THE BULLETIN OF OKAYAMA UNIVERSITY OF SCIENCE No.37 A pp.145-154 (2001)

Simple Enhancements of Local Search Methods for the Binary Quadratic Programming Problem

Kengo KATAYAMA and Hiroyuki NARIHISA

Department of Information and Computer Engineering,
Faculty of Engineering,
Okayama University of Science,
1-1 Ridai-cho, Okayama 700-0005, Japan

(Received November 1, 2001)

1. Introduction

Iterated Local Search (ILS) is one of the representative metaheuristics that indicate an algorithmic framework based on enhancements of (simple) local search and greedy heuristics for combinatorial optimization problems. This metaheuristic algorithm is often referred to as *Parthenogenetic Algorithm* (PA), composed of a local search heuristic and a random mutation technique to escape from local optima found by the local search. For example, for the traveling salesman problem (TSP), PA is one of the most successful approximation algorithms, so-called *Iterated Lin-Kernighan* heuristic[9], which is composed of a Lin-Kernighan's local search[15] and a random four-opt escape technique.

On the other hand, it is well known that the *unconstrained binary quadratic programming problem* (BQP) is equivalent to many classical combinatorial optimization problems such as maximum cut problem, maximum clique problem, etc. See[22] for more details. Given a symmetric rational $n \times n$ matrix $Q = (q_{ij})$, the objective of the BQP is to find a binary vector x of length n that maximizes the following quantity:

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} q_{ij} x_i x_j, \qquad x_i \in \{0,1\} \ \forall i = 1, \ldots, n. \tag{1}$$

Several exact methods have been developed to solve the BQP. However, the BQP belongs to the class of $\mathcal{NP}$-hard problems as well as the related classical combinatorial optimization problems. Due to the computational complexity of the problem, at the present time it is only capable of solving the small size instances. For larger problem instances, such methods would become prohibitively expensive to apply, whereas high-performance heuristic algorithms might find high-quality solutions in a short time. To obtain near-optimal solutions in reasonable time, several metaheuristic approaches such as tabu search[4, 6], simulated annealing[4, 11], evolutionary algorithms[16, 18, 12, 21], and scatter search[1] have been proposed for the BQP so far. Although these algorithms do not deliver a guarantee to find optimum solutions, they have been shown to be highly effective in practice. In addition, the algorithms for the BQP can be utilized to solve the related combinatorial optimization problems.

In this paper, we consider various PA implementations to the BQP. Each PA performs each of four local search heuristics (deterministic *1-opt*, randomized *1-opt*, deterministic *k-opt*, and randomized *k-opt*) known for the BQP so far and has a simple mutation controlled by a probability parameter. To observe behaviors of PAs induced by several parameter values used in the mutation, we test each PA on test problem instances of up to 2500 variables from the literature and then choose the best algorithm from these PAs. Computational results after this extensive testing indicate that search abilities of PAs with *k-opt* local search are not as sensitive in the parameter values as PAs with *1-opt* local search. Furthermore,

it turns out that the PA incorporating the randomized *k-opt* local search with the near-optimal parameter value set for the mutation is superior or *at least* competitive to the other existing powerful metaheuristic approaches, such as simulated annealing and genetic local search, on the same test instances.

2. Basic Idea of Parthenogenetic Algorithm

Local Search (LS) is a generally applicable approach that can be used to find approximate solutions to hard optimization problems, and many powerful heuristics that belong to a class of metaheuristics are based on it. The basic idea of the LS is to start from a randomly generated solution x and to repeatedly replace x with a better cost chosen from a set of neighbor solutions x' that can be reached by a slight modification of a current solution. If no better neighbor solutions can be found, the LS immediately stops and finally returns to the best solution found during the search. Thus, the resulting solution cannot be improved by a slight modification. This modification is often referred to as *neighborhood* N, and the resulting solution is called the *locally optimal solution* under the neighborhood N. The quality of the locally optimal solution found by the LS heuristic substantially depends on a structure of the predefined neighborhood.

LS can be trapped in local optima and be unable to reach the global optima. To reach the global optimum or very good approximate solutions, LS process should be enlarged in some sense. One of such enlargements is a multi-start technique of LS that starts from random solutions and may promise a satisfactory solution with a larger amount of computation time. However, the use of relatively good (or previously found) solutions rather than randomly generated ones is a more natural way in applying LS itself. Better final solutions can be expected even with the same amount of computation times for each of the multi-start LS (MLS) and the following algorithm.

Among the powerful heuristic algorithms using previously found solutions with relatively good costs, *Parthenogenetic Algorithm* (PA) or *Iterated Local Search* may be one of the simplest methods that enhance the LS for various optimization problems. Generally, the PA first generates a random solution, and then the solution (or mutated solution, see below) is locally optimized by the LS heuristic, obtaining a *locally optimal* solution. The locally optimal solution (or previously found one) is slightly mutated by a mutation technique, obtaining a mutated solution that is not locally optimal with respect to neighborhood definitions of the LS heuristics. These processes, except for initial generation of a random solution in the first step, are repeated until a predefined terminal condition is satisfied. Thus, the PA can be simply composed of a *local search* and a *mutation* (kick) to escape from previously found local optima. This idea lies in focusing the search not on the full space of solutions but on a smaller subspace defined by locally optimal solutions.

To get out the local optimality of the local optimum solution, the PA requires a mutation (kick) technique that produces the other (worse) solution from the given local optimum. The kick technique may be considered as a kind of neighborhoods, and we usually require the other neighborhood structure that differs from one used in the LS of the PA. If we adopt the same (or quite similar) neighborhood structure in both the LS and the kick technique (e.g., *1-opt* neighborhood for the BQP, see §**3.3** for more details), the same local optimum would be reproduced by LS that starts from a newly mutated solution after the kick with the same neighborhood. In such case, we can not expect that PA provides very good solutions or better ones than even the best one found by MLS that performs LS starting with newly and randomly generated points.

Let us turn to optimization problems briefly. The PA should be applied to a problem characterized by the following assumption (or fact): "there are good (local optimum) solutions around the other good (local optimum) one in a search space of the given problem". Such assumption implies that in the search space many local optima found by LS are distributed in a cluster. For such optimization problem, e.g., the TSP and graph partitioning problem (GPP)[5], it is quite expected that the PA is more favorable in

PA 1

1 Generate a random solution, and run a local search on the solution, obtaining x. Copy x to x_{best}.
2 Perform a random mutation on x, obtaining x'.
3 Run the local search on x', obtaining x''.
4 If $f(x'') > f(x_{best})$, then set $x_{best} = x''$. Copy x_{best} to x.
5 If a predefined terminal condition is met, then output x_{best}, otherwise go to Step2.

Figure 1 The flow of Parthenogenetic Algorithm 1

PA 2

1 Generate a random solution, and run a local search on the solution, obtaining x. Copy x to x_{best}.
2 Perform a random mutation on x, obtaining x'.
3 Run the local search on x', obtaining x''.
4 If $f(x'') > f(x_{best})$, then set $x_{best} = x''$. Copy x'' to x.
5 If a predefined terminal condition is met, then output x_{best}, otherwise go to Step2.

Figure 2 The flow of Parthenogenetic Algorithm 2

terms of final solution qualities and running times than the MLS. The BQP in this paper also can be considered as such a problem[20, 21].

3. Parthenogenetic Algorithm for BQP

3.1 Fitness and Representation

When applying PAs to a specific problem, it is important to determine the fitness function and the representation. For the BQP, Eq. (1) can be used as the fitness function.

On the other hand, 0-1 binary representation is an obvious choice for the BQP since it represents the underlying 0-1 integer variables. In a bit string of length n, where n is the number of variables in a given BQP instance, a value of 0 or 1 at the j-th bit implies that $x_j = 0$ or 1 in the individual used in the PA, respectively.

3.2 PAs for BQP

We mainly consider two types of PAs for the BQP, but each of four local search heuristics is incorporated into each type of PAs. The two main PA's flows are shown in Fig.1 and Fig.2, and we here refer to each of them as PA1 and PA2, respectively. The difference between PA1 and PA2 shown is only a final process in Line 4 of each figure. This means that in each iteration the solutions given for a random mutation in Line 2 of each figure are different. In PA1 shown in Fig.1, the best found solution x_{best} is always used in Line 2 since x_{best} is copied to a current solution x in Line 4. In PA2, for each iteration, the currently found solution x (that is x'' in Line 4) rather than the best found one is always used in Line 2 of Fig.2. Generally, PA1 can be interpreted as the standard PA due to Johnson[9] who has presented the powerful heuristic so-called *Iterated Lin-Kernighan* for the TSP. However, the alternative flow of PA2 is also investigated due to the first PA implementations to the BQP.

As other existing PAs, there is a *Large Step Markov Chain* (LSMC) (or *Chained Local Optimization* (CLO))[17, 2]. This PA operates on a solution in Line 2 that is chosen by an idea based on a simulated annealing. If a parameter *temperature* in LSMC is set to zero, LSMC becomes the same as PA1 in our study. On the other hand, if the temperature is set to a fixed large value so as to always accept the solution found by local search in each iteration, it is equivalent to PA2. Alternatively, a novel heuristic called *Genetic Iterated Local Search* (GILS) can be found in[10]. GILS operates on two solutions in Line 2: the best found solution corresponds to the one in Line 2 of Fig.1 and the current solution corresponds to the one in Line 2 of Fig.2. These PA variants will be future issues. In the following, local search heuristics and random mutation are described to achieve our simple PAs for the BQP.

3.3 Local Search Algorithms

To incorporate a local search into the PAs provided above, we show four local search heuristics for the BQP. They have been clearly stated in the recent literatures and each of their performances in terms of solution qualities and running speeds has been practically confirmed under an implementation of the multi-start technique with each local search. Thus, we describe here each feature of these local search mechanisms.

Four local search heuristics for the BQP may be categorized by two neighborhood classes: *1-opt* neighborhood and *k-opt* neighborhood. In the BQP, since a solution x to the problem is a binary vector of fixed length n, the simplest form is the *1-opt* neighborhood, i.e., *1-opt* neighbor solution is reached by flipping a single bit in a current solution. Therefore, the Hamming distance [1] d_H between the current solution x and the *1-opt* neighbor solution x' is one, $d_H(x, x') = 1$. The local search that has the *1-opt* neighborhood for the BQP is called the *1-opt* local search. It searches new solutions with better cost in the *1-opt* neighbor solutions that can be reached by flipping a single bit of the current solution in each step and the search is performed until no improved neighbor solution is found.

On the other hand, the *k-opt* neighborhood for the BQP can be extended from the *1-opt* one. The *k-opt* local search heuristic for the BQP was first presented by Merz and Freisleben[19] that was based on well-known ideas used in Lin-Kernighan algorithm for the TSP[15] and Kernighan-Lin algorithm for the GPP[14]. The basic idea of the heuristic is to find a solution by flipping a variable number of k bits in the solution vector per iteration. In each step, a sequence of n solutions is generated by flipping the bit with the highest associated gain. The best solution in the sequence is accepted as the new solution for the next iteration. In the *k-opt* local search, the best solution found in each iteration can be interpreted as the *k-opt* neighbor solution. The search is performed until no better new *k-opt* solution is found. Therefore, the Hamming distance d_H between a current solution x and a resulting *k-opt* neighbor solution x' depends on a variable number k.

Both neighborhoods are further divided into deterministic and randomized versions that are characterized by two move strategies: best improvement and first improvement. In the BQP, the deterministic *1-opt*[18, 19, 20] and *k-opt*[19, 20] local search heuristics have been presented by Merz *et al.* In both deterministic heuristics, new solutions are found by always flipping the bits with the highest gain value by the best improvement move strategy. In the *k-opt*, roughly speaking, the *1-opt* with the best improvement is performed with a tabu fashion to achieve a variable *k-opt* neighbor solution. The others, the randomized *1-opt*[11] and *k-opt*[12, 13] local search heuristics have been presented by ourselves. The randomized *1-opt* local search randomly finds new solutions with good (positive) gain value by the first improvement in a fixed number of iterations ($\leq n$). The randomized *k-opt* local search finds a new better *k-opt* neighbor solution reached by a combination of the randomized *1-opt* and deterministic *k-opt* neighborhoods in each iteration. In this combination, the first improvement *1-opt* is executed before the highest gain found by the best improvement *1-opt* with the tabu fashion, which is a part of the deterministic *k-opt*, is negative. The search conducted by the randomized *1-opt* neighborhood in this *k-opt* local search contributes to oscillations that are performed so as to randomly move to gainful solutions with positive gain values before temporarily moving to tentative solutions with negative gains. More details can be found in[11, 12, 13, 18, 19, 20].

When performing two multi-start methods of *1-opt* and *k-opt* local searches that start from random solutions, each result obtained by each of the randomized versions (randomized *1-opt* and *k-opt*) is superior to each of the deterministic versions (deterministic *1-opt* and *k-opt*), respectively, in terms of resulting solution qualities on average for the test instances ranging from medium size (500 variables) to large size (2500 variables)[11, 13]. It is also confirmed that in a comparison of both *1-opt* local searches (or both *k-opt* local searches), each of both running times consumed by two *1-opt* (or two *k-opt*) is almost

[1] The Hamming distance between the binary vectors $u = \{u_1, u_2, \ldots, u_n\}$ and $v = \{v_1, v_2, \ldots, v_n\}$ is the indices i such that $1 \leq i \leq n$ and $u_i \neq v_i$, where n is a length of the vectors. We denote the Hamming distance by $d_H(u, v)$.

the same when starting from random solutions.

3.4 Random Mutation

The mutation in the PA is interpreted as one neighborhood to generate (or transform) a new solution that is not a local optimum (with a worse cost) from locally optimal solution found by local search. In other words, it is a technique to *kick* a local search solution, that is, to perturb it slightly.

In the TSP, for example, one of the mutation techniques used very often in the PA is a random 4-opt move (or non-sequence 4-change, double-bridge)[15, 9, 17, 2]. The mutation for the TSP is very useful in that it is possible to generate a suitable initial solution that can be given even for the powerful *Lin-Kernighan*'s local search and it can be expected that the powerful heuristic starting from such solution produces the other local optimum which is different with the solution before the mutation performed previously. Hong *et al.* [8] have investigated suitable edge numbers in the kick that are exchanged to generate a suitable initial solution for well-known TSP local searches. They have reported that the suitable edge numbers depended on types of local search heuristics and TSP instances after the investigation that, for each of given instances, the edges ranging from 2 to 50 were randomly exchanged for the local search solutions.

Such investigations are not yet conducted for the BQP. Therefore, ones as in[8] may be required for the first study of the PA to the BQP, since the useful mutation in the TSP is impossible to apply to the problem considered here.

To achieve such investigations for the PAs to the BQP, we perform a random mutation that flips bits randomly chosen in a given locally optimal solution and generates a new solution which is an initial solution for the next local search process in the PA. A number r of the bits to be flipped is defined by $r = n * prmt$, where n denotes a number of variables of a given instance size and $prmt$ ($0 < prmt < 1$) is a probability parameter for determining the number of the bits to be randomly flipped. For each of the BQP instances investigated in the following experiment, we test several parameter values to show which is the most suitable in the PAs. We set $prmt \in \{$ 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95 $\}$. The 20 parameter values in the 0.01 to 0.95 range mean to flip the bits ranging from 1% to 95% for the given solution of length n. Given an instance of 1000 variables and if $prmt$ is set to 0.1, for example, 100 bits ($= 1000 * 0.1$) are randomly chosen from the solution of length 1000, and these 100 bits chosen are all flipped to generate the new solution. Thus, $d_H(x_{new}, x_{old}) = 100$, where x_{new} is the new solution and x_{old} is the given solution.

4. Computational Results

Here, two experiments are mainly conducted. The aim of the first experiment is to compare several PAs and to observe behaviors of the random mutation controlled by probability parameter values. The second experiment is to show the search performance of the PA, which is the most promising algorithm chosen from the first one, in order to compare with the other existing metaheuristics for the BQP.

For the experiments, it may be convenient to name each of several PAs. There are 8 candidates of the PAs and the entries are listed as follows.

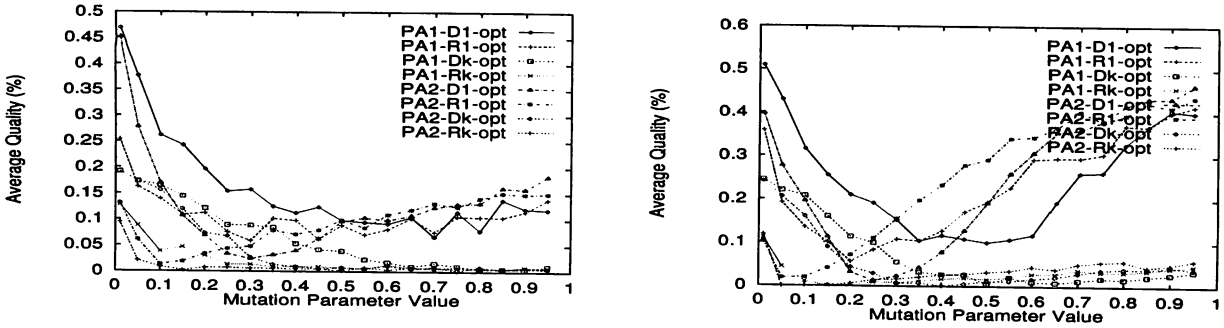
- PA1-D1-opt — PA1 with the deterministic *1-opt* local search (see Fig.1 for details of the PA1)
- PA1-R1-opt — PA1 with the randomized *1-opt*
- PA1-Dk-opt — PA1 with the deterministic *k-opt*
- PA1-Rk-opt — PA1 with the randomized *k-opt*
- PA2-D1-opt — PA2 with the deterministic *1-opt* local search (see Fig.2 for details of the PA2)
- PA2-R1-opt — PA2 with the randomized *1-opt*
- PA2-Dk-opt — PA2 with the deterministic *k-opt*
- PA2-Rk-opt — PA2 with the randomized *k-opt*

4.1 Results of the First Experiment

In the first experiment, we consider four of the BQP's test instances with varied densities and problem

Table 1 Information of test problem instances

instance	bkv	n	$dens(Q)$	Ref.
glov500-4	172771	500	0.75	6)3)
kb-g09	262658	1000	0.9	7)1)
beas1000-8	351994	1000	0.1	4)3)
beas2500-2	1471392	2500	0.1	4)3)



(a) glov500-4 (b) beas2500-2

Figure 3 Results of eight PAs for average quality versus mutation parameter values

size ranging from 500 to 2500 variables from the literature. The four test instances chosen as one from each of four problem sets are glov500-4, kb-g09, beas1000-8 and beas2500-2, which are known to be relatively hard to solve. In Table 1, these instances are listed with the best-known value (bkv), the problem size n, where n denotes a number of variables, matrix Q density ($dens(Q)$), where $dens(Q)$ is defined as the number of non-zero entries divided by the number of total entries in the matrix, and references that previously used each instance or a library we can take the instance. Other information of each problem set is given in the next subsection.

We imposed a time limit of PA running according to each of the problem sizes: 10 seconds for 500 variable instance (i.e., glov500-4), 30 seconds for 1000 variable instances (i.e., kb-g09 and beas1000-8) and 60 seconds for 2500 variable instance (i.e., beas2500-2), on a Sun Ultra 5/10 (UltraSPARC-IIi 440MHz). Each of eight PAs was run 30 times with each of *prmt* values for each instance. An initial solution for each run of the algorithm was randomly generated with a different random seed. All the PA heuristics were implemented in *C*.

Figure 3 reports overall comparison results of eight PAs on the instances: (a) glov500-4 and (b) beas2500-2 (The results of kb-g09 and beas1000-8 instances were omitted since we observed almost the same behaviors with (a) and (b) in Fig. 3). For each of (a) and (b), the vertical line shows average quality (%) that is the average percentage excess over the best-known solution value and the horizontal line denotes *prmt* value. Here, we observe the behaviors of how the final solutions obtained by the PAs change when the *prmt* value is changed. In all the results in this experiment, interesting behaviors are observed in eight PAs: average qualities of solutions finally obtained by PAs with *k-opt* are not sensitive even if the mutation parameter values are increased, but in cases of PAs with *1-opt*, the qualities are gradually inferior. From the behavior of the PAs with *1-opt*, we note that it is difficult to confidently choose the best value of the parameters for obtaining final good solutions. In comparison of two types of PAs with each *k-opt*, PA2 type algorithms seem to obtain better solutions than another type PA1 in most cases. In addition, these better solutions by PA2 are found from relatively smaller values of the parameter without depending on problem size, although it seems that the most promising *prmt* value depends on a type of the local search heuristics and others (problem densities). This tendency in each of PA2 type algorithms with *k-opt* may be good in that we set confidently the parameter values.

From the results shown in the figures, we choose the best algorithm among the eight PAs in order to show performances of the best algorithm in the next experiment. The criteria to choose it are as follows. 1) an algorithm among the PAs obtains the highest-quality solutions and 2) the mutation parameter value *prmt* should be small. The first criterion is quite natural in that the high-quality solutions should be always required. For the second one, since a larger value of *prmt* causes expensive computational tasks of local search to reach local optimum in each iteration of the PA, it may be necessary to use a small value rather than a larger one, if good solutions are obtained with the small value. From the criteria provided above, we recommend PA2-Rk-opt algorithm because the algorithm has obtained the best result among them and the best result was always obtained when *prmt* is a relatively small value for each instance tested. In addition, a range of the small values is almost the same without depending on problem sizes, i.e., it shows to be robust, but it seems that the near-optimal value of *prmt* depends on the problem densities.

Our recommended *prmt* values for PA2-Rk-opt are as follows: without depending on the problem size n, for lower density problems we simply recommend *prmt* = 0.05 up to 0.15, and for higher density problems *prmt* = 0.15 up to 0.25. From these, we also recommend for the middle density problem *prmt* = 0.1 up to 0.2. These are near-optimal values for PA2-Rk-opt, not optimal. Note that these values are substantially acquired to only the instances considered here. Therefore, it is difficult to justify the values to all problem instances of the BQP. However, we consider that they may be used as one standard near-optimal value for other instances with corresponding densities.

4.2 Results of the Second Experiment

To show empirical performances of the best PA for many test problem instances from the literature, we test PA2-Rk-opt with near-optimal value of mutation chosen from the results of the previous experiment. A number of the instances considered here are 35 in each problem set considered in the previous experiment: 5 instances from `glov500` set and 30 instances from each of `kb-g`, `beas1000` and `beas2500`. Five instances in the first set `glov500` are with $n = 500$ and a density $dens(Q)$ of the instance in the set are between 0.1 and 1.0. Ten instances of `kb-g` are with $n = 1000$ and $dens(Q)$ varied between 0.1 and 1.0. Last 20 instances: `beas1000` and `beas2500` are with 1000 and 2500 variables, and each of the sets consists of 10 instances. Each density of the 20 instances is 0.1. According to such density information and our recommended value shown above, *prmt* values used in PA2-Rk-opt are set as follows: *prmt* = 0.05 for `glov500-1`, `kb-g01`—`kb-g02`, and all `beas1000` and `beas2500` instances, *prmt* = 0.1 for `glov500-2`, `glov500-3` and `kb-g03`—`kb-g07` instances, and *prmt* = 0.15 for `glov500-4`, `glov500-5` and `kb-g08`—`kb-g10` instances.

For the PA running in this experiment, a time limit is also imposed on the same computer used above, but a larger time limit was set because the relatively large running times have been permitted in the previously reported results of the powerful metaheuristics. Our setting of the time limit depends on the problem size: 30(s) for the instances with $n = 500$, 60(s) for the instances with $n = 1000$ and 360(s) for the instances with $n = 2500$. These setting are the same as that in[12)].

Table 2 summarizes the results of PA2-Rk-opt for the 35 instances. 30 runs were performed and the running times to reach the best-known solutions were recorded. If the best-known solution could not be found for each run, the run was performed until the time limit described above was reached. In this table, the best found solution "best", the average final solution "avg.", the number of times in which the best-known solution could be found "b/30", and the average running time "t1" in seconds in the case which the PA could find the best-known solution, were provided.

From Table 2, it is observed that the best PA, PA2-Rk-opt, was capable of finding the best-known solution with high frequency in the predefined time limit (see the column denoted "b/30"), except for `kb-g` instances with high densities. For particularly `kb-g09` and `kb-g10`, it appears to be relatively hard to find the best-known solution by the PA, since the numbers of "b/30" are less than 10. On the other

Table 2 Parthenogenetic Algorithm incorporating randomized *k-opt* local search (PA2-Rk-opt) for glov500, kb-g, beas1000 and beas2500 instances

BQP instance	PA2-Rk-opt best	avg. (%)	b/30	t1(s)
glov500-1	61194	61194.0 (0.00000)	30	0.1
glov500-2	100161	100161.0 (0.00000)	30	0.2
glov500-3	138035	138035.0 (0.00000)	30	0.6
glov500-4	172771	172771.0 (0.00000)	30	6.4
glov500-5	190507	190507.0 (0.00000)	30	5.6
kb-g01	131456	131456.0 (0.00000)	30	8.7
kb-g02	172788	172788.0 (0.00000)	30	8.7
kb-g03	192565	192565.0 (0.00000)	30	1.0
kb-g04	215679	215584.5 (0.04383)	25	23.8
kb-g05	242367	242367.0 (0.00000)	30	2.6
kb-g06	243293	243291.0 (0.00081)	29	19.1
kb-g07	253590	253432.6 (0.06209)	18	16.8
kb-g08	264268	264220.3 (0.01805)	19	19.0
kb-g09	262658	262488.1 (0.06473)	8	7.7
kb-g10	274375	274256.5 (0.04321)	5	3.7
beas1000-1	371438	371438.0 (0.00000)	30	2.1
beas1000-2	354932	354932.0 (0.00000)	30	1.4
beas1000-3	371236	371236.0 (0.00000)	30	1.6
beas1000-4	370675	370675.0 (0.00000)	30	0.6
beas1000-5	352760	352760.0 (0.00000)	30	5.4
beas1000-6	359629	359629.0 (0.00000)	30	2.6
beas1000-7	371193	371193.0 (0.00000)	30	4.3
beas1000-8	351994	351990.3 (0.00106)	29	8.2
beas1000-9	349337	349254.5 (0.02362)	25	8.7
beas1000-10	351415	351415.0 (0.00000)	30	3.4
beas2500-1	1515944	1515944.0 (0.00000)	30	13.9
beas2500-2	1471392	1471392.0 (0.00000)	30	89.5
beas2500-3	1414192	1414156.4 (0.00252)	26	88.5
beas2500-4	1507701	1507701.0 (0.00000)	30	7.0
beas2500-5	1491816	1491816.0 (0.00000)	30	14.9
beas2500-6	1469162	1469162.0 (0.00000)	30	30.1
beas2500-7	1479040	1479039.9 (0.00001)	29	93.1
beas2500-8	1484199	1484199.0 (0.00000)	30	35.0
beas2500-9	1482413	1482412.2 (0.00005)	28	109.1
beas2500-10	1483355	1483355.0 (0.00000)	30	85.9

Table 3 Previous results of alternative heuristics: genetic local search (GLS-KTN) by Katayama, Tani, and Narihisa, simulated annealing (SA-KN) by Katayama and Narihisa, genetic local search (GLS-MF) by Merz and Freisleben, tabu search (TS-B) and simulated annealing (SA-B) by Beasley for beas2500 instances

BQP	GLS-KTN[12]			SA-KN[11]		GLS-MF[18]		TS-B[4]	SA-B[4]
instance	avg.(b/30)	t1(s)	t2(s)	avg.(b/30)	t(s)	avg.	t(s)	best	best
beas2500-1	1515944.0 (30)	32	—	1515828.9 (9)	15.1	1514804.6	1200	1514971	1515011
beas2500-2	1471195.1 (13)	215	360	1470600.9 (1)	15.2	1469721.0	1200	1468694	1468850
beas2500-3	1414111.9 (21)	117	360	1413657.1 (8)	15.1	1412943.0	1200	1410721	1413083
beas2500-4	1507701.0 (30)	22	—	1507630.3 (21)	14.6	1507674.2	1200	1506242	1506943
beas2500-5	1491816.0 (30)	51	—	1491692.8 (6)	15.2	1491623.4	1200	1491796	1491465
beas2500-6	1469162.0 (30)	52	—	1468810.3 (5)	15.3	1467918.2	1200	1467700	1468427
beas2500-7	1479038.8 (29)	117	360	1478397.4 (2)	15.4	1477101.7	1200	1476059	1478654
beas2500-8	1484197.1 (25)	86	360	1483907.9 (6)	15.0	1483226.9	1200	1484199	1482953
beas2500-9	1482411.3 (27)	176	360	1482192.0 (1)	15.1	1481622.9	1200	1482306	1481834
beas2500-10	1483172.8 (16)	178	360	1482522.4 (1)	15.4	1481899.2	1200	1482354	1482166

hand, for the instances with lower densities such as beas1000 and beas2500 instances with $dens(Q) = 0.1$, it seems that the PA with $prmt = 0.05$ provided satisfactory solutions in reasonable running times.

To compare our best PA with the other metaheuristics, we provide several results of the alternative heuristics. Table 3 displays the results of the metaheuristics previously reported by others and us. The heuristics are two genetic local search (GLS), two simulated annealing (SA), and a tabu search (TS), and have been tested on large test instances of up to 2500 variables contained in ORLIB[3], except the set of kb-g.

The previous results for only 2500 variable instances are shown in Table 3. GLS-KTN[12] incorporates

the randomized *k-opt*. For GLS-KTN, 30 runs were performed and the running times to reach the best-known solutions were recorded. If the best-known solution could not be found for each run, the run was performed until the time limit of 360(s) was reached. In this table, for GLS-KTN, the average final solution "avg.", the number of times in which the best-known solution could be found "(b/30)", the average running time "t1" in seconds in the case which the GLS could find the best-known solution, and the time limit "t2" in seconds except for the case which the GLS could find the best-known solution, were provided. The GLS-KTN was performed on the same computer used above, Sun Ultra 5/10. For SA-KN[11], the average final solution in 30 runs "avg.", "(b/30)" and the average running times in seconds which were required by SA-KN on the Sun Ultra 5/10, were provided. In other results by other researchers, for GLS-MF[18] incorporating the deterministic *1-opt* local search, the average final solution in 30 runs was provided for each instance. Their GLS was performed until 1,200(s) for each of beas2500 was reached on Pentium II PC (300MHz). For TS-B and SA-B, in the study[4], Beasley provided the best result for each instance. These algorithms (TS-B and SA-B) for each instance of the set beas2500 has consumed about 14 hours and 17 hours on Silicon Graphics (R4000 CPU with 100MHz), respectively.

When comparing both the results of PA2-Rk-opt for the beas2500 set in Table 2 and GLS-KTN for the same set that may be one of the best heuristics for solving the BQP, it is clear that a number of "b/30" by the PA for each instance is better or at least competitive to GLS-KTN or others. In addition, the running times to find the best-known solution is relatively fast in comparison to SA-KN, which is one of the fastest heuristics for finding the best-known solution, particularly for beas2500-1, beas2500-4, etc. Note again that both results of PA2-Rk-opt and SA-KN (and GLS-KTN) have been obtained on the same computational circumstance.

5. Conclusion

This paper have attempted to design several Parthenogenetic Algorithms, which are simple enhancements of local search methods, for the unconstrained binary quadratic programming problems (BQP). We demonstrated the search ability of the most promising PA, after an extensive testing of the mutation parameter values. Finally, we showed that the best PA —the PA2 framework incorporating the randomized *k-opt* local search (PA2-Rk-opt)— is one of the most powerful metaheuristics for the large BQP benchmark instances.

References

1) M.M. Amini and B. Alidaee, and G.A. Kochenberger, "A scatter search approach to unconstrained quadratic binary programs," New Ideas in Optimization (eds, D. Corne, M. Dorigo and F. Glover), McGraw-Hill, pp.317–329, 1999.

2) D. Applegate, W. Cook, and A. Rohe, "Chained Lin-Kernighan for large traveling salesman problems," taken from http://www.caam.rice.edu/~keck/reports/chained_lk.ps , Jul. 2000.

3) J.E. Beasley, "OR-Library: distributing test problems by electronic mail," Journal of the Operational Research Society, vol.41, no.11, pp.1069–1072, 1990.

4) J.E. Beasley, "Heuristic algorithms for the unconstrained binary quadratic programming problem," Technical Report, Management School, Imperial College, UK, 1998.

5) K.D. Boese, A.B. Kahng, and S. Muddu, "A new adaptive multi-start technique for combinatorial global optimizations," Operations Research Letters, vol.16, pp.101–113, 1994.

6) F. Glover, G.A. Kochenberger, and B. Alidaee, "Adaptive memory tabu search for binary quadratic programs," Management Science, vol.44, no.3, pp.336–345, 1998.

7) F. Glover, G. Kochenberger, B. Alidaee, and M. Amini, "Tabu search with critical event memory: An enhanced application for binary quadratic programs," Meta-Heuristics, Advances and Trends in

Local Search Paradigms for Optimization (eds, S. Voss, S. Martello, I.H. Osman, and C. Roucairol), Kluwer Academic Pub., pp.93–109, 1999.

8) I. Hong, A.B. Kahng, and B.-R. Moon, "Improved large-step Markov chain variants for the symmetric TSP," Journal of Heuristics, vol.3, no.1, pp.63–81, 1997.

9) D.S. Johnson, "Local optimization and the traveling salesman problem," Proc. 17th. Colloquium on Automata, Lang., and Prog., pp.446–461, 1990.

10) K. Katayama and H. Narihisa, "Iterated local search approach using genetic transformation to the traveling salesman problem," Proc. of the Genetic and Evolutionary Computation Conference, vol.1, pp.321–328, 1999.

11) K. Katayama and H. Narihisa, "Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem," European Journal of Operational Research, vol.134, no.1, pp.103–119, 2001.

12) K. Katayama, M. Tani, and H. Narihisa, "Solving large binary quadratic programming problems by effective genetic local search algorithm," Proc. of the 2000 Genetic and Evolutionary Computation Conference, pp.643–650, 2000.

13) K. Katayama and H. Narihisa, "A variant *k-opt* local search heuristic for binary quadratic programming," Trans. IEICE (A), vol.J84-A, no.3, pp.430–435, 2001. (*in Japanese*)

14) B.W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell System Technical Journal, vol.49, pp.291–307, 1970.

15) S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," Operations Research, vol.21, pp.498–516, 1973.

16) A. Lodi, K. Allemand, and T.M. Liebling, "An evolutionary heuristic for quadratic 0-1 programming," European Journal of Operational Research, vol.119, pp.662–670, 1999.

17) O. Martin, S.W. Otto, and E.W. Felten, "Large-step Markov chains for the TSP incorporating local search heuristics," Operations Research Letters, vol.11, pp.219–224, 1992.

18) P. Merz and B. Freisleben, "Genetic algorithms for binary quadratic programming," Proc. of the 1999 Genetic and Evolutionary Computation Conference, vol.1, pp.417–424, 1999.

19) P. Merz and B. Freisleben, "Greedy and local search heuristics for unconstrained binary quadratic programming," Technical Report No.99-01, Informtik-Berichte, 1999. (to appear in *Journal of Heuristics*)

20) P. Merz, "Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies," Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany, 2000.

21) P. Merz and K. Katayama, "Memetic algorithms for the unconstrained binary quadratic programming problem," (provisional title) in preparation, 2001.

22) P.M. Pardalos and J. Xue, "The maximum clique problem," Journal of Global Optimization, vol.4, pp.301–328, 1994.