A Study of Memetic Search with Multi-parent Combination for UBQP

Zhipeng Lü[1], Jin-Kao Hao[1], and Fred Glover[2]

[1] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
[2] OptTek Systems, Inc., 2241 17th Street Boulder, CO 80302, USA
lu@info.univ-angers.fr, hao@info.univ-angers.fr, glover@opttek.com

Abstract. We present a multi-parent hybrid genetic–tabu algorithm (denoted by GTA) for the Unconstrained Binary Quadratic Programming (UBQP) problem, by incorporating tabu search into the framework of genetic algorithm. In this paper, we propose a new multi-parent combination operator for generating offspring solutions. A pool updating strategy based on a quality-and-distance criterion is used to manage the population. Experimental comparisons with leading methods for the UBQP problem on 25 large public instances demonstrate the efficacy of our proposed algorithm in terms of both solution quality and computational efficiency.

Keywords: UBQP, Memetic Algorithm, Tabu Search, Genetic Algorithm, multi-parent combination.

1 Introduction

The unconstrained binary quadratic programming problem may be written

$$\text{UBQP: Maximize } f(x) = x'Qx$$
$$x \text{ binary}$$

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

The formulation of UBQP is notable for its ability to represent a wide range of important problems, including those from financial analysis [1], computer aided design [2], traffic management [3], machine scheduling [4]), cellular radio channel allocation [5] and molecular conformation [6]. 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 [7]. A review of additional applications and formulations can be found in [8].

Given the interest in the UBQP and its NP-hard nature [9], a large number of solution procedures have been reported in the literature. Some representative examples include exact algorithms (such as [7,10]), local search based approaches (such as direct local search [11], Simulated Annealing [12,13,14] and Tabu Search

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 154–165, 2010.

[13,15,16,17,18]) and population-based approaches (such as Evolutionary Algorithms [19,20,21,22], Scatter Search [23] and Memetic Algorithms [24]).

In the current paper, we study a memetic algorithm for the UBQP, which integrates a tabu search procedure with a genetic search approach. The proposed algorithm is characterized by several original features. First, we introduce a "logic" combination operator using multiple parents called MSX to produce a combination scheme that more fully exploits the problem structure within the present context. Second, the proposed MSX operator is jointly employed with the conventional uniform crossover to generate diversified new solutions. Finally, our algorithm relies on a quality-and-distance replacement strategy for population updates to maintain the population diversity.

To assess the performance and the competitiveness of our memetic algorithm in terms of both solution quality and efficiency, we provide computational results on the 10 largest benchmark instances with 2 500 variables from ORLIB as well as 15 larger instances with up to 5 000 variables, comparing our outcomes with the best results of the literature.

The remaining part of the paper is organized as follows. In Section 2, our memetic algorithm is described, including the tabu search procedure, the multi-parent combination operator and the pool updating rule. Sections 3 is dedicated to the computational results and concluding remarks are given in Section 4.

2 Hybrid Genetic–Tabu Algorithm

2.1 Main Scheme and Initial Population

Memetic algorithms such as hybrid evolutionary algorithms are known to be highly effective for solving a large number of constraint satisfaction and optimization problems [25]. By combining the more global recombinant search and the more intensive local search, the memetic framework is expected to offer a better balance between the exploration and exploitation of the search space.

In principle, our genetic-tabu algorithm (GTA) 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: the solution quality and the diversity of the population.

The general framework of our GTA algorithm is described in Algorithm 1. GTA contains four main components: population initialization, a tabu search procedure, a multi-parent combination operator and population updating. Starting from an initial random population, GTA uses the tabu search procedure to optimize each individual to reach a local optimum (see Sect. 2.2, lines 4-6 in Algorithm 1). Then, a combination operator is employed to generate new offspring solutions (see Sect. 2.3, line 10 in Algorithm 1), whereupon a new round of tabu search is launched to improve the new solutions. Subsequently, the population updating rule will decide whether an improved solution should be inserted into

Algorithm 1. Pseudo-code of the GTA algorithm for the UBQP problem

```
1: Input: matrix Q
2: Output: the best solution x* found so far
3: P = {x^1, ..., x^p} ← Population_Initialization( )
4: for i = {1, ..., p} do
5:    x^i ← Tabu_Search(x^i)
6: end for
7: x* = arg max{f(x^i)|i = 1, ..., p}
8: repeat
9:    randomly choose a subset of individuals E from P
10:   x^0 ← Combination_Operator(E)
11:   x^0 ← Tabu_Search(x^0)
12:   if f(x^0) > f(x*) then
13:      x* = x^0
14:   end if
15:   {x^1, ..., x^p} ← Pool_Updating(x^0, x^1, ..., x^p)
16: until a stop criterion is met
```

the population and which existing individual should be replaced (line 15 in Algorithm 1). Throughout the search process, x^* records the best solution found (lines 7, 12-14 in Algorithm 1).

The individuals of the initial population are generated randomly (i.e., each variable x_i of the n-vector x receives a value of 0 or 1 with equal probability). To build a diversified initial 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. The *distance threshold* for executing this rule is discussed in Section 2.3.

2.2 Tabu Search Procedure

As demonstrated in [15] and more recently in [16,17,18], TS is one of the more successful approaches for the UBQP. Our tabu search procedure uses a *neighborhood* defined by the simple *one-flip move*, which is widely used in local search algorithms for binary problems such as the UBQP problem and the satisfiability problem [26]. The one-flip move consists of changing (flipping) the value of a single variable x_i to its complementary value $1 - x_i$. The implementation of this neighborhood uses a fast incremental evaluation technique [15,27] to calculate the cost (move value) of transitioning to each neighboring solution.

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:

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

In addition, once a move is performed, one needs just to update a subset of move values affected by the move. Specifically, it is possible to update the move values upon flipping a variable x_i by performing the following abbreviated calculation:

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

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 [28]. 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

$$TabuTenure(i) = tt + rand(10) \quad (2)$$

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 (by the criterion established by (2)), 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, a random best move is selected. Meanwhile, 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*.

2.3 Combination Operator

In our GTA algorithm, we jointly use two kinds of combination operators to generate suitable offspring: one is the uniform crossover widely used in the literature; the other is a "logic" multi-parent 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.

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 [29]. In our case, the application of uniform crossover is controlled by the Hamming distance d_{ij} between two parent solutions x^i and x^j (i.e., d_{ij} equals the number of variables that receive different values in the parents. We require that two solutions chosen as parents must satisfy $d_{ij} > \overline{d}$, where $\overline{d}$ denotes the average

distance between pairs of solutions in the population. Therefore, we have $\overline{d} = \frac{2}{p(p-1)} \sum_{i=1}^{p} \sum_{j=i+1}^{p} d_{ij}$, where p denotes the population size.

Our "logic" multi-parent combination operator, called MSX, relies on information extracted from diversified and elite solutions. Let $E = \{x^{(1)}, \ldots, x^{(s)}\}$, where $x^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$ and the solutions in E are ordered in terms of their quality, i.e., $x^{(1)}$ is the best solution in E and $x^{(s)}$ is the worst. The value s giving the number of solutions in E is allowed to vary randomly between 4 and 8. E itself is generated by randomly selecting elements from the pool one at a time, subject to the restriction that each new element added to E must be separated by a distance of at least $\overline{d}$ from all elements of E previously added, as a basis for assuring the diversity of E. (In some cases, this requirement may compel E to have fewer than s elements.)

Associated with each $x^{(i)}$ in E, we identify the value

$$sum(i) = \sum_{j=1}^{n} x_j^{(i)} \tag{3}$$

and define a weight $w(i)$ for the solution $x^{(i)}$ as the inverse of $sum(i)$:

$$w(i) = 1/sum(i) \tag{4}$$

The weighted quantity $w(i)x_j^{(i)}$, which equals $w(i)$ if $x_j^{(i)} = 1$ and equals 0 otherwise, may be interpreted as the "relative contribution" of setting $x_j = 1$ in the solution $x^{(i)}$. In other words, if $x_j^{(i)} = 1$ for all $j \in N$ then each assignment $x_j = 1$ contributes only $1/n$ to the weighted quantity, whereas if $x_j^{(i)} = 1$ for only two $j \in N$ then each $x_j = 1$ assignment contributes $1/2$, disclosing that the relative contribution of any given assignment $x_j = 1$ in the latter solution is significantly greater than in the former.

For the elite set E ($|E| = s$), define the value $Strength(j)$ to be the weighted sum of the values $x_j^{(i)}$ (hence of the values $x_j^{(i)} = 1$) over the solutions $x^{(i)}$ in E:

$$Strength(j) = \sum_{i=1}^{s} w(i)x_j^{(i)} \tag{5}$$

The value $Strength(j)$ gives a relative indication of the tendency of the solutions in E to favor $x_j = 1$ or $x_j = 0$. That is, we may say that the larger the value of $Strength(j)$, the greater is the degree that "E favors $x_j = 1$". We allow the use of different weights for different solutions to reflect the fact that some solutions may deserve greater influence in determining the strength assigned a given variable than other solutions. The use of different weights also permits the use of strategies that amend the emphasis placed on various solutions as a function of search history, though we have not exploited this feature in the present study.

For the goal of generating a solution x from the set of solutions E, the value $Strength(j)$ by itself is not enough to determine that x_j should be 1

or 0. To make this determination, we need to order the vector $Strength$ ($= (Strength(1), \ldots, Strength(n))$) from its largest to smallest component:

$$Strength(j_1) \geq Strength(j_2) \geq \ldots \geq Strength(j_n) \tag{6}$$

To complete the determination of a suitable vector x to be derived from the set E, we select for the number of components x_j of x that should receive a value $x_j = 1$. We take an average of the $sum(i)$ values over E to get a value for the number of x_j components that should be 1 in an "average" solution. Specifically, let

$$Avg = \sum_{i=1}^{s} sum(i)/s \tag{7}$$

Making use of these elements, we can now compute the vector x created from combining the solutions of E as follows:

1. For each $j = j_k$, $k \leq Avg - r_1$, set $x_j = 1$;
2. For each $j = j_k$, $k \geq Avg + r_2$, set $x_j = 0$;
3. For each $j = j_k$, $Avg - r_1 < k < Avg + r_2$, randomly set $x_j = 1$ or $x_j = 0$.

where r_1 or r_2 denotes a randomly generated number from 3 to 10.

This idea is inspired from the intuition that it is preferable to shift Avg slightly in one direction or another to make the generation of offspring solutions more varied. In such a way, an offspring solution x is generated based on the elite set E, to which the tabu search procedure can be applied to further optimize the solution.

2.4 Pool Updating

In our 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. For this purpose, we define a quality-and-distance goodness score of the offspring x^0 with respect to the population.

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 solution currently in the population. In order to make things clearer, we make use of the following definitions:

Definition 1. Distance Between a 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 2. 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 reasonable that the greater the goodness score $g(i, P)$, the better solution x^i, since we should not only maintain a pool of good quality solutions but also emphasize the importance of the diversity of the solutions to avoid a premature convergence of the population. Therefore, if the goodness score of the offspring solution is good enough, it will have high probability to replace the worst solution in the population. Interested readers are referred to [30] for more details about this quality-and-distance based pool updating strategy.

3 Experimental Results

3.1 Instances and Experimental Protocol

To assess the efficiency of our proposed GTA algorithm, we carry out experiments on two sets of benchmarks. The first set of benchmarks is composed of the 10 largest instances of size $n = 2\,500$ introduced in [13] and available in the ORLIB [31]. These instances are used in the literature by many authors (e.g., [13,14,16,17,18,24]). Note that the small test instances from the ORLIB whose sizes range from n=50 to 1 000 present no challenge for our GTA algorithm, since all their best known results can be obtained within 2 seconds by our algorithm. The second set of benchmarks consists of a set of 15 randomly generated large problem instances named p3000.1,. . .,p5000.5 with sizes ranging from n=3 000 to 5 000 [16,17]. These instances are available at: http://www.soften.ktu.lt/∼gintaras/ubqop_its.html.

Our algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.66GHz CPU and 512M RAM. Given the stochastic nature of our GTA procedure, each problem instance is independently solved 20 times. For the instances with 2 500, 3 000, 4 000 and 5 000 variables, the CPU time limit is set to be 40, 500, 800 and 1 500 seconds, respectively.

3.2 Computational Results and Comparisons

Based on preliminary testing, we observed that the following parameter settings give satisfying results: population size $p = 20$, tabu tenure constant $tt = n/150$, tabu search improvement cutoff $\alpha = 2n$ and goodness score constant parameter $\beta = 0.6$. 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.

Our first experiment aims to evaluate the overall performance of our GTA algorithm on the tested instances. The results of this experiment are summarized in Table 1, showing the computational statistics of our GTA algorithm. Columns 2 and 3 respectively give the density (dens) of the Q matrix and the previous best known objective values (f_{prev}). Columns 4 to 10 give our results: the best objective value (f_{best}), the best solution gap to the previous best known values g_{best} ($= f_{best} - f_{prev}$), the average solution gap to the previous best known value g_{avr} ($= f_{avr} - f_{prev}$) (where f_{avr} represents the average objective value over 20 runs), the standard deviations of the solution gaps over 20 runs (σ), the number of success runs (suc) for reaching the best known results f_{prev}, the best and the average CPU time (seconds) for reaching the best results f_{best} (t_{best} and t_{avr}) over 20 independent runs. Furthermore, for each set of benchmarks, the summary of our algorithm's average performance is indicated in the row "Average". Note that the previous best known objective values f_{prev} are extracted from [17] and [18], which are obtained by allowing a time limit of up to *several hours*. These reference algorithms are among the best performing algorithms for the tested instances.

The results shown in Table 1 disclose that our GTA algorithm can stably reach the previous best known results within a very short CPU time, demonstrating the high efficiency of our method. For the 10 medium size ORLIB instances with 2 500 variables, our algorithm can easily reach all the previous best known objective values within 4 seconds on our computer. For the 15 remaining large and difficult instances, our algorithm can also easily reach the previous best known objective values within the given time limit. The average CPU time to obtain the best known objective values is only 352 seconds and the average number of success runs is about 15 out of 20 runs for this set of benchmarks.

As indicated in Section 2, one of the original features of our approach is the multi-parent "logic" combination operator and its joint use with the uniform crossover. In order to check the effect of this strategy, we conducted our second experiment to compare GTA with a variant of GTA, where the multi-parent combination operator is disabled and the remaining components are kept unchanged. We denote this algorithm by GTA_a, where we disable our multi-parent combination operator MSX and keep only the uniform crossover.

We run this second experiment using GTA_a under exactly the same conditions as before and the results are reported in Table 2 together with those of GTA extracted from Table 1. 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}, the standard deviations of the solution gaps over 20 runs (σ), and the number of success runs (suc) for reaching the best known objective values f_{prev} over 20 runs.

One observes that GTA performs better than GTA_a in terms of all the performance criteria. In particular, for the large instance p5000.4, GTA_a failed to reach the best known objective value $f_{prev} = 12252318$ within the given time limit, while GTA reaches this best solution 3 times out of 20 runs. The average gap to the previous best known objective value is 214.3 for GTA against 301.1 for GTA_a. Moreover, GTA obtains the previous best known objective values more often than GTA_a does (15.3 versus 13.7 over 20 independent runs). For the best objective values obtained over 20 runs, we performed a 95% confidence t-test to assess the difference between these two algorithms and found that GTA is statistically superior to GTA_a in 6 instances while it is inferior to GTA_a only in 1 instance. For all the 18 remaining cases, there is no clear difference between these two algorithms.

Table 1. Overall performance of our GTA algorithm over 20 runs

instance	dens	f_{prev}	GTA						
			f_{best}	g_{best}	g_{avr}	σ	suc	t_{best}	t_{avr}
b2500.1	0.1	1515944	1515944	0	0.0	0.0	20	0.45	2.52
b2500.2	0.1	1471392	1471392	0	9.9	43.2	19	3.87	26.7
b2500.3	0.1	1414192	1414192	0	0.0	0.0	20	0.60	6.70
b2500.4	0.1	1507701	1507701	0	0.0	0.0	20	0.33	1.32
b2500.5	0.1	1491816	1491816	0	0.0	0.0	20	0.41	3.50
b2500.6	0.1	1469162	1469162	0	0.0	0.0	20	0.71	4.14
b2500.7	0.1	1479040	1479040	0	0.0	0.0	20	1.06	12.3
b2500.8	0.1	1484199	1484199	0	0.0	0.0	20	0.55	5.83
b2500.9	0.1	1482413	1482413	0	0.0	0.0	20	0.57	11.0
b2500.10	0.1	1483355	1483355	0	0.0	0.0	20	1.44	13.2
Average				0	0.99	4.32	19.9	0.999	8.72
p3000.1	0.5	3931583	3931583	0	0.0	0.0	20	8.83	42.8
p3000.2	0.8	5193073	5193073	0	0.0	0.0	20	5.39	38.6
p3000.3	0.8	5111533	5111533	0	7.7	33.6	19	13.0	82.2
p3000.4	1.0	5761822	5761822	0	0.0	0.0	20	36.1	79.8
p3000.5	1.0	5675625	5675625	0	298.2	373.9	14	16.3	85.6
p4000.1	0.5	6181830	6181830	0	0.0	0.0	20	4.92	52.0
p4000.2	0.8	7801355	7801355	0	194.1	471.5	17	186.3	276.7
p4000.3	0.8	7741685	7741685	0	0.0	0.0	20	42.9	208.4
p4000.4	1.0	8711822	8711822	0	3.0	13.1	19	43.7	168.5
p4000.5	1.0	8908979	8908979	0	260.2	455.1	15	171.3	420.6
p5000.1	0.5	8559680	8559680	0	507.4	313.4	4	137.7	636.3
p5000.2	0.8	10836019	10836019	0	425.6	250.1	11	81.6	562.8
p5000.3	0.8	10489137	10489137	0	356.4	164.3	9	264.7	726.0
p5000.4	1.0	12252318	12252318	0	1035.6	456.6	3	826.7	1326.1
p5000.5	1.0	12731803	12731803	0	126.3	401.6	18	423.9	568.3
Average				0	214.3	195.5	15.4	150.9	351.6

Table 2. Performance comparison of GTA algorithm with GTA_a

instance	f_{prev}	GTA				GTA_a			
		g_{best}	g_{avr}	σ	suc	g_{best}	g_{avr}	σ	suc
b2500.1	1515944	0	0.0	0.0	20	0	0.0	0.0	20
b2500.2	1471392	0	**9.9**	**43.2**	**19**	0	35.7	87.0	17
b2500.3	1414192	0	0.0	0.0	20	0	0.0	0.0	20
b2500.4	1507701	0	0.0	0.0	20	0	0.0	0.0	20
b2500.5	1491816	0	0.0	0.0	20	0	0.0	0.0	20
b2500.6	1469162	0	0.0	0.0	20	0	0.0	0.0	20
b2500.7	1479040	0	0.0	0.0	20	0	0.0	0.0	20
b2500.8	1484199	0	0.0	0.0	20	0	0.0	0.0	20
b2500.9	1482413	0	0.0	0.0	20	0	0.0	0.0	20
b2500.10	1483355	0	0.0	0.0	20	0	0.0	0.0	20
Average		0	**0.99**	**4.32**	**19.9**	0	3.57	8.70	19.7
p3000.1	3931583	0	0.0	0.0	20	0	0.0	0.0	20
p3000.2	5193073	0	0.0	0.0	20	0	0.0	0.0	20
p3000.3	5111533	0	7.7	33.6	19	0	**0.0**	**0.0**	**20**
p3000.4	5761822	0	0.0	0.0	20	0	0.0	0.0	20
p3000.5	5675625	0	**298.2**	373.9	**14**	0	387.3	**311.3**	6
p4000.1	6181830	0	0.0	0.0	20	0	0.0	0.0	20
p4000.2	7801355	0	**194.1**	**471.5**	**17**	0	286.5	649.9	14
p4000.3	7741685	0	0.0	0.0	20	0	0.0	0.0	20
p4000.4	8711822	0	**3.0**	**13.1**	19	0	6.0	26.2	19
p4000.5	8908979	0	**260.2**	**455.1**	**15**	0	865.2	1174.9	12
p5000.1	8559680	0	507.4	313.4	**4**	0	**415.8**	**132.1**	3
p5000.2	10836019	0	**425.6**	**250.1**	**11**	0	673.8	313.8	8
p5000.3	10489137	0	**356.4**	**164.3**	**9**	0	552.1	872.1	6
p5000.4	12252318	**0**	**1035.6**	456.6	**3**	608	1205.3	**280.6**	0
p5000.5	12731803	0	126.3	**401.6**	18	0	**124.5**	652.7	18
Average		**0**	**214.3**	**195.5**	**15.4**	40.5	301.1	294.2	13.7

Let us finally comment that we also carried out a similar experiment on another important feature of our GTA algorithm — the quality-and-distance based pool updating strategy. We compare GTA with another variant of GTA, where the pool updating strategy is disabled and is replaced by one that randomly takes the place of one of the parent individuals in the population. The remaining components are kept unchanged. Once again we observe that GTA performs better than this variant of GTA relative to all the four criteria shown in Table 2, implying the importance of our population updating strategy.

These results provide evidence of the benefit of our multi-parent combination operator and quality-and-distance based pool updating strategy.

4 Conclusions and Discussion

In this paper, we have presented the GTA algorithm, a hybrid genetic–tabu algorithm for solving the UBQP problem. The proposed algorithm integrates

a "logic" multi-parent combination operator for generating offspring solutions and an effective Tabu Search procedure. GTA uses also a pool updating strategy considering both solution quality and diversity. Tested on two sets of 25 well-known benchmark instances with 2 500 to 5 000 variables, we have shown that this hybrid algorithm obtains highly competitive outcomes in comparison with the previous best known results from the literature.

There are several directions to extend this work. One immediate possibility is to examine other dedicated combination operators by considering more detailed semantic information of the UBQP problem. Furthermore, more advanced adaptive memory strategies from tabu search afford opportunities for creating improvements of the local search part. Finally, given that the multi-parent combination introduced in this paper is independent of the UBQP problem, it is worthwhile to verify its effectiveness on other problems and to compare it with other conventional recombinant operators.

Acknowledgement

We would like to thank the referees for their helpful comments. 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 (2008-2011).

References

1. McBride, R.D., Yormark, J.S.: An implicit enumeration algorithm for quadratic integer programming. Management Science 26, 282–296 (1980)
2. Krarup, J., Pruzan, A.: Computer aided layout design. Mathematical Programming Study 9, 75–94 (1978)
3. Gallo, G., Hammer, P., Simeone, B.: Quadratic knapsack problems. Mathematical Programming 12, 132–149 (1980)
4. Alidaee, B., Kochenberger, G.A., Ahmadian, A.: 0-1 quadratic programming approach for the optimal solution of two scheduling problems. International Journal of Systems Science 25, 401–408 (1994)
5. Chardaire, P., Sutter, A.: A decomposition method for quadratic zero-one programming. Management Science 41(4), 704–712 (1994)
6. Phillips, A.T., Rosen, J.B.: A quadratic assignment formulation of the molecular conformation problem. Journal of Global Optimization 4, 229–241 (1994)
7. Pardalos, P., Rodgers, G.P.: Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45, 131–144 (1990)
8. Kochenberger, G.A., Glover, F., Alidaee, B., Rego, C.: A unified modeling and solution framework for combinatorial optimization problems. OR Spectrum 26, 237–250 (2004)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
10. Boros, E., Hammer, P.L., Sun, R., Tavares, G.: A max-flow approach to improved lower bounds for quadratic 0-1 minimization. Discrete Optimization 5(2), 501–529 (2008)

11. Boros, E., Hammer, P.L., Tavares, G.: Local search heuristics for Quadratic Unconstrained Binary Optimization (QUBO). Journal of Heuristics 13, 99–132 (2007)
12. Alkhamis, T.M., Hasan, M., Ahmed, M.A.: Simulated annealing for the unconstrained binary quadratic pseudo-boolean function. European Journal of Operational Research 108, 641–652 (1998)
13. Beasley, J.E.: Heuristic algorithms for the unconstrained binary quadratic programming problem. In: Working Paper, The Management School, Imperial College, London, England (1998)
14. Katayama, K., Narihisa, H.: Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. European Journal of Operational Research 134, 103–119 (2001)
15. Glover, F., Kochenberger, G.A., Alidaee, B.: Adaptive memory tabu search for binary quadratic programs. Management Science 44, 336–345 (1998)
16. Palubeckis, G.: Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Annals of Operations Research 131, 259–282 (2004)
17. Palubeckis, G.: Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2), 279–296 (2006)
18. Glover, F., Lü, Z., Hao, J.K.: Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR (2010); doi: 10.1007/s10288-009-0115-y
19. Merz, P., Freisleben, B.: Genetic algorithms for binary quadratic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 417–424. Morgan Kaufmann, San Francisco (1999)
20. Lodi, A., Allemand, K., Liebling, T.M.: An evolutionary heuristic for quadratic 0-1 programming. European Journal of Operational Research 119(3), 662–670 (1999)
21. Katayama, K., Tani, M., Narihisa, H.: Solving large binary quadratic programming problems by an effective genetic local search algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000), pp. 643–650. Morgan Kaufmann, San Francisco (2000)
22. Borgulya, I.: An evolutionary algorithm for the binary quadratic problems. Advances in Soft Computing 2, 3–16 (2005)
23. Amini, M., Alidaee, B., Kochenberger, G.A.: A scatter search approach to unconstrained quadratic binary programs. In: New Methods in Optimization, pp. 317–330. McGraw-Hill, New York (1999)
24. Merz, P., Katayama, K.: Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78, 99–118 (2004)
25. Moscato, P.: Memetic algorithms: a short introduction. In: New Ideas in Optimization, pp. 219–234. Mcgraw-Hill Ltd., Maidenhead (1999)
26. Hoos, H., Stützle, T.: Stochastic Local Search Foundations and Applications. Morgan Kaufmann / Elsevier (2004)
27. Glover, F., Hao, J.K.: Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems. To appear in International Journal of Metaheuristics 1(1) (2009)
28. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
29. Syswerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 2–9 (1989)
30. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. European Journal of Operational Research 203(1), 241–250 (2010)
31. Beasley, J.E.: Obtaining test problems via internet. Journal of Global Optimization 8, 429–433 (1996)