

---

# Tabu Search with Strategic Oscillation for the Maximally Diverse Grouping Problem

MICAEL GALLEGO

Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.  
Micael.Gallego@urjc.es

MANUEL LAGUNA

Leeds School of Business, University of Colorado at Boulder, USA  
laguna@colorado.edu

RAFAEL MARTÍ

Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain  
Rafael.Marti@uv.es

ABRAHAM DUARTE

Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.  
Abraham.Duarte@urjc.es

---

## ABSTRACT

We propose new heuristic procedures for the maximally diverse grouping problem (MDGP). This NP-hard problem consists of forming maximally diverse groups —of equal or different size— from a given set of elements. The most general formulation, which we address, allows for the size of each group to fall within specified limits. The MDGP has applications in academics, such as creating diverse teams of students, or in training settings where it may be desired to create groups that are as diverse as possible. Search mechanisms, based on the tabu search methodology, are developed for the MDGP, including a strategic oscillation that enables search paths to cross a feasibility boundary. We evaluate construction and improvement mechanisms to configure a solution procedure that is then compared to state-of-the-art solvers for the MDGP. Extensive computational experiments with medium and large instances show the advantages of a solution method that includes strategic oscillation.

---

**Keywords:** Diversity problems, metaheuristics, strategic oscillation.

## 1. Introduction

The maximally diverse grouping problem (MDGP) consists of grouping a set of  $M$  elements into  $G$  mutually disjoint groups in such a way that the diversity among the elements in each group is maximized. The diversity among the elements in a group is calculated as the sum of the individual distance between each pair of elements, where the notion of distance depends on the specific application context. The objective of the problem is to maximize the overall diversity, i.e., the sum of the diversity of all groups. Feo and Khellaf (1990) proved that the MDGP is NP-hard.

The MDGP is called the  $k$ -partition problem in Feo et al. (1992) and the equitable partition problem in O'Brien and Mingers (1995). It arises in a wide range of real world settings; such as the design of VLSI circuits (Chen 1986; Feo and Khellaf 1990) or the storage of large programs onto paged memory (Kral, 1965), where the subroutines of a program have to be partitioned onto pages of available memory. In this particular application, the objective is to maximize the data transfer between subroutines on the same page (minimizing in this way the data transfers between different pages). One of the most popular MDGP applications appears in the academic context when forming student groups (Weitz and Jelassi, 1992). Specifically, in business schools is nowadays common to create diverse student workgroups or training teams in order to provide students a diverse environment (Weitz and Lakshminarayanan, 1998). The MDGP also applies to forming diverse groups of peer reviewers in scientific publications or project evaluation in scientific funding agencies (Hettich and Pazzani, 2006). Finally, workforce diversity is an increasing phenomenon in organizations. Creating diverse groups, in which people with different background work together, is a way to deal with this heterogeneity and facilitate their understanding and communication (Bhadury et al., 2000).

In order to formulate the MDGP in mathematical terms, we assume that each element can be represented by a set of attributes. Let  $s_{ik}$  be the state or value of the  $k^{th}$  attribute of element  $i$ , where  $k = 1, \dots, K$  and  $i = 1, \dots, M$ . Then, the distance  $d_{ij}$  between element  $i$  and  $j$  may be simply defined by the Euclidean calculation:

$$d_{ij} = \sqrt{\sum_{k=1}^K (s_{ik} - s_{jk})^2}$$

We have identified two variants of the MDGP. The first one (MDGP1) is the better known and forces all groups to have the same number  $S$  of elements, with  $S = M/G$ . The second variant (MDGP2) allows the size  $S_g$  of each group  $g$  to be in the interval  $[a_g, b_g]$ , where  $a_g \leq b_g$  for  $g = 1, \dots, G$ . Clearly, MDGP1 is a special case of the MDGP2 for which  $S_g = a_g = b_g$  for all  $g$ . Our procedure is designed for the MDGP2 but tested on both MDGP1 and MDGP2. In the remainder, MDGP will refer to the general case MDGP2. Both variants can be formulated as quadratic integer programs with binary variables  $x_{ig}$  that take the value of 1 if element  $i$  is in group  $g$  and 0 otherwise. A quadratic integer programming formulation of MDGP1 is:

$$\begin{aligned}
&\text{Maximize} && \sum_{g=1}^G \sum_{i=1}^{M-1} \sum_{j>i}^M d_{ij} x_{ig} x_{jg} \\
&\text{subject to} && \sum_{g=1}^G x_{ig} = 1 \quad i = 1, 2, \dots, M \\
&&& \sum_{i=1}^M x_{ig} = S \quad g = 1, 2, \dots, G \\
&&& x_{ig} \in \{0, 1\} \quad i = 1, \dots, M \quad g = 1, \dots, G
\end{aligned}$$

The objective function adds the distance of all pairs of elements that belong to the same group. The first set of constraints forces the assignment of each element to a group. The second set of constraints forces the size of all groups to be equal to  $S$ . In the more general case, MDGP2, the second set of constraints is replaced with:

$$a_g \leq \sum_{i=1}^M x_{ig} \leq b_g \quad g = 1, 2, \dots, G$$

In terms of the mathematical formulation, both problems (MDGP1 and MDGP2) are equivalent. However, the generalization that allows groups of different sizes has implications with respect to developing search procedures because the search space is larger for MDGP2. In particular, procedures for the MDGP1 can focus on the area of the solution space for which all groups are of the same size.

The next section summarizes the most relevant MDGP literature. It is followed by a description of four proposed methods for constructing feasible solutions, two of which are based on memory mechanisms and two on Greedy Randomized Adaptive Search Procedures (GRASP). The proposed tabu search with strategic oscillation is described in Section 3, followed by computational experiments with both instances from the literature and new larger instances with  $M = 480$  and  $M = 960$ . Statistical analysis shows the merit of the approach when compared to existing methods.

## 2. Previous Methods

The MDGP has been the subject of study for at least 21 years, beginning with the multistart algorithm introduced by Arani and Lotfi (1989). This procedure consists of a random construction followed by an improving phase that partially deconstructs the random solution and scans all possible reconstructions to select the best one. This process is repeated until the solution does not change between reconstructions (i.e., when the current solution cannot be improved). Feo and Khellaf (1990) proposed several heuristics based on graph theory for the special case of even-sized groups, odd-sized groups and  $2^i$  sized groups. The authors also show that the values obtained by their heuristics are within a bounded percentage of the optimal solution. The most recent solution procedure introduced to the OR literature

is a memetic algorithm due to Fan et al. (2011). Chen et al. (2011) apply this procedure to a practical application of the MDGP known as the Reviewer Group Construction problem. Weitz and Lakshminarayanan (1998) carried out extensive experimentation to compare all heuristics for the MDGP known at the time in addition to the new ones that they introduced in their work. They identified the Lotfi-Cerveney-Weitz (LCW) heuristic as the best. The LCW is an improvement method that may be initiated from a random solution or a solution generated with the Weitz-Jelassi (WJ) construction procedure, as tested by Weitz and Lakshminarayanan (1998). The authors did not find significant differences in solution quality when LCW was started from a random solution and when the WJ construction method was used to initiate the search. They do, however, report a significant difference in computational time, with WJ taking considerably longer than a random starting point. Therefore for purpose of comparison with previous methods, we have implemented the LCW improvement method, which we start from a random solution.

The LCW is a modified version of the Lofti-Cerveney (LC) method, originally published by Lofti and Cerveney (1991) as a part of a comprehensive method for scheduling final exams with the objective of minimizing (instead of maximizing) the diversity in each group. Weitz and Lakshminarayanan (1996) discovered and corrected a number of errors in the LC method. The adaptation of the LC method to the MDGP is presented by Weitz and Lakshminarayanan (1998) and summarized in Figure 1.

---

```

Construct and initial solution
do {
  for ( $i = 1, \dots, M$ ) {
    1. Identify group  $g$  for which diversity is maximized when adding element  $i$ 
    2. Find the element  $j$  in  $g$  for which a switch of group assignments between elements
        $i$  and  $j$  results in the largest increase in the objective function value
    3. If the increase in the objective function is strictly positive, make the switch
  }
} while at least one switch is made

```

---

**Figure 1.** LC method

LCW is a variation of LC in which the search for element  $j$  is not limited to group  $g$ , as identified in step 1 of Figure 1. The motivation behind step 1 in LC is to minimize the search for element  $j$ , because the contributions of each element to the diversity of any group  $g$  is pre-calculated at the beginning of the procedure, updated after step 3 if a switch is made and stored in a matrix labeled  $R$ . Instead, LCW considers all groups when searching for element  $j$ , except for the group to which element  $i$  is currently assigned. The LCW method is summarized in Figure 2.

---

```

Construct and initial solution
do {
  for ( $i = 1, \dots, M$ ) {
    1. Find the element  $j$  in any group for which a switch of group assignments between
       elements  $i$  and  $j$  results in the largest increase in the objective function value
    2. If the increase in the objective function is strictly positive, make the switch
  }
} while at least one switch is made

```

---

**Figure 2.** LCW method

Weitz and Jelassi (1992) developed a basic constructive heuristic. Its philosophy is to avoid the assignment of very similar elements to the same group. WJ starts with the random assignment of an element to the first group. The heuristic then selects the element with the smallest distance to the previously selected element and assigns it to the next group. When a sweep of all groups has been completed, the procedure goes back to the first group. The construction finishes when all the elements have been assigned. Figure 3 summarizes this procedure.

- 
1. Randomly select a starting element, and assign it to group 1.
  2. From those elements still unassigned, select the element most similar to the last assigned element, where ties are broken arbitrarily. Assign this element to the next group, where the next group after group  $G$  is group 1.
  3. Stop if all elements have been assigned; otherwise go to step 2.
- 

**Figure 3.** WJ method

Fan et al. (2011) present a hybrid genetic algorithm (LSGA) for the solution of the MDGP. LSGA combines a genetic algorithm and a local search procedure, thus creating a hybrid method. This hybridization is usually known as a memetic algorithm (MA) and has been previously suggested for other problems (see for example Miller et al. 1993 and Vasko et al. 2005). The genetic aspect of LSGA is based on the encoding scheme for grouping problems proposed by Falkenauer (1998). The local search within LSGA implements a *best improvement* strategy based on exchanging elements between groups. To the best of our knowledge, Fan et al.'s is the first publication that describes a method for the general version of the MDGP, which allows for different group sizes. The basic structure of the memetic algorithm is shown in Figure 4.

Extensive experiments were conducted in Fan et al. (2011) to compare the relative merit of LSGA with a pure genetic algorithm (i.e., without local search) and LCW from random initial points (labeled R+LCW). Their experiments showed the effectiveness of LSGA when solving MDGP instances with equal and different group sizes.

- 
1. Start with a randomly generated population of *populationSize* solutions.
  2. Calculate the objective function of each solution in the population.
  3. Repeat the following steps until *populationSize* offspring have been created:
    - a. Select a pair of parent solutions from the current population, with the probability of selection being an increasing function of the solution value. Selection is done “with replacement,” meaning that the same solution can be selected more than once to become a parent.
    - b. Combine the selected parents to create a new solution.
    - c. Mutate the new offspring with probability *mp* (the mutation probability or mutation rate).
    - d. Repair the solution if it is not feasible.
    - e. Improve the solution with an improvement method.
  4. Create a new population with the best *populationSize* solutions in the original population and offspring population.
  5. Go to step 2.
- 

**Figure 4.** LSGA Memetic Algorithm

We have identified two main limitations of previous approaches that became the motivation for developing a new procedure to tackle the MDGP. First, all previous procedures with the exception of the one by Fan et al. (2011) were designed for the special version of the problem for which all groups are required to be of the same size (i.e., they were designed to solve MDGP1). Even Fan et al.’s (2011) MA limits the local search to solutions for which the size of each group is preserved by constricting the neighborhoods to those defined by swap moves. Solutions with different group sizes are found by their construction procedure and the crossover mechanisms but the local search preserves the sizes of the trial solution to which it is applied. Second, previous procedures preserve feasibility during the search. Once again, the exception is Fan et al.’s MA. Both the first stage of their initialization procedure (i.e., the procedure that builds the initial population) and the crossover operator allow for the violation of the group size restrictions. The infeasible solutions, however, are immediately repaired by their so-called *group size adjustment algorithm*. We overcome these limitations by designing a procedure for the general problem that can also be applied to the special case for which all groups have the same size. Our procedure searches the solution space both from within and coming from outside the feasible region, as described next.

### 3. Constructions, Neighborhoods and Strategic Oscillation

The main goal of our work is the development of a procedure for the MDGP that is based on the tabu search methodology. This section describes the elements of our proposed procedure: 1) construction of the initial solution, 2) neighborhood search and 3) strategic oscillation. We describe these elements separately because we later combine them to test several solution methods. A solution method may consist of simply constructing solutions with one of our construction procedures or both constructing and improving solutions by in addition applying an improvement method based on one of our neighborhood searches. Furthermore, the strategic oscillation framework is structured to use any of the construction or improvement methods. Instead of trying all possible combination, in the next section,

we sequentially evaluate the construction and improvement methods to choose the best to embed in the strategic oscillation framework.

Our construction method is greedy and accommodates both versions of the problem, namely, the one for which all groups are of the same size and the one for which the cardinality of each group is bounded. The method (GC, for greedy construction) starts by randomly selecting  $G$  elements and assigning each of these elements to a separate group. Therefore, at the end of the first step, each group has one element assigned to it. Then, the procedure performs  $M - G$  iterations to assign the remaining unassigned elements to groups. In order to generate a feasible solution, the iterations are divided into two phases. In the first phase, the elements are assigned to groups that currently contain fewer elements than the desired minimum number of elements  $a_g$ . In the second phase, the remaining elements are assigned to groups with a number of elements that is smaller than the desired maximum number of elements  $b_g$ .

Let  $E_g$  be the set of elements currently assigned to group  $g$ . The iterations in each phase start with the identification of the groups that have either fewer elements than the desired minimum in the first phase (i.e., all  $g$  for which  $|E_g| < a_g$ ) or fewer elements than the maximum allowed in the second phase (i.e., all  $g$  for which  $|E_g| < b_g$ ). Then, an unassigned element  $i$  is selected at random and added to the group (within the identified set) for which the average distance between element  $i$  and all the elements in the group is maximized. That is,  $i$  is assigned the group  $g$  for which  $D_{ig}$  is maximized:

$$D_{ig} = \frac{\sum_{j \in E_g} d_{ij}}{|E_g|}$$

The first phase finishes when all groups contain at least the desired minimum number of elements (i.e., when for each group  $g, |E_g| \geq a_g$ ). The second phase finishes when all elements have been assigned. Figure 5 summarizes the GC method.

- 
1. Randomly select  $G$  elements and assign one to each group
  2. Repeat the following steps until  $|E_g| \geq a_g$  for all groups
    - a. Randomly select an unassigned element  $i$
    - b. Assign element  $i$  to the group  $g$  with  $|E_g| < a_g$  that maximizes  $D_{ig}$
  3. Repeat the following steps until all elements are assigned
    - a. Randomly select an unassigned element  $i$
    - b. Assign element  $i$  to the group  $g$  with  $|E_g| < b_g$  that maximizes  $D_{ig}$
- 

**Figure 5.** GC Method

We developed two GC variants, one (GC-FULL) as a combination of GC and the FULL method proposed by Mingers and O'Brien (1995) and another one that includes elements from the tabu search methodology (GC-Tabu). In GC-FULL, instead of the random element selection in steps 2.a and 3.a of Figure 5, the procedure searches for the  $(i, g)$  pair that maximizes  $D_{ig}$  in order to make the assignment of element  $i$  to group  $g$ . Since the construction procedures are used within a multi-start framework, GC-Tabu utilizes two memory structures to record relevant information associated with previously

generated solutions and then applies this knowledge to the construction of new solutions. Specifically,  $freq(i,j)$  records the number of times that elements  $i$  and  $j$  are assigned to the same group in previous constructions and  $quality(i,j)$  records the average quality (i.e., objective function value) of the previous constructions for which elements  $i$  and  $j$  belonged to the same group. The distance value  $D_{ig}$  is modified according to the information in both memory structures to favor the assignment to the same group of pairs of elements with low frequency and high quality values.

Solutions are improved by performing a sequence of steps that are based on examining the neighborhood of the current solution and selecting the *best* move to make. The definition of best depends on the context and is an important design choice when implementing local searches. Fan et al. (2011), for their local search, use a neighborhood definition proposed by Baker and Powell (2002). The neighborhood consists of all possible switches (or swaps) of elements  $i$  and  $j$  belonging to different groups. The entire neighborhood is calculated at each step and the switch that produces the largest improvement to the objective function value is selected. That is, the method follows what it is known as *best improvement* (BI) strategy. Because, the procedure is applied as a true local search, it stops when no improving switch is found and the resulting solution is a local maximum. An alternative to BI is to stop the neighborhood search as soon as an improving switch is identified. This is the so-called *first improvement* (FI) strategy. Other possibilities for neighborhood searches are those provided by LC and LCW. We report experimental results with these four alternatives, that is, local searches defined by swap neighborhoods with the BI and FI strategies, as well as LC and LCW local searches.

All neighborhood searches in the MDGP literature are based on switching (or swapping) the group assignments of a pair of elements. This is the obvious design to preserve feasibility during the search when tackling the variant of the MDGP for which all groups must have the same size. For the general case, however, the neighborhood could be augmented with moves that allow transferring a single element from its current group to another group. These moves are typically called *insertions* in the literature and could be limited to only those for which the resulting solution is feasible. We have added these moves to the LCW, BI and FI neighborhoods described above to create T-LCW, T-BI and T-FI. The “T” in these procedures stands for “Tabu” because in addition to expanding the original neighborhoods to include insertions, we have added a short-term tabu memory to allow the search to continue beyond the first local optimum point. Specifically, when the local search reaches a point where no improving moves are available, the best (according to the specific rules of the procedure) non-improving move is selected and executed. At this point, elements that are moved from their current group to another are not allowed to move again for *tabuTenure* iterations. The process terminates when *maxIter* consecutive iterations have been performed without improving the best solution found during the search.

The construction and the improvement methods described above are combined in a straightforward way. That is, a construction method is executed to create a trial solution to which an improvement method is applied. The best solution found is kept and this process is performed until a time limit is reached. Note that the improvement methods labeled BI, FI, LC and LCW have a “natural” termination (i.e., they finish when no improving solution is found in the neighborhood of the current solution) while

the termination of their “T” counterparts (i.e., T-BI, T-FI and T-LCW) depends on the *maxIter* parameter value. Therefore, for a specified time limit, the number of constructions depends on both the choice of the improvement method and the value of *maxIter*.

So far, we have assumed that all neighborhood searches under consideration start from a feasible solution (i.e., one for which all elements are assigned and  $a_g \leq |E_g| \leq b_g$  for all groups) and feasibility is maintained through the search. An element of the tabu search methodology that has not been explored as thoroughly as others is the so-called *strategic oscillation* (SO), which Glover and Laguna (1997) describe as follows:

“Strategic oscillation operates by orienting moves in relation to a critical level, as identified by a stage of construction or a chosen interval of functional values. Such a critical level or oscillation boundary often represents a point where the method would normally stop. Instead of stopping when this boundary is reached, however, the rules for selecting moves are modified, to permit the region defined by the critical level to be crossed.”

The boundary that we intend to cross in the current context is the one defined by the feasibility of the solutions encountered during the search. While we would like to limit the search to solutions for which all the elements have been assigned, we would also like to explore a search space that includes solutions for which the group cardinality restrictions may be violated. Our goal is to design a strategic oscillation mechanism that we are able to couple with any of the construction and improvement methods described above (i.e., LC, LCW, BI and FI and their tabu variants). The oscillation between feasibility and infeasibility is defined by an integer parameter  $k$  that ranges between 0 and  $k_{max}$  and that is applied as follows:

$$a_g - k \leq |E_g| \leq b_g + k$$

This definition means that when  $k > 0$  the search is allowed to visit cardinality-infeasible solutions. To create the oscillation pattern, the value of  $k$  is reset to one after every successful application of the improvement method, otherwise  $k$  is increased by one unit in the manner described in Figure 6.

---

Repeat until search time expires

1. Construct an initial feasible solution.
2. Set  $k = 0$  and apply an improvement method. Let  $s$  be the resulting solution.
3. Set  $k = 1$ .

while ( $k \leq k_{max}$ )

4. Make solution  $s'$  the result of applying the improvement method to solution  $s$  when enforcing  $a_g - k \leq |E_g| \leq b_g + k$  for all groups
  5. Repair solution  $s'$  if infeasible and apply the improvement method with  $k = 0$  (i.e., with  $a_g \leq |E_g| \leq b_g$ )
  6. If solution  $s'$  is better than  $s$  then make  $s = s'$  and set  $k = 1$ ; otherwise  $k = k + 1$
- 

**Figure 6.** Strategic Oscillation

The repair mechanism consists of removing elements from groups  $g$  for which  $|E_g| > b_g$  and adding elements to groups  $g$  for  $|E_g| < a_g$ . The elements are selected at random and the process continues until the cardinality of the groups is feasible. This is the repair mechanism implemented in LSGA (Fan et al. 2011). Step 1 in Figure 6 may be performed by applying GC-FULL or GC-Tabu. Also, solutions may be improved (see steps 2, 4 and 5 in Figure 6) with any of the improvement methods described above.

#### 4. Computational Experiments

This section describes the computational experiments that we performed to test the effectiveness and efficiency of the procedures discussed above. All methods were implemented in Java SE 6 and we solved the integer quadratic programming formulations described in Section 1 with Cplex 12.1 and Gurobi 4.01 using a single processor for each run. All experiments were conducted on an Intel Core 2 Quad CPU Q 8300 with 6 GiB of RAM and Ubuntu 9.04 64 bits OS.

We employed 480 instances in our experimentation. This benchmark set of instances, referred to as MDGPLIB, is available at <http://www.opticom.es/mdgp>. The set is divided into three subsets:

1. *RanReal* — This set consists of 160  $M \times M$  matrices in which the distance values  $d_{ij}$  are real numbers generated using a Uniform distribution  $U(0,100)$ . The number of elements  $M$ , the number of groups  $G$  and the limits of each group  $a_g$  and  $b_g$  are shown in Table 1. There are 20 instances for each combination of parameters (i.e., each row in Table 1), 10 for instances with equal group size (EGS) and 10 for instances with different group size (DGS). For the 10 instances in EGS, the group size is equal for all instances and is calculated as  $a_g = b_g = M/G$ . For the 10 instances in DGS, the limits of each group ( $a_g$  and  $b_g$ ) for each instance are generated randomly in the predefined interval. That is, the value of  $a_g$  is generated in the interval  $[a_g^{min}, a_g^{max}]$  and the value of  $b_g$  is generated in the interval  $[b_g^{min}, b_g^{max}]$ . This data set was introduced by Fan et al. (2011) with  $M$  ranging from 10 to 240. We have generated larger instances with  $M = 480$  and  $M = 960$ .

$M$	$G$	EGS	DGS			
		$a_g = b_g$	$a_g^{min}$	$a_g^{max}$	$b_g^{min}$	$b_g^{max}$
10	2	5	3	5	5	7
12	4	3	2	3	3	5
30	5	6	5	6	6	10
60	6	10	7	10	10	14
120	10	12	8	12	12	16
240	12	20	15	20	20	25
480	20	24	18	24	24	30
960	24	40	32	40	40	48

**Table 1.** Summary of parameters to generate problem instances

2. *RanInt* — This set has the same structure and size as *RanReal* (shown in Table 1) but distances are generated with an integer Uniform distribution  $U(0,100)$ .
3. *Geo* — This set follows the same structure and size as the previous two, however,  $d_{ij}$  values are calculated as Euclidean distances between pair of points with coordinates randomly generated in  $[0,10]$ . The number of coordinates for each point is generated randomly in the 2 to 21 range. Glover et al. (1998) introduced this generator for the MDP.

In our first experiment we attempt the solution of some of the problem instances by means of applying the commercial solvers Cplex 12.1 and Gurobi 4.01 to the integer quadratic programming formulations of MDGP1 and MDGP2 given in Section 1. For this experiment we employed 48 instances, one from each subset (*RanReal*, *RanInt* and *Geo*), type (EGS and DGS) and size ( $M = 10$  to  $M = 960$ ).

	Integer Program				Cplex Output			
	nz	rows	cols	LB	UB	gap	time	nodes
Geo_n010_ss_01.txt	40	12	20	3660.67	3660.67	0%	0.64	503
Geo_n012_ss_01.txt	96	16	48	716.46	716.46	0%	45.27	701993
Geo_n030_ss_01.txt	300	35	150	13776.34	60354.76	77%	1802.55	5291729
Geo_n060_ss_01.txt	720	66	360	45374.32	268663.1	83%	1800.61	661408
Geo_n120_ss_01.txt	2400	130	1200	99906.5	1044060.53	90%	1800.81	10389
Geo_n240_ss_01.txt	5760	252	2880	185973.83	2280971.91	92%	1811.05	149
Geo_n480_ss_01.txt	19200	500	9600	-	-	-	-	-
Geo_n960_ss_01.txt	46080	984	23040	-	-	-	-	-
RanInt_n010_ss_01.txt	40	12	20	1292.00	1292.00	0%	0.03	225
RanInt_n012_ss_01.txt	96	16	48	985.00	985.00	0%	25.51	398594
RanInt_n030_ss_01.txt	300	35	150	5324.00	17084.75	69%	1802.57	5511470
RanInt_n060_ss_01.txt	720	66	360	18408.00	83352.74	78%	1800.77	806515
RanInt_n120_ss_01.txt	2400	130	1200	40577.00	355806.92	89%	1800.84	16140
RanInt_n240_ss_01.txt	5760	252	2880	129877.00	1426655.06	91%	1811.47	130
RanInt_n480_ss_01.txt	19200	500	9600	-	-	-	-	-
RanInt_n960_ss_01.txt	46080	984	23040	-	-	-	-	-
RanReal_n010_ss_01.txt	40	12	20	1427.85	1427.85	0%	0.03	221
RanReal_n012_ss_01.txt	96	16	48	956.43	956.43	0%	25.82	398022
RanReal_n030_ss_01.txt	300	35	150	5503.12	16982.96	68%	1802.63	5664732
RanReal_n060_ss_01.txt	720	66	360	18164.17	82653.29	78%	1800.77	813849
RanReal_n120_ss_01.txt	2400	130	1200	42047.6	352947.83	88%	1800.24	16368
RanReal_n240_ss_01.txt	5760	252	2880	128619.53	1424593.19	91%	1812.2	177
RanReal_n480_ss_01.txt	19200	500	9600	-	-	-	-	-
RanReal_n960_ss_01.txt	46080	984	23040	-	-	-	-	-

**Table 2.** Cplex 12.1 results of MDGP1 formulation on EGS instances

Tables 2 and 3 report, for each instance the number of rows (*rows*), columns (*cols*), and nonzeros (*nz*) of the integer program, the number of nodes generated in the branch and bound tree (*nodes*), the CPU

time in seconds (*time*), the lower bound (*LB*), the upper bound (*UB*) and the gap (*gap*). The gap is computed as the upper bound minus the lower bound (best solution found) —both returned by the solver when the time limit is reached— divided by the upper bound and multiplied by 100. We limit each solver run to at most 1800 seconds of CPU time.

File name	Integer Program			Cplex Output				
	nz	rows	cols	LB	UB	gap	time	nodes
Geo_n010_ds_01.txt	60	14	20	3864.69	3864.69	0%	0.72	912
Geo_n012_ds_01.txt	144	20	48	807.68	807.68	0%	93.21	1433717
Geo_n030_ds_01.txt	450	40	150	14358.40	60152.22	76%	1803.38	8434509
Geo_n060_ds_01.txt	1080	72	360	48163.77	267789.57	82%	1801.18	1481547
Geo_n120_ds_01.txt	3600	140	1200	108971.98	1042936.21	90%	1800.79	20729
Geo_n240_ds_01.txt	8640	264	2880	190288.26	2291376.02	92%	1826.76	246
Geo_n480_ds_01.txt	28800	520	9600	-	-	-	-	-
Geo_n960_ds_01.txt	69120	1008	23040	-	-	-	-	-
RanInt_n010_ds_01.txt	60	14	20	1325.00	1325.00	0%	0.03	304
RanInt_n012_ds_01.txt	144	20	48	1059.00	1059.00	0%	47.08	739027
RanInt_n030_ds_01.txt	450	40	150	5607.00	17064.49	67%	1803.21	8539329
RanInt_n060_ds_01.txt	1080	72	360	19080.00	83421.05	77%	1801.20	1469352
RanInt_n120_ds_01.txt	3600	140	1200	44589.00	355761.49	87%	1800.82	30723
RanInt_n240_ds_01.txt	8640	264	2880	137150.00	1432488.61	90%	1824.05	311
RanInt_n480_ds_01.txt	28800	520	9600	-	-	-	-	-
RanInt_n960_ds_01.txt	69120	1008	23040	-	-	-	-	-
RanReal_n010_ds_01.txt	60	14	20	1437.81	1437.81	0%	0.03	319
RanReal_n012_ds_01.txt	144	20	48	1050.35	1050.35	0%	45.75	676416
RanReal_n030_ds_01.txt	450	40	150	5595.16	17005.14	67%	1803.29	8907437
RanReal_n060_ds_01.txt	1080	72	360	18967.72	82530.84	77%	1801.19	1505282
RanReal_n120_ds_01.txt	3600	140	1200	43420.36	352789.62	88%	1800.19	31022
RanReal_n240_ds_01.txt	8640	264	2880	133756.40	1430450.46	91%	1807.11	264
RanReal_n480_ds_01.txt	28800	520	9600	-	-	-	-	-
RanReal_n960_ds_01.txt	69120	1008	23040	-	-	-	-	-

**Table 3.** Cplex 12.1 results of MDGP2 formulation on DGS instances

Tables 2 and 3 show that Cplex 12.1 is capable of solving only the 12 out of the 48 instances, producing a  $gap = 0\%$  for all instances with  $M \leq 12$ . For the remaining 36 instances, the solver produces a large positive gap that increases with the problem size. For the largest problems (with  $M = 480$  and  $M = 960$ ) Cplex is unable to find a single integer solution. The behavior of the solver and the associated performance is similar in both DGS and EGS instances. Also similar are the results obtained when applying Gurobi 4.01 to these 48 instances. The only noticeable difference in performance is Gurobi's longer running times and larger gaps when tackling small and medium size problems. On the other hand, Gurobi is capable of finding at least one integer solution to problems with  $M = 480$  and  $M = 960$  when Cplex is not. These experiments show that the use of commercial optimization software to solve

the math programming formulation in section 1 seems to be practical only for small problems. It is possible, however, that the math formulation could be strengthened with valid inequalities or that the branch-and-bound process could be improved by rules and strategies that are specialized to the MDGP. As part of this research project, we did not follow this line of thought and therefore we can't claim that finding optimal solutions to larger problems is not possible or practical in general.

Moving to the heuristic approaches, a series of preliminary experiments were conducted to set the values of the key search parameters. In each experiment, we compute for each instance the overall best solution value, *BestValue*, obtained by the execution of all methods under consideration. Then, for each method, we compute the relative percentage deviation between the best solution value found by the method and the *BestValue*. We report the average of this relative percentage deviation (*Dev*) across all the instances considered in each particular experiment. We also report the number of instances (*#Best*) for which the value of the best solution obtained by a given method matches *BestValue*.

With the purpose of fine-tuning our methods, we employed a training set consisting of 10 instances (5 from EGS and 5 from DGS) from each subset of instances with  $M = 10$  to  $M = 240$ . Therefore, the training set has a total of 180 instances, 60 *RanReal*, 60 from *RanInt* and 60 from *Geo*. The fine-tuning included both finding the most effective combination of construction plus improvement and also determining the best values for the search parameters *tabuTenure*, *maxIter* and  $k_{max}$ . All methods were stopped using a time limit, which varied according to problem size, as specified in Table 4.

$M$	Seconds
$\leq 60$	1
120	3
240	20
480	120
960	600

**Table 4.** CPU time limits

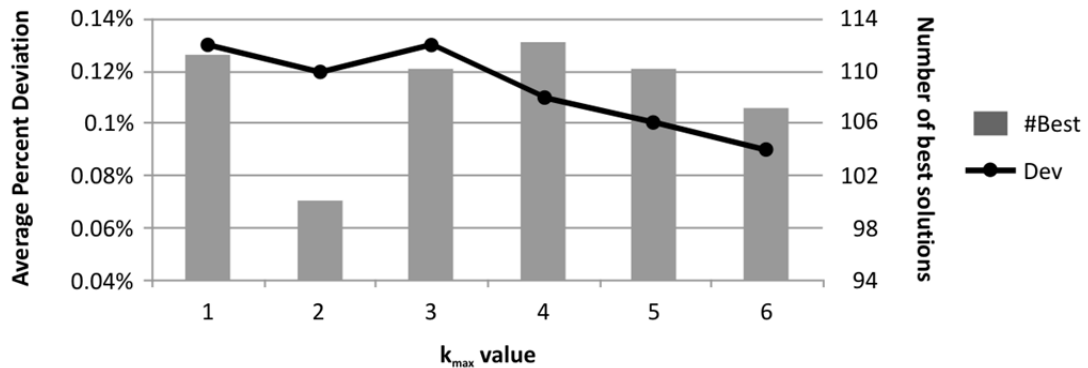
The goal of the first preliminary experiment is to identify the best combination of constructive and improvement methods. Specifically, we couple WJ and the three variants of the greedy-based construction procedures (GC, GC-FULL and GC-Tabu) with the improvement methods LCW, BI and FI as well as with their tabu counterparts (T-LCW, T-BI and T-FI). The tabu parameters *tabuTenure* and *maxIter* are set to  $0.1M$  and  $0.5M$  respectively according to the results of a preliminary experiment not reported here for the sake of brevity. Table 5 summarizes the results, where the construction methods are in rows and the improvement procedures in columns. The results in bold in Table 5 identify the best construction/improvement combination with and without memory structures.

The results in Table 5 show that the best outcomes are obtained when the construction method GC is coupled with T-LCW. This combination results in the smallest average deviation (0.13%) and the largest number of best solutions (103) among all the combinations considered in the experiment. Moreover, the results show that the tabu version of the improvement methods outperforms the straight local search versions, regardless of the construction method used. For example, while GC+LCW achieves an average percent deviation of 0.84%, GC+T-LCW's deviation is only 0.13%.

Construction		Improvement					
		Without memory			With memory		
		LCW	BI	FI	T-LCW	T-BI	T-FI
GC	<i>Dev</i>	<b>0.84%</b>	0.90%	0.89%	<b>0.13%</b>	0.13%	0.14%
	<i>#Best</i>	<b>82</b>	81	82	<b>103</b>	100	99
GC-FULL	<i>Dev</i>	0.86%	0.88%	0.87%	0.13%	0.13%	0.15%
	<i>#Best</i>	79	82	84	99	100	97
GC-Tabu	<i>Dev</i>	0.88%	0.91%	0.90%	0.14%	0.15%	0.16%
	<i>#Best</i>	79	79	78	97	97	94
WJ	<i>Dev</i>	4.31%	4.30%	3.69%	0.62%	0.55%	0.20%
	<i>#Best</i>	26	28	49	57	58	94

**Table 5.** Construction plus improvement multi-start methods

The second preliminary experiment studies the effect of the  $k_{max}$  parameter associated with the strategic oscillation method. In particular, building from the results of our previous experiments, we apply the strategic oscillation procedure employing GC constructions and T-LCW improvements. For the sake of simplicity, we refer to this procedure as SO. Figure 7 shows the *Dev* and *#Best* values for SO with  $k_{max} = 1, \dots, 6$ . The figure shows that SO with  $k_{max} = 4$  obtains the largest number of best solutions (*#Best* = 112) and a relatively robust average deviation value (*Dev* = 0.11%). Based on these results, we choose  $k_{max} = 4$  for all remaining experiments involving SO.



**Figure 7.** Preliminary experiment with SO

We have obtained the parameters values through preliminary experimentation with a training set of 180 instances, which as indicated before, it consists of a subset of all available instances. To test performance, we employ the set of 300 instances that we did not use for calibration purposes in the preliminary experiments reported above. The experiment consists of comparing the following procedures:

- LSGA: implemented in Java as described by Fan et al. (2011)
- LCW: implemented with random restarts as described by Weitz and Lakshminarayanan (1998)
- T-LCW: GC+T-LCW with  $tabuTenure = 0.1M$  and  $maxIter = 0.5M$
- SO: strategic oscillation with GC restarts, T-LCW improvements and  $k_{max} = 4$

The outcome of this experiment is presented in Tables 6 to 9 and in Figures 8 and 9. Results in these tables are obtained with 240 of the 300 instances in our test set (5 EGS instances and 5 DGS instances for  $M = 10$  to 960). Figures 8 and 9 were built with the remaining 60 instances (30 with  $M = 480$  and 30 with  $M = 960$ ).

Tables 6 to 9 compare the performance of the four procedures listed above on the basis of the values of *Dev* and *#Best*. In addition, we report the *Score* achieved by each method. As formulated by Ribero et al. (2002), the *Score* for a particular set of instances is the number of methods that obtained results that are strictly better than those obtained by the method being evaluated, and hence, the lower the *Score* the better the method. The minimum *Score* is zero while the maximum *Score* is given by the product of the number of instances in the group and the number of competing methods minus one (i.e., the one being scored). The number of instances in each group is indicated as a reference point to interpret the *Score* values. In order to make a fair comparison, all procedures were executed for the same amount of time, which depends on the problem size, as indicated in Table 4.

Table 6 summarizes the results of the entire experiment. These results show that when considering the three metrics of *Dev*, *#Best* and *Score*, both T-LCW and SO outperform the existing procedures. It is also evident that there is a performance difference between T-LCW and SO that indicates the advantage of the search mechanisms embedded in the SO implementation.

Method	<i>Dev</i>	<i>#Best</i>	<i>Score</i>
LSGA	0.61%	82	339
LCW	1.01%	80	438
T-LCW	0.17%	141	108
SO	0.04%	192	55

**Table 6.** Summary of results for 240 instances

In order to provide additional insight into the information in Table 6, we calculate the performance metrics for subsets of the 240 instances according to three different criteria. Specifically, Table 7 shows the results by problem size; Table 8 summarizes the performance metrics according to problem type (EGS and DGS); and Table 9 partitions the results according to data set (*RanReal*, *RanInt* and *Geo*).

In Table 7 we observe that differences in performance between the existing methods (LSGA and LCW) and the proposed procedures (T-LCW and SO) increase with problem size. For instance, in the subset with the largest instances ( $M = 480$  & 960), SO outperforms the existing procedures by at least an order of magnitude across all the proposed performance metrics.

Table 8 shows that T-LCW and SO obtain better results than LSGA and LCW on both type of instances (EGS and DGS). This is an interesting result because both T-LCW and SO are designed to exploit the MDGP2 characteristic that allows for the size of the groups to vary (see DGS results). However, they compete well against LCW, a procedure that was originally designed to tackle instances of the MDGP1 for which the groups must have equal size (see EGS results).

Instance Size	Method	<i>Dev</i>	<i>#Best</i>	<i>Score</i>
$M \leq 60$ (120 instances)	LSGA	0.08%	82	77
	LCW	0.28%	80	110
	T-LCW	0.01%	107	16
	SO	0.01%	106	17
$M = 120$ & 240 (60 instances)	LSGA	1.04%	0	119
	LCW	1.91%	0	172
	T-LCW	0.30%	19	47
	SO	0.08%	41	22
$M = 480$ & 960 (60 instances)	LSGA	1.25%	0	143
	LCW	1.58%	0	156
	T-LCW	0.34%	15	45
	SO	0.06%	45	16

**Table 7.** Results by problem size

Type	Method	<i>Dev</i>	<i>#Best</i>	<i>Score</i>
EGS (120 instances)	LSGA	0.37%	43	179
	LCW	0.41%	43	191
	T-LCW	0.16%	71	57
	SO	0.01%	97	25
DGS (120 instances)	LSGA	0.85%	39	160
	LCW	1.61%	37	247
	T-LCW	0.17%	70	51
	SO	0.07%	95	30

**Table 8.** Results by problem type

Data Set	Method	<i>Dev</i>	<i>#Best</i>	<i>Score</i>
<i>RanReal</i> (80 instances)	LSGA	0.79%	29	111
	LCW	1.17%	28	142
	T-LCW	0.23%	40	43
	SO	0.02%	71	9
<i>RanInt</i> (80 instances)	LSGA	0.84%	29	111
	LCW	1.20%	28	143
	T-LCW	0.26%	38	47
	SO	0.01%	75	5
<i>Geo</i> (80 instances)	LSGA	0.21%	24	117
	LCW	0.66%	24	153
	T-LCW	0.01%	63	18
	SO	0.08%	46	41

**Table 9.** Results by data set

Results in Table 9 show the summary of the performance metrics when considering each data set (*RanReal*, *RanInt* and *Geo*) separately. Interestingly, the dominance that the strategic oscillation variant exhibits in the results shown in Tables 6-8 is not present when *Geo* instances are isolated (as shown in Table 9). This is the only case in which SO does not outperform all other methods, with T-LCW exhibiting a more robust behavior. This seems to indicate that the solution space of the *Geo* instances is such that it favors procedures with a relatively large number of restarts during the allotted execution time. We point out that for a given solution time, the number of T-LCW restarts is at least an order of magnitude larger than the SO restarts.

With the goal of supporting our conclusions about the performance of the proposed procedures, we performed three statistical tests. First, we applied the *non-parametric Friedman test* for multiple correlated samples to the best solutions obtained by each of the 4 methods. This test computes, for each instance, the rank value of each method according to solution quality (where rank 4 is assigned to the best method and rank 1 to the worst). Then, it calculates the average rank values for each method across all instances. If the averages differ greatly, the associated *p*-value or level of significance is small. The resulting *p*-value of 0.000 obtained in this experiment clearly indicates that there are statistically significant differences among the 4 methods. The rank values produced by this test are 3.23 (SO), 3.02 (T-LCW), 2.08 (LSGA) and 1.67 (LCW).

Second, we employed the *Wilcoxon test* and *Sign test* to make a pairwise comparison of SO and T-LCW, which consistently provide the best solutions in our experiments. The results of the *Wilcoxon* test (with a *p*-value of 0.000) determined that the solutions obtained by the two methods indeed represent two different populations. The *Sign* test (with a *p*-value of 0.000) indicated that the solutions obtained with SO tend to be better (i.e., larger) than those obtained with T-LCW.

An interesting feature of the strategic oscillation strategy relates to its ability to add a long term component to the search. Typically, searches that include strategic oscillation are capable of finding improved solutions late in the search, while maintaining an aggressive search trajectory early in the search. In other words, when implemented carefully, the strategic oscillation strategy adds a diversification component that complements the search intensification that is typical to procedures based on making moves selected via a thorough exploration of effective neighborhood structures. Figure 8 shows the progression of the average percent deviation of the best solutions found by four methods (T-LCW, LSGA, LCW and SO) for 30 problem instances with  $M = 480$  during 120 seconds of search time. The deviation values are calculated against the best-known solutions and are plotted on a logarithmic scale. Figure 9 shows a similar plot for 30 instances with  $M = 960$  and 600-second runs.

Figure 8 and 9 show how most improvements on the best solutions are achieved early in the search (i.e., within 10 percent of the total search time, corresponding to 12 seconds in Figure 8 and 60 seconds in Figure 9). After that point, LSGA and LCW stagnate, while T-LCW makes some minor additional progress toward improving the incumbent solutions. In contrast, SO exhibits an improving trajectory throughout the entire search horizon.

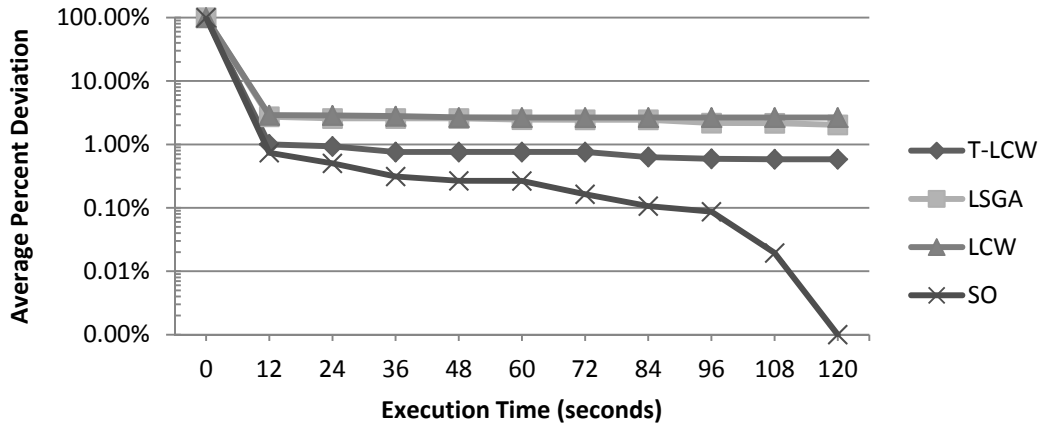


Figure 8. Best-solution profile for a 120-second run on the  $M = 480$  instances

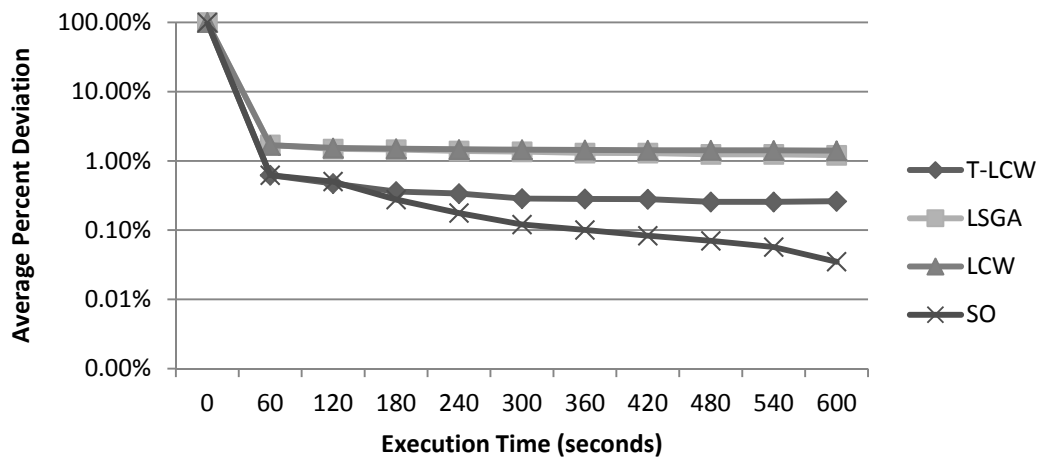


Figure 9. Best-solution profile for a 600-second run on the  $M = 960$  instances

## 5. Conclusions

The MDGP is a difficult combinatorial optimization problem and a perfect platform to study the effectiveness of search mechanisms. Of particular interest in our work has been testing the effect of expanding search neighborhoods, by including additional moves, and search spaces, by allowing the search to visit infeasible solutions. Through extensive experimentation, we have been able to determine the benefits of adding enhanced search strategies to basic procedures. We purposefully added these mechanisms sequentially in order to measure their effects and studied the combinations that resulted in effective solution procedures with improved outcomes. We believe that our findings can be translated to other settings and will help in the development of more robust searches of combinatorial spaces.

## Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Ref. TIN2009-07516) and by the University Rey Juan Carlos (in the program “Ayudas a la Movilidad 2010”).

## References

- Arani, T. and V. Lotfi (1989) “A three phased approach to final exam scheduling,” *IIE Transactions*, vol. 21, pp. 86-96.
- Bhadury, J., E. J. Mighty and H. Damar (2000) “Maximizing workforce diversity in project teams: A network flow approach,” *Omega*, vol. 28, pp. 143–153.
- Chen, C. C. (1986) “Placement and partitioning methods for integrated circuit layout,” Ph.D. Dissertation, EECS Department, University of California, Berkeley.
- Chen, Y., Z. P. Fan, J. Ma, S. Zeng (2011) “A hybrid grouping genetic algorithm for reviewer group construction problem”, *Expert Systems with Applications*, vol. 38, pp. 2401-2411.
- Falkenauer, E. (1998) *Genetic Algorithms for Grouping Problems*, Wiley: New York.
- Fan, Z. P., Y. Chen, J. Ma and S. Zeng (2011) “A hybrid genetic algorithmic approach to the maximally diverse grouping problem”, *Journal of the Operational Research Society*, vol. 62, pp. 92-99.
- Feo, T., O. Goldschmidt and M. Khellaf (1992) “One-half approximation algorithms for the k-partition problem,” *Operations Research*, 1992, vol. 40, pp. S170-S173.
- Feo, T. and M. Khellaf (1990) “A class of bounded approximation algorithms for graph partitioning,” *Networks*, vol. 20, pp. 181-195.
- Glover, F. and Laguna, M. (1997) *Tabu Search*, Kluwer Academic Publisher: Boston.
- Glover, F., C. C. Kuo and K. S. Dhir (1998) “Heuristic algorithms for the maximum diversity problem,” *Journal of Information and Optimization Sciences*, vol. 19, no. 1, pp. 109–132.
- Hettich S. and M. J. Pazzani (2006) “Mining for element reviewers: Lessons learned at the national science foundation,” In: *Proceedings of the KDD’06*, ACM: New York, NY, pp. 862–871.
- Kral, J. (1965) “To the problem of segmentation of a program,” *Information Processing Machines*, vol. 2, pp. 116-127.
- Lotfi V. and R. Cerveny (1991) “A final exam scheduling package,” *Journal of the Operational Research Society*, vol. 42, pp. 205-216.
- Miller, J., W. Potter, R. Gandham and C. Lapena (1993) “An evaluation of local improvement operators for genetic algorithms,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 23, no. 5, pp. 1340–1351.
- Mingers, J. and F. A. O’Brien (1995) “Creating students groups with similar characteristics: a heuristic approach,” *Omega*, vol. 23, pp. 313-321.

- O'Brien, F. A. and J. Mingers (1995) "The equitable partitioning problem: a heuristic algorithm applied to the allocation of university student accommodation," Warwick Business School, Research Paper no. 187.
- Ribeiro, C. C., E. Uchoa and R. F. Werneck (2002) "A hybrid GRASP with perturbations for the Steiner problem in graphs," *INFORMS Journal on Computing*, vol. 14, pp. 228–246.
- Vasko, F. J., P. J. Knolle and D. S. Spiegel (2005) "An empirical study of hybrid genetic algorithms for the set covering problem," *The Journal of the Operational Research Society*, vol. 56, pp. 1213–1223.
- Weitz, R. R. and M. T. Jelassi (1992) "Assigning students to groups: a multi-criteria decision support system approach," *Decision Sciences*, vol. 23, no. 3, pp. 746-757.
- Weitz, R. R. and S. Lakshminarayanan (1996) "On a heuristic for the final exam scheduling problem," *Journal of the Operational Research Society*, vol. 47, no. 4, pp. 599-600.
- Weitz, R. R. and S. Lakshminarayanan (1997) "An empirical comparison of heuristic and graph theoretic methods for creating maximally diverse groups, VLSI design, and exam scheduling" *Omega*, vol. 25, no. 4, pp. 473-482.
- Weitz, R. R. and S. Lakshminarayanan (1998) "An empirical comparison of heuristic methods for creating maximally diverse groups," *Journal of the Operational Research Society*, vol. 49, no. 6, pp. 635-646.