Journal of the Operational Research Society (2015) 66, 579–592

www.palgrave-journals.com/jors/

Maximally diverse grouping: an iterated tabu search approach

Gintaras Palubeckis*, Armantas Ostreika and Dalius Rubliauskas

Kaunas University of Technology, Kaunas, Lithuania

The maximally diverse grouping problem (MDGP) consists of finding a partition of a set of elements into a given number of mutually disjoint groups, while respecting the requirements of group size constraints and diversity. In this paper, we propose an iterated tabu search (ITS) algorithm for solving this problem. We report computational results on three sets of benchmark MDGP instances of size up to 960 elements and provide comparisons of ITS to five state-of-the-art heuristic methods from the literature. The results demonstrate the superiority of the ITS algorithm over alternative approaches. The source code of the algorithm is available for free download via the internet.

Journal of the Operational Research Society (2015) **66**(4), 579–592. doi:10.1057/jors.2014.23

Published online 26 March 2014

Keywords: optimization; maximally diverse grouping; metaheuristics; tabu search

Introduction

A problem that arises in many applications is that of finding a partition of a set of elements into a given number of mutually disjoint groups, while respecting the requirements of group size constraints and diversity. In the literature, it is called the *maximally diverse grouping problem* (MDGP) (Fan *et al*, 2011, Gallego *et al*, 2013). Mathematically, the MDGP can be formulated as follows. Let $\mathbf{D}=(d_{ij})$ be an $n\times n$ symmetric matrix representing the dissimilarities between n elements to be grouped. Let the set of elements be denoted by V. Each element from V needs to be assigned to one of m groups. The latter are restricted in size: for $k\in\{1,\ldots,m\}$, the k-th group must contain at least a_k and at most b_k elements. By introducing binary variables x_{ik} to indicate whether element $i\in V$ is assigned to group k, the MDGP can be stated in the following form:

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

$$\text{subject to} \quad \sum_{k=1}^{m} x_{ik}=1,\ i=1,\ldots,n, \tag{2}$$

$$a_k\leqslant\sum_{i=1}^{n} x_{ik}\leqslant b_k,\ k=1,\ldots,m, \tag{3}$$

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

The objective function of this binary program captures the dissimilarity between all pairs of elements that belong to the same group. Constraints (2) ensure that each element will be assigned to a unique group. Constraints (3) are imposed to restrict the size of groups. An instance of the MDGP is defined by a matrix $\mathbf{D}$ and vectors $(a_1,\ldots,a_m)$ and $(b_1,\ldots,b_m)$. It is assumed that the dissimilarity matrix $\mathbf{D}$ has zero main diagonal and non-negative off-diagonal entries. The MDGP takes a slightly simpler form when $n\equiv 0(\text{mod } m)$ and $a_k=b_k=n/m$ for all k in (3). Yet another variation of the problem is the case where $a_k=b_k$ for $k=1,\ldots,m$, but not necessarily $a_k=a_l$ for $k,l\in\{1,\ldots,m\}$.

The MDGP models a variety of real-world situations, perhaps the best known of these being in academic or training settings where the task is to create diverse student work-groups or training teams (Weitz and Jelassi, 1992; Weitz and Lakshminarayanan, 1997, 1998; Baker and Powell, 2002; Yeoh and Nor, 2011). In this case, V stands for the set of students. The computation of the dissimilarity value d_{ij} for students i and j depends on the requirements of the application as well as the data used. Typically, d_{ij} is obtained by additively combining the values of a number of attributes (see, for example, Weitz and Jelassi (1992) for details). Lotfi and Cerveny (1991) considered the MDGP in one of the stages of their approach for scheduling final exams at a large university. Specifically, in that stage, exam blocks are assigned to days while minimizing the number of students with two or more exams per day. The entry d_{ij} of the matrix $\mathbf{D}$ is the number of students with exams in both blocks i and j. The resulting problem is the minimization version of (1)–(4). By performing a simple transformation, it can be reduced to the MDGP. Chen *et al* (2011) applied the MDGP as a suitable model for the construction of reviewer

*Correspondence: Gintaras Palubeckis, Faculty of Informatics, Kaunas University of Technology, Studentu 50-408, Kaunas, 51368, Lithuania.
E-mail: gintaras@soften.ktu.lt

groups. They proposed to build the dissimilarity matrix **D** by using the values of a set of attributes describing the reviewers. The MDGP also arises in the context of Very Large Scale Integration (VLSI) design (Li and Behjat, 2006; Areibi *et al*, 2007). In this application, the purpose is to group highly interconnected cells into disjoint clusters. Bhadury *et al* (2000) studied the problem of assigning employees to workgroups so as to maximize the groups' diversity. They proposed a somewhat simplified model and companion algorithm to deal with it. However, this problem can also be formulated as the binary program (1)–(4) and thus is of the type we consider in this paper.

The MDGP can be considered as a generalization of the maximum diversity problem (MDP) (Palubeckis, 2007; Aringhieri and Cordone, 2011; Martí *et al*, 2013). The latter asks to select a specified number of elements from a given set so that the sum of dissimilarities between the selected elements is as large as possible. Indeed, the MDP is obtained from (1)–(4) by setting $m = 1$, removing constraints (2) and assuming equalities in (3). Certainly, in this case, the index k can be dropped everywhere. Computational experience shows that the MDGP is a substantially more difficult problem than the MDP (Palubeckis, 2007; Palubeckis *et al*, 2011; Martí *et al*, 2013). An important special case of the MDGP is the K_r-factor problem (Guruswami *et al*, 2001), which asks for the existence of a partition of the vertex set of a graph into copies of the complete graph on r vertices, denoted K_r. For this case, the matrix **D** is binary and $a_k = b_k = r$ in (3) for all k. If a K_r-factor does not exist, then one can consider the K_r-packing problem, which is to find the maximum number of mutually vertex-disjoint copies of K_r in a given graph. This problem is not a special case of the MDGP, though both are similar in nature.

The MDGP is an NP-hard problem as it contains the K_r-factor problem as a special case, which is known to be NP-complete for $r \geqslant 3$ (Kirkpatrick and Hell, 1983). In fact, the MDGP is difficult to solve not only theoretically, but also practically. As Gallego *et al* (2013) report, using the commercial solver CPLEX, they were able to solve to guaranteed optimality only instances of size 10 and 12. All attempts to solve instances with 30 elements were consistently unsuccessful. This shows that efficient heuristic algorithms, which provide good but not necessarily optimal solutions, are required. Traditional approaches to combinatorial optimization include construction heuristics and iterative improvement heuristics. One of the first construction heuristics for the MDGP was developed by Weitz and Jelassi (1992). In their algorithm, the groups are considered in a fixed order. Initially, all the elements are unmarked. At each step, the algorithm selects an unmarked element that is least different from the last marked one and assigns it to the next group in the list. The step ends by marking the selected element as already assigned. Another construction heuristic for the MDGP was proposed by Mingers and O'Brien (1995). In their approach, the elements are assigned to groups one by one in a greedy fashion. However, it is widely acknowledged that construction heuristics are not able to produce solutions of high quality. They can be applied in situations where computation time is a critical factor.

The other way to approach the problem is to use iterative improvement algorithms. Lotfi and Cerveny (1991) presented an algorithm of this type for the MDGP. It starts with an arbitrary initial solution and repeats a sequence of steps in which the pairwise interchange neighbourhood of the current solution is searched. While doing this, the algorithm runs through the set of elements and, for each of them, finds a group that is most likely to contain it. Then the algorithm determines which element in this group should be interchanged with the examined element. The interchange is accepted if and only if it results in an improvement of the objective function value. Weitz and Lakshminarayanan (1996) have remarked that the Lotfi–Cerveny algorithm at each step explores only a portion of the pairwise interchange neighbourhood. They proposed a variation of this algorithm, which checks all pairwise interchanges of elements belonging to different groups (see also Weitz and Lakshminarayanan, 1998). Arani and Lotfi (1989) developed another improvement heuristic for solving the MDGP. At each iteration, their algorithm first removes m elements, one from each group, and then reassigns them optimally by solving a linear assignment problem of size m. The algorithm terminates when no element is moved from its current group to another one in a full cycle of iterations (assignments). Fan *et al* (2011) proposed a hybrid genetic algorithm (HGA) incorporating a local search (LS) procedure. The encoding scheme in their algorithm comprises information about both the elements and the groups. Parents for crossover operation are selected using a rank-based roulette-wheel strategy. The crossover operation is similar to that described by Falkenauer (1998). To make the generated offspring legal, a group size adjustment procedure is applied. Recently, Gallego *et al* (2013) proposed tabu search (TS) with strategic oscillation for solving the MDGP. Their algorithm repetitively goes through two steps. In the first of them, the algorithm applies an improvement method to intensify the search. If the resulting solution is infeasible, then, in the second step, a mechanism to repair the solution is used. Such a strategy enables search paths to cross a feasibility boundary. The best results were obtained when an improvement method based on TS was employed.

The current paper is an important extension of our recent work (Palubeckis *et al*, 2011) on metaheuristics for solving the MDGP. The previous study by Palubeckis *et al* (2011) has investigated computationally the applicability of three quite different metaheuristics to the MDGP, namely, simulated annealing (SA), HGA and variable neighbourhood search (VNS). Since the CPU time-based stopping criterion was used, the SA procedure was run in a multistart mode. However, through experimentation we have experienced that all of these three methods have some limitations. In particular, the weakness of SA for the MDGP is that it is frequently trapped into local maxima of rather poor quality. Certainly, by applying a slower cooling schedule better solutions can be obtained. However,

such a strategy leads to a significant increase in computation time. Moreover, the choice of SA parameters is a very delicate issue. One set of parameter values give best results for small MDGP instances and quite a different set of parameter values should be used for large instances. The main drawback of the developed HGA is slow convergence. This algorithm provided good results for small (up to size 500) MDGP instances; however, its performance decreased dramatically for instances of size 2000. Obtaining good solutions with HGA for such instances is unduly time consuming. As expected, the performance of the pure genetic algorithm, that is HGA without the LS component, is very poor compared with the proposed HGA implementation. The third algorithm considered by Palubeckis *et al* (2011) is based on the VNS scheme. As the experiments show, for the largest problem instances in the test set, this algorithm performed better than both SA and HGA. However, it was found that the VNS intensification phase, applying the LS heuristic, is a little bit too weak to produce solutions of high quality. In order to reach such solutions, the VNS technique for instances of size 2000 required very long computation times. In general, it was discovered from the experiments that all the methods investigated by Palubeckis *et al* (2011) did not perform very well for large problem instances. The authors have concluded that there remains a lot of room for improvement in the heuristics for solving the MDGP. The goal is to devise algorithms for the MDGP that would be able to produce close-to-optimal solutions in a relatively short time. In the pursuit of faster algorithms for the problem, we made a decision to try the TS metaheuristic. We were motivated by the fact that the TS method provides a better search intensification strategy than the LS procedure used in HGA and VNS techniques. On the other hand, we believe that a successful algorithm should also incorporate a sufficiently powerful search diversification component. Integrating such a component with TS procedure leads to an approach called *iterated tabu search* (ITS) method. Another argument that motivated us to choose the TS scheme is an excellent performance of ITS for the MDP (Palubeckis, 2007), which, as alluded to earlier, is a simplification of the MDGP we deal with in this paper. The developed ITS algorithm for the MDGP is described in the next section. The algorithm was experimentally compared with several recent techniques from the literature. The empirical results were obtained by testing the algorithms on the MDGP instances used by Gallego *et al* (2013). The paper is ended with the concluding remarks section.

Before closing this introduction, let us give a couple of notations used in the rest of the paper. We denote by $P=(V_1, \ldots, V_m)$ a feasible solution to the problem. Thus P is a partition of the set V into mutually disjoint subsets V_k, $k=1, \ldots, m$, satisfying the size constraints. Referring to (1)–(4), $V_k=\{i \in V \mid x_{ik}=1\}$, $k=1, \ldots, m$. The subsets in the partition will be called groups. We will use Π to denote the set of all feasible partitions. We will write $F(P)$ for the value of the objective function (1) for a solution P. With these notations, the problem (1)–(4) can be reformulated as $\max_{P \in \Pi} F(P)$.

The algorithm

The basic components of the ITS algorithm are TS and solution perturbation procedures. The first of them is enhanced with the integration of a LS technique. Good-quality solutions are obtained by applying these procedures repeatedly. Implementation of the algorithm can be broken into the following four steps.

ITS

1. Generate a feasible solution P to the given instance of the MDGP at random. Set $P^*:=P$ and $F^*:=F(P)$.
2. Apply the tabu search procedure TS(P, P^*, F^*).
3. Check if a specified termination condition is met. If yes, then stop with the partition P^* of value F^*. If not, then proceed to 4.
4. Choose the values of the solution perturbation parameters $\overline{\alpha}$ and $\overline{\beta}$ and make a call to the procedure GSP$\left(P, \overline{\alpha}, \overline{\beta}\right)$. Go to 2.

The algorithm starts by generating a random initial solution to the problem. This is accomplished by first randomly generating a permutation of elements. Then a solution is constructed in two passes over the groups, each of which initially is empty. In the first pass, the algorithm assigns the minimum required number of elements to each of the groups. To this end, the first a_1 elements of the permutation are assigned to the first group, the next a_2 elements are assigned to the second group, and so on. If $a:=\sum_{k=1}^{m} a_k < n$, then the second pass needs to be performed. The last $n-a$ elements of the permutation are distributed among groups in a similar fashion as in the first pass of the generator.

In the description of the algorithm, P^* and F^* are used to denote the best found solution and its value, respectively. At the beginning, P^* is initialized to a randomly generated solution. In order to improve P^*, the TS procedure is run multiple times with different starting partitions. In fact, running TS in Step 2 of ITS may be viewed as a search intensification phase of the algorithm. A detailed description of the TS procedure is presented later in this section. Step 3 of ITS requires a stopping rule to be specified. The rule used in our implementation is to stop the algorithm after a prescribed time period has elapsed. In Step 4 of ITS, a solution perturbation procedure, named Get Start Partition (GSP), is invoked. This procedure has to be regarded as a search diversification mechanism embedded in the ITS algorithm. The goal of GSP is to generate a new starting solution for TS. This process is controlled by perturbation parameters $\overline{\alpha}$ and $\overline{\beta}$. The meaning of these parameters is explained in the second part of this section, where also a description of the GSP procedure is given.

An important feature of our algorithm is that the current solution is kept feasible throughout the search process. In other words, the size of each group $V_k, k \in \{1, \ldots, m\}$, in the current partition is forced to belong to the interval $[a_k, b_k]$. This allows us to avoid using special procedures to repair infeasible solutions.

As it is typical in the development of LS-based algorithms for combinatorial optimization problems, we have to choose an appropriate neighbourhood structure to explore the solution space. Our approach uses the two smallest neighbourhoods, the *relocation neighbourhood* $N_1(P)$ and the *pairwise interchange* (or *2-interchange*) neighbourhood $N_2(P)$ (here P is a solution to the MDGP). To define them, let $g(i,P)$ denote the group in P the element i belongs to. Then $N_1(P)=\{P'\in\Pi \mid \exists\, i\in V, g(i,P')\neq g(i,P),$ and $g(l,P')=g(l,P), l=1,\ldots,n, l\neq i\}$, $N_2(P)=\{P'\in\Pi \mid \exists\, i,j\in V,\ g(i,P)\neq g(j,P),\ g(i,P')=g(j,P), g(j,P')=g(i,P),$ and $g(l,P')=g(l,P), l=1,\ldots,n, l\neq i,j\}$. As we will see in the next subsections, exploration of the neighbourhoods N_1 and N_2 is a key task in all search procedures employed in our approach.

To continue with the description of the algorithm, we further need two more notations. Suppose $i\in V$ and $r\in\{1,\ldots,m\}$ are such that $|V_{g(i,P)}|>a_{g(i,P)}$ and $|V_r|<b_r$. Then we denote by $P_r(i)$ the solution obtained from P by relocating the element i from the group $V_{g(i,P)}$ to the group V_r. Similarly, assuming $g(i,P)\neq g(j,P)$ for $i,j\in V$, we denote by $P(i,j)$ the solution constructed from P by interchanging the elements i and j. Certainly, the solutions $P_r(i)$ and $P(i,j)$ are the members of the neighbourhoods $N_1(P)$ and $N_2(P)$, respectively.

Tabu search

Throughout the search process, the algorithm compares the differences between the values of the objective function at the solutions in the neighbourhoods $N_1(P)$ and $N_2(P)$ and the value of the objective function at the current solution P. These differences can be efficiently computed using an auxiliary $n\times m$ matrix $\mathbf{C}=(c_{ik})$, where $c_{ik}=\sum_{j\in V_k} d_{ij}$. The entry c_{ik} of $\mathbf{C}$ represents the sum of the dissimilarities between the element i and all elements in the group V_k. Given a solution $P\in\Pi$, let $\Delta(P,i,j)$ (respectively, $\delta(P,i,r)$) denote the change in the value of the objective function caused by interchanging the elements $i\in V_q$ and $j\in V_k, k\neq q$ (respectively, by relocating the element i from the group V_q to the group $V_r, r\neq q$). We can express $\Delta(P,i,j)$ in terms of the entries of the matrices $\mathbf{C}$ and $\mathbf{D}$

$$\Delta(P,i,j) = F(P(i,j)) - F(P) = c_{ik} - c_{iq} + c_{jq} - c_{jk} - 2d_{ij}. \quad (5)$$

Analogously, we can write

$$\delta(P,i,r) = F(P_r(i)) - F(P) = c_{ir} - c_{iq}. \quad (6)$$

Thus both $\Delta(P,i,j)$ and $\delta(P,i,r)$ can be computed in constant time. Consequently, the time complexity of exploring the neighbourhoods N_1 and N_2 is $O(nm)$ and $O(n^2)$, respectively. After interchanging the elements i and j, the matrix $\mathbf{C}$ is updated according to the following formulas:

$$c_{lq} := c_{lq} + d_{lj} - d_{li},\ c_{lk} := c_{lk} + d_{li} - d_{lj},\ l\in V\setminus\{i,j\}, \quad (7)$$

$$c_{iq} := c_{iq} + d_{ij},\ c_{ik} := c_{ik} - d_{ij}, \quad (8)$$

$$c_{jq} := c_{jq} - d_{ij},\ c_{jk} := c_{jk} + d_{ij}. \quad (9)$$

Similarly, after relocating the element i from the group V_q to the group V_r, the following calculations are done:

$$c_{lq} := c_{lq} - d_{li}, c_{lr} := c_{lr} + d_{li}, l\in V\setminus\{i\}. \quad (10)$$

The TS procedure for the MDGP consists of repeatedly performing a move, that is, either an interchange or a relocation operation. The reverse moves are forbidden for $\overline{\tau}$ iterations to avoid cycling. The parameter $\overline{\tau}$ is called *tabu tenure*. In our implementation of TS, it is a constant (except the cases of very small n). Another parameter of TS is the number of iterations $\overline{K}$. The TS procedure can be stated as follows.

$$\mathrm{TS}(P,P^*,F^*)$$

1. Initialize τ_{ij}, $i,j=1,\ldots,n$, and τ'_{ir}, $i=1,\ldots,n$, $r=1,\ldots,m$, with 0. Set $K:=0$ and $f:=F(P)$.
2. Increment K by 1. Set $\tilde{\Delta}:=F^*-f$, $h:=-\infty$, $\rho:=0$ and $i:=1$.
3. Iterating through all elements $j>i$ such that $g(j,P)\neq g(i,P)$, perform the following steps.
   - 3.1. Compute $\Delta(P,i,j)$ by Equation (5). If $\Delta(P,i,j)>\tilde{\Delta}$, then proceed to 3.2. Otherwise, check whether $\rho=0$ and $\tau_{ij}=0$. If so, go to 3.3; if not, go to 3.4.
   - 3.2. If $\rho=0$, then set $h:=\Delta(P,i,j)$, $k:=i$, $l:=j$ and $\rho:=1$. Otherwise, increment ρ by 1 and set $h:=\Delta(P,i,j)$, $k:=i$, $l:=j$ with probability $1/\rho$. In both cases, go to 3.4.
   - 3.3. If $\Delta(P,i,j)>h$, then set $h:=\Delta(P,i,j)$, $k:=i$, $l:=j$ and $z:=1$. Otherwise, check whether $\Delta(P,i,j)=h$. If so, then increment z by 1 and set $k:=i$, $l:=j$ with probability $1/z$.
   - 3.4. Repeat 3.1–3.3 for the next element j, if any.
4. If $|V_{g(i,P)}|=a_{g(i,P)}$, then go to 5. Otherwise, iterating through all groups $r\neq g(i,P)$ such that $|V_r|<b_r$, perform the following steps.
   - 4.1. Compute $\delta(P,i,r)$ by Equation (6). If $\delta(P,i,r)>\tilde{\Delta}$, then proceed to 4.2. Otherwise, check whether $\rho=0$ and $\tau'_{ir}=0$. If so, go to 4.3; if not, go to 4.4.
   - 4.2. If $\rho=0$, then set $h:=\delta(P,i,r)$, $k:=i$, $l:=-1$, $q:=r$ and $\rho:=1$. Otherwise, increment ρ by 1 and set $h:=\delta(P,i,r)$, $k:=i$, $l:=-1$, $q:=r$ with probability $1/\rho$. In both cases, go to 4.4.
   - 4.3. If $\delta(P,i,r)>h$, then set $h:=\delta(P,i,r)$, $k:=i$, $l:=-1$, $q:=r$ and $z:=1$. Otherwise, check whether $\delta(P,i,r)=h$. If so, then increment z by 1 and set $k:=i$, $l:=-1$, $q:=r$ with probability $1/z$.
   - 4.4. Repeat 4.1–4.3 for the next group r, if any.
5. Increment i by 1. If $i\leqslant n$, then go to 3.
6. If $l<0$, then save $g(k,P)$ as g'. Let $\tilde{P}=P(k,l)$ if $l>0$ and $\tilde{P}=P_q(k)$ if $l<0$. Set $P:=\tilde{P}$ and $f:=f+h$. If $\rho>0$, then proceed to 7. Otherwise, go to 8.

7. Apply the local search procedure LS(P). Let P' denote the partition returned by it. Set $P^* := P'$, $F^* := F(P')$, $P := P'$ and $f := F^*$.
8. If $K = \overline{K}$, then return. Otherwise, decrement each positive τ_{ij}, $i, j = 1, \ldots, n$, as well as each positive τ'_{ir}, $i = 1, \ldots, n$, $r = 1, \ldots, m$, by 1. If $l > 0$, then set $\tau_{kl} := \overline{\tau}$; else set $\tau'_{kg'} := \overline{\tau}$. Go to 2.

In the above description, τ_{ij} and τ'_{ir} denote the tabu values of interchanges and relocations, respectively. If $\tau'_{ir} > 0$, then adding element i to group r is forbidden. The usage of τ_{ij} is similar to τ'_{ir}. The other variables introduced in Step 1 of TS are the iteration counter K and the value f of the current solution P. Additional variables involved in TS include $\tilde{\Delta}$, h and ρ. They are reinitialized each time the iteration is repeated. In the algorithm, h is used to denote the change in the objective function value calculated for a move being selected, ρ serves as a flag to indicate that a new best solution in the search is found, and $\tilde{\Delta}$ is the target value for both Δ and δ. The search for the best solution in the neighbourhoods N_1 and N_2 is organized in n rounds by performing Steps 3 and 4 for each element $i \in V$. In Step 3, the neighbourhood N_2 is explored by examining pairs of the form (i, j), where $j > i$ is an element belonging to a group different from the one containing the element i. The fulfillment of the condition $\Delta(P, i, j) > \tilde{\Delta}$ means that the algorithm has found a solution improving the best partition P^* available at the beginning of the current iteration. If two or more such solutions are found, then one of them is selected in a random fashion. Such a strategy helps to diversify the search. Of course, when looking for an improving solution, the tabu status is ignored. A non-improving move (interchange in Step 3.3 or relocation in Step 4.3) is considered only if both the corresponding tabu value and the flag ρ are equal to zero. A move is chosen according to the maximum of increase in the objective function value, breaking ties randomly if they occur. In Step 4 of TS, the neighbourhood N_1 is explored provided the size of the group containing the element i is greater than the minimum allowed size. The δ value is computed only for groups that can accommodate at least one additional element. Steps 4.1, 4.2 and 4.3 are similar to Steps 3.1, 3.2 and 3.3, respectively. Step 5 initiates the next round of the search. Upon termination of the loop over all elements in V, the algorithm proceeds to Step 6. The move to be performed is either the interchange of the elements k and l or the relocation of the element k from its current group to the group V_q. In the latter case, l is set to -1. This allows the algorithm to act accordingly in Step 6. In particular, if $l < 0$, then the group currently owning the element k is saved as g' for future reference. This is needed to temporarily disable the possibility of moving the element k backwards from its new group to the old one (see Step 8). In Step 6, the current solution P is replaced with either $P(k, l)$ or $P_q(k)$, depending on which type of move has been selected. Also, the matrix $\mathbf{C}$ is updated using either (7)–(9) or (10). Then the algorithm checks if $\rho > 0$. Recall that positive ρ indicates the fact that a new best solution in the ITS run has been found. Before saving this solution, an attempt is made to improve it locally by applying the LS procedure. It returns a solution that, if different from P, becomes the new current one. The TS procedure stops after $\overline{K}$ iterations. If, however, $K < \overline{K}$, then, in Step 8, the tabu values are updated. The reverse of the last performed move is forbidden for the next $\overline{\tau}$ iterations.

It should be noted that the described TS procedure can be simplified when dealing with a special case of the MDGP in which the size of each group is fixed. In this case, Step 4 can be removed from TS. Moreover, the tabu values τ'_{ir}, $i = 1, \ldots, n$, $r = 1, \ldots, m$, are not used and thus can be eliminated.

Our implementation of the LS algorithm for the MDGP is based on interchanging and relocating elements. It terminates at a solution that is locally optimal with respect to both the neighbourhood N_1 and the neighbourhood N_2. The procedure LS was already used by Palubeckis *et al* (2011). Therefore, to save space, we only outline it in brief. The procedure works by evaluating the neighbours of the current solution P. To choose a move, it applies a first-improvement search strategy. In other words, it accepts the first solution discovered in the neighbourhoods N_1 and N_2 that demonstrates an improvement. If such a solution is found, it replaces the current solution P and then starts a new iteration. If, however, all the neighbours of P are rejected, then LS will halt at a local maximum. The TS heuristic makes multiple calls to the procedure LS. Because of this, we decided to introduce some randomization into the LS process. Specifically, in the initialization phase of LS, a permutation of elements and that of groups are randomly generated. The first of them defines the order in which the pairwise interchanges of elements are checked. The second permutation is used when examining the relocation neighbourhood N_1.

Solution perturbation

An important ingredient of the ITS algorithm is a solution perturbation procedure. Its task is to direct the search towards unexplored regions of the solution space. The procedure, named GSP, is applied to partition P that existed at the end of the most recent run of TS. Besides P, the input to GSP includes several parameters. The parameter $\overline{\alpha}$ is the number of elements that have to be moved from their current groups in P to different ones. To generate its value in Step 4 of ITS, the algorithm uses three secondary parameters, λ_1, λ_2 and α_0. First, it chooses uniformly at random an integer α' in $[\lfloor \lambda_1 n \rfloor, \lfloor \lambda_2 n \rfloor]$ and, if $\alpha' > \alpha_0$, then uniformly and randomly chooses an integer $\overline{\alpha}$ in $[\alpha_0, \alpha']$. If, however, $\alpha' \leqslant \alpha_0$, then $\overline{\alpha}$ is set to α'. Thus, generally, $\overline{\alpha}$ is drawn from the interval whose right endpoint is not fixed throughout the search process. The precise values of λ_1, λ_2 and α_0 should be selected experimentally. The parameter $\overline{\beta}$ is the upper bound on the size of the list containing the best moves. We call this list the candidate list. In Step 4 of ITS, the value assigned to $\overline{\beta}$ is an integer randomly drawn from the interval $[\mu_1, \mu_2]$ whose endpoints are constants. The role of the parameters λ_1, λ_2, α_0, μ_1 and μ_2 is to guide the diversification of the search. In order to correctly generate $\overline{\alpha}$, the parameter λ_1 must

be a positive number less than λ_2, and the latter must be less than or equal to 1. It is expected (and verified experimentally) that choosing values of λ_1 and λ_2 close to any one of the endpoints of the interval [0, 1] has a negative effect on the performance of the ITS algorithm. If both λ_1 and λ_2 are close to zero, then GSP may generate a solution that is close to the input partition P. The TS procedure applied to this solution may have less chances to improve the best partition, P^*, found so far. On the other hand, if λ_1 and λ_2 are close to one, then GSP tends to generate starting solutions of low quality. In this case, TS works similarly to how it does when applied to random starting points. The parameter α_0 has the same effect as that of λ_1 and λ_2. Both large and very small values of α_0 have a negative impact on the performance of ITS. Like λ_1, λ_2 and α_0, the parameters μ_1 and μ_2 influence the level of diversification of the search as well. With larger values of μ_1 and μ_2, more diverse solutions are generated. However, μ_1 and μ_2 also have a different effect on the solutions produced by GSP. Basically, these parameters are used to control the level of degradation of the objective function value due to the perturbation of the partition P. The quality of generated solutions deteriorates with the increase of μ_1 and μ_2. Thus, these parameters ensure a certain balance between search diversification and solution quality degradation.

Possible moves in the algorithm are represented by triplets having one of the two forms: $(i, j, 0)$ to denote the case of interchanging the elements i and j, and $(i, r, 1)$ to denote the case when the element i is moved from its current group to the group V_r. The last component of the triplet helps to distinguish between two types of move. The GSP procedure also uses a $0-1$ flag γ, which is assigned a value depending on the variation of the problem being solved: $\gamma = 0$ if $a_r = b_r$ for all $r = 1, \ldots, m$; $\gamma = 1$ otherwise. If $\gamma = 1$, then GSP provides an option to consider only relocation moves when constructing the candidate list of possible actions. The choice of this option is controlled by the probability parameter Q. If, at the beginning of an iteration, a randomly generated number between 0 and 1 is less than or equal to Q, then exploration of the neighbourhood N_2 is simply bypassed and a relocation move is selected. Unlike pairwise interchanges, relocations of elements affect the sizes of groups. The purpose behind the introduction of the parameter Q is to achieve more diversity in the solutions generated, by forcing the sizes of groups to change more frequently during perturbation. If $Q = 1$ and the sizes of groups are allowed to vary, that is, $\gamma = 1$, then interchange moves are not used at all, and the partition P is perturbed by performing exclusively relocation moves. The solution perturbation procedure goes as follows.

$$GSP(P, \overline{\alpha}, \overline{\beta}).$$

1. Set $W := \varnothing$.
2. Set $A := V \backslash W$ and $T := \varnothing$. If $\gamma = 1$, then with probability Q set $R := 1$. Otherwise, including the case where $\gamma = 0$, keep $R = 0$.
3. Choose any element i of A. Remove it from A. If $R = 1$, go to 5; else proceed to 4.
4. Iterating through all elements $j \in A$ such that $g(j, P) \neq g(i, P)$, perform the following steps.
   4.1. Compute $\Delta(P, i, j)$. If $|T| < \overline{\beta}$, then go to 4.3. Otherwise, proceed to 4.2.
   4.2. Choose a triplet $t' \in T$ such that $h(t') \leqslant h(t)$ for all $t \in T$. If $\Delta(P, i, j) > h(t')$, then remove t' from T and go to 4.3. Otherwise, repeat from 4.1 for the next element j.
   4.3. Create a triplet $t'' = (i, j, 0)$ with $h(t'') = \Delta(P, i, j)$ attached to it. Add t'' to T.
5. If $|V_{g(i,P)}| = a_{g(i,P)}$, then go to 6. Otherwise, iterating through all groups $r \neq g(i, P)$ such that $|V_r| < b_r$, perform the following steps.
   5.1. Compute $\delta(P, i, r)$. If $|T| < \overline{\beta}$, then go to 5.3. Otherwise, proceed to 5.2.
   5.2. Choose a triplet $t' \in T$ such that $h(t') \leqslant h(t)$ for all $t \in T$. If $\delta(P, i, r) > h(t')$, then remove t' from T and go to 5.3. Otherwise, repeat from 5.1 for the next group r.
   5.3. Create a triplet $t'' = (i, r, 1)$ with $h(t'') = \delta(P, i, r)$ attached to it. Add t'' to T.
6. If the set A is empty, then proceed to 7. Otherwise, go to 3.
7. If the set T is non-empty, then proceed to 8. Otherwise, if $R = 0$, then return with P; else go to 2.
8. Select a triplet t from T at random. If t is of the form $(k, l, 0)$, then set $P = P(k, l)$ and add both k and l to W. Otherwise, t has the form $(k, q, 1)$. In this case, set $P = P_q(k)$ and add k to W. If $|W| < \overline{\alpha}$, then go to 2. Otherwise, return with the partition P.

In the procedure, W stands for the set of elements that were involved either in the interchange or relocating operation. The complement of the set W with respect to V is denoted by A. During the run of GSP, each element is allowed to be moved from one group to another at most once. In Step 2, the move selection process is initialized. Depending on the γ and Q values, one of the following two scenarios for execution of an iteration is chosen: both the neighbourhoods $N_1(P)$ and $N_2(P)$ are explored (choice indicator R takes value 0); the search is restricted to the neighbourhood $N_1(P)$ only (in this case $R = 1$). The candidate list in the description of GSP is denoted by T. The algorithm assigns to each move (triplet) t in T a measure of its attractiveness $h(t)$, which is calculated, depending on the type of move, using either (5) or (6). Generally, the loop comprising Steps 2–8 is repeated until the size of the set W reaches $\overline{\alpha}$. Premature exit from the loop is possible in the situation where $R = 0$ and the candidate list T is empty. This may happen only if $\overline{\alpha}$ is very close to n. The inner loop, indexed by i, runs until the set A is exhausted. In Step 4 of GSP, pairwise interchanges of elements are evaluated. If the size of the candidate list T equals $\overline{\beta}$, then the interchange of the elements i and j is compared with the worst move in T, denoted by t'. If the Δ value of the pair (i, j) is greater than $h(t')$, then the candidate list is updated by substituting t' with the triplet t'' representing the pairwise interchange of the elements i and j. If, however,

$|T|<\overline{\beta}$, then the new triplet is simply added to the list T. Step 5 of GSP evaluates the relocation moves. This is done in a manner similar to Step 4. In this case, a triplet of the second type is created, provided it is worth including into T. In Step 8, the move is performed for a randomly selected triplet from the list T. The elements involved in the move are added to the set W.

We notice that in the case of MDGP instances with fixed group size the presented GSP procedure can be made simpler. In particular, the variable R, the parameters γ and Q, and the entire Step 5 can be eliminated.

An interesting question is whether GSP can be embedded in other techniques for solving the MDGP. The VNS algorithm of Palubeckis *et al* (2011) is a possible candidate. Comparing the structure of ITS to that of VNS, we see that the counterpart of GSP in the VNS method is the shaking operation. By replacing this operation with GSP we can obtain a variation of the VNS heuristic. Some numerical results on this variation will be given in the next section.

Computational experiments

In this section, we report the results of an empirical evaluation of ITS and several state-of-the-art algorithms proposed recently. The algorithms used for comparison are multistart simulated annealing (MSA) (Palubeckis *et al*, 2011), HGA (Palubeckis *et al*, 2011), VNS (Palubeckis *et al*, 2011), T-LCW (Lotfi–Cerveny–Weitz heuristic coupled with TS) (Gallego *et al*, 2013) and SO (strategic oscillation with T-LCW improvements) (Gallego *et al*, 2013). We do not consider algorithms presented in older papers because they are outperformed by more advanced methods (see, for example, the study of Gallego *et al*, 2013). It should be noted that in Palubeckis *et al* (2011) the MSA, HGA and VNS algorithms were used to solve a variation of the MDGP in which $a_k = a$, $b_k = b$, $k = 1, \ldots, m$, for fixed positive integers a and $b \geqslant a$. We have modified the code slightly in order to be able to solve the more general case of the MDGP we address in this paper.

Experimental setup

The described algorithm has been coded in the C programming language. The same language was used in Palubeckis *et al* (2011) to implement MSA, HGA and VNS. The sources of these algorithms as well as ITS are publicly available at http://www.proin.ktu.lt/~gintaras/mdgp.html. The code of SO and T-LCW was obtained from Gallego *et al* (2011). All the tests have been carried out on a PC with an Intel Core 2 Duo CPU running at 3.0 GHz.

We performed computational experiments on the following three datasets introduced by Gallego *et al* (2013): *RanReal*, *RanInt* and *Geo*. The first two sets consist of matrices with entries being real and, respectively, integer numbers chosen randomly from a uniform distribution between 0 and 100. In the case of *Geo*, the (i, j)-th entry of the matrix $\mathbf{D}$ is calculated as Euclidean distance between points i and j with coordinates randomly drawn from the interval $[0, 10]$. The number of coordinates for each point is picked randomly from the range $[2, 21]$. We compared the algorithms on the instances of size 120, 240, 480 and 960, which are the largest instances in the *RanReal*, *RanInt* and *Geo* datasets. For each $n \in \{120, 240, 480, 960\}$ and each dataset, we used five instances with varying group size and five instances with fixed group size. In the former case, a_k and b_k, $k = 1, \ldots, m$, are drawn at random from the predefined intervals. In the latter case, $a_k = b_k = a$, $k = 1, \ldots, m$, and a is equal to 12, 20, 24 and 40 for $n = 120$, 240, 480 and 960, respectively. In both cases, the number of groups m for these n values is fixed, respectively, at 10, 12, 20 and 24. For more details on the *RanReal*, *RanInt* and *Geo* data series, see Gallego *et al* (2013).

The ITS algorithm described in the previous section has several parameters that control its performance. Their values were fixed on the basis of preliminary experiments, which were carried out on a training set provided by Gallego *et al* (2013). This set comprises *RanReal*, *RanInt* and *Geo* subsets that are generated in precisely the same manner as datasets used in the main experiments. In fact, the training set is obtained by generating five additional instances for each combination of characteristics, namely the type of instance (*RanReal*, *RanInt* or *Geo*), the number of elements $n \in \{120, 240, 480, 960\}$, and the type of groups (fixed size or varying size). For each parameter setting, we run the ITS algorithm once for each instance in the training set. We applied the same cutoff times as in Gallego *et al* (2013): 3 s (respectively, 20 s, 120 s, and 600 s) for each run of the algorithm on an instance of size 120 (respectively, 240, 480, and 960). To select the values of the parameters, we applied a simple procedure. We allowed one or two related parameters to vary, while keeping the other parameters fixed at reasonable values on the basis of the results obtained in a limited number of runs of the algorithm on the smallest instances in the training set. We started by investigating the parameters λ_1, λ_2 and their ratio λ_2/λ_1. We have found that the ITS algorithm is quite robust to variations of these parameters. It performed well when the above ratio was in the range from 3 to 10 and λ_2 was in the range from 0.3 to 0.7. In our main experiments, we fixed λ_2 at 0.5 and λ_1 at 0.1. The next step was to assess the influence of the parameter α_0 on the quality of solutions. Experimentally it has been determined that acceptable values for α_0 are in the interval $[3, 30]$. We observed that larger values of α_0, say 100 or higher, lead to worse results. We set α_0 to 10. In the next experiment, we studied the parameters μ_1 and μ_2. The results obtained were quite robust to the choice of μ_1 and μ_2 provided that μ_2 was not very large (say, less than 1000). We decided to set μ_1 to 10 and μ_2 to 300. While tuning GSP parameters, we fixed the iteration number $\overline{K}$ at 100. However, more detailed investigation has shown that, for the largest instances in the datasets, slightly better results were obtained when the iteration number $\overline{K}$ was increased. Therefore in the main experiments, the value of $\overline{K}$ was made dependent on the size of the problem: $\overline{K} = 100$ if $n < 300$, and $\overline{K} = 200$ otherwise.

We notice that a more significant increase of $\overline{K}$ (for example, to 500) led to worse results. Another TS parameter is the tabu tenure $\overline{\tau}$. We have experimented with several values of this parameter. On the basis of the results, we set $\overline{\tau}$ to $\min(10, n/4)$. Our final preliminary experiment was aimed at investigating the probability parameter Q. Our tests have shown that setting Q to a positive value can be advantageous for instances in the *Geo* dataset only. We remind that the parameter Q, when positive, is used to increase the number of relocation moves performed during execution of the GSP procedure. Let w denote the number of such moves summed over all invocations of GSP in a run of the ITS algorithm. We have found that when Q is set to zero, $w = 0$ for 17 (out of 20) *Geo* instances (with varying group size) in the training set. In the remaining three instances, w ranging from 4 to 22 is vanishingly small with respect to the number of interchange moves. This means that the GSP procedure, when applied to *Geo* instances with varying group size, almost always generates a partition in which the sizes of groups are identical to those of the initial partition submitted to the procedure. This slightly weakens the diversification capabilities of the solution perturbation process of the ITS algorithm. We performed a series of tests for $Q \in \{0, 0.1, 0.2, \ldots, 1.0\}$ on the *Geo* instances. The results suggest that any value of the probability parameter Q from the interval $[0.1, 0.7]$ is acceptable, whereas $Q = 0$ is not, as in this case the performance of ITS is noticeably worse than for the case of bounded positive Q. In the main experiments, we fixed Q at 0.4. The zero values of w (for $Q = 0$) observed for *Geo* instances allow us to assume that, for these instances, pairwise interchanges consistently give higher objective function values than relocation moves. Such a behaviour probably is determined by the fact that the dissimilarity matrix in the *Geo* case is composed of entries representing point-to-point distances. We notice that for two other sets of instances with varying group size, *RanInt* and *RanReal*, w was always positive and sufficiently large. For instances of size 960, for example, this value was about 1000. Therefore, setting Q to a positive value for these sets does not lead to an increase in the performance of the ITS algorithm. Our code implementing ITS is tuned to set the Q value automatically. If $\gamma = 1$, the algorithm makes an attempt to recognize MDGP instances in which the matrix $\mathbf{D}$ is formed by calculating pairwise distances between points. It examines a small number of triplets of elements. If, for each triplet in the sample, the triangle inequality is satisfied, then Q is set to 0.4; otherwise, Q is set to 0. Thus, the only parameter required by the ITS code is a maximum CPU time limit for a run.

The results summarized in the next subsection were obtained by running each algorithm 10 times on each problem instance in the test sets. The cutoff times were the same as in the preliminary experiments.

Results

The MDGP instances used in our experiments are by far too large to be solved to guaranteed optimality. Therefore, the optimal values for them are not known. In this regard, we decided to perform one long run of the ITS algorithm for each dissimilarity matrix in the test suite. For a subset of instances, namely, *Geo* instances with varying group size, good results were also achieved by applying the MSA algorithm. Therefore, we also performed one long run of MSA for these problem instances. The solution values obtained (or better of the two values in the case of *Geo*) act as reference points when evaluating the outcome of the algorithms involved in the comparison. The time limit imposed on a long run of an algorithm was 300 s (respectively, 1800 s, 7200 s and 36000 s) for an instance of size 120 (respectively, 240, 480 and 960). Let the reference point be denoted by F_{best} and let P stand for a solution delivered by an algorithm being evaluated. To assess the quality of P, we use the following measure

$$M = 100(F_{\text{best}} - F(P))/F_{\text{best}}. \tag{11}$$

Tables 1 and 2 summarize the results obtained by the tested algorithms on the *RanInt* dataset. In the name of an instance, the integer following n indicates the number of elements. The names are displayed in the first column of the tables. The second column provides the best values F_{best} achieved by the ITS algorithm. We should remark that the F_{best} values given in the tables of this paper are better than those reported by Gallego *et al* (2011) for all 120 MDGP instances used in our computational experiments. The third column of Tables 1 and 2 contains, for each instance, the value of M calculated using (11) for the best solution (and, in parentheses, the average value of M) obtained in 10 runs of ITS. The remaining columns show these statistics for the other algorithms. The results, averaged over 20 instances, are presented in the last row.

From Table 1, we see that ITS is superior to the other five algorithms. Among them, VNS and MSA are ranked second and third, but the HGA is not far behind. In fact, HGA shows better results compared to both VNS and MSA for instances of size 120 and 240. Its performance, however, deteriorates rapidly when the number of elements is large. The more general conclusion of the experiment on *RanInt* dataset is that the problem instances are difficult for the tested algorithms to attain the F_{best} values within a reasonable amount of time. The only instance for which ITS was able to match the best solution is *RanInt_n120_ds_09*. The results presented in Table 2 show that again the quality of solutions is significantly in favour of the ITS algorithm. According to these results, the second and third best algorithms are HGA and VNS. However, the difference between the M values for HGA and VNS and those for ITS is larger than the difference between the M values for the worst of the algorithms, T-LCW, and those for HGA and VNS. Comparing HGA and VNS, we see that the former is better for smaller instances, while the latter is better for larger instances. By analysing the results in Tables 1 and 2, we also find that the *RanInt* instances with varying group size are more difficult for each of the six algorithms than those with fixed group size. The ITS algorithm matched the best solution for all five

Table 1 Performance of ITS and other algorithms on *RanInt* instances with varying group size

Instance	*Best value*	*Best and average (in parentheses) values of M*					
	(*by ITS*)	*ITS*	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
RanInt_n120_ds_06	49 767	0.07(0.44)	0.85(1.49)	0.23(0.55)	0.61(1.12)	1.82(2.19)	0.92(1.74)
RanInt_n120_ds_07	50 349	0.33(1.12)	0.78(1.47)	1.12(1.43)	0.86(1.26)	1.36(1.94)	1.20(1.95)
RanInt_n120_ds_08	50 434	0.24(0.41)	0.51(1.07)	0.37(0.69)	0.26(0.83)	1.28(1.75)	0.83(1.30)
RanInt_n120_ds_09	50 499	0.00(0.32)	0.66(1.35)	0.22(0.42)	0.17(0.98)	1.42(2.04)	1.06(1.93)
RanInt_n120_ds_10	50 405	0.16(0.40)	0.53(1.06)	0.42(0.65)	0.44(1.19)	1.24(1.76)	1.07(1.32)
RanInt_n240_ds_06	161 417	0.29(0.51)	1.07(1.33)	0.55(0.96)	0.82(1.32)	1.89(2.13)	1.49(1.72)
RanInt_n240_ds_07	160 333	0.35(0.57)	0.71(1.16)	0.27(0.53)	0.76(1.41)	1.78(2.15)	1.36(1.85)
RanInt_n240_ds_08	158 249	0.06(0.32)	1.00(1.11)	0.48(0.73)	0.94(1.31)	1.54(1.89)	1.43(1.61)
RanInt_n240_ds_09	160 804	0.36(0.49)	1.26(1.42)	0.49(0.71)	1.19(1.57)	1.93(2.16)	1.38(1.86)
RanInt_n240_ds_10	160 364	0.39(0.60)	0.90(1.29)	0.38(0.78)	1.00(1.30)	2.07(2.22)	1.56(1.82)
RanInt_n480_ds_06	389 997	0.38(0.55)	1.23(1.55)	1.89(2.13)	1.51(1.79)	2.06(2.50)	1.84(2.10)
RanInt_n480_ds_07	390 809	0.59(0.81)	1.58(1.79)	2.29(2.45)	1.66(1.98)	2.66(3.00)	2.36(2.57)
RanInt_n480_ds_08	391 441	0.34(0.47)	1.49(1.64)	1.98(2.20)	1.44(1.63)	2.30(2.60)	1.89(2.19)
RanInt_n480_ds_09	389 440	0.34(0.63)	1.57(1.72)	1.81(2.15)	1.38(1.90)	2.24(2.69)	1.92(2.19)
RanInt_n480_ds_10	394 050	0.30(0.82)	1.28(1.47)	2.10(2.30)	1.38(1.91)	1.98(2.54)	1.75(2.24)
RanInt_n960_ds_06	1 238 068	0.29(0.45)	1.35(1.47)	1.94(2.10)	1.39(1.56)	1.97(2.18)	1.70(1.93)
RanInt_n960_ds_07	1 243 424	0.44(0.64)	1.47(1.58)	2.18(2.30)	1.22(1.61)	2.10(2.37)	2.10(2.33)
RanInt_n960_ds_08	1238 714	0.41(0.50)	1.42(1.50)	2.04(2.22)	1.23(1.57)	1.91(2.09)	1.68(1.91)
RanInt_n960_ds_09	1 238 770	0.34(0.48)	1.45(1.55)	1.83(2.14)	1.21(1.51)	1.91(2.15)	1.68(1.99)
RanInt_n960_ds_10	1 240 981	0.31(0.46)	1.39(1.50)	1.96(2.06)	1.34(1.53)	1.99(2.18)	1.67(2.01)
Average		0.30(0.55)	1.13(1.43)	1.23(1.48)	1.04(1.46)	1.87(2.23)	1.54(1.93)

Table 2 Performance of ITS and other algorithms on *RanInt* instances with fixed group size

Instance	*Best value*	*Best and average (in parentheses) values of M*					
	(*by ITS*)	*ITS*	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
RanInt_n120_ss_06	46 647	0.00(0.11)	0.53(1.05)	0.00(0.12)	0.28(0.65)	1.07(1.30)	0.29(0.95)
RanInt_n120_ss_07	47 142	0.00(0.05)	0.45(0.95)	0.01(0.23)	0.32(0.58)	1.20(1.48)	0.53(1.01)
RanInt_n120_ss_08	47 390	0.00(0.05)	0.55(0.87)	0.00(0.11)	0.14(0.61)	1.10(1.25)	0.69(0.87)
RanInt_n120_ss_09	47 660	0.00(0.06)	0.61(0.99)	0.00(0.09)	0.20(0.62)	1.07(1.43)	0.58(0.89)
RanInt_n120_ss_10	47 807	0.00(0.01)	0.26(0.86)	0.00(0.04)	0.24(0.58)	0.79(1.21)	0.61(0.89)
RanInt_n240_ss_06	155 644	0.12(0.33)	0.77(1.13)	0.15(0.29)	0.70(1.05)	1.62(1.84)	1.21(1.43)
RanInt_n240_ss_07	155 849	0.07(0.26)	0.80(1.05)	0.05(0.23)	0.67(1.16)	1.83(1.93)	1.07(1.40)
RanInt_n240_ss_08	155 398	0.15(0.32)	1.00(1.18)	0.17(0.24)	0.84(1.28)	1.58(1.82)	1.00(1.38)
RanInt_n240_ss_09	156 043	0.02(0.04)	0.88(1.26)	0.00(0.08)	0.99(1.30)	1.77(2.04)	1.05(1.58)
RanInt_n240_ss_10	155 993	0.11(0.21)	0.65(0.97)	0.11(0.19)	0.66(1.00)	1.17(1.66)	0.86(1.38)
RanInt_n480_ss_06	380 170	0.43(0.56)	1.61(1.76)	2.02(2.16)	1.43(1.67)	2.31(2.50)	1.74(1.84)
RanInt_n480_ss_07	380 132	0.34(0.43)	1.46(1.62)	1.69(1.95)	1.45(1.75)	2.16(2.44)	1.47(1.86)
RanInt_n480_ss_08	380 278	0.28(0.45)	1.37(1.57)	1.90(2.04)	1.33(1.64)	2.14(2.38)	1.48(1.85)
RanInt_n480_ss_09	379 285	0.24(0.41)	1.41(1.60)	1.70(1.99)	1.55(1.70)	2.29(2.43)	1.38(1.79)
RanInt_n480_ss_10	381 090	0.38(0.52)	1.21(1.60)	1.93(2.12)	1.51(1.89)	2.10(2.55)	1.69(1.89)
RanInt_n960_ss_06	1 220 773	0.24(0.43)	1.39(1.49)	1.95(2.03)	1.39(1.54)	1.79(2.03)	1.27(1.51)
RanInt_n960_ss_07	1 220 684	0.19(0.37)	1.24(1.40)	1.95(2.05)	1.44(1.59)	1.73(2.10)	1.39(1.58)
RanInt_n960_ss_08	1 220 503	0.34(0.44)	1.28(1.42)	2.00(2.05)	1.15(1.48)	1.93(2.08)	1.29(1.53)
RanInt_n960_ss_09	1 219 765	0.39(0.49)	1.37(1.49)	2.08(2.15)	1.39(1.67)	1.92(2.24)	1.24(1.61)
RanInt_n960_ss_10	1220 084	0.49(0.55)	1.52(1.59)	2.00(2.16)	1.38(1.59)	1.96(2.19)	1.44(1.62)
Average		0.19(0.30)	1.02(1.29)	0.99(1.12)	0.95(1.27)	1.68(1.94)	1.11(1.44)

instances of size 120 in Table 2. For four of them as well as for *RanInt_n240_ss_09*, solutions of value F_{best} have also been obtained by HGA.

The results of solving MDGP instances in the *RanReal* dataset are summarized in Tables 3 and 4. Their structure is the same as that of Tables 1 and 2. On the basis of the results

Table 3 Performance of ITS and other algorithms on *RanReal* instances with varying group size

Instance	*Best value*	*Best and average (in parentheses) values of M*					
	(*by ITS*)	*ITS*	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
RanReal_n120_ds_06	50287.42	0.13(0.33)	0.51(1.43)	0.37(0.65)	0.79(1.22)	1.51(1.86)	1.16(1.71)
RanReal_n120_ds_07	50305.71	0.04(1.29)	0.87(1.46)	1.17(1.47)	0.86(1.34)	1.55(2.12)	1.20(1.88)
RanReal_n120_ds_08	50491.67	0.17(0.40)	0.47(1.32)	0.23(0.46)	0.55(1.27)	1.45(1.69)	0.79(1.45)
RanReal_n120_ds_09	50455.54	0.13(0.43)	0.88(1.52)	0.31(0.78)	0.48(1.04)	1.29(1.84)	1.33(1.80)
RanReal_n120_ds_10	49818.84	0.19(0.59)	0.98(1.49)	0.93(1.17)	0.69(1.26)	1.21(1.77)	0.67(1.61)
RanReal_n240_ds_06	161343.04	0.28(0.53)	0.87(1.19)	0.70(1.15)	1.00(1.39)	1.54(1.91)	1.14(1.70)
RanReal_n240_ds_07	160274.40	0.55(0.74)	1.01(1.34)	0.63(0.86)	1.06(1.72)	1.91(2.25)	1.40(1.88)
RanReal_n240_ds_08	158642.90	0.08(0.28)	0.52(1.04)	0.47(0.71)	0.78(1.45)	1.33(1.99)	1.16(1.63)
RanReal_n240_ds_09	160031.39	0.21(0.59)	0.90(1.31)	1.03(1.25)	1.03(1.42)	1.82(2.13)	1.52(1.95)
RanReal_n240_ds_10	160380.44	0.32(0.51)	1.03(1.37)	0.47(0.79)	1.11(1.46)	1.69(2.07)	0.85(1.75)
RanReal_n480_ds_06	389766.50	0.53(0.67)	1.53(1.65)	2.17(2.29)	1.44(1.73)	2.19(2.57)	1.92(2.20)
RanReal_n480_ds_07	389279.42	0.45(0.61)	1.42(1.69)	2.09(2.27)	1.27(1.83)	2.32(2.71)	2.01(2.21)
RanReal_n480_ds_08	389731.31	0.51(0.71)	1.55(1.70)	1.82(2.28)	1.47(1.61)	2.20(2.60)	1.78(2.12)
RanReal_n480_ds_09	388116.74	0.56(0.71)	1.60(1.75)	2.04(2.37)	1.43(1.83)	2.32(2.68)	1.84(2.19)
RanReal_n480_ds_10	392854.78	0.45(0.64)	1.07(1.39)	2.11(2.29)	1.36(1.78)	2.28(2.44)	1.45(2.05)
RanReal_n960_ds_06	1234653.27	0.34(0.54)	1.35(1.58)	2.10(2.18)	1.36(1.59)	1.83(2.18)	1.65(1.99)
RanReal_n960_ds_07	1239682.66	0.53(0.65)	1.53(1.62)	2.16(2.30)	1.38(1.59)	2.04(2.26)	1.64(2.05)
RanReal_n960_ds_08	1233411.63	0.43(0.60)	1.35(1.54)	2.09(2.20)	1.25(1.58)	1.93(2.09)	1.67(1.93)
RanReal_n960_ds_09	1238854.27	0.40(0.61)	1.50(1.68)	2.10(2.26)	1.40(1.68)	1.82(2.21)	1.88(2.13)
RanReal_n960_ds_10	1241230.26	0.41(0.55)	1.40(1.63)	1.98(2.11)	1.41(1.59)	1.97(2.13)	1.73(1.95)
Average		0.33(0.60)	1.12(1.48)	1.35(1.59)	1.11(1.52)	1.81(2.17)	1.44(1.91)

Table 4 Performance of ITS and other algorithms on *RanReal* instances with fixed group size

Instance	*Best value*	*Best and average (in parentheses) values of M*					
	(*by ITS*)	*ITS*	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
RanReal_n120_ss_06	47256.73	0.06(0.17)	0.77(1.20)	0.16(0.21)	0.25(0.64)	1.04(1.36)	0.74(1.13)
RanReal_n120_ss_07	47085.87	0.00(0.10)	0.41(0.99)	0.05(0.13)	0.28(0.67)	0.85(1.23)	0.69(1.07)
RanReal_n120_ss_08	47460.13	0.00(0.05)	0.33(0.70)	0.00(0.01)	0.17(0.72)	0.75(1.25)	0.42(0.89)
RanReal_n120_ss_09	47686.34	0.00(0.17)	0.41(1.02)	0.00(0.24)	0.33(0.65)	0.97(1.38)	0.00(1.10)
RanReal_n120_ss_10	47415.35	0.00(0.12)	0.43(0.94)	0.00(0.19)	0.34(0.87)	0.66(1.48)	0.58(1.15)
RanReal_n240_ss_06	155671.23	0.23(0.34)	0.83(1.07)	0.33(0.58)	0.82(1.13)	1.65(1.84)	1.15(1.39)
RanReal_n240_ss_07	155739.51	0.05(0.24)	0.90(1.11)	0.12(0.35)	0.72(1.18)	1.61(1.78)	1.06(1.41)
RanReal_n240_ss_08	155675.74	0.13(0.28)	0.70(1.11)	0.15(0.20)	0.75(1.24)	1.75(1.92)	0.91(1.41)
RanReal_n240_ss_09	155174.95	0.03(0.28)	0.99(1.23)	0.24(0.59)	0.62(1.02)	1.46(1.79)	0.82(1.26)
RanReal_n240_ss_10	155927.91	0.02(0.18)	0.54(1.12)	0.12(0.23)	1.00(1.36)	1.51(1.88)	0.75(1.31)
RanReal_n480_ss_06	379476.32	0.42(0.54)	1.47(1.63)	2.09(2.18)	1.47(1.70)	2.16(2.39)	1.49(1.74)
RanReal_n480_ss_07	379366.93	0.30(0.45)	1.53(1.64)	1.69(1.93)	1.24(1.69)	2.14(2.44)	1.53(1.86)
RanReal_n480_ss_08	378207.97	0.25(0.44)	1.45(1.54)	1.88(2.04)	0.98(1.46)	2.15(2.28)	1.53(1.71)
RanReal_n480_ss_09	377963.50	0.33(0.45)	1.34(1.55)	1.87(2.12)	1.35(1.76)	2.08(2.35)	1.35(1.74)
RanReal_n480_ss_10	379715.99	0.25(0.48)	1.23(1.54)	1.75(2.03)	1.29(1.66)	2.12(2.35)	1.34(1.68)
RanReal_n960_ss_06	1216435.30	0.31(0.45)	1.17(1.42)	1.92(2.07)	1.20(1.54)	1.85(2.02)	1.23(1.47)
RanReal_n960_ss_07	1217879.08	0.30(0.52)	1.49(1.59)	2.03(2.12)	1.36(1.56)	1.87(2.09)	1.29(1.57)
RanReal_n960_ss_08	1215839.78	0.38(0.48)	1.40(1.56)	2.03(2.10)	1.21(1.52)	1.99(2.08)	1.30(1.58)
RanReal_n960_ss_09	1218792.87	0.43(0.53)	1.40(1.61)	2.08(2.18)	1.15(1.60)	1.89(2.10)	1.35(1.55)
RanReal_n960_ss_10	1218735.95	0.33(0.49)	1.36(1.56)	2.01(2.10)	1.40(1.53)	1.85(2.04)	1.10(1.53)
Average		0.19(0.34)	1.01(1.30)	1.03(1.18)	0.90(1.27)	1.62(1.90)	1.03(1.43)

obtained, we find that the overall ranking of the algorithms for *RanReal* is the same as in the experiment with the *RanInt* instances. From Tables 1–4, we can observe that for *RanReal* slightly better results were achieved by both T-LCW and SO and slightly worse results were achieved by applying HGA. The best value has been reached for *RanReal_n120_ss_07* (by ITS),

for *RanReal_n120_ss_08* and *RanReal_n120_ss_10* (by ITS and HGA), and for *RanReal_n120_ss_09* (by ITS, HGA and SO).

In Tables 5 and 6, we report the results of an empirical evaluation of tested algorithms for instances in the *Geo* dataset. The second column of Table 5 indicates, for each instance, the objective function value of better of the two solutions, one of which was found during one long run of ITS and another was produced by performing one long run of MSA. In most cases, the choice was to take the solution delivered by the ITS algorithm. As it can be seen from Table 5, the MSA algorithm comes out as a winner on the subset of *Geo* instances with varying group size. Slightly worse results were obtained with ITS and HGA. We note that ITS performed poorly for two instances, *Geo_n240_ds_07* and *Geo_n480_ds_10*. This led to a strong increase in the average of the M values for ITS (last row in Table 5). Another remark concerns the data shown in the second column. When the cutoff time is significantly increased, the results of ITS seem to be better than those of MSA. By inspecting Table 5, one can also see that VNS is the worst algorithm in the comparison.

Table 6 shows the results for the *Geo* instances with fixed group size. They indicate that ITS significantly outperforms other approaches. For this subset of the *Geo* dataset, VNS is the second best algorithm. It can also be noticed that T-LCW, SO and MSA are on the same level in terms of solution quality. Comparing the data in Tables 1–6, we can see that for *Geo* the algorithms produce solutions with considerably smaller relative error computed with respect to F_{best} than in the case of *RanInt* and *RanReal* data series. However, the best value has been reached in only one case: ITS did this for *Geo_n480_ss_06*. The other zero entries in Tables 5 and 6 stand for values of M, which are positive, yet less than 0.005. In other words, they are obtained by rounding M to a number with two fractional digits.

In Table 7, we compare the performance of the algorithms directly with each other, thereby ignoring the data contained in the second column of the previous tables. Given an MDGP instance I, let $F^*(I)$ denote the value of the best solution obtained from all runs of ITS, MSA, HGA, VNS, T-LCW and SO. Of course, when calculating $F^*(I)$, the solutions found during long runs of ITS and MSA are not taken into account. The first integer of the first entry in the second column of Table 7 shows the number of instances I in the *RanInt* dataset for which ITS produced at least one solution of value $F^*(I)$ in 10 runs. The integer in parentheses counts the number of instances in *RanInt* for which the average value of 10 solutions found by ITS is larger than the same parameter calculated for each of the other five heuristics. The other entries in the main part of the table are obtained analogously. The last row gives the sum of achieved scores across all three datasets. The statistics in Table 7 suggest the unequivocal superiority of the proposed algorithm over alternative approaches. The advantage of ITS is more pronounced for MDGP instances in the *RanInt* and *RanReal* data series.

We also conducted several experiments to investigate various components of our approach. We have attempted to

Table 5 Performance of ITS and other algorithms on *Geo* instances with varying group size

Instance	*Best value*	*Best and average (in parentheses) values of M*					
		ITS	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
Geo_n120_ds_06	136924.86*	0.06(0.16)	0.12(0.15)	0.19(0.21)	0.03(0.23)	0.15(0.23)	0.14(0.52)
Geo_n120_ds_07	108583.42*	0.04(0.05)	0.06(0.14)	0.15(0.19)	1.68(1.72)	0.12(0.18)	0.07(0.98)
Geo_n120_ds_08	88236.18*	0.01(0.04)	0.09(0.13)	0.15(0.17)	0.02(0.05)	0.10(0.13)	0.08(0.20)
Geo_n120_ds_09	95525.13*	0.03(0.06)	0.10(0.15)	0.19(0.21)	0.00(0.14)	0.16(0.36)	0.15(0.72)
Geo_n120_ds_10	65562.31*	0.00(0.07)	0.08(0.14)	0.11(0.13)	0.30(0.32)	0.08(0.13)	0.07(0.40)
Geo_n240_ds_06	358657.62*	0.01(0.02)	0.04(0.10)	0.09(0.11)	0.01(0.36)	0.06(0.12)	0.07(0.28)
Geo_n240_ds_07	342049.58*	0.41(0.50)	0.06(0.08)	0.12(0.14)	0.40(0.57)	0.12(0.41)	0.08(0.65)
Geo_n240_ds_08	131036.00†	0.01(0.01)	0.01(0.01)	0.02(0.02)	0.00(0.01)	0.02(0.19)	0.22(0.54)
Geo_n240_ds_09	410611.60*	0.01(0.17)	0.07(0.08)	0.08(0.12)	0.20(0.70)	0.08(0.19)	0.25(0.62)
Geo_n240_ds_10	355302.40*	0.01(0.10)	0.07(0.11)	0.12(0.13)	0.38(0.52)	0.09(0.34)	0.06(0.59)
Geo_n480_ds_06	1012706.46*	0.10(0.11)	0.06(0.07)	0.10(0.11)	0.12(0.13)	0.06(0.38)	0.26(0.53)
Geo_n480_ds_07	865014.39*	0.02(0.04)	0.06(0.07)	0.11(0.12)	0.06(0.07)	0.12(0.40)	0.19(0.50)
Geo_n480_ds_08	587750.02†	0.02(0.07)	0.00(0.02)	0.04(0.06)	0.23(0.66)	0.21(0.41)	0.23(0.49)
Geo_n480_ds_09	667404.35*	0.17(0.23)	0.02(0.04)	0.07(0.08)	0.34(0.78)	0.17(0.33)	0.09(0.35)
Geo_n480_ds_10	937286.16†	0.69(0.82)	0.00(0.02)	0.07(0.08)	0.86(0.97)	0.09(0.38)	0.33(0.66)
Geo_n960_ds_06	3153420.65*	0.00(0.06)	0.04(0.04)	0.06(0.07)	0.02(0.11)	0.16(0.26)	0.19(0.43)
Geo_n960_ds_07	1301869.45*	0.01(0.30)	0.00(0.01)	0.01(0.09)	0.29(0.49)	0.54(0.78)	0.48(0.79)
Geo_n960_ds_08	1723467.35†	0.12(0.26)	0.00(0.01)	0.02(0.03)	0.18(0.37)	0.07(0.27)	0.07(0.40)
Geo_n960_ds_09	1897532.78†	0.08(0.30)	0.00(0.03)	0.02(0.07)	0.51(0.64)	0.10(0.38)	0.29(0.55)
Geo_n960_ds_10	2618296.60*	0.08(0.09)	0.03(0.04)	0.05(0.05)	0.10(0.30)	0.07(0.17)	0.11(0.46)
Average		0.09(0.17)	0.05(0.07)	0.09(0.11)	0.29(0.46)	0.13(0.30)	0.17(0.53)

*Found by ITS.
†Found by MSA.

Table 6 Performance of ITS and other algorithms on *Geo* instances with fixed group size

Instance	*Best value*	*Best and average (in parentheses) values of M*					
	(*by ITS*)	*ITS*	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
Geo_n120_ss_06	125447.54	0.03(0.04)	0.07(0.12)	0.10(0.11)	0.02(0.05)	0.08(0.11)	0.07(0.12)
Geo_n120_ss_07	98529.63	0.04(0.05)	0.12(0.14)	0.10(0.13)	0.02(0.05)	0.09(0.12)	0.10(0.13)
Geo_n120_ss_08	79998.82	0.03(0.04)	0.08(0.11)	0.12(0.13)	0.00(0.05)	0.08(0.10)	0.08(0.11)
Geo_n120_ss_09	87290.10	0.01(0.04)	0.05(0.11)	0.11(0.12)	0.02(0.05)	0.09(0.11)	0.08(0.11)
Geo_n120_ss_10	60259.68	0.00(0.02)	0.04(0.08)	0.09(0.10)	0.01(0.04)	0.06(0.08)	0.06(0.09)
Geo_n240_ss_06	338613.71	0.01(0.01)	0.04(0.06)	0.07(0.07)	0.02(0.03)	0.04(0.05)	0.04(0.05)
Geo_n240_ss_07	326085.67	0.01(0.02)	0.04(0.06)	0.06(0.07)	0.01(0.03)	0.03(0.05)	0.05(0.05)
Geo_n240_ss_08	126918.24	0.01(0.01)	0.01(0.01)	0.02(0.02)	0.01(0.01)	0.02(0.02)	0.02(0.02)
Geo_n240_ss_09	391486.88	0.01(0.02)	0.05(0.07)	0.06(0.07)	0.02(0.03)	0.04(0.05)	0.04(0.05)
Geo_n240_ss_10	339545.82	0.00(0.01)	0.05(0.07)	0.06(0.07)	0.01(0.02)	0.03(0.04)	0.04(0.05)
Geo_n480_ss_06	966654.45	0.00(0.01)	0.04(0.04)	0.05(0.06)	0.01(0.03)	0.04(0.04)	0.04(0.04)
Geo_n480_ss_07	827717.19	0.00(0.01)	0.04(0.05)	0.06(0.07)	0.02(0.03)	0.03(0.04)	0.04(0.04)
Geo_n480_ss_08	556675.10	0.00(0.01)	0.03(0.03)	0.05(0.05)	0.03(0.04)	0.03(0.03)	0.02(0.03)
Geo_n480_ss_09	636374.07	0.00(0.01)	0.04(0.04)	0.05(0.06)	0.03(0.04)	0.03(0.04)	0.03(0.04)
Geo_n480_ss_10	883447.90	0.00(0.01)	0.04(0.05)	0.06(0.07)	0.03(0.03)	0.04(0.04)	0.04(0.05)
Geo_n960_ss_06	3069734.23	0.00(0.00)	0.03(0.03)	0.03(0.04)	0.02(0.02)	0.02(0.02)	0.02(0.02)
Geo_n960_ss_07	1257789.58	0.00(0.01)	0.01(0.01)	0.02(0.02)	0.01(0.01)	0.01(0.01)	0.01(0.01)
Geo_n960_ss_08	1674031.86	0.00(0.00)	0.02(0.02)	0.02(0.02)	0.01(0.02)	0.01(0.01)	0.01(0.01)
Geo_n960_ss_09	1835514.54	0.00(0.00)	0.02(0.02)	0.02(0.03)	0.01(0.02)	0.01(0.01)	0.01(0.01)
Geo_n960_ss_10	2529028.02	0.00(0.00)	0.02(0.03)	0.03(0.03)	0.01(0.02)	0.01(0.02)	0.02(0.02)
Average		0.01(0.02)	0.04(0.06)	0.06(0.07)	0.02(0.03)	0.04(0.05)	0.04(0.05)

Table 7 The number of instances for which the best solution found by each algorithm achieves the largest objective function value (in parentheses, the average objective function value is the largest among the six algorithms)

Set of instances	*ITS*	*MSA*	*HGA*	*VNS*	*T-LCW*	*SO*
RanInt	35(35)	0(0)	9(5)	0(0)	0(0)	0(0)
RanReal	40(38)	0(0)	3(2)	0(0)	0(0)	1(0)
Geo	23(26)	8(12)	0(0)	8(2)	1(0)	0(0)
Total	98(99)	8(12)	12(7)	8(2)	1(0)	1(0)

Table 8 Effect of replacing shaking operation of VNS with GSP procedure

Set of instances of size 960	*VNS*	*VNS–GSP*
RanInt_ds	1.28(1.56)	1.39(1.59)
RanInt_ss	1.35(1.57)	1.31(1.61)
RanReal_ds	1.36(1.61)	1.39(1.71)
RanReal_ss	1.26(1.55)	1.33(1.62)
Geo_ds	0.22(0.38)	0.28(0.48)
Geo_ss	0.01(0.02)	0.01(0.01)

incorporate the GSP procedure into the VNS method described in Palubeckis *et al* (2011). As mentioned in the previous section, GSP was used to play the role of the shaking operation of VNS. In Table 8, the last column lists the results of this variation of VNS, named VNS–GSP, on instances of size 960. The numerical entries of the table are obtained using formula (11), as in previous experiments. The results of the main version of VNS (second column) are extracted from Tables 1–6. In each row of Table 8, statistics are averaged over five instances of the same type. From the results, it can be seen that VNS–GSP does not perform better than the basic VNS method. Actually, in most cases, VNS–GSP provides slightly worse results. A deeper investigation has shown that VNS–GSP updates the best solution found so far significantly more times than VNS. As it is known from the description of the VNS method (Hansen *et al*, 2010), when an improvement of the currently best solution is obtained, the variable specifying the neighbourhood size is returned to its initial value, which is usually as small as possible. From the above two facts it is possible to deduce that VNS–GSP spends more time than VNS exploring neighbourhoods of small size. This means that, in the case of VNS–GSP, search diversification is slightly weaker than in the case of the basic VNS method. Another observation is that the GSP procedure is slower than the shaking operation of VNS. This also negatively influences the performance of the VNS–GSP algorithm.

The quite disappointing results of VNS–GSP made us pose the question about efficiency of the GSP procedure when embedded in the ITS scheme. To clarify the situation, we carried out an experiment with two variants of ITS. One of them is obtained by replacing GSP with a random perturbation procedure, which is similar to the shaking operation of VNS.

Table 9 Comparison of GSP with random perturbation and random restart strategies

Set of instances of size 960	*ITS*	*ITS–RP*	*ITS–RR*
RanInt_ds	0.36(0.51)	0.54(0.73)	1.94(2.02)
RanInt_ss	0.33(0.46)	0.55(0.70)	1.89(1.97)
RanReal_ds	0.42(0.59)	0.61(0.76)	1.91(2.04)
RanReal_ss	0.35(0.49)	0.57(0.71)	1.86(1.95)
Geo_ds	0.06(0.20)	0.01(0.02)	0.14(0.19)
Geo_ss	0.00(0.00)	0.01(0.02)	0.02(0.02)

We denote this variant by ITS–RP. Like in the case of ITS, the perturbation depth in ITS–RP is controlled by the parameters λ_1, λ_2 and α_0. Another variant of ITS, named ITS–RR, is the random restart TS method. It proceeds by repeatedly applying the TS procedure to randomly generated solutions. The results for instances of size 960 are presented in Table 9. We tried ITS–RP with α_0 fixed at 10 and different settings of parameters λ_1 and λ_2. The best results (shown in the third column of the table) were obtained when $\lambda_1 = 0.1$ and $\lambda_2 = 0.5$. They are significantly better than, for example, for ITS–RP with $\lambda_1 = 0.06$ and $\lambda_2 = 0.3$. On the other hand, when λ_2 gets closer to its limit 1 and λ_1 increases as well, the performance of ITS–RP becomes similar to that of ITS–RR. The entries in the second column of Table 9 are calculated by averaging the results of ITS reported in Tables 1–6. From Table 9, we see that GSP significantly contributes to the performance of the ITS method. Thus, during solution perturbation, it is advantageous to favour moves (interchanges and relocations) that minimize, to some degree, the degradation of the objective function value of the problem. When applied to the MDGP, such a strategy appears to be much more effective than random perturbation and random restart techniques.

In addition, we performed several experiments aimed at investigating various randomization strategies that are used in the TS and LS procedures. Specifically, the following four versions of ITS differing from the basic ITS method have been tested: (1) with LS procedure that does not use random permutations of elements and groups for search diversification (see the end of subsection 'Tabu search'); (2) with TS procedure in which the neighbourhood exploration process gets out of the loop comprised of Steps 3–5 as soon as a solution improving the best partition P^* is found (after processing the case of $\rho = 0$ in Steps 3.2 and 4.2, the algorithm immediately proceeds to Step 6); (3) with TS evaluating all solutions (if any) that improve the best partition P^* and selecting the best of them (thus no randomization in Steps 3.2 and 4.2 is involved); (4) with TS selecting the best move fully deterministically (the algorithm ignores moves that are equally good as the currently best one in Steps 3.3 and 4.3). We notice that all the above-listed versions of ITS are obtained from the basic algorithm by making modifications to eliminate precisely one kind of randomization. We tested these versions on a set of MDGP instances of size 960. Comparing with the results of ITS reported in Table 9 (second column), the increase in the value of the measure M, on the average, is 0.01 for the first three versions and 0.02 for the fourth one. Thus, the performance gain achieved by using randomization in ITS is quite small. However, on the other hand, the randomization strategies used in this paper can be implemented with almost no overhead.

Concluding remarks

In this paper we have developed an ITS algorithm for solving the MDGP. The algorithm significantly differs from the approaches described in Palubeckis *et al* (2011). To intensify the search, the ITS algorithm employs the TS technique, whereas MSA uses a stochastic process to narrow down the search and both HGA and VNS method take advantage of an LS procedure, which is fast but yields solutions of lower quality than those of TS. Another distinguishing feature of our algorithm is the use of the search diversification mechanism (procedure GSP), which perturbs a relatively good solution in such a way that the degradation of its quality is minimized to some degree. Meanwhile, VNS applies the random shaking strategy, HGA diversifies the search by generating new offspring, and MSA simply restarts from a new random solution. The use of GSP also makes a major difference between the ITS algorithm and the approach of Gallego *et al* (2013).

The algorithm we proposed is quite simple and easy to implement. At the same time, it is capable of producing high-quality solutions at a moderate computational cost. On the basis of the results of the experiments, we can conclude that the ITS algorithm is definitely superior to the current state-of-the-art methods for the MDGP. We believe that the better performance of our method resulted from the strategy of executing a large number of relatively short runs of TS, each of which is applied to a better-than-random partition generated by the solution perturbation procedure GSP. According to our experience, in particular that gained from the experiments comparing ITS against VNS with GSP incorporated, TS is the component that contributes most to the efficiency of the approach. However, as the comparison of ITS with random restart TS method shows, GSP plays an important role in achieving high performance of the method as well. Thus, good results are obtained when TS and GSP are acting in concert.

Using ITS and, in a few occasions, the SA algorithm, we improved the best known solutions reported by Gallego *et al* (2011) for all 120 MDGP instances we experimented with. We should note, however, that finding solutions of very high quality takes a quite big amount of computer time.

One of the possible directions of future research could be to apply an algorithm portfolio approach for dealing with the MDGP. Such an approach was found to be promising for solving several combinatorial problems (see, for example, Gomes and Selman (2001), Streeter and Smith (2008), Pillay (2012), and Remde *et al* (2012)). Using a portfolio of algorithms can lead to discovery of new algorithm design strategies.

References

Arani T and Lotfi V (1989). A three phased approach to final exam scheduling. *IIE Transactions* **21**(1): 86–96.

Areibi S, Grewal G, Banerji D and Du P (2007). Hierarchical FPGA placement. *Canadian Journal of Electrical and Computer Engineering* **32**(1): 53–64.

Aringhieri R and Cordone R (2011). Comparing local search metaheuristics for the maximum diversity problem. *Journal of the Operational Research Society* **62**(2): 266–280.

Baker KR and Powell SG (2002). Methods for assigning students to groups: A study of alternative objective functions. *Journal of the Operational Research Society* **53**(4): 397–404.

Bhadury J, Mighty EJ and Damar H (2000). Maximizing workforce diversity in project teams: A network flow approach. *Omega* **28**(2): 143–153.

Chen Y, Fan ZP, Ma J and Zeng S (2011). A hybrid grouping genetic algorithm for reviewer group construction problem. *Expert Systems with Applications* **38**(3): 2401–2411.

Falkenauer E (1998). *Genetic Algorithms and Grouping Problems*. Wiley: New York.

Fan ZP, Chen Y, Ma J and Zeng S (2011). A hybrid genetic algorithmic approach to the maximally diverse grouping problem. *Journal of the Operational Research Society* **62**(7): 1423–1430.

Gallego M, Laguna M, Martí R and Duarte A (2011). Maximally diverse grouping problem. http://www.optsicom.es/mdgp/, accessed 15 May 2012.

Gallego M, Laguna M, Martí R and Duarte A (2013). Tabu search with strategic oscillation for the maximally diverse grouping problem. *Journal of the Operational Research Society* **64**(5): 724–734.

Gomes CP and Selman B (2001). Algorithm portfolios. *Artificial Intelligence* **126**(1–2): 43–62.

Guruswami V, Pandu Rangan C, Chang MS, Chang GJ and Wong CK (2001). The K_r-packing problem. *Computing* **66**(1): 79–89.

Hansen P, Mladenović N and Moreno Pérez JA (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research* **175**(1): 367–407.

Kirkpatrick DG and Hell P (1983). On the complexity of general graph factor problems. *SIAM Journal on Computing* **12**(3): 601–609.

Li J and Behjat L (2006). A connectivity based clustering algorithm with application to VLSI circuit partitioning. *IEEE Transactions on Circuits and Systems–II: Express Briefs* **53**(5): 384–388.

Lotfi V and Cerveny R (1991). A final-exam-scheduling package. *Journal of the Operational Research Society* **42**(3): 205–216.

Martí R, Gallego M, Duarte A and Pardo EG (2013). Heuristics and metaheuristics for the maximum diversity problem. *Journal of Heuristics* **19**(4): 591–615.

Mingers J and O'Brien FA (1995). Creating student groups with similar characteristics: A heuristic approach. *Omega* **23**(3): 313–321.

Palubeckis G (2007). Iterated tabu search for the maximum diversity problem. *Applied Mathematics and Computation* **189**(1): 371–383.

Palubeckis G, Karčiauskas E and Riškus A (2011). Comparative performance of three metaheuristic approaches for the maximally diverse grouping problem. *Information Technology and Control* **40**(4): 277–285.

Pillay N (2012). Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society* **63**(1): 47–58.

Remde S, Cowling P, Dahal K, Colledge N and Selensky E (2012). An empirical study of hyperheuristics for managing very large sets of low level heuristics. *Journal of the Operational Research Society* **63**(3): 392–405.

Streeter M and Smith SF (2008). New techniques for algorithm portfolio design. In: McAllester DA and Myllymäki P (eds). *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*. AUAI Press: Helsinki, Finland, pp 519–527.

Weitz RR and Jelassi MT (1992). Assigning students to groups: A multi-criteria decision support system approach. *Decision Sciences* **23**(3): 746–757.

Weitz RR and Lakshminarayanan S (1996). On a heuristic for the final exam scheduling problem. *Journal of the Operational Research Society* **47**(4): 599–600.

Weitz RR and Lakshminarayanan S (1997). An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design, and exam scheduling. *Omega* **25**(4): 473–482.

Weitz RR and Lakshminarayanan S (1998). An empirical comparison of heuristic methods for creating maximally diverse groups. *Journal of the Operational Research Society* **49**(6): 635–646.

Yeoh HK and Nor MIM (2011). An algorithm to form balanced and diverse groups of students. *Computer Applications in Engineering Education* **19**(3): 582–590.

Received 29 August 2012;
accepted 17 February 2014 after one revision