" (in terms of its objective function evaluation) and is not too *similar* to any current solution in the population. The approach of accounting for (and hence avoiding) similarity in population updating is commonly employed in scatter search and path-relinking methods. (See, e.g., the survey [43].)

For the UBQP problem, the similarity between two solutions is generally measured by the Hamming distance between two solutions, which is defined as the number of positions at which the corresponding values are different. In other words, it measures the minimum number of flip moves required to change one solution into the other. In this paper, we propose a different type of distance definition for estimating the similarity between two solutions. The basic idea is that, in contrast to treating each variable the same as in the Hamming distance, we account for the importance of the variables when calculating the distance. It is reasonable to think that a variable x_i associated with large values of $|q_{ii}|$ and $|\sum_{j=1}^{n} q_{ij}|$ is more important than another variable associated with small values of these quantities. This idea is also in accordance with the spirit of the basic one-pass heuristic of [17]. In order to make things clearer, we make use of the following definitions:

Definition 1 (*Variable's importance*). The importance of the ith variable x_i of a solution x is defined as:

$$VI_i = \left(|q_{ii}| + \varphi \sum_{j=1}^{n} |q_{ij}| \right)^{\frac{1}{2}}, \tag{6}$$

where φ is a small constant parameter. In this paper, we empirically set $\varphi = 0.2$.

Definition 2 (*Distance between two solutions*). Given two solutions $x^a = \{x_1^a, \ldots, x_n^a\}$ and $x^b = \{x_1^b, \ldots, x_n^b\}$, the distance between x^a and x^b is defined as the total sum of the importance of the variables having different values in the two solutions, denoted by d_{ab}:

$$d_{ab} = \sum_{i=1, x_i^a \neq x_i^b}^{n} VI_i. \tag{7}$$

Definition 3 (*Distance between one solution and a population*). Given a population $P = \{x^1, \ldots, x^p\}$ and the distance d_{ij} between any two solutions x^i and x^j ($i, j = 1, \ldots, p, i \neq j$), the distance between a solution x^i ($i = 1, \ldots, p$) and the population P is defined as the minimum distance between x^i and any other solution in P, denoted by $D_{i,P}$:

$$D_{i,P} = min\{d_{ij} | x^j \in P,\ j \neq i\}. \tag{8}$$

Definition 4 (*Goodness score of a solution for a population*). Given a population $P = \{x^1, \ldots, x^p\}$ and the distance $D_{i,P}$ for any solution x^i ($i = 1, \ldots, p$), the goodness score of solution x^i for population P is defined as:

$$g(i, P) = \beta \widetilde{A}(f(x^i)) + (1 - \beta)\widetilde{A}(D_{i,P}), \tag{9}$$

where $f(x^i)$ is the objective function value of solution x^i and $\widetilde{A}(\cdot)$ represents the normalized function:

$$\widetilde{A}(y) = \frac{y - y_{min}}{y_{max} - y_{min} + 1}, \tag{10}$$

where y_{max} and y_{min} are respectively, the maximum and minimum values of y in the population P. The number "1" is used to avoid the possibility of a 0 denominator. β is a constant parameter and we empirically set $\beta = 0.6$ in this paper.

It is clear that the greater the goodness score $g(i,P)$, the better solution x^i. This goodness score function simultaneously considers the factors of solution quality and diversity of the population. On the one hand, we should maintain a pool of elite solutions. On the other hand, we have to emphasize the importance of the diversity of the solutions to avoid a premature convergence of the population.

The Pseudo-code of our pool updating strategy is presented in Algorithm 3. Given an offspring x^0 optimized by tabu search and a population $P = \{x^1, \ldots, x^p\}$, we use the following rule to decide whether x^0 should be inserted into the population. First of all, x^0 is temporarily inserted into the population P (line 3). Then, the goodness score for each solution $x^i \in P'$ is calculated according to Eq. (9) (lines 4–7) and the worst solution in the original population P (with the smallest value of goodness score) is identified, denoted by x^w (line 8). If the goodness score of the offspring solution x^0 is no smaller than that of the worst solution x^w, then x^0 will be inserted into the population and replace x^w. Otherwise, the worst solution x^w is replaced by x^0 with a small probability $wp = 0.3$ (lines 9–11). This strategy is used to avoid the premature of the population.

Algorithm 3. Pseudo-code of the pool updating strategy

1: **Input**: Population $P = \{x^1, \ldots, x^p\}$ and offspring solution x^0
2: **Output**: Updated Population $P = \{x^1, \ldots, x^p\}$
3: Tentatively add x^0 to Population P: $P' = P \cup \{x^0\}$
4: **for** $i = 0, \ldots, p$ **do**
5: Calculate the distance between x^i and P' according to Eq. (8)
6: Calculate the goodness score of x^i ($g(i,P')$) according to Eq. (9)
7: **end for**
8: Identify the solution with the worst goodness score in the original population P:
$x^w = arg\,min\{g(i,P') | i = 1, \ldots, p\}$
9: **if** $g(0,P') \geqslant g(w,P')$ or $rand(0,1) < 0.3$ **then**
10: Replace x^w with x^0: $P = P \cup \{x^0\} \backslash \{x^w\}$
11: **end if**

3. Computational results

In this section, we report intensive experimental results of the HMA algorithm on 40 large and difficult instances available in the literature and compare them with those of the best performing algorithms.[1]

3.1. Test instances

Three sets of test problems are considered in the experiments, in total constituting 40 instances. The first set of benchmarks is composed of 10 large instances of size $n = 2500$ introduced in [6] and available in the ORLIB [5]. 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 [6,24,36,37,40,41]. Note that the small test instances from the ORLIB whose sizes range from $n = 50$–1000 and the similarly small instances from [16] are not considered here, since they present no challenge for the HMA algorithm. Specifically, all their best known objective values can be obtained within 2 seconds by our algorithm.

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

Table 1
Settings of important parameters.

Parameters	Section	Description	Values	
			Random	SPP
tt	2.4	Tabu tenure constant	$n/150$	$n/50$
α	2.4	Improvement cutoff of TS	$5n$	$15n$
φ	2.6	Variable importance coefficient	0.2	0.2
β	2.6	Goodness score coefficient	0.6	0.6
p	2.6	Population size	20	20

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 $n = 3000$–7000 and with densities from 0.5 to 1.0 [40,41]. 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 [40,41] showed that these large instances are particularly challenging UBQP problems.

The third set of benchmarks include nine instances derived from the set partitioning problem, named spp1,...,spp9, with variable sizes ranging from 600 to 1000. This set of instances and the best results obtained by CPLEX solver are available at: http://academic.missouriwestern.edu/mlewis14/ProblemSets/summarytable.htm. For the approach to convert a set-partitioning problem into a UBQP problem, interested readers are referred to [30].

3.2. 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 512 M RAM. All computational results were obtained without special tuning of the parameters, i.e., all the parameters used in our algorithm are fixed (constant) for all instances considered. Table 1 gives the descriptions and settings of the parameters used in the HMA algorithm, where the last two columns respectively, denote the settings for the set of 31 random instances and the set of 9 set-partitioning instances. Given the stochastic nature of the HMA algorithm, each problem instance is independently solved 20 times.

3.3. Computational results on the random instances

Our first experiment aims to evaluate the HMA algorithm on the 31 random instances with 2500–7000 variables (the first two sets). The results of this experiment are summarized in Tables 2 and 3. The stop condition for the 10 Beasley instances is set to be 40 seconds. For the instances with 3000, 4000, 5000, 6000, and 7000 variables, the time limit for a single run is respectively set to be 5, 10, 20, 30 and 50 minutes on our computer. Note that this time cutoff is the same as in [41], which uses a Pentium III 800 PC and the time limit in [41] was set to be three times of the above values (since our computer is about three times faster than theirs).[2]

Tables 2 and 3, respectively show the computational statistics of the HMA algorithm on the 10 ORLIB instances with 2500 variables and the 21 large random instance with 3000 to 7000 variables. In both tables, columns 2 and 3, respectively give the density (dens) and the previous best objective values (f_{prev}). Note that in Table 3, the best objective values f_{prev} are compiled from Tables 4 and 7 in [41] and Table 8 in [20]. To our knowledge, they are the current best known results for these 21 problem instances.

Columns 4–9 give our results: the best objective value (f_{best}), the best solution gap to the previous best known objective values g_{best} (i.e., $f_{prev} - f_{best}$), the average solution 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), the success rate (suc) for reaching the best known results f_{prev}, the best and the average CPU time (seconds) for reaching the best results f_{best} (denoted by t_{best} and t_{avr}, respectively) over 20 runs. Furthermore, the last row "Average" indicates the summary of our algorithm's average performance.

Table 2 discloses that HMA can stably reach all the previous best objective values for the 10 largest Beasley instances within 9.41 seconds on average. In particular, our algorithm can reach all the previous best objective values within 1.47 seconds in the best case out of the 20 runs. Table 3 shows that for the 21 large and difficult random instances, our algorithm can also easily reach the previous best known objective values within the given time limit. The average gap to the previous best objective values (g_{avr}) is 489.4 for these instances. The average success rate is about 12 out of 20 runs for this set of benchmarks. The average CPU time to obtain the best known objective values is 746.6 seconds. It should be noticed that our computing time to reach the best known objective values is much shorter than those reported in previous studies like [41,20].

In order to further evaluate our HMA algorithm, we compare our results with some best performing algorithms in the literature. For this purpose, we restrict attention to comparisons with five methods that have reported the best results for the more challenging problems. These methods are respectively named ITS [41], MST1 [40], MST2 [40], SA [24] and D^{2}TS [20].

Table 4 shows the best results of our HMA algorithm compared with the reference algorithms. The results of the first four reference algorithms are directly extracted from [41] and those of D^{2}TS are from [20]. Note that the results of all these algorithms are obtained under the same time limit. Table 4 displays the solution difference between the best objective values obtained by these six algorithms with the best known objective values. The averaged results over the 11 instances are presented in the last row.

From Table 4 it may be observed that the HMA algorithm outperforms the four reference algorithms, named ITS, MST1, MST2 and SA, in terms of the quality of the best solution. Specifically, these four algorithms have an average solution gap from 306.8 to 3634.9, while HMA can find all the best known objective values within the given time limit. Moreover, HMA also performs slightly better than our previous algorithm D^{2}TS, which still has an average solution gap of 20.4 to the best known objective values. These results show the efficacy of the HMA algorithm in finding the best objective values.

3.4. Computational results on the structured instances

In this section, we test the HMA algorithm on the nine structured instances derived from the set-partitioning problem. The results of this experiment are summarized in Table 5. The time limit for a single run is set to be 30 minutes on our computer.

In Table 5, columns 2–4, respectively give the density (dens), the best objective values (f_{prev}) obtained by CPLEX and the MIP gap to the optimal objective value. The six instances marked "*optimal*" are solved to optimality by CPLEX, while for the remaining three instances no optimal solutions can be found by CPLEX when allowing a solution time of 64,000 seconds. Columns 5–10 give the computational statistics of our algorithm, using the same criteria as in Tables 2 and 3.

Table 5 demonstrates that HMA can reach all the previous best objective values obtained by CPLEX within the given time limit (30 minutes). Moreover, our algorithm can obtain new better

[2] We tested a benchmark program on our computer and a Pentium III 800 PC with 512 M 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 2
Computational results on the 10 large Beasley instances with 2500 variables.

Instance	Dens	f_{prev}	HMA algorithm					
			f_{best}	g_{best}	g_{avr}	*suc*	t_{best}	t_{avr}
b2500.1	0.1	1,515,944	1,515,944	0	0.0	20/20	1.41	2.56
b2500.2	0.1	1,471,392	1,471,392	0	20.2	18/20	1.87	20.6
b2500.3	0.1	1,414,192	1,414,192	0	0.0	20/20	1.72	12.7
b2500.4	0.1	1,507,701	1,507,701	0	0.0	20/20	0.86	1.23
b2500.5	0.1	1,491,816	1,491,816	0	0.0	20/20	0.49	1.56
b2500.6	0.1	1,469,162	1,469,162	0	0.0	20/20	1.21	3.45
b2500.7	0.1	1,479,040	1,479,040	0	0.0	20/20	1.77	13.2
b2500.8	0.1	1,484,199	1,484,199	0	0.0	20/20	1.19	5.34
b2500.9	0.1	1,482,413	1,482,413	0	0.0	20/20	2.76	15.9
b2500.10	0.1	1,483,355	1,483,355	0	0.0	20/20	1.37	17.6
Average				0	2.02	19.8/20	1.47	9.41

Table 3
Computational results on the 21 larger random instances with 3000–7000 variables.

Instance	Dens	f_{prev}	HMA algorithm					
			f_{best}	g_{best}	g_{avr}	*suc*	t_{best}	t_{avr}
p3000.1	0.5	3,931,583	3,931,583	0	0.0	20/20	3.63	42.5
p3000.2	0.8	5,193,073	5,193,073	0	0.0	20/20	5.52	54.8
p3000.3	0.8	5,111,533	5,111,533	0	32.6	17/20	8.58	92.9
p3000.4	1.0	5,761,822	5,761,822	0	0.0	20/20	7.78	106.1
p3000.5	1.0	5,675,625	5,675,625	0	144.8	15/20	22.6	123.5
p4000.1	0.5	6,181,830	6,181,830	0	0.0	20/20	4.99	48.8
p4000.2	0.8	7,801,355	7,801,355	0	142.4	17/20	34.4	267.4
p4000.3	0.8	7,741,685	7,741,685	0	6.0	19/20	35.4	276.1
p4000.4	1.0	8,711,822	8,711,822	0	37.8	18/20	53.1	273.7
p4000.5	1.0	8,908,979	8,908,979	0	546.2	12/20	89.7	305.2
p5000.1	0.5	8,559,680	8,559,680	0	506.8	4/20	153.2	587.3
p5000.2	0.8	10,836,019	10,836,019	0	512.3	6/20	98.7	463.8
p5000.3	0.8	10,489,137	10,489,137	0	332.1	14/20	364.5	758.3
p5000.4	1.0	12,252,318	12,252,318	0	1228.2	3/20	789.6	1452.6
p5000.5	1.0	12,731,803	12,731,803	0	284.3	16/20	212.3	685.6
p6000.1	0.5	11,384,976	11,384,976	0	139.7	12/20	727.7	994.3
p6000.2	0.8	14,333,855	14,333,855	0	525.7	6/20	965.3	1332.1
p6000.3	1.0	16,132,915	16,132,915	0	2310.5	3/20	676.5	1405.6
p7000.1	0.5	14,478,676	14,478,676	0	818.5	5/20	987.3	1435.2
p7000.2	0.8	18,249,948	18,249,948	0	1322.9	2/20	1254.7	1769.8
p7000.3	1.0	20,446,407	20,446,407	0	1385.6	7/20	1868.5	2456.3
Average				0	489.4	12.2/20	418.1	746.6

Table 4
Best results comparison between HMA and other state-of-the-art algorithms for larger problem instances.

Instance	Dens	f_{prev}	Best solution gap (i.e., $f_{prev} - f_{best}$)					
			ITS	MST1	MST2	SA	D^2TS	HMA
p3000.1	0.5	3,931,583	0	0	0	0	0	0
p3000.2	0.8	5,193,073	0	0	0	0	0	0
p3000.3	0.8	5,111,533	0	357	0	0	0	0
p3000.4	1.0	5,761,822	0	0	0	0	0	0
p3000.5	1.0	5,675,625	0	478	0	0	0	0
p4000.1	0.5	6,181,830	0	0	0	0	0	0
p4000.2	0.8	7,801,355	0	1686	0	504	0	0
p4000.3	0.8	7,741,685	0	54	0	0	0	0
p4000.4	1.0	8,711,822	0	0	0	0	0	0
p4000.5	1.0	8,908,979	0	0	0	0	0	0
p5000.1	0.5	8,559,680	700	3016	325	1432	325	0
p5000.2	0.8	10,836,019	0	0	582	582	0	0
p5000.3	0.8	10,489,137	0	3277	0	354	0	0
p5000.4	1.0	12,252,318	934	3785	1643	444	0	0
p5000.5	1.0	12,731,803	0	5150	0	1025	0	0
p6000.1	0.5	11,384,976	0	3198	0	430	0	0
p6000.2	0.8	14,333,855	88	10001	0	675	0	0
p6000.3	1.0	16,132,915	2729	11658	0	0	0	0
p7000.1	0.5	14,478,676	340	7118	1607	2579	0	0
p7000.2	0.8	18,249,948	1651	8902	2330	5552	104	0
p7000.3	1.0	20,446,407	0	17652	0	2264	0	0
Average			306.8	3634.9	308.9	754.3	20.4	0

Table 5
Computational results on the nine set-partitioning problem instances.

Instance	Dens	f_{prev}	Gap	HMA algorithm					
				f_{best}	g_{best}	g_{avr}	*suc*	t_{best}	t_avr
spp1	0.1	9489	70%	**9520**	−31	−31	20/20	10.5	211.6
spp2	0.5	9232	Optimal	9232	0	42.6	4/20	51.3	358.1
spp3	0.9	9939	Optimal	9939	0	0.0	20/20	8.64	123.2
spp4[a]	0.1	22,095	84%	**22,174**	−79	170.1	5/20	2.85	181.5
spp5	0.5	23,032	Optimal	23,032	0	487.1	3/20	28.3	246.8
spp6	0.9	24,776	Optimal	24,776	0	13.9	16/20	69.2	524.3
spp7	0.1	42,927	74%	**43,282**	−355	179.7	4/20	42.8	268.9
spp8	0.5	45,656	Optimal	45,656	0	210.5	12/20	2.15	138.4
spp9	0.9	49,372	Optimal	49,372	0	217.8	6/20	26.7	256.7
Average					−51.7	143.4	10/20	26.9	256.6

[a] As shown in Table 6, a better value (22,335) can be obtained.

objective values for the three instances whose optimal objective values cannot be achieved by CPLEX. The average gap to the best known objective values over all the nine instances is only 143.4 and the average success rate is about 10 out of 20 runs. These results further provide evidence of the benefit of our hybrid metaheuristic approach.

4. Discussion and analysis

We now turn our attention to discussing and analyzing several important features of the proposed HMA algorithm, including the diversification-guided combination operator, the distance-and-quality population updating strategy and the tradeoffs between the Tabu Search procedure and the combination operations.

4.1. Uniform crossover vs. DG/PR combination

As indicated in Section 2.5.3, the DG/PR combination operator employs a version of path-relinking that alternates between two parent guiding solutions to replace components of the initiating solution. In order to be sure this new operator makes a meaningful contribution, we conduct additional experiments to compare the performance of the DG/PR operator and the uniform crossover operator on the 11 largest random instances with variables from 5000 to 7000 and the six difficult structured instances named spp4,...,spp9. We implement two variants of the HMA algorithm for this experiment, respectively denoted as $HMA_{dg/pr}$ and HMA_{ux}, where the DG/PR and uniform crossover operators are used separately and the remaining components are kept unchanged.

We run this experiment using $HMA_{dg/pr}$ and HMA_{ux} under exactly the same conditions as before and the results are reported in Table 6. Once again, the following information is provided for each instance: the best solution gap to the previous best known objective values g_{best}, the average solution gap to the previous best known objective values g_{avr} and the success rate (*suc*) for reaching the best known objective values f_{prev} over 20 runs.

For the 11 random instances, $HMA_{dg/pr}$ performs better than HMA_{ux} in terms of the best solution gap g_{best} (13.4 for $HMA_{dg/pr}$ against 188.3 for HMA_{ux}) and the success rate *suc* (4.9 vs. 3.5 over 20 runs) while $HMA_{dg/pr}$ performs worse in terms of the average solution gap g_{avr} (1424.1 for $HMA_{dg/pr}$ against 1018.5 for HMA_{ux}). In particular, $HMA_{dg/pr}$ fails to reach the best known objective values for two instances (p5000.4 and p7000.2), while HMA_{ux} fails for six instances. This indicates that $HMA_{dg/pr}$ is superior for finding the best solutions but HMA_{ux} is more stable in its performance in the case of these random problems.

For six structured instances, $HMA_{dg/pr}$ performs better (respectively worse) than HMA_{ux} for 4 (respectively 2) instances in terms of the best and average solution gaps g_{best} and g_{avr}. $HMA_{dg/pr}$ reaches or surpasses the best known objective values for three instances (spp4, spp8, and spp9), while HMA_{ux} reaches or surpasses the best known objective values for two instances (spp6 and spp7). In particular, it is remarkable that for spp4 whose optimum is unknown, $HMA_{dg/pr}$ finds an even better objective value than HMA (f_{best} = 22,335 against 22,174), which already improves on the current best objective value (22,095).

Table 6
Performance comparison of the DG/PR operator with uniform crossover.

Instance	f_{prev}	$HMA_{dg/pr}$			HMA_{ux}		
		g_{best}	g_{avr}	*suc*	g_{best}	g_{avr}	*suc*
p5000.1	8,559,680	0	578.9	6/20	325	467.2	0/20
p5000.2	10,836,019	0	715.4	5/20	0	534.3	5/20
p5000.3	10,489,137	0	712.3	7/20	0	338.2	11/20
p5000.4	12,252,318	43	1434.9	0/20	608	1204.6	0/20
p5000.5	12,731,803	0	632.7	8/20	0	511.9	8/20
p6000.1	11,384,976	0	196.4	9/20	0	111.9	11/20
p6000.2	14,333,855	0	798.5	5/20	88	542.9	0/20
p6000.3	16,132,915	0	2902.6	3/20	950	2594.3	0/20
p7000.1	14,478,676	0	1259.5	5/20	0	958.9	3/20
p7000.2	18,249,948	104	1988.1	0/20	41	2010.3	0/20
p7000.3	20,446,407	0	4445.6	6/20	59	1928.6	0/20
Average		**13.4**	1424.1	**4.9/20**	188.3	**1018.5**	3.5/20
spp4	22,095	**−240**	175.8	2/20	62	179.8	0/20
spp5	23,032	293	482.2	0/20	354	485.9	0/20
spp6	24,776	101	142.5	0/20	0	40.5	15/20
spp7	42,927	15	383.5	0/20	−27	280.7	3/20
spp8	45,656	0	184.5	5/20	37	217.1	0/20
spp9	49,372	0	173.4	4/20	61	210.3	0/20
Average		101.3	330.2	1.8/20	154.3	308.9	2.5/20

These results demonstrate that while uniform crossover has an overall good performance (with better average solution gaps), it may have difficulties in finding the best solutions on both random and structured instances. The design of our DG/PR operator, by contrast, favors the creation of diversified and promising offspring solutions in many situations, though it is less successful in a few cases.

This experiment highlights the complementary nature of the two combination operators and provides an empirical justification of their joint use in the HMA algorithm. From Tables 3 and 5, we observe that employing these two operators together proves useful for overall performance.

4.2. Population updating strategy

In order to evaluate the distance-and-quality based population updating strategy (denoted by *DisQual*), we compare it with a popular updating strategy widely used in the literature (denoted by *PoolWorst*), where the new offspring replaces the worst solution of the population only in terms of the objective function.

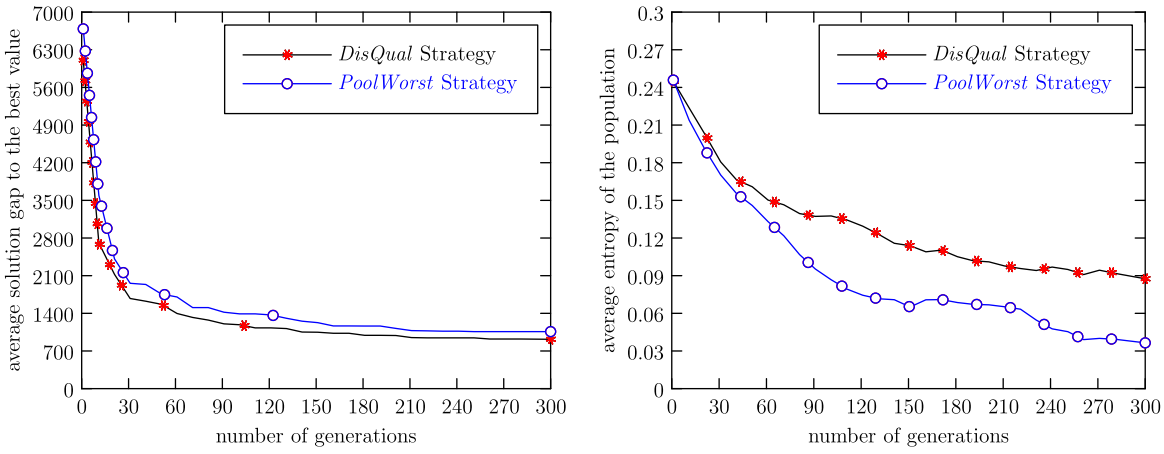

Fig. 1. Comparison between two population updating strategies.

The experiments are presented on the large random instance p5000.4 (which proves to be one of the most difficult instances for most algorithms). Similar results are observed on other random and structured instances. The stopping criterion is the number of generations (i.e., the number of applications of the combination operation) which is limited to 300.

We keep other ingredients unchanged in the HMA algorithm and observe two characteristics of the two population updating strategies: one is the best solution gap vs. the number of generations; the other is the population diversity measured in terms of *entropy* vs. the number of generations. The entropy, taking into account the value of each variable in each individual of the population, is calculated as follows [11]:

$$entropy(P) = \frac{-\sum_{i=1}^{n}\sum_{j=0}^{1}\frac{n_{ij}}{p}log\frac{n_{ij}}{p}}{nlog2}, \quad (11)$$

where n is the number of variables and n_{ij} is the number of times the variable x_i is set to j in the population P. In this definition, $entropy(P) \in [0,1]$. 0 indicates a population of identical individuals whereas 1 means that all possible assignments are almost uniformly distributed in the population.

Fig. 1 shows how the best solution gap (left) and the entropy of the population (right) evolve with the number of generations. We see that the population converges more quickly towards high-quality solutions with the distance-and-quality based population updating strategy than with the *PoolWorst* strategy. In addition, the population diversity is better preserved during the evolution process with *DisQual* than with *PoolWorst*, which is directly correlated to the evolution of the solution quality.

Therefore, following the theme of scatter search and path-relinking, an efficient population updating strategy is not necessarily a strategy that can quickly improve the whole population but rather is one that ensures a good tradeoff between quality and the diversity of the population. In other words, the diversification process introduced in our approach allows the algorithm to benefit from a better exploration of the search space and prevents the population from stagnating in poor local optima.

4.3. Tradeoffs between TS and the combination operations

We study now another important aspect of the proposed hybrid algorithm, i.e., the tradeoffs between Tabu Search and the combination operations. In fact, the performance and the behavior of HMA are influenced by the value of the *improvement cutoff* α of TS. Under a limited computational resource, the *improvement cutoff* of TS α reflects the relative proportion of combination operations and TS in the algorithm. In this section, we analyze the influence of the parameter α on the performance of the HMA algorithm. For this purpose, experiments are performed on various instances. We present below in detail the results on a single instance p5000.4, but these results are valid for other cases.

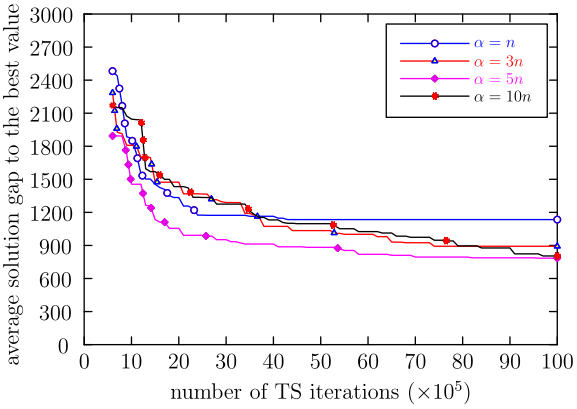

Fig. 2. Tradeoffs between TS and combination procedure.

To implement this experiment, we consider four different values of the parameter α: $\alpha = n$, $3n$, $5n$, and $10n$. For each of these values, we perform 20 independent runs, each run being given a total number of 10^7 iterations. Fig. 2 shows the average evolution of the best solution gaps during the search obtained with these different values for α.

From Fig. 2, we first notice that HMA performs much worse with $\alpha = n$ than with other values. Specifically, the solution quality improves only very slightly after the first 2.3×10^6 iterations. For $\alpha = 3n$ and $\alpha = 10n$, the algorithm can still improve the solutions after the first 4×10^6 iterations, but needs more iterations than the case with $\alpha = 5n$ to get the same quality solutions. We observe also that with $\alpha = 5n$, the average value of g_{best} decreases more quickly at the beginning than other three cases. In addition, the algorithm can reach high-quality solutions very quickly and this improvement can last for a long time. Thus, this experiment shows a clear advantage to setting an appropriate value for α in order to achieve a desired tradeoff between intensification and diversification.

5. Conclusion

In this paper, we have presented the HMA algorithm, a hybrid metaheuristic method for solving the UBQP problem. The proposed

algorithm integrates a diversification-guided combination operator for generating offspring solutions and an effective TS procedure. Based on a new definition of the distance between two solutions, HMA uses also a pool updating strategy that considers both solution quality and diversity of individuals.

Tested on three sets of 40 well-known random and structured benchmark instances, we have shown that this hybrid algorithm obtains highly competitive outcomes in comparison with the previous best known results from the literature. For the random instances with up to 7000 variables, the HMA algorithm reaches easily all the previous best known objective values within a short computation time (less than 13 minutes in average on a 2.66 GHz PC with 512 M RAM). For the nine SPP instances, we improve the best known objective values for three instances whose optimum solutions are still unknown.

Furthermore, we investigated several essential parts of our proposed algorithm, including the diversification-guided combination operator, the distance-and-quality population updating strategy and the tradeoffs between the tabu search procedure and the combination operations.

The success of the HMA algorithm on the UBQP problem reminds us that it is essential to introduce meaningful diversification mechanisms, highlight the tradeoffs between intensification and diversification and incorporate the problem specific knowledge in designing heuristic search algorithms. We anticipate that the exploitation of additional forms of path-relinking and more advanced tabu search mechanisms will provide further gains along these lines. Finally, given that the ideas behind the diversification-guided path-relinking operator and the pool updating strategy introduced in this paper are independent of the UBQP problem, there may be value in examining their application to other binary problems.

Acknowledgments

We are grateful for the referees for their comments and questions which helped us to improve the paper. The authors thank Dr. Gary Kochenberger for providing us with the SPP instances. 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

[1] B. Alidaee, G.A. Kochenberger, A. Ahmadian, 0-1 Quadratic programming approach for the optimal solution of two scheduling problems, International Journal of Systems Science 25 (1994) 401–408.
[2] B. Alidaee, G.A. Kochenberger, K. Lewis, M. Lewis, H. Wang, A new approach for modelling and solving set packing problems, European Journal of Operational Research 86 (2) (2008) 504–512.
[3] T.M. Alkhamis, M. Hasan, M.A. Ahmed, Simulated annealing for the unconstrained binary quadratic pseudo-boolean function, European Journal of Operational Research 108 (1998) 641–652.
[4] M. Amini, B. Alidaee, G.A. Kochenberger, A Scatter Search Approach to Unconstrained Quadratic Binary Programs, New Methods in Optimization, McGraw-Hill, New York, NY, 1999. 317–330.
[5] J.E. Beasley, Obtaining test problems via internet, Journal of Global Optimization 8 (1996) 429–433.
[6] J.E. Beasley, Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem, Working Paper, The Management School, Imperial College, London, England.
[7] I. Borgulya, An evolutionary algorithm for the binary quadratic problems, Advances in Soft Computing 2 (2005) 3–16.
[8] 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.
[9] E. Boros, P.L. Hammer, G. Tavares, Local search heuristics for quadratic unconstrained binary optimization (QUBO), Journal of Heuristics 13 (2007) 99–132.
[10] P. Chardaire, A. Sutter, A decomposition method for quadratic zero-one programming, Management Science 41 (4) (1994) 704–712.
[11] C. Fleurent, J.A. Ferland, Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability, in: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, 1996, pp. 619–652.
[12] G. Gallo, P. Hammer, B. Simeone, Quadratic knapsack problems, Mathematical Programming 12 (1980) 132–149.
[13] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, New York, 1979.
[14] F. Glover, M. Laguna, Tabu Search, Kluwer Academic Publishers, Boston, 1997.
[15] F. Glover, M. Laguna, R. Marti, Fundamentals of scatter search and path-relinking, Control and Cybernetics 39 (2000) 654–684.
[16] F. Glover, G.A. Kochenberger, B. Alidaee, Adaptive memory tabu search for binary quadratic programs, Management Science 44 (1998) 336–345.
[17] F. Glover, B. Alidaee, C. Rego, G. Kochenberger, One-pass heuristics for large-scale unconstrained binary quadratic problems, European Journal of Operational Research 137 (2) (2002) 272–287.
[18] F. Glover, J.K. Hao, Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems, International Journal of Metaheuristics 1 (1) (2010) 3–10.
[19] F. Glover, J.K. Hao, Fast 2-flip move evaluations for binary unconstrained quadratic optimization problems, International Journal of Metaheuristics 1 (2) (2010) 100–107.
[20] F. Glover, Z. Lü, J.K. Hao, Diversification-driven tabu search for unconstrained binary quadratic problems, 4OR: A Quarterly Journal of Operations Research (2010), doi:10.1007/s10288-009-0115-y.
[21] F. Harary, On the notion of balanced of a signed graph, Michigan Mathematical Journal 2 (1953) 143–146.
[22] C. Helmberg, F. Rendl, Solving quadratic (0,1)-problem by semidefinite programs and cutting planes, Mathematical Programming 82 (1998) 388–399.
[23] R. Horst, P.M. Pardalos, N.V. Thoai, Introduction to Global Optimization, second ed., Kluwer Academic Publishers, 2000.
[24] 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.
[25] K. Katayama, M. Tani, H. Narihisa, Solving large binary quadratic programming problems by an effective genetic local search algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO00), Morgan Kaufmann, 2000, pp. 643–650.
[26] 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.
[27] G.A. Kochenberger, F. Glover, B. Alidaee, C. Rego, An unconstrained quadratic binary programming approach to the vertex coloring problem, Annals of Operations Research 139 (2005) 229–241.
[28] J. Krarup, A. Pruzan, Computer aided layout design, Mathematical Programming Study 9 (1978) 75–94.
[29] D.J. Laughunn, Quadratic binary programming, Operations Research 14 (1970) 454–461.
[30] M. Lewis, G.A. Kochenberger, B. Alidaee, A new modelling and solution approach for the set-partitioning problem, Computers and Operations Research 35 (3) (2008) 807–813.
[31] M. Lewis, B. Alidaee, F. Glover, G.A. Kochenberger, A note on xQx as a modelling and solution framework for the Linear Ordering Problem, International Journal of Operational Research 5 (2) (2009) 152–162.
[32] 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.
[33] Z. Lü, J.K. Hao, A memetic algorithm for graph coloring, European Journal of Operational Research 203 (1) (2010) 241–250.
[34] R.D. McBride, J.S. Yormark, An implicit enumeration algorithm for quadratic integer programming, Management Science 26 (1980) 282–296.
[35] P. Merz, B. Freisleben, Genetic algorithms for binary quadratic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), Morgan Kaufmann, 1999, pp. 417–424.
[36] P. Merz, B. Freisleben, Greedy and local search heuristics for unconstrained binary quadratic programming, Journal of Heuristics 8 (2002) 197–213.
[37] P. Merz, K. Katayama, Memetic algorithms for the unconstrained binary quadratic programming problem, BioSystems 78 (2004) 99–118.
[38] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program, C3P Report 826, 1989.
[39] P. Moscato, Memetic Algorithms: A Short Introduction, New Ideas in Optimization, Mcgraw-Hill Ltd., Maidenhead, UK, 1999. pp. 219–234.
[40] G. Palubeckis, Multistart tabu search strategies for the unconstrained binary quadratic optimization problem, Annals of Operations Research 131 (2004) 259–282.
[41] G. Palubeckis, Iterated tabu search for the unconstrained binary quadratic optimization problem, Informatica 17 (2) (2006) 279–296.
[42] P. Pardalos, G.P. Rodgers, Computational aspects of a branch and bound algorithm for quadratic zero-one programming, Computing 45 (1990) 131–144.
[43] M.G.C. Resende, C. Ribeiro, F. Glover, R. Marti, Scatter search and path-relinking: Fundamentals, advances and applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Second ed., Springer, 2010.
[44] G. Syswerda, Uniform crossover in genetic algorithms, in: Proceedings of the 3rd International Conference on Genetic Algorithms, 1989, pp. 2–9.
[45] C. Witsgall, Mathematical Methods of Site Selection for Electronic System (ems), NBS Internal Report, 1975.