An Evolutionary Approach for the Maximum Diversity Problem

Kengo Katayama[1] and Hiroyuki Narihisa[2]

[1] Okayama University of Science, 1 - 1 Ridai-cho, Okayama, 700-0005 Japan
katayama@ice.ous.ac.jp
[2] Okayama University of Science, 1 - 1 Ridai-cho, Okayama, 700-0005 Japan
narihisa@ice.ous.ac.jp

Summary. The objective of the maximum diversity problem (MDP) is to select a set of m-elements from larger set of n-elements such that the selected elements maximize a given diversity measure. The paper presents an evolutionary algorithm incorporating local search — memetic algorithm (MA) — for the MDP which consists of a greedy method, simple evolutionary operators, a repair method, and a k-flip local search based on *variable depth search*. In the MA, the k-flip local search starts with a feasible solution and obtains a local optimum in the feasible search space. Since infeasible solutions may be created by the simple crossover and mutation operators even if they start with feasible ones found by the local search, the repair method is applied to such infeasible solutions after the crossover and the mutation in order to guarantee feasibility of solutions to the problem. To show the effectiveness of the MA with the k-flip local search, we compare with a MA with 2-flip local search for large-scale problem instances (of up to $n = 2500$) which are larger than those investigated by other researchers. The results show that the k-flip local search based MA is effective particularly for larger instances. We report the best solution found by the MA as this is the first time such large instances are tackled.

1 Introduction

We consider the following maximum diversity problem (MDP). Given a symmetric $n \times n$ matrix d_{ij} ($d_{ij} = d_{ji}$ and $d_{ii} = 0$) and a predetermined number of size m ($n > m > 1$), the objective of the MDP is to select a subset of m-elements from n-elements such that the selected elements maximize a *diversity* measure. The MDP is represented as the following quadratic zero-one integer program:

$$\begin{aligned} \text{maximize } & f(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_i x_j, \\ \text{subject to } & \sum_{i=1}^{n} x_i = m, \\ & x_i \in \{0,1\},\ \forall i = 1,\ldots,n. \end{aligned} \tag{1}$$

The first model of the MDP has been formulated by Kuo, Glover, and Dhir [15] in which the concept of diversity is quantifiable and measurable. The concept of diversity is described as follows: consider a set of elements $S = \{s_i : i \in N\}$ with the index set $N = \{1, 2, \ldots, n\}$ and their common r attributes that each element possesses, denoted by s_{ik}, $k \in R = \{1, 2, \ldots, r\}$. To measure the diversity of a selected set of elements, a specified distance d_{ij} between each pair of elements s_i and s_j is required. One of the most commonly used distances may be the Euclidean distance, $d_{ij} = [\sum_{k=1}^{r}(s_{ik} - s_{jk})^2]^{1/2}$. In the MDP, it is assumed that the matrix d can be given by such a distance.

Kuo et al. proved the problem to be NP-hard, both with and without restricting the d_{ij} coefficients to non-negative values. Moreover, they transformed the maximum diversity model into two equivalent linear integer programming models and maximin diversity model in order to solve the problem by integer programming approaches. The MDP shown above is a general diversity maximization model that arises in data mining [14] and is substantially equivalent to the model of Kuo et al.

The MDP has a large number of applications. For example, such applications are immigration and admissions policies, committee formation, curriculum design, market planning and portfolio selection [5, 15]. Moreover, there are VLSI design and exam timetabling problems [23]. Others are environmental balance, medical treatment, genetic engineering, molecular structure design, agricultural breeding stocks, right sizing the firm, and composing jury panels [14].

The form of the MDP is quite similar to that of the unconstrained binary quadratic programming problem (BQP) in that they are both problems of maximizing a quadratic objective by suitable choice of binary (zero-one) variables. The BQP can be expressed as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_i x_j,\ x_i \in \{0,1\},\ \forall i = 1,\ldots,n. \tag{2}$$

Thus, the MDP can be interpreted as a constrained version of the BQP.

Applications of the BQP are known to be abundant as well as the MDP. They appear in machine scheduling, traffic message management, CAD, capital budgeting and financial analysis, and molecular conformation [3]. Furthermore, it has been known that several classical combinatorial optimization problems can be formulated as a BQP, such as maximum cut problem, maximum clique problem, maximum vertex packing problem, minimum vertex cover problem, and maximum (weight) independent set problem [21, 22].

Since the problems MDP and BQP are NP-hard, exact methods would become prohibitively expensive to apply for large scale problem instances, whereas the heuristic or metaheuristic approaches may find high quality solutions of near-optimum with reasonable times. For the BQP, several heuristic and metaheuristic approaches have been developed; for example, greedy methods [7, 19], local searches [11, 19], and the metaheuristics, tabu search [3, 6], simulated annealing [3, 9], iterated local search [12], and evolutionary methods such as scatter search [1] and genetic algorithms incorporating local search [10, 17, 18, 20].

On the other hand, studies on such approaches for the MDP seem much more limited. Ghosh [8] showed a greedy randomized adaptive search procedure (GRASP) for the MDP. The GRASP metaheuristic was tested on small problem instances with $n \leq 40$. Glover et al. [5] proposed two constructive heuristics and two destructive heuristics for the MDP. They tested them for several instances of up to $n = 30$. Kochenberger et al. [14] dealt with large instances of the general MDP from $n = 100$ to $n = 1000$, which are randomly generated, and tested a tabu search metaheuristic taken into account searching *infeasible* space. Their tabu search to the MDP is based on the algorithm that has been developed for the BQP. The tabu search includes the strategic oscillation with constructive and destructive heuristics. The details of the tabu search can be found in [6].

This paper presents an evolutionary approach to the MDP. To the best of our knowledge, such an evolutionary approach is the first attempt to the MDP. Our approach consists of a greedy method to create initial solutions, simple evolutionary operators such as uniform crossover and bit-flip mutation, a repair method to turn an infeasible solution created by crossover or mutation into a feasible one, and a sophisticated k-flip local search based on variable depth search [13, 16], to be an effective memetic algorithm (MA) for the MDP. To show the effectiveness of the MA, computational experiments are conducted on large problem instances of up to $n = 2,500$ compared to the previous studies for the MDP. The results demonstrate that the k-flip local search based MA is more effective than a MA based on 2-flip local search in terms of final solutions, particularly for large-scale problem instances.

The paper is organized as follows. In the next section, we show the k-flip local search incorporated in the memetic algorithm for the MDP. In section 3, a flow of the memetic algorithm is given, and each operation in the algorithm is described. In section 4, we report experimental results for the memetic algorithms tested on our new problem instances and on several benchmarks derived from well-known BQP's ones in which the d_{ij} coefficients are not restricted to non-negative values. The final section contains concluding remarks.

2 Local Search for MDP

Local Search (LS) is a generally applicable approach that can be used to find approximate solutions to hard optimization problems. Many powerful heuristics are so-called *metaheuristics* such as memetic algorithm are based on LS.

The basic idea of LS is to start with a feasible solution x (e.g., randomly generated solution) and to repeatedly replace x with a better solution x' selected from the set of neighboring solutions that can be reached by a slight modification of the current solution. If no better solutions can be found in the set of neighbors, LS immediately stops and finally returns the best solution found during the search. Thus, a resulting solution cannot be improved by the slight modification. This modification is achieved by a predefined structure often referred to as *neighborhood* $\mathcal{NB}$. The resulting solution is called *locally optimal* with respect to the neighborhood. LS is an integral process in the memetic framework. The remainder of this section describes a LS, k-flip local search, for the MDP. We begin by describing the fitness (objective) function and the solution representation on which the local search and the evolutionary operators in the memetic algorithm are based.

2.1 Fitness Function and Solution Representation

In our memetic algorithm incorporating the local search for the MDP, the fitness, i.e., a solution cost, is evaluated by equation (1).

A solution to the MDP can be represented in a binary string x of length $n = |N|$, where N denotes an index set of elements $N = \{1, 2, \ldots, n\}$. In this representation, a value of 0 or 1 at the i-th bit (element) implies that $x_i = 0$ or 1 in the solution, respectively.

Let S_1 be an index set of elements with $x_i = 1$ for all $i \in N$ and S_0 be an index set of elements with $x_i = 0$ for all $i \in N$. In the MDP, we thus note that $S_1 \cup S_0 = N$ and $S_1 \cap S_0 = \emptyset$. To be a feasible solution, it is restricted that a sum of $x_i = 1$ for all $i \in N$ is equal to m $(= |S_1| = |N| - |S_0|)$ due to the constraint in the formulation (1). Note that in this paper the solution representation x always corresponds to a representation S_1 and S_0.

2.2 Neighborhoods

Although we use a k-flip local search heuristic in the LS process of the memetic framework for the MDP, 2-flip based neighborhood is mainly used in the k-flip local search as a basic move structure. In the crossover and mutation operators in the memetic algorithm, 1-flip based neighborhood is used. Thus, we here describe the two neighborhoods for the MDP.

Given a solution x, the 1-flip neighborhood $\mathcal{NB}_1$ is defined by the set of solutions that can be obtained by flipping a single bit x_i in the current solution. Thus, a hamming distance $d_H(x, x')$ between the current solution

x and the neighboring solution x' is 1. The number of all possible solutions that can be created from a current solution by the 1-flip neighborhood $\mathcal{NB}_1$ at a time is equal to n. Even if a given solution is feasible, a feasibility of the neighboring solutions that can be reached by the neighborhood is not preserved since the number of '0' and that of '1' in the neighboring solutions are changed from the feasible condition of $\sum_{i=1}^{n} x_i = m$. In order to guarantee feasibility of solutions, several considerations should be taken into account.

The 2-flip neighborhood $\mathcal{NB}_2$ is defined by the set of all solutions that can be reached by simultaneously flipping two bits x_i ($i \in S_1$) and x_j ($j \in S_0$) in the current solution x. The hamming distance between the current solution x and its neighboring solution x' can be $d_H(x, x') = 2$. Note in this neighborhood that it is not allowed to flip two bits i and j in the same set (e.g., $i, j \in S_0$). Given a feasible solution, therefore, the feasibility of neighboring solutions by the neighborhood can be always preserved. The number of possible neighboring solutions at a time is equal to $|S_1||S_0|$.

2.3 Gain Calculation for Neighbors

In order to perform an efficient search for a problem, it is crucial to calculate the difference $\Delta = f(x') - f(x)$, where f is an objective function of the problem, and x' denotes a neighboring solution obtained from a current one x by reference of a neighborhood, instead of naively calculating the cost of x' by $f(x')$ from scratch. In this paper, we refer the difference Δ to the term *gain* for a given neighboring solution x'. For the MDP, the gain can be computed much faster than the naive calculation $f(x')$.

Fast Gain Calculation for 1-flip Neighborhood

To achieve a fast calculation for the gain in the local search or memetic algorithm to the MDP, we refer to the paper of Merz and Freisleben [19] (see also [6, 20] as other related references). They showed that a calculation of all gains for the 1-flip (or 1-opt) neighbors to the BQP can be computed in linear time. The gain calculation for the BQP can be used for the MDP without modification.

Naively, the gain value g_j of flipping a single j-th bit in a current solution x can be computed by the difference between the objective function values of $f(x')$ and $f(x)$, i.e., $g_j = f(x') - f(x)$, where $x' = 1 - x_j$. However, the gain g_j can be calculated by the following formula:

$$g_j = d_{jj}(\overline{x}_j - x_j) + 2 \sum_{i=1, i \neq j}^{n} d_{ij}\, x_i\, (\overline{x}_j - x_j), \tag{3}$$

with $\overline{x}_j = 1 - x_j$. In this case, the gain g_j of flipping j-th bit in the current solution can be calculated in $O(n)$. However, the calculation of the all gains for the n candidates takes $O(n^2)$ time by using this formula.

Using the information of the all gains that have been already computed, all of new gains can be calculated efficiently, instead of recalculating them by (3). To achieve such a calculation, we take into account the update of the gains that is based on calculating the difference of gains Δg_i ($\forall i \in N$). Assuming that all g_i have been calculated and the bit j is flipped, the new gains g'_i can be computed efficiently by

$$g'_i = \begin{cases} -g_i & \text{if } i = j \\ g_i + \Delta g_i(j) & \text{otherwise} \end{cases} \quad \text{with} \quad \Delta g_i(j) = 2\, d_{ij}\, (\overline{x}_i - x_i)\, (x_j - \overline{x}_j). \tag{4}$$

The update of the gains for the n candidates of the 1-flip neighboring solutions can be performed in linear time. Furthermore, only the gains g_i for $d_{ij} \neq 0$ have to be updated [19].

This update technique [19] for the 1-flip neighbors is basically embedded with our local search and memetic algorithms for the MDP. If we consider a k-flip ($1 < k < n$) based neighborhood as used in the k-flip local search for the MDP, the following can be useful.

Generalized Gain Calculation

We now show a generalized gain calculation for a k-flip neighbor ($1 < k < n$) in the current solution in order to efficiently perform k-flip neighborhood search. The information of the matrix of a given problem instance and the gains for the 1-flip neighbors in a solution is fully used in the generalized gain calculation. Assuming that all gains g for the 1-flip neighbors are calculated and several k bits are flipped for a current solution x, a gain G of flipping the k bits (we assume in the following that the bits are stored in flip[ ] for convenience and all the bits are different) can be computed by

$$\begin{aligned} G = {} & g_\alpha && (\textit{1-flip}) \\ & + g_\beta + 2d_{\alpha\beta}(1-2x_\alpha)(1-2x_\beta) && (\textit{2-flip}) \\ & + g_\gamma + 2d_{\beta\gamma}(1-2x_\beta)(1-2x_\gamma) + 2d_{\gamma\alpha}(1-2x_\gamma)(1-2x_\alpha) && (\textit{3-flip}) \\ & + g_\delta + 2d_{\gamma\delta}(1-2x_\gamma)(1-2x_\delta) + 2d_{\delta\alpha}(1-2x_\delta)(1-2x_\alpha) \\ & + 2d_{\delta\beta}(1-2x_\delta)(1-2x_\beta) && (\textit{4-flip}) \\ & \qquad\qquad \vdots && \vdots \\ & = \sum_{i=1}^{k} g_{\mathrm{flip}[i]} + 2\sum_{i=1}^{k-1}\sum_{j=i+1}^{k} d_{\mathrm{flip}[i]\mathrm{flip}[j]}(1-2x_{\mathrm{flip}[i]})(1-2x_{\mathrm{flip}[j]}) && (\textit{k-flip}) \\ & \qquad (1 < k \leq n, \mathrm{flip}[\,] = \{\alpha, \beta, \gamma, \delta, \ldots\}). \end{aligned} \tag{5}$$

For example, assuming that two bits of α-th and β-th are flipped (i.e., 2-flip neighborhood) in the current solution, the gain G for the two bits can be given by a sum of g_α, g_β, and $2d_{\alpha\beta}(1-2x_\alpha)(1-2x_\beta)$ if the gains g for the 1-flip neighbors in the current solution is provided in advance. As shown

above, this calculation for the 2-flip neighbor can be extended to several k bits for the generalized k-flip neighborhood. The update of the gains have to be performed after each flip of k bits. The update can be done by (4).

This generalized gain calculation is valid for the MDP and the BQP.

2.4 k-flip Local Search

The larger sized neighborhoods such as k-flip neighborhood ($1 \ll k < n$) for the MDP may yield better local optima but the effort needed to search the neighborhood is too computationally expensive. An idea of the variable depth search (VDS) [13, 16] is based on efficiently searching a small fraction of the large neighborhood.

A basic idea of VDS based local search for the MDP is described as follows. Given a feasible solution x as an initial solution, in each iteration a sequence of m (or $n-m$, see below) solutions is produced by 2-flip based sub-moves leading from one solution to another, and the best solution x_{best} in the sequence is adopted as a new initial solution x for the next iteration. Such a process is repeated until no better solution is found.

To produce the sequence, the 2-flip based moves are sequentially performed so that each bit of x is flipped no more than once. All m solutions in the sequence are different and each solution x' differs two to k bits from the initial solution x. Thus, the hamming distance d_H between the initial solution x and each solution x' is $d_H(x, x') = k$, where $k = \{2, 4, \ldots, 2m-2, 2m\}$. Since the solution x_{best} with the highest cost is selected from the resulting sequence, the hamming distance $d_H(x, x_{best})$ is variable in each iteration of the algorithm, i.e., $d_H(x, x_{best}) = k$.

This specialized neighborhood may be called *k-flip neighborhood*. The k-flip neighborhood, denoted by $\mathcal{NB}_k$, for the MDP can be defined as follows:

$\mathcal{NB}_k(x) := \{x' \mid x'$ is obtained from a sequence of m solutions that can be obtained from x by exchanging an index i in one set S_1 with an index j in the other set S_0 under the following prohibition: all of the exchanged i and j are not re-exchanged$\}$.

Note in this neighborhood that the number m of the solutions in the sequence described above depends on the problem constraint. Throughout the paper, we assume that a given number m in the problem constraint is greater than one and fewer than a half of n variables [3]. If the given number m is greater than $n/2$, the elements of each S_0 and S_1 are all swapped before the search and the number of solutions produced in each iteration should be $n - m$.

[3] If the given number m is just $n/2$ in the problem constraint, we should produce $m-1$ solutions for the sequence in each iteration of the algorithm, because both a given current solution and a resulting m-th solution in the sequence become the same, that is, all elements in each set S_1 and S_0 in an initial state are only exchanged each other, if the 2-flip move is embedded with the k-flip neighborhood search.

```
    procedure k-Flip-Local-Search-QuasiBstImp2-FlipMove(x, g)
    begin
 1    repeat
 2        x_prev := x, G_max := 0, G := 0, C1 := S_1, C0 := S_0;
 3        repeat
 4            find j with g_j = max_{j∈C1} g_j;
 5            find k with gain = max_{k∈C0} (g_k + g_j + 2d_jk(1 − 2x_k)(1 − 2x_j));
 6            G := G + gain;
 7            x_j := 1 − x_j, x_k := 1 − x_k, and update gains g for each flipping;
 8            C1 := C1\{j}, C0 := C0\{k};
 9            if G > G_max then G_max := G, x_best := x;
10        until new x_best is not found for several repeats or C1 = ∅;
11        if G_max > 0 then x := x_best else x := x_prev;
12    until G_max ≤ 0;
13    return x;
    end;
```

Fig. 1. *k-flip* Local Search with Quasi-Best Improvement 2-flip Move

Quasi-Best Improvement k-flip Local Search

Our k-flip local search used in our memetic algorithm is based on the above basic idea. To produce a sequence of different m solutions in each iteration, we perform the 2-flip based sub-move with quasi-best improvement. We thus call it the *quasi-best improvement k-flip local search.* The meaning of the *quasi-best* is mentioned later.

Figure 1 shows the pseudo-code of the quasi-best improvement k-flip local search heuristic for the MDP. In the figure, we assume that 1) a feasible solution x and an associated gain vector g are provided in advance. 2) the gain vector is maintained and updated using (4) after each flip, and the generalized gain calculation (5) is used for solutions by 2-flip moves. 3) the solution x always corresponds to the representation of the sets S_1 and S_0 as mentioned in the section 2.1.

The local search consists of two loops: an inner loop in which a sequence of solutions is produced and the best solution is selected from the sequence and an outer loop in which the best solution found in the inner loop is evaluated.

To produce a sequence of different solutions in the inner loop, two candidate sets of $C1$ and $C0$ are used to ensure that each bit of a given initial solution x is flipped no more than once. Therefore, a basic stopping criterion of the search in the inner loop is expressed as $C1 = \emptyset$. To choose the best solution in the sequence, the inner loop involves a judgment process (line 9) whether a current solution x' is better than the incumbently stored best-solution x_{best}. Such a judgment plays a key role in a reduction of running time for the local search. In the k-flip local search, we change the stopping criterion of the search in the inner loop as follows: the inner loop (line 10) is terminated if new x_{best} is not found for more than t repeats or if $C1 = \emptyset$. A parameter t for the MDP is set to a range $1 < t < m$ in advance and is fixed during

```
   procedure MA
   begin
 1    initialize a population P ∈ {I_1, ..., I_PS};
 2    foreach individual I ∈ P do I := Local-Search(I);
 3    repeat
 4        for i := 1 to #crossovers do
 5            choose two parents I_a, I_b ∈ P randomly;
 6            I_c := Crossover(I_a, I_b);
 7            I_c := Repair(I_c);
 8            I_c := Local-Search(I_c);
 9            add an individual I_c to P_c;
10        endfor
11        P := Selection(P, P_c);
12        if diversification=true then
13            foreach individual I ∈ P\{best individual } do
14                I := Mutation(I);
15                I := Repair(I);
16                I := Local-Search(I);
17        endif
18    until terminate=true;
19    return best individual ∈ P;
   end;
```

Fig. 2. An outline of our evolutionary approach to the MDP

the local search. When choosing a suitable value of the parameter, it is quite expected that the running time of the local search is considerably reduced in comparison with only the basic criterion, but a sacrifice may be made in the guarantee of choosing the *true* best solution in the sequence which might be produced with the criterion $C1 = \emptyset$. Such a parameter setting is derived from the k-opt local search for the BQP [19]. In our k-flip local search for the MDP, we adopt a parameter value $t = m/5$ for larger MDP instances ($n \geq 500$), for smaller instances ($n < 500$) t is set to m. These setting show good behavior in our initial experiments.

In the outer loop, the solution x_{best} selected is evaluated whether x_{best} is better than the initial solution given at beginning of the inner loop. This can be done by a check $G_{max} > 0$. If satisfied, the k-flip neighborhood search is performed after x_{best} is set to x, otherwise, the local search is terminated after the return of the best solution found during the search.

The *quasi-best* improvement k-flip local search is a faster variant of the best improvement k-flip local search. In the best improvement version, each solution in the sequence is produced by selecting the best pair with the highest gain in the sets $C1$ and $C0$. This local search takes $O(m|S_1||S_0|)$ time per iteration. However, in the quasi-best improvement version, a first bit with the highest gain is selected from $C1$ and then a second bit with the highest gain in the 2-flip move is determined from $C0$. Thus, the time complexity of each iteration in the quasi-best improvement version is $O(m|S_1| + m|S_0|)$ time.

3 Memetic Algorithm for MDP

Memetic framework for the MDP shown in this paper is similar to one for other difficult optimization problems, which consists of a local search procedure and evolutionary operators. However, each operation in the framework has to be devised so as to work well for the MDP.

An outline of our memetic algorithm is shown in Figure 2. After the initialization of the population, new offspring are created by application of crossover and local search a predefined number of times. A new population is produced by selecting individuals from the old population and the set of generated offspring (P_c). Unless the search has converged, this process is repeated until a predefined time limit is exceeded.

In the following, the evolutionary operators are described in detail.

3.1 Creating the Initial Population

In our approach, the initial solutions (individuals) $(I_1, \ldots, I_{PS})$ of a population P are created by a randomized greedy method, where PS is a predetermined number of the individuals. The method is a variant of the randomized greedy heuristic for the BQP described in [19]. The greedy method for the MDP is devised so that m bits with '1' are appeared in a solution of length n in order to create a feasible solution. Afterwards, each of these feasible individuals is locally optimized by a local search, i.e., the quasi-best improvement k-flip local search, to create an initial population of locally optimum solutions.

3.2 Crossover

In the MDP, classical crossover operators, such as one-point, two-point, or uniform crossover can be applied, but it is not preserved in many cases that a new solution created by such a crossover is feasible even if starting with two feasible parents.

In our approach, we use the uniform crossover in which a single offspring is created from two parents, as shown in Figure 3. In the crossover process, two parents are chosen randomly from a current population such that all individuals are used with a restriction that no individual in the population is used twice in each generation. Therefore, the number of crossover processes depends on the size of population, i.e., $PS/2$.

After each crossover process, a repair method is applied to turn an infeasible offspring into a feasible one, and each feasible solution is locally improved by the k-flip local search. The repair method is described in 3.4.

3.3 Selection and Diversification/Restart Strategy

In each generation, a new population has to be formed after offspring have been generated. In our approach, the PS individuals with the highest fitness

parent1	1	0	0	1	1	0	1	0
parent2	0	0	1	1	0	1	1	0
	*	0	*	1	*	*	1	0
offspring	0	0	1	1	1	0	1	0

Comment : For the offspring, a value of 0 or 1 at each position '*' is chosen with probability 0.5.

Fig. 3. An example of uniform crossover

```
   procedure Repair(x, g)
   begin
 1    calculate a violation v := m − |S1|;
 2    if v = 0 then return x;
 3    else if v < 0 then
 4        repeat
 5            find j with g_j = max_{j∈S1} g_j;
 6            x_j := 1 − x_j, S1 := S1\{j}, and update gains g;
 7        until Σ_{i=1}^{n} x_i = m;
 8        return x;
 9    else
10        repeat
11            find j with g_j = max_{j∈S0} g_j;
12            x_j := 1 − x_j, S0 := S0\{j}, and update gains g;
13        until Σ_{i=1}^{n} x_i = m;
14        return x;
15    endif
   end;
```

Fig. 4. Repair Method

of the old population P and the set of offspring P_c are selected. However, the duplicates from the temporary set containing P and P_c are removed to ensure that no MDP solution is contained in the new population more than once.

A general drawback in evolutionary approach may be a premature convergence of the algorithm, especially in the absence of mutation. We thus perform a diversification/restart strategy, which is borrowed from [4], in order to move to other points of the search space if no new best individual in the population was found for more than 30 generations. In response to this requirement, the individuals except for the best one in the population are mutated by flipping randomly chosen $n/2$ bits for each individual of length n. After that, each of the mutated individuals is applied to the repair method. Then each individual after the repair method is locally improved by the k-flip local search to obtain a renewal set of local optima and the search is started again with the new, diverse population.

3.4 Repair Method

A repair method should be applied to an infeasible solution after the crossover and the mutation in the diversification strategy, since the feasibility of the solution by such an operator is not preserved for the MDP.

Analogous to the k-flip local search procedure shown above, the gains g corresponding to a given solution x is managed and updated in the repair procedure. When a solution x given for the repair method is infeasible, it is repeated that a violated bit with the highest gain is flipped in each repair iteration even if the highest one is negative. Such a repair process is executed until x becomes feasible. By using this repair algorithm, the given infeasible solution is turned to be feasible one that is as better cost as possible.

Our repair algorithm for the MDP is given in pseudo-code in Figure 4. At first, the violation number of the given solution x is calculated. If no violation, the solution is immediately returned as a feasible one at line 2, otherwise, the repair process is performed to obtain a feasible solution from the given infeasible one by flipping v bits in the set of S_1 or S_0 according to a judgment of line 3. The number of the repair iterations therefore depends on the number v. The each iteration consists of selecting a bit with the highest gain and flipping the bit with the update of the gains. The time complexities of each repair iteration in the algorithm are $O(|S_1|)$ or $O(|S_0|)$ time for the line 5 or 11 and $O(n)$ time for updating gains g after each flip.

4 Computational Experiments

4.1 Test Instances

For the experiments, we newly provide six problem sets for the MDP. Each problem set is characterized by the following problem sizes: n = 100, 250, 500, 750, 1,000, and 2,500 variables. We name them `mdp00100`, `mdp00250`, `mdp00500`, `mdp00750`, `mdp01000`, and `mdp02500`, respectively. The each set, i.e., the matrix d, is generated in the following way: each of d_{ij} $(i < j)$ values is given randomly between 1 and 50. Therefore, each matrix is 100% dense problem but the diagonal is off, $d_{ii} = 0$. Each problem set consists of four instances, and each of the four instances in the set is characterized by a different value of m. The four values of m are set to 10%, 20%, 30%, 40% of the variable size n, respectively.

In addition, we also use three test problem sets, which we modified benchmark instances of the BQP contained in ORLIB [2]. (This is the first attempt in research for the MDP.) Their variable sizes are n = 500, 1,000, and 2,500, and their names are `beas500-1`, `beas1000-1`, and `beas2500-1`, respectively, which are first used as test instances for BQP's heuristic algorithms in [3]. Note in each problem set that the d_{ij} coefficients in the original matrix are not restricted to non-negative values. In the three sets a density of each matrix

is 10%. However, we modified the matrix as follows: the diagonal is off, i.e., $d_{ii} = 0$, due to the definition in the MDP. Each set consists of four instances that are characterized by the four values of m as well as the new test problem instances we provided above.

The variable sizes of the first five sets in our new test problems and the first two sets in ORLIB are competitive with [14] that reported results for randomly generated instances of up to 1,000 variables, but the remainders are much larger than any reported in the literature for the MDP.

The test instances we newly provided are available from the following web page: `http://k2x.ice.ous.ac.jp/~katayama/bench/`.

4.2 Results and Discussions

We imposed a time limit for the memetic algorithm. The time limit was chosen for each variable size of the problem sets: 10 seconds for 100 variable instances, 30 (sec) for 250 variable instances, 100 (sec) for 500 variable instances, 300 (sec) for 750 variable instances, 1000 (sec) for 1,000 variable instances, and 3000 (sec) for 2,500 variable instances, on a Sun Ultra 5/10 (UltraSPARC-IIi 440MHz). The algorithm was run 30 times for each instance. Each run of the algorithm is performed with a different seed. The value of the best solution found by the algorithm in each run was saved with their corresponding generation number, running time, etc. The algorithm was implemented in *C*. The program code was compiled with the gcc compiler using the optimization flag -O2 on Solaris 8.

The parameters contained in the memetic algorithm have already described in the previous sections except for the population size. The population size PS was set to 40, which is a commonly used population size for evolutionary algorithms incorporating local search.

Table 1 shows the results for the memetic algorithm with the k-flip local search obtained for the test problem instances. In the first three columns of the table, the name of the problem sets, the variable size n, and the number of the problem constraint m are given. In the following columns, we provide the best solution value (quality % of the best solution), the average solution value (quality % of the average solution) of 30 runs, the number of times in which the best solution could be found by the algorithm "b/run", the average running time "t1" in seconds in case the algorithm could find the best solution, and the time-limit "t2" in seconds (exclusive of the case the algorithm could find the best solution). In addition, "t1" and "t2" are provided with their corresponding average generation numbers "(gens)". In the table, the number of 30/30 shown in the column of "b/run" indicates that the best-known solution could be found by the algorithm within the predefined time limit in *all* 30 trials. As an additional result, the final line in this table shows the result of the MA with longer time limit of 30000 seconds for `mdp02500` with $m = 750$. In the result, the MA found a better solution of $f(x) = 14988436$

Table 1. Results for the quasi-best improvement k-flip local search based MA

instance (off diag.)			Memetic Algorithm with k-flip Local Search						
name	n	m	best (%)	avg. (%)	b/run	t1	(gens)	t2	(gens)
mdp00100	100	10	3606 (0.000000)	3606.0 (0.000000)	30/30	0.1	(4)	—	(—)
	100	20	12956 (0.000000)	12956.0 (0.000000)	30/30	0.1	(1)	—	(—)
	100	30	27036 (0.000000)	27036.0 (0.000000)	30/30	0.1	(1)	—	(—)
	100	40	4587discussion2 (0.000000)	45872.0 (0.000000)	30/30	0.2	(2)	—	(—)
mdp00250	250	25	20834 (0.000000)	20834.0 (0.000000)	30/30	1.0	(6)	—	(—)
	250	50	75816 (0.000000)	75816.0 (0.000000)	30/30	1.8	(5)	—	(—)
	250	75	162252 (0.000000)	162252.0 (0.000000)	30/30	14.4	(40)	—	(—)
	250	100	279470 (0.000000)	279470.0 (0.000000)	30/30	2.5	(4)	—	(—)
mdp00500	500	50	78898 (0.000000)	78898.0 (0.000000)	30/30	5.1	(15)	—	(—)
	500	100	291916 (0.000000)	291916.0 (0.000000)	30/30	4.5	(5)	—	(—)
	500	150	631898 (0.000000)	631898.0 (0.000000)	30/30	12.9	(24)	—	(—)
	500	200	1096092 (0.000000)	1096092.0 (0.000000)	30/30	1.7	(2)	—	(—)
mdp00750	750	75	171704 (0.000000)	171704.0 (0.000000)	30/30	74.5	(99)	—	(—)
	750	150	641140 (0.000000)	641140.0 (0.000000)	30/30	12.6	(6)	—	(—)
	750	225	1395672 (0.000000)	1395672.0 (0.000000)	30/30	106.3	(71)	—	(—)
	750	300	2430660 (0.000000)	2430660.0 (0.000000)	30/30	34.4	(16)	—	(—)
mdp01000	1000	100	299730 (0.000000)	299721.7 (0.002758)	28/30	456.7	(249)	1000	(524)
	1000	200	1125264 (0.000000)	1125264.0 (0.000000)	30/30	165.4	(57)	—	(—)
	1000	300	2458316 (0.000000)	2458316.0 (0.000000)	30/30	375.0	(102)	—	(—)
	1000	400	4292438 (0.000000)	4292438.0 (0.000000)	30/30	81.3	(16)	—	(—)
mdp02500	2500	250	1775366 (0.000000)	1774418.9 (0.053349)	3/30	2313.4	(163)	3000	(203)
	2500	500	6794586 (0.000000)	6793782.4 (0.011827)	2/30	930.3	(30)	3000	(123)
	2500	750	14988200 (0.001575)	14987346.8 (0.007267)	0/30	—	(—)	3000	(88)
	2500	1000	26332276 (0.000000)	26332274.3 (0.000007)	29/30	927.1	(19)	3000	(82)
beas500-1	500	50	21750 (0.000000)	21750.0 (0.000000)	30/30	1.7	(25)	—	(—)
	500	100	48738 (0.000000)	48738.0 (0.000000)	30/30	1.0	(6)	—	(—)
	500	150	73544 (0.000000)	73544.0 (0.000000)	30/30	9.6	(62)	—	(—)
	500	200	95516 (0.000000)	95516.0 (0.000000)	30/30	0.5	(1)	—	(—)
beas1000-1	1000	100	62102 (0.000000)	62102.0 (0.000000)	30/30	22.1	(64)	—	(—)
	1000	200	141512 (0.000000)	141512.0 (0.000000)	30/30	41.6	(79)	—	(—)
	1000	300	218478 (0.000000)	218478.0 (0.000000)	30/30	177.8	(245)	—	(—)
	1000	400	284688 (0.000000)	284688.0 (0.000000)	30/30	57.0	(65)	—	(—)
beas2500-1	2500	250	246678 (0.000000)	246678.0 (0.000000)	30/30	581.9	(248)	—	(—)
	2500	500	566928 (0.000000)	566473.1 (0.080245)	9/30	1323.0	(318)	3000	(709)
	2500	750	882276 (0.000000)	882240.0 (0.004080)	28/30	720.2	(118)	3000	(506)
	2500	1000	1153068 (0.000000)	1151871.1 (0.103798)	1/30	1245.2	(189)	3000	(397)
mdp02500	2500	750	14988436 (0.000000)	14988307.6 (0.000857)	20/30	15000.3	(432)	30000	(807)

than the case of 3000 seconds. Thus, it indicates that the MA is capable of finding better solutions if longer running times are allowed.

Since the optimal solution for each instance is unknown yet, we reported the value of the best solution found by the algorithm for each instance as a result. It is expected that each of these best-known solutions is likely to be the very near-optimal or the optimal solution for each of the instances.

To show the effectiveness of the k-flip local search based memetic algorithm (MA-k-flip) for the MDP, we test a 2-flip local search based variant algorithm (MA-2-flip). The difference between them is only the local search process. In the variant, the same parameters and time limits set in the memetic algorithm with k-flip local search are adopted to be fair. This 2-flip local search performs the quasi-best improvement strategy as moves in each iteration.

Table 2. Results for the quasi-best improvement 2-flip local search based MA

instance (off diag.)			Memetic Algorithm with 2-flip Local Search							
name	n	m	best	(%)	avg.	(%)	b/run	t1	(gens)	t2	(gens)
mdp00100	100	10	3606	(0.000000)	3606.0	(0.000000)	30/30	0.1	(14)	—	(—)
	100	20	12956	(0.000000)	12956.0	(0.000000)	30/30	0.1	(4)	—	(—)
	100	30	27036	(0.000000)	27036.0	(0.000000)	30/30	0.1	(3)	—	(—)
	100	40	45872	(0.000000)	45872.0	(0.000000)	30/30	0.1	(7)	—	(—)
mdp00500	500	50	78898	(0.000000)	78898.0	(0.000000)	30/30	6.8	(78)	—	(—)
	500	100	291916	(0.000000)	291916.0	(0.000000)	30/30	7.1	(84)	—	(—)
	500	150	631898	(0.000000)	631898.0	(0.000000)	30/30	8.5	(100)	—	(—)
	500	200	1096092	(0.000000)	1096092.0	(0.000000)	30/30	7.6	(77)	—	(—)
mdp01000	1000	100	299730	(0.000000)	299642.5	(0.029204)	8/30	459.4	(717)	1000	(1610)
	1000	200	1125264	(0.000000)	1125180.9	(0.007382)	18/30	516.3	(832)	1000	(1518)
	1000	300	2458316	(0.000000)	2458290.6	(0.001033)	11/30	444.2	(756)	1000	(1557)
	1000	400	4292438	(0.000000)	4292438.0	(0.000000)	30/30	49.4	(100)	—	(—)
beas500-1	500	50	21750	(0.000000)	21750.0	(0.000000)	30/30	2.1	(112)	—	(—)
	500	100	48738	(0.000000)	48738.0	(0.000000)	30/30	0.9	(55)	—	(—)
	500	150	73544	(0.000000)	73544.0	(0.000000)	30/30	22.8	(955)	—	(—)
	500	200	95516	(0.000000)	95516.0	(0.000000)	30/30	0.7	(23)	—	(—)
beas1000-1	1000	100	62102	(0.000000)	62097.9	(0.006548)	29/30	188.1	(2013)	1000	(9906)
	1000	200	141512	(0.000000)	141512.0	(0.000000)	30/30	161.9	(1601)	—	(—)
	1000	300	218478	(0.000000)	218406.5	(0.032742)	1/30	46.0	(529)	1000	(9344)
	1000	400	284688	(0.000000)	284653.9	(0.011990)	14/30	179.8	(1796)	1000	(9546)

Table 2 shows the results obtained by MA-2-flip, the memetic algorithm with the 2-flip local search, only for the 100, 500 and 1000 variables instances among the nine problem sets. In the table, we give the same column entries as in Table 1.

From the results of Tables 1 and 2, the performance of MA-k-flip may be comparable with that of MA-2-flip for the instances of $n \leq 500$ since the best solution found by MA-k-flip can be obtained by MA-2-flip with a high frequency (see the column of "b/run"). However, MA-k-flip has a better advantage for the larger instances of $n > 500$: the numbers of "b/run" in MA-k-flip are greater and the running times for reaching the best-known solutions are less than those of MA-2-flip in many cases, although it seems that MA-k-flip spends more running times per generation in the predefined time limit for the computation. Thus, the effectiveness of the k-flip local search based MA is superior to MA-2-flip.

Since the difference between MA-k-flip and MA-2-flip is only the process of local search, the results of Tables 1 and 2 make it clear that a design in the local search process is quite important to obtain good solutions for the problem. Moreover, better results are expected if the optimal value of the parameters is determined and if we devise evolutionary operators instead of simple ones used in our algorithms.

Unfortunately, a comparison with the previously proposed approaches to the MDP is difficult because in most cases their algorithms were tested on smaller instances generated by them and without using of publicly available instances such as BQP contained in ORLIB as tried in this paper.

Finally, we give our experience on the setting value of mutation in the diversification/restart strategy. Although our strategy with the default setting of $n/2$ in the mutation is considerably disruptive, we believe that this setting value is a better choice than that of a smaller value in this memetic framework for the MDP. In our additional experiments, we have attempted to flip smaller bits of $n/3$ chosen randomly in the mutation for each individual except for the best one of the current population, instead of the default setting. The results showed that the default setting gave better solutions, particularly with MA-k-flip and MA-2-flip for large instances.

5 Conclusion

In this paper, we have presented a memetic algorithm for solving the maximum diversity problem. Although most of the components of our algorithm were comparable in a standard memetic framework, newly developed methods, i.e., the powerful k-flip local search, the repair method, etc. were incorporated to obtain good solutions and to preserve the feasibility of solutions for the MDP. The results showed that the k-flip local search based memetic algorithm outperformed the 2-flip local search based variant particularly for larger instances we newly provided and contained as the BQP instances in ORLIB. Due to the first report for such instances, the values of the best solution found by the algorithm were also reported for the problem instances investigated.

One of the most important issues for future research is to compare the MA with other (meta-)heuristics for the same problem instances in order to assert the effectiveness of memetic approach to the MDP.

References

1. Amini, M.M., Alidaee, B., Kochenberger, G.A. (1999) A Scatter Search Approach to Unconstrained Quadratic Binary Programs. In Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill, London 317–329
2. Beasley, J.E. (1990) OR-Library: Distributing Test Problems by Electronic Mail. Journal of the Operational Research Society **41** : 1069–1072
3. Beasley, J.E. (1998) Heuristic Algorithms for the Unconstrained Binary Quadratic Programming Problem. Tech. Rep., Management School, Imperial College, UK
4. Eshelman, L. (1991) The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Rawlings, G.J.E., ed. Foundations of Genetic Algorithms 265–283
5. Glover, F., Kuo, C.-C., Dhir, K.S. (1998) Heuristic Algorithms for the Maximum Diversity Problem. Journal of Information and Optimization Sciences **19** : 109–132

6. Glover, F., Kochenberger, G., Alidaee, B., Amini, M. (1999) Tabu Search with Critical Event Memory: An Enhanced Application for Binary Quadratic Programs. In Voss, S., et al., eds.: Meta-Heuristics, Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Pub. 93–109
7. Glover, F., Alidaee, B., Rego, C., Kochenberger, G. (2002) One-Pass Heuristics for Large-Scale Unconstrained Binary Quadratic Problems. European Journal of Operational Research **137** : 272–287
8. Ghosh, J.B. (1996) Computational Aspects of the Maximum Diversity Problem. Operations Research Letters **19** : 175–181
9. Katayama, K., Narihisa, H. (2001) Performance of Simulated Annealing-Based Heuristic for the Unconstrained Binary Quadratic Programming Problem. European Journal of Operational Research **134** : 103–119
10. Katayama, K., Tani, M., Narihisa, H. (2000) Solving Large Binary Quadratic Programming Problems by Effective Genetic Local Search Algorithm. In: Proceedings of the 2000 Genetic and Evolutionary Computation Conference 643–650
11. Katayama, K., Narihisa, H. (2001) A Variant *k-opt* Local Search Heuristic for Binary Quadratic Programming. Trans. IEICE (A) **J84-A** : 430–435 (*in Japanese*)
12. Katayama, K., Narihisa, H. (2001) On Fundamental Design of Parthenogenetic Algorithm for the Binary Quadratic Programming Problem. In: Proceedings of the 2001 Congress on Evolutionary Computation 356–363
13. Kernighan, B.W., Lin, S. (1970) An Efficient Heuristic Procedure for Partitioning Graphs. Bell System Technical Journal **49** : 291–307
14. Kochenberger, G., Glover, F. (1999) Diversity Data Mining. Tech. Rep. HCES-03-99, Hearin Center for Enterprise Science College of Business Administration
15. Kuo, C.-C., Glover, F., Dhir, K.S. (1993) Analyzing and Modeling the Maximum Diversity Problem by Zero-One Programming. Decision Sciences **24** : 1171–1185
16. Lin, S., Kernighan, B.W. (1973) An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research **21** : 498–516
17. Lodi, A., Allemand, K., Liebling, T.M. (1999) An Evolutionary Heuristic for Quadratic 0-1 Programming. European Journal of Operational Research **119** : 662–670
18. Merz, P., Freisleben, B. (1999) Genetic Algorithms for Binary Quadratic Programming. In: Proceedings of the 1999 Genetic and Evolutionary Computation Conference 417–424
19. Merz, P., Freisleben, B. (2002) Greedy and Local Search Heuristics for Unconstrained Binary Quadratic Programming. Journal of Heuristics **8** : 197–213
20. Merz, P., Katayama, K. (2002) Memetic Algorithms for the Unconstrained Binary Quadratic Programming Problem. BioSystems (submitted for publication)
21. Pardalos, P.M., Rodgers, G.P. (1992) A Branch and Bound Algorithm for the Maximum Clique Problem. Computers and Operations Research **19** : 363–375
22. Pardalos, P.M., Xue, J. (1994) The Maximum Clique Problem. Journal of Global Optimization **4** : 301–328
23. Weitz, R.R., Lakshminarayanan, S. (1997) An Empirical Comparison of Heuristic and Graph Theoretic Methods for Creating Maximally Diverse Groups, VLSI Design, and Exam Scheduling. Omega **25** : 473–482