ISSN 1392–124X INFORMATION TECHNOLOGY AND CONTROL, 2011 Vol. 40, No. 4

COMPARATIVE PERFORMANCE OF THREE METAHEURISTIC APPROACHES FOR THE MAXIMALLY DIVERSE GROUPING PROBLEM

Gintaras Palubeckis

Multimedia Engineering Department, Kaunas University of Technology
Studentu St. 50, LT-51368 Kaunas, Lithuania
E-mail: gintaras@soften.ktu.lt

Eimutis Karčiauskas

Software Engineering Department, Kaunas University of Technology
Studentu St. 50, LT-51368 Kaunas, Lithuania
E-mail: eimutis.karciauskas@ktu.lt

Aleksas Riškus

Multimedia Engineering Department, Kaunas University of Technology
Studentu St. 50, LT-51368 Kaunas, Lithuania
E-mail: aleksas@soften.ktu.lt

crossref http://dx.doi.org/10.5755/j01.itc.40.4.977

Abstract. Given a set of elements and a symmetric matrix representing dissimilarities between them, the maximally diverse grouping problem asks to find a partitioning of the elements into a fixed number of restricted size-groups such that the sum of pairwise dissimilarities between elements in the same group is maximized. We present multistart simulated annealing, hybrid genetic and variable neighborhood search algorithms for solving this problem. We report on computational experiments that compare the performance of these algorithms on benchmark instances of size up to 2000 elements.

Keywords: combinatorial optimization; maximally diverse grouping; metaheuristics; simulated annealing; genetic algorithm; variable neighborhood search.

1. Introduction

The *maximally diverse grouping problem* (MDGP for short) is the problem of partitioning a set of elements into a given number of pairwise disjoint subsets called groups, such that the groups are bounded in size and are as diverse as possible. It can be stated as follows. Suppose there is a set V of n elements to be grouped. Let $D = (d_{ij})$ be an $n \times n$ symmetric matrix with zero main diagonal and nonnegative off-diagonal entries. It is assumed that $d_{ij},\ i,j \in \{1,\ldots,n\}$, $i \neq j$, summarizes the dissimilarity between element i and element j. Additionally, suppose that the set V is partitioned into m groups, each of size at least a and at most b. Then the MDGP can be stated as follows:

$$\text{maximize} \quad F = \sum_{k=1}^{m} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_{ik} x_{jk} \qquad (1)$$

$$\sum_{k=1}^{m} x_{ik} = 1,\ i = 1,\ldots,n, \qquad (2)$$

$$a \leqslant \sum_{i=1}^{n} x_{ik} \leqslant b,\ k = 1,\ldots,m, \qquad (3)$$

$$x_{ik} \in \{0,1\},\ i = 1,\ldots,n,\ k = 1,\ldots,m. \qquad (4)$$

In this formulation, the binary variable x_{ik} is equal to 1 if and only if element i is assigned to group k. Constraints (2) guarantee that each element belongs to a unique group and constraints (3) enforce each group to be of a specified size.

One of the most studied applications of the MDGP is the assignment of students to groups [2, 6, 18, 20–22]. In this context, the goal is to create diverse groups of students, where the diversity is measured on the basis of the dissimilarity matrix D. Also, several other applications of the MDGP have been identified in the literature. These include exam scheduling [10, 21], the construction of reviewer groups [4], the assignment of employees to project teams [2], and VLSI design [20, 21].

It can be seen that the model (1)–(4) is a special case of the binary quadratic optimization problem (BQOP), as formulated, for example, in [16]. In this case, the underlying set of elements is $S = \{(i,k) \mid i = 1,\ldots,n, k = 1,\ldots,m\}$. The feasible region of the problem is a certain subset of the power set of S. Using binary variables, this subset is specified by the constraints (2) and (3). The coefficients of the objective function (1) are assigned to pairs of elements in S in one-to-many fashion. The most widely studied cases of the BQOP include unconstrained binary quadratic programming, the maximum diversity problem [3, 5, 12, 14], and the Max-2-SAT problem [15]. Actually, the MDGP is a generalization of the maximum diversity problem. Indeed, the latter is obtained from the MDGP by setting $m = 1$ and removing the constraint (2). Then the variables x_{ik} and x_{jk} can be replaced by the variables x_i and x_j, respectively. Moreover, since all the entries of the matrix D are nonnegative, the first inequality in (3) is redundant and the second one transforms into an equality.

The MDGP is an NP-hard problem and is difficult to solve. The application of exact methods to larger instances of the MDGP is unduly time consuming. Therefore, heuristic algorithms have been developed for finding good but not necessarily optimal solutions. Perhaps the simplest heuristic techniques for the MDGP are the so-called construction algorithms. An algorithm of this type starts with a set of empty groups and constructs an assignment by adding one or more elements to groups at a time. Such algorithms were given in [13, 18]. The construction algorithms are very fast but, however, the quality of solutions produced by them is generally unsatisfactory. A quite different approach to the MDGP is to use iterative improvement algorithms. There are several algorithms of this type developed in the past [1,10,19,21]. The results of computational testing presented in [21] have shown the superiority of the algorithm suggested in [19]. Recently, Fan et al. [6] proposed a hybrid genetic algorithm (GA) for solving the MDGP. In their approach, they used a specific crossover and a related encoding scheme. The GA was hybridized with a local search procedure. However, this procedure makes only one pass through the elements and thus it does not guarantee that the returned solution is locally optimal (see the appendix in [6]). Hence, the power of local search is exploited insufficiently.

In the current paper, we investigate computationally the applicability of three quite different metaheuristics to the MDGP, namely, simulated annealing, genetic algorithm and variable neighborhood search. The latter was selected as a representative of a group of non-evolutionary metaheuristics based on the local search (LS) technique. Our genetic algorithm is hybridized with an effective LS procedure. The empirical results summarized in this paper were obtained by testing the developed algorithms on the dissimilarity matrices taken from the literature.

The remainder of this paper is structured as follows. In Sections 2 to 4, we describe simulated annealing, hybrid genetic and variable neighborhood search algorithms, respectively. In Section 5, we report the results of numerical experiments comparing these algorithms. Finally, Section 6 ends the paper with a few concluding remarks.

2. Simulated annealing

The *simulated annealing* (SA) technique is a metaheuristic search method exploiting an analogy between the physical process of annealing and the process of searching for the global extremum of a function. During annealing, a material is first heated up to a very high temperature and then slowly cooled down to obtain a minimum-energy crystalline structure. The simulated annealing metaheuristic mimics this process by generating a sequence of solutions that eventually converges to the optimum of the objective function.

Before presenting the SA algorithm for the MDGP, we introduce some necessary notations. We will use $P = (V_1, \ldots, V_m)$ to denote a feasible solution to the problem. Certainly, if the same solution is represented by a 0-1 vector $(x_{ik} \mid i = 1, \ldots, n, k = 1, \ldots, m)$ satisfying (2) and (3), then $V_k = \{i \in V \mid x_{ik} = 1\}$, $k = 1, \ldots, m$. We denote by $F(P)$ the value of the objective function F for a solution P. Suppose that elements i and j belong to different groups of P. Then by interchanging the elements i and j we get a new feasible solution to the problem. We denote it by $P(i, j)$. The set of all solutions obtained from P in this way is the *pairwise interchange* (or *2-interchange*) *neighborhood* $N_2(P)$ of P. Suppose now that $a < b$ in (3). Consider an element $i \in V_r$ and a group V_k, $k \neq r$. If $|V_r| > a$ and $|V_k| < b$, then the solution obtained from P by relocating the element i from group V_r to group V_k is a feasible one. Let this particular solution be denoted by $P_k(i)$. All such solutions constitute the *relocation neighborhood* $N_1(P)$ of P.

In order to efficiently compute the difference between the values of the objective function at the solutions $P(i, j)$ (or $P_k(i)$) and P, we use an auxiliary $n \times m$ matrix $C = (c_{qk})$. Its entry c_{qk} represents the sum of the dissimilarities between the element q and all elements in the group V_k. Formally, $c_{qk} = \sum_{s \in V_k} d_{qs}$. Assume that $i \in V_r$ and $j \in V_l$. Then, using C, we can write

$$\begin{aligned} \Delta(P, i, j) &:= F(P(i, j)) - F(P) = \\ &= c_{il} - c_{ir} + c_{jr} - c_{jl} - 2d_{ij}, \end{aligned} \tag{5}$$

$$\delta(P, i, k) := F(P_k(i)) - F(P) = c_{ik} - c_{ir}. \tag{6}$$

At each iteration of the SA algorithm, the choice between selecting a solution $P(i, j)$ from the neighborhood $N_2(P)$ or a solution $P_k(i)$ from the neighborhood $N_1(P)$ is made probabilistically. For that purpose, the algorithm is supplied with a probability parameter p. Other parameters include the cooling rate α, the minimum temperature $t_{\min}$, and the repetition factor R_0. Typically, $t_{\min}$ is fixed at a very small positive number. The maximum temperature, $t_{\max}$, is set to the largest absolute value of $\Delta(P,i,j)$ over a sample of solutions randomly drawn from the 2-interchange neighborhood $N_2(P)$ of a randomly generated initial solution P.

The computation time of the SA algorithms largely depends on the cooling rate as well as on the repetition factor. However, the stopping rule based on these parameters is not suitable when we want to fairly compare the performance of the SA algorithm with that of other approaches for the MDGP. A universal termination rule is to stop an algorithm after a prescribed time period has elapsed. We adopted this rule in our computational study. In order to be able to apply it in the case of SA, we execute the simulated annealing procedure repetitively. The main algorithm, called MSA (Multistart Simulated Annealing), can be stated as follows.

```
MSA
1. Randomly generate an initial solution $P$ to the given instance of the MDGP. Initialize $P^*$ with $P$ and $F^*$ with $F(P)$.
2. Compute $t_{\max} = \max\{|\Delta(P,i,j)| \mid P(i,j) \in Q\}$, where $Q$ is a set of solutions randomly selected from the neighborhood $N_2(P)$. Set $T := \lfloor(\log(t_{\min}) - \log(t_{\max}))/\log\alpha\rfloor$, and $R := R_0 n$.
3. Apply SA($P$, $P^*$, $F^*$, $t_{\max}$, $T$, $R$, $\alpha$).
4. Check if the termination condition is satisfied. If so, then stop with the solution $P^*$ of value $F^*$. If not, then randomly generate a new starting solution $P$ for SA and return to 3.
```

Our implementation of Step 1 of MSA is based on using a randomly generated permutation of elements. Suppose, for simplicity, that $z = n/m$ is an integer number. The algorithm splits the permutation into m equally sized parts. Specifically, the first z elements are assigned to the first group, the next z elements are assigned to the second group, and so on. The same procedure involving generation of a permutation and splitting it into m parts is used also in Step 4. The purpose of Step 2 is to prepare the parameters to be passed to the simulated annealing algorithm. These parameters are the maximum temperature $t_{\max}$, the number of temperature reductions T, and the number of solutions evaluated at a temperature level (denoted by R). The size of the set Q in our experiments was fixed at 1000. Throughout both MSA and simulated annealing procedure SA, the best solution found so far is denoted by P^* and its value by F^*. The procedure SA can be described as follows.

```
SA($P, P^*, F^*, t_{\max}, T, R, \alpha$)
1. Initialize $f$ with $F(P)$, $t$ with $t_{\max}$ and $K$ with 1.
2. Set $L := 1$.
3. With probability $p$ go to 5, otherwise go to 4.
4. Randomly select elements $i$ and $j$ such that $P(i,j) \in N_2(P)$. Compute $h = \Delta(P,i,j)$ by equation (5). Set $P' := P(i,j)$. If $h \geqslant 0$, then go to 7. Otherwise go to 6.
5. Randomly select an element $i$ and group $V_k$ such that $P_k(i) \in N_1(P)$. Compute $h = \delta(P,i,k)$ by equation (6). Set $P' := P_k(i)$. If $h \geqslant 0$, then go to 7. Otherwise proceed to 6.
6. Randomly draw a number $\xi$ from the uniform distribution on $[0,1]$. If $\xi \leqslant \exp(h/t)$, then proceed to 7; else go to 8.
7. Replace $P$ with $P'$. Set $f := f + h$. If $f > F^*$, then set $P^* := P'$ and $F^* := f$.
8. Increment $L$ by 1. If $L \leqslant R$, then go to 3.
9. Increment $K$ by 1. If $K \leqslant T$, then set $t := \alpha t$ and go to 2. Otherwise return with $P^*$ and $F^*$.
```

The body of SA consists of the initialization step and two nested loops. In the initialization step, the temperature is set equal to $t_{\max}$. The outer loop successively modifies the temperature by multiplying it by the cooling factor α. Each execution of the inner loop starts with the random selection of the neighborhood of the current solution P. The relocation neighborhood $N_1(P)$ is selected with probability p. If this does not happen, then the 2-interchange neighborhood $N_2(P)$ is used. Such a strategy helps to increase a level of diversification in the search process. Of course, if $a = b$ in (3), then the probability p must be forced to zero. The current solution P is compared with a solution randomly selected from an appropriate neighborhood. The new solution P' is accepted to replace P if either it is not worse than P or the condition defined in Step 6 is fulfilled. When moving from P to P', the matrix C needs to be updated. Suppose that $P' = P(i,j)$. Furthermore, let us assume that $i \in V_r$ and $j \in V_l$. Then the formulas used for $q \in V \setminus \{i,j\}$ are: $c_{qr} := c_{qr} + d_{qj} - d_{qi}$, $c_{ql} := c_{ql} + d_{qi} - d_{qj}$. Also, d_{ij} is added to both c_{ir} and c_{jl} and subtracted from both c_{il} and c_{jr}. Similar manipulations are performed when $P' = P_k(i)$. Step 7 also updates the current objective function value f and, if this value exceeds the previous best, saves P' as the best solution found so far. It is easy to see that the time complexity of an iteration of the inner loop is $O(n)$. The most expensive operations are done in Step 7 of SA.

3. Hybrid genetic algorithm

One of the techniques that offers an alternative to traditional search methods working with only one solution at a time is *genetic algorithm* (GA). The key feature of GA is that it works with a population of individuals representing solutions to a problem. Frequently, in order to make the GA smarter, it is hybridized with a local search procedure. In this section, we propose a hybrid genetic algorithm (HGA) for solving the MDGP. The algorithm includes the following main steps: creating an initial population, reproducing offspring, applying local search to offspring, and updating the current population. In the description given below, the population is denoted by Π and its size by *pop_size*. For the ease of presentation, we denote by $H = \{(l, r) \mid l, r = 1, \dots, m\}$ the Cartesian square of the set $\{1, \dots, m\}$. The algorithm can be stated as follows.

```
HGA
1. (Initialization) Set $\Pi := \emptyset$, $\lambda := 0$ and $F^* := -\infty$. While $\lambda <$ pop_size do the following:
   1.1. Randomly generate a solution $P$ that satisfies (3).
   1.2. Apply the local search procedure LS to $P$. Let $P'$ denote the solution returned by it.
   1.3. Add $P'$ to the population $\Pi$. Increment $\lambda$ by 1. If $F(P') > F^*$, then set $P^* := P'$ and $F^* := F(P')$.
2. (Parents selection) Randomly choose two individuals, say $P^1 = (V_1^1, \dots, V_m^1)$ and $P^2 = (V_1^2, \dots, V_m^2)$, from the current population $\Pi$.
3. (Mating) Perform the following steps:
   3.1. Set $W := V$.
   3.2. Compute $e_{lr} = |V_l^1 \cap V_r^2|$ for each pair $(l, r) \in H$.
   3.3. Form a set $H^*$ of $m$ pairs $(l, r) \in H$ such that $e_{lr} \geqslant e_{uv}$ for each $(l, r) \in H^*$ and each $(u, v) \in H \setminus H^*$ (in other words, pick the $m$ largest values $e_{lr}$).
   3.4. Let $\rho$ be a one-to-one mapping from $\{1, \dots, m\}$ onto $H^*$. For each $k \in \{1, \dots, m\}$, set $V_k := V_l^1 \cap V_r^2$, where $(l, r) = \rho(k)$. Remove all elements of each $V_k$, $k \in \{1, \dots, m\}$, from $W$.
   3.5. For each group $V_k$ of size less than $a$, perform the following steps:
        3.5.1. Assuming that $\rho(k) = (l, r)$, form the set $U_k = (V_l^1 \cup V_r^2) \cap W$.
        3.5.2. If $|U_k| \leqslant g := a - |V_k|$, then add all the elements of $U_k$ to $V_k$. Otherwise, randomly select $g$ elements from the set $U_k$ and add them to $V_k$. In both cases, remove the added elements from the set $W$.
   3.6. For each group $V_k$ of size less than $a$, randomly select $a - |V_k|$ elements from the set $W$ and move them from $W$ to $V_k$.
   3.7. If the set $W$ is empty, then go to 4. Otherwise, for each element $i \in W$, perform the following steps:
        3.7.1. Identify the groups $V_l^1 \in P^1$ and $V_r^2 \in P^2$ the element $i$ belongs to.
        3.7.2. Consider the groups $V_u$, $u \in \{1, \dots, m\}$, satisfying the following two conditions: 1) $|V_u| < b$; 2) either $l = l'$ or $r = r'$, where $(l', r') = \rho(u)$. If such groups exist, then randomly select one of them and move the element $i$ from the set $W$ to this selected group.
   3.8. If the set $W$ is empty, then go to 4. Otherwise, for each element $i \in W$, perform the following operations. Randomly select a group $V_k$ such that $|V_k| < b$. Assign $i$ to $V_k$.
4. (Local search) Apply the local search procedure LS to the offspring $P = (V_1, \dots, V_m)$ constructed in the previous step. Let $P'$ denote the solution returned by LS.
5. (Offspring evaluation) Check whether $F(P') > F^*$. If so, then set $P^* := P'$ and $F^* := F(P')$. Accept the offspring $P'$ only if it is not worse than the worst individual in the current population $\Pi$. In such a case, replace the worst individual in $\Pi$ by $P'$.
6. Check if the termination condition is satisfied. If so, then stop with the solution $P^*$ of value $F^*$. If not, then return to 2.
```

In the initialization phase of HGA, a starting population of individuals is created. Random solutions to the problem are generated using precisely the same mechanism as in Step 1 of MSA. These solutions are improved by a local search procedure and gathered to form the initial population Π. Thus, already at the start of the evolution phase, Π entirely consists of locally optimal solutions. A member of Π with the maximum objective function value is saved as the best solution found so far. This solution is denoted by P^* and its value by F^*. Each iteration of the evolution process starts by choosing two individuals from Π, called parents. In our implementation of the genetic algorithm, they are selected randomly. From the selected parents, an offspring is generated. This is done in Step 3 of HGA. There, the offspring is denoted by $(V_1, \ldots, V_m)$. The groups V_k, $k = 1, \ldots, m$, are initialized with seeds formed in Steps 3.2–3.4. Actually, each seed is obtained as the result of a set intersection operation on two groups, one from each of the two parents. Each such pair of groups is evaluated by calculating the number of common elements. For $V_l^1 \in P^1$ and $V_r^2 \in P^2$, this number is denoted by e_{lr}. It is natural to favor pairs of groups with the greatest value of this quantity. When selecting m such pairs (Step 3.3), ties are broken at random. In Step 3 of the algorithm, W stands for the set of elements that are not yet assigned to groups V_k, $k = 1, \ldots, m$. This set is gradually reduced and finally emptied by performing a sequence of steps.

Usually all or almost all groups V_k, $k = 1, \ldots, m$, after initialization contain less than a elements. The minimum required size of each group is achieved in Steps 3.5 and 3.6. First, for each group $V_k = V_l^1 \cap V_r^2$ such that $|V_k| < a$, an attempt is made to enlarge it by adding some still unassigned elements from either V_l^1 or V_r^2. After this operation, the group V_k still may violate the size lower bound. All such groups are expanded to the required size in Step 3.6 using the random selection rule.

Steps 3.7 and 3.8 of HGA are needed to deal with the case of $a < b$. The elements of W are distributed to groups in two passes. In the first pass, the algorithm strives to assign element $i \in W$ to a group which is constructed starting from the seed $V_{l'}^1 \cap V_{r'}^2$ such that i belongs either to $V_{l'}^1$ or $V_{r'}^2$. In the second pass, each unassigned element, if any, is moved to a randomly selected group of size less than b. Certainly, if $a = b$, Steps 3.7 and 3.8 are skipped.

The produced offspring is submitted to a local search procedure for possible improvement. In fact, this procedure can be regarded as a kind of mutation operator. In Step 5, the population Π is updated by replacing the worst individual in Π by the offspring, unless the latter is worse than all members of Π. The evolution process is stopped when the same termination criterion (based on the CPU clock) as in the case of SA is met.

The described algorithm makes multiple calls to a local search heuristic LS. There are various ways to implement it for the MDGP. The notable features of our implementation are the following: first, LS performs an exploration of both the relocation neighborhood N_1 and the 2-interchange neighborhood N_2; second, the search is stochastic by nature, that is, LS considers elements (and groups) in the order given by a random permutation. The LS procedure consists of the following steps.

```
LS($P$)
1. Randomly generate a permutation of elements, denoted by $(\beta(1), \ldots, \beta(n))$, and a permutation of groups, denoted by $(\gamma(1), \ldots, \gamma(m))$. Initialize $f$ with $F(P)$.
2. For $i' = 1, \ldots, n$ do the following:
   2.1. Set $i := \beta(i')$. Let $V_r$ be the group the element $i$ belongs to. If $|V_r| = a$, then go to 2.3. Otherwise proceed to 2.2.
   2.2. For $k' = 1, \ldots, m$ do the following:
        2.2.1. Set $k := \gamma(k')$. If $k \neq r$ and $|V_k| < b$, then go to 2.2.2. Otherwise repeat from 2.2.1 for the next value of $k'$.
        2.2.2. Compute $h = \delta(P, i, k)$ by equation (6). If $h > 0$, then set $\tilde{P} := P_k(i)$ and go to 3.
   2.3. For $j' = i' + 1, \ldots, n$ do the following:
        2.3.1. Set $j := \beta(j')$. If $j \in V_r$, then repeat from 2.3.1 for the next value of $j'$. Otherwise proceed to 2.3.2.
        2.3.2. Compute $h = \Delta(P, i, j)$ by equation (5). If $h > 0$, then set $\tilde{P} := P(i, j)$ and go to 3.
3. If $h \leqslant 0$, then return with the solution $P$ of value $f$. Otherwise, replace $P$ with $\tilde{P}$, set $f := f + h$, and go to 2.
```

At each iteration of LS, both the neighborhood N_1 and the neighborhood N_2 are explored (respectively, in Steps 2.2 and 2.3). The result of the relocating operation is evaluated only for elements in the groups of size exceeding a. Of course, such an element is allowed to be moved to groups of size less than b only. Obviously, if $a = b$, then Step 2.2 is bypassed. Step 2.3 evaluates pairwise interchanges of elements. The increase in the objective function value, denoted by h, is calculated using formula (5). If $h > 0$ in Step 3, then LS starts the search for an improving interchange or relocation from the beginning. Upon termination, LS returns a locally optimal solution with respect to both neighborhoods N_1 and N_2.

We note that the pure genetic algorithm can be obtained from HGA simply by removing Step 4. In this case, the local search procedure is used solely as a tool for improving individuals in the initial population.

4. Variable neighborhood search

In this section, we describe an implementation of the *variable neighborhood search* (VNS) algorithm for solving the MDGP. The VNS metaheuristic is a general-purpose optimization method combining neighborhood change mechanism with local search technique. In recent years, algorithms based on the VNS framework have been successfully applied to a variety of optimization problems. The basic schemes of the approach and typical applications are reviewed in [7–9].

Before presenting our VNS algorithm, we first define the r-th neighborhood of a solution to the considered problem. Given such a solution $P = (V_1, \ldots, V_m)$, let $l(i, P)$ be the index of the group containing element $i \in V$. Thus, if $i \in V_k$, then $l(i, P) = k$. Let Ψ be the set of all solutions to the MDGP. Then its subset $N_r(P) = \{P' \in \Psi \mid l(i, P') \neq l(i, P)$ for exactly r elements $i \in V\}$ is called the *r-th neighborhood* of P in the search space. Notice that the neighborhoods N_1 and N_2 considered in the previous sections fit this definition.

The developed algorithm comprises initialization step and three phases executed iteratively: shaking, local search, and neighborhood change. The system of neighborhoods used in the algorithm is $\{N_r\}$, $r \in [r_{\min}, r_{\max}]$. For the sake of simplicity in the exposition, we will assume that for any value of r from the above interval, in the shaking phase, the algorithm always succeeds in selecting $\lfloor r/2 \rfloor$ pairs of elements such that elements in the same pair belong to different groups and no element is selected more than once. Without this assumption, the description given below should be modified slightly. The algorithm can be stated as follows.

```
VNS
1. Randomly generate an initial solution $P$ to the given instance of the MDGP. Apply the local
   search procedure LS to $P$. Let $P'$ denote the solution returned by it. Initialize $P^*$ with $P'$ and
   $F^*$ with $F(P')$.
2. Set $r := r_{\min}$.
3. Set $P := P^*$ and $W := V$.
4. Repeat $\lfloor r/2 \rfloor$ times the following steps:
   4.1. Randomly select elements $i, j \in W$ such that $l(i, P) \neq l(j, P)$ and interchange
        them (set $P := P(i, j)$).
   4.2. Remove $i$ and $j$ from $W$.
5. If either $a = b$ or $r$ is even, then go to 6. Otherwise, search for the element-group pairs $(j, u)$
   such that $j \in W$, $l(j, P) \neq u$, $|V_{l(j,P)}| > a$ and $|V_u| < b$. If no such pair exists,
   then go to 6. Otherwise, select one at random (let it be denoted by $(i, k)$). Move the element $i$
   from $V_{l(i,P)}$ to $V_k$ (set $P := P_k(i)$).
6. Apply the local search procedure LS to $P$. Let $P'$ denote the solution returned by it.
7. Check whether $F(P') > F^*$. If so, then set $P^* := P'$, $F^* := F(P')$ and $r := r_{\min}$.
   If not, then increase $r$ by $r_{\text{step}}$.
8. Check if the termination condition is satisfied. If so, then stop with the solution $P^*$ of value $F^*$.
   Otherwise, if $r \leqslant r_{\max}$, then go to 3; else go to 2.
```

Earlier in this section, we did make an assumption regarding the selection of element pairs in the shaking phase of VNS. If this assumption is abandoned, then only a couple of modifications to the description of VNS should be made. First, in Step 4, it is needed to check if $W \subseteq V_k$ for some $k \in \{1, \ldots, m\}$. If this condition is satisfied (which may occur only when r is very close to n), then the algorithm must exit from this step. Second, in Step 5, it may happen that more than one element needs to be moved from its current group to a randomly selected one. Of course, Step 5 is executed only if a is less than b.

At the beginning of VNS, the initial assignment of elements to groups is generated randomly using the same method as for the SA algorithm. This solution is improved by applying the local search procedure LS described in Section 3. The shaking phase consists of Steps 3 through 5. The algorithm strives to obtain a solution in the neighborhood $N_r(P)$ by performing the maximum number of pairwise interchanges of elements. During this process, W is used to denote the set of elements that were not involved in the interchange operation. If r is odd and $a < b$, then, in addition, one randomly selected element from W is moved from its current group to a different one. Notice that if $a = b$, then it makes sense to consider the neighborhoods $N_r(P)$ for even values of r only (since Step 5 in this case is skipped). The conditions in Step 5 guarantee that the group to which the selected element i is currently assigned differs from the target group and, after the move, for both of them, the size constraints are met. Step 7 implements the neighborhood change principle of the VNS method. If P' does not improve the best solution obtained over the previous iterations, then the value of r is increased. If

r becomes larger than $r_{\max}$, the process proceeds to Step 2, and r is switched to $r_{\min}$. The parameters of the algorithm are $r_{\min}$, r_{step} and $r_{\max}$. In our implementation, we take $r_{\max} = \tilde{r}n$, where $\tilde{r}$ is a number from the interval $[r_{\min}/n, 1]$. Thus, essentially, $\tilde{r}$ replaces $r_{\max}$ in the parameter list.

5. Experimental results

In this section, we report the results of computational experiments aiming at comparing the performance of the algorithms we have described. All the algorithms have been coded in the C programming language and all the tests have been carried out on a PC with an Intel Core 2 Duo CPU running at 3.0GHz. As a testbed we have chosen two sets of randomly generated dissimilarity matrices that have been used in the recent past for empirical evaluation of various algorithms for solving the maximum diversity problem (see, for example, [3, 5, 14]). The first set was introduced by Silva et al. in [17]. The second set is one of the four sets generated by Duarte and Martí [5]. These data can be obtained from the internet (for example, [11]).

In the experiments, we have used the following parameter setting: $\alpha = 0.95$, $t_{\min} = 0.0001$, $R_0 = 300$, $p = 0.3$ for MSA; *pop_size*=100 for HGA; $r_{\min} = 1$, $\tilde{r} = 0.3$, $r_{\text{step}} = 1$ for VNS. These parameters were fixed on the basis of preliminary tests. The results presented in this section were obtained by performing 10 runs of each algorithm on each problem instance in the chosen datasets. Maximum CPU time limits for a run were as follows: 60s for $n = 100$, 300s for $n = 200$, 600s for $n = 300$, 900s for $n = 400$, 1800s for $n = 500$, and 3600s for $n = 2000$.

Table 1 summarizes the results of MSA, HGA and VNS on the *Silva* instances. Each entry of the dissimilarity matrix defining an instance in this series is an integer randomly and uniformly drawn from the interval $[0, 9]$. The number of elements $n \in \{100, 200, 300, 400, 500\}$ is included in the name of an instance (first column in the table). For example, *Silva_100_1* denotes the first (out of four) instance with 100 elements. The numerical experiment was conducted for the case where the number of groups m was fixed at 10. The size of each group was bounded from below by $a = 0.8n/m$ and from above by $b = 1.2n/m$. The second column of Table 1 contains, for each instance, the value of the best solution obtained from all runs of MSA, HGA and VNS. The third (respectively, fourth) column shows the difference between the value displayed in the second column and the value of the best solution out of 10 runs (respectively, the average value of 10 solutions) found by MSA. The remaining columns give these differences for HGA and VNS. The results, averaged over all problem instances, are presented in the last row of the table.

As seen in Table 1, HGA is definitely superior to both the MSA and VNS algorithms. Basically, the solutions found by HGA appear to be the best in our experiment. Compared with the results of HGA, the MSA and VNS algorithms tie in one and, respectively, three cases and produce inferior solutions in all other cases. We can also see from Table 1 that VNS performs better than MSA for instances of size 100 but is worse than MSA for all larger instances in the dataset. Overall, the ranking of the results from best to worst is as follows: "*Min* HGA", "*Ave* HGA", "*Min* MSA", "*Ave* MSA", "*Min* VNS", and "*Ave* VNS".

In Table 2 we show the results obtained for the *Duarte-Martí* instances. All the entries d_{ij} of the dissimilarity matrix D for these instances are integers generated randomly from a uniform distribution between 0 and 10. We should mention that Duarte and Martí have generated 20 dissimilarity matrices of size 2000×2000. We present the results for the first 10 of them. The results of MSA, HGA and VNS for the other 10 matrices are very similar to those reported in Table 2. For our experimentation, we have fixed the number of groups to 50, the minimum group size to $0.8n/m = 32$, and the maximum group size to $1.2n/m = 48$. The structure of Table 2 is the same as that of Table 1.

From Table 2, we find that, according to solution quality, the algorithms can be ranked in the reverse order of that observed in the experiment with the *Silva* instances. Now VNS is the best-performing algorithm of the three. Meanwhile, the HGA algorithm shows the worst performance. After analyzing the results of the experiment, we found out that the LS procedure was quite time consuming for the *Duarte-Martí* instances, especially when applied to a solution that is far from local optima. This is probably the main reason why the performance of HGA deteriorates significantly on large-size instances. When applied to *DM_1*, for example, HGA was able to invoke the LS procedure within the allotted one hour only 332 times on the average (excluding the calls to LS throughout the construction of the initial population). For comparison, the average number of LS invocations in the case of the *Silva_500_1* instance exceeded 688000. When solving *DM_1* by the VNS algorithm, the average number of calls to LS was 1748.

We do not present the results of the pure genetic algorithm, which is obtained from HGA by deleting Step 4 (see Section 3) where hybridization with local search occurs. The pure GA generates a huge num-

Table 1. Results of running MSA, HGA and VNS on the *Silva* instances ($m = 10$, $a = 0.08n$, $b = 0.12n$)

Instance	Best value	Solution difference					
		MSA		HGA		VNS	
		Min	Ave	Min	Ave	Min	Ave
Silva_100_1	3136	13	20.7	0	1.9	2	15.0
Silva_100_2	3215	0	10.3	0	11.3	0	9.5
Silva_100_3	3175	1	13.4	0	3.5	0	11.6
Silva_100_4	3171	6	10.5	0	3.7	0	8.1
Silva_200_1	12167	30	63.1	0	17.8	44	78.5
Silva_200_2	12206	36	59.1	0	7.3	54	92.3
Silva_200_3	12151	34	51.8	0	8.1	43	88.4
Silva_200_4	12282	20	51.6	0	19.3	21	84.2
Silva_300_1	26604	102	136.7	0	39.2	102	193.9
Silva_300_2	26550	87	132.6	0	52.9	157	218.5
Silva_300_3	26565	58	105.3	0	32.4	111	209.9
Silva_300_4	26682	85	109.7	0	31.1	163	223.8
Silva_400_1	46266	170	267.3	0	71.3	332	454.0
Silva_400_2	46295	52	141.1	0	19.7	240	350.5
Silva_400_3	46272	127	214.5	0	53.2	305	400.8
Silva_400_4	46196	142	197.5	0	59.9	219	389.8
Silva_500_1	70955	265	348.5	0	123.2	339	523.1
Silva_500_2	71109	247	291.0	0	83.5	306	523.6
Silva_500_3	71092	208	292.2	0	96.1	351	458.9
Silva_500_4	71027	240	285.2	0	86.3	363	533.3
Average		96.1	140.1	0	41.1	157.6	243.4

Table 2. Results of running MSA, HGA and VNS on the *Duarte-Martí* instances ($n = 2000$, $m = 50$, $a = 32$, $b = 48$)

Instance	Best value	Solution difference					
		MSA		HGA		VNS	
		Min	Ave	Min	Ave	Min	Ave
DM_1	269990	1160	1424.4	2665	2823.7	0	588.8
DM_2	270409	1509	1877.9	3060	3246.3	0	908.1
DM_3	269710	768	1121.7	2382	2559.6	0	460.1
DM_4	269790	430	1135.5	2053	2507.4	0	644.1
DM_5	269675	495	1175.1	1983	2455.8	0	365.7
DM_6	269933	877	1225.7	2537	2772.8	0	553.5
DM_7	269781	952	1275.8	2405	2705.8	0	558.9
DM_8	270325	1535	1833.4	2718	3050.6	0	748.5
DM_9	269830	866	1209.8	2357	2565.5	0	535.1
DM_10	269892	1106	1417.9	2447	2643.3	0	547.4
Average		969.8	1369.7	2460.7	2733.1	0	591.0

ber of offspring, but each of them is worse than its own parents. This is because the parents are sampled from the initial population which is built using the local search procedure. The mating mechanism (Step 3 in the description of HGA) is too weak to be able to produce an offspring that could outperform the fitness of its parents. Thus the performance of the pure GA is poor compared with the described MSA, HGA and VNS implementations.

6. Conclusions

In this paper we have presented simulated annealing (MSA), hybrid genetic (HGA) and variable neighborhood search (VNS) algorithms for the maximally diverse grouping problem. In order to evaluate the performance of these algorithms we conducted computational experiments on two sets of problem instances of size up to 2000 elements. The results show that neither of the algorithms is a clear winner in all cases. For smaller instances, the best results were achieved using HGA. However, the comparison of the heuristics on larger MDGP instances favors the VNS algorithm. In both cases, MSA is the second best method. Certainly some more experiments, especially by varying group size, could be run to better evaluate the potential of various approaches.

In an additional experiment, we let HGA and VNS run for longer on a number of problem instances in the datasets we have used. The result was that in all cases except for the *Silva_100_2* and *Silva_100_3* instances the solutions obtained were better than those reported in Tables 1 and 2. This means that there is some room for improvements and further research on the heuristics for solving the MDGP. For example, different mating mechanism in HGA and various local search procedures both in HGA and VNS could be tried. Also, the development of new algorithms for the MDGP, based on other metaheuristics than those considered in this paper, is an important line of future work.

References

[1] **T. Arani, V. Lotfi**. A three phased approach to final exam scheduling. *IIE Transactions*, 1989, *Vol*.21, 86–96.

[2] **K.R. Baker, S.G. Powell**. Methods for assigning students to groups: a study of alternative objective functions. *Journal of the Operational Research Society*, 2002, *Vol*.53, 397–404.

[3] **J. Brimberg, N. Mladenović, D. Urošević, E. Ngai**. Variable neighborhood search for the heaviest k-subgraph. *Computers and Operations Research*, 2009, *Vol*.36, 2885–2891.

[4] **Y. Chen, Z.-P. Fan, J. Ma, S. Zeng**. A hybrid grouping genetic algorithm for reviewer group construction problem. *Expert Systems with Applications*, 2011, *Vol*.38, 2401–2411.

[5] **A. Duarte, R. Martí**. Tabu search and GRASP for the maximum diversity problem. *European Journal of Operational Research*, 2007, *Vol*.178, 71–84.

[6] **Z.P. Fan, Y. Chen, J. Ma, S. Zeng**. A hybrid genetic algorithmic approach to the maximally diverse grouping problem. *Journal of the Operational Research Society*, 2011, *Vol*.62, 1423–1430.

[7] **P. Hansen, N. Mladenović**. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 2001, *Vol*.130, 449–467.

[8] **P. Hansen, N. Mladenović, J.A. Moreno Pérez**. Variable neighbourhood search: methods and applications. *4OR*, 2008, *Vol*.6, 319–360.

[9] **P. Hansen, N. Mladenović, J.A. Moreno Pérez**. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 2010, *Vol*.175, 367–407.

[10] **V. Lotfi, R. Cerveny**. A final-exam-scheduling package. *Journal of the Operational Research Society*, 1991, *Vol*.42, 205–216.

[11] **R. Martí, M. Gallego, A. Duarte**. Maximum diversity problem. http://www.optsicom.es/mdp/. Accessed 25 May 2011.

[12] **R. Martí, M. Gallego, A. Duarte, E.G. Pardo**. Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics*, 2011, in press, DOI: 10.1007/s10732-011-9172-4.

[13] **J. Mingers, F.A. O'Brien**. Creating student groups with similar characteristics: a heuristic approach. *Omega*, 1995, *Vol*.23, 313–321.

[14] **G. Palubeckis**. Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation*, 2007, *Vol*.189, 371–383.

[15] **G. Palubeckis**. A new bounding procedure and an improved exact algorithm for the Max-2-SAT problem. *Applied Mathematics and Computation*, 2009, *Vol*.215, 1106–1117.

[16] **G. Palubeckis, D. Rubliauskas, A. Targamadzė**. Metaheuristic approaches for the quadratic minimum spanning tree problem. *Information Technology and Control*, 2010, *Vol*.39, 257–268.

[17] **G.C. Silva, L.S. Ochi, S.L. Martins**. Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. *Lecture Notes in Computer Science*, 2004, *Vol*.3059, 498–512.

[18] **R.R. Weitz, M.T. Jelassi**. Assigning students to groups: a multi-criteria decision support system approach. *Decision Sciences*, 1992, *Vol*.23, 746–757.

[19] **R.R. Weitz, S. Lakshminarayanan**. On a heuristic for the final exam scheduling problem. *Journal of the Operational Research Society*, 1996, *Vol*.47, 599–600.

[20] **R.R. Weitz, S. Lakshminarayanan**. An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design, and exam scheduling. *Omega*, 1997, *Vol*.25, 473–482.

[21] **R.R. Weitz, S. Lakshminarayanan**. An empirical comparison of heuristic methods for creating maximally diverse groups. *Journal of the Operational Research Society*, 1998, *Vol*.49, 635–646.

[22] **H.K. Yeoh, M.I.M. Nor**. An algorithm to form balanced and diverse groups of students. *Computer Applications in Engineering Education*, 2011, *Vol*.19, 582–590.

Received June 2011.