Journal of the Operational Research Society (2011) 62, 1423–1430

www.palgrave-journals.com/jors/

Erratum

A hybrid genetic algorithmic approach to the maximally diverse grouping problem

ZP Fan[1], Y Chen[2]*, J Ma[3] and S Zeng[4]

[1]Northeastern University, Shenyang, China; [2]Shanghai University of Finance & Economics, Shanghai, China; [3]City University of Hong Kong, Kowloon, Hong Kong; and [4]University of Arizona, USA

Correction to: *Journal of the Operational Research Society* (2010). doi:10.1057/jors.2009.168
Published online 6 January 2010

It has come to our notice that several aspects of the above paper were incorrect. The correct version of the paper is reproduced here.

The maximally diverse grouping problem (MDGP) is a NP-complete problem. For such NP-complete problems, heuristics play a major role in searching for solutions. Most of the heuristics for MDGP focus on the equal group-size situation. In this paper, we develop a genetic algorithm (GA)-based hybrid heuristic to solve this problem considering not only the equal group-size situation but also the different group-size situation. The performance of the algorithm is compared with the established Lotfi–Cerveny–Weitz algorithm and the non-hybrid GA. Computational experience indicates that the proposed GA-based hybrid algorithm is a good tool for solving MDGP. Moreover, it can be easily modified to solve other equivalent problems.

Journal of the Operational Research Society (2011) **62,** 1423–1430. doi:10.1057/jors.2010.92
Published online 23 June 2010

Keywords: genetic algorithm; maximally diverse grouping problem; local neighbourhood search

Introduction

The maximally diverse grouping problem (MDGP) refers to the problem of grouping a given set of elements into several mutually disjoint subsets (groups) to maximize the overall diversity between elements. It arises in a wide range of real-world settings. Firstly stressed by Weitz and Jelassi (1992) in their work on forming student workgroups, MDGP has a variety of applications in different contexts, such as education, industry, scientific funding agencies or organizations. In business schools, it becomes increasingly co mmon to create diverse student workgroups or training teams in order to provide students a diverse environment (Weitz and Lakshminarayanan, 1998). Moreover, in the very large scale integration (VLSI) design, it needs to group highly connected modules onto the same circuit. Because of the mathematical identity, VLSI design is considered as an application of MDGP (Weitz and Lakshminarayanan, 1998). The MDGP also appears in the project selection in scientific funding agencies. In most cases, the decisions of project selection are made based on the review results of peer reviewers. So, forming the diverse group of reviewers is desired by the scientific funding agencies to insure multiple perspectives are represented in the review process (Hettich and Pazzani, 2006). In addition, the workforce diversity is an increasing common phenomenon in most organizations today. Bhadury *et al* (2000) suggested a way of dealing with this heterogeneity by making people with different backgrounds work on common projects so as to facilitate their understanding and communication.

The MDGP can be formally stated as follows. Consider $P = \{p_i : i \in \{1, 2, \ldots, M\}\}$ a set of elements, and p_{ik}, $k \in \{1, 2, \ldots, K\}$, the K attributes associated with each element p_i. Let u_g denote the gth group, where $g \in \{1, 2, \ldots, G\}$. The size of each group u_g is in the interval $[a_g, b_g](a_g \leqslant b_g)$. Let d_{ij} be the distance between elements p_i and p_j, which is measured by a function applied on their attributes. For example, $d_{ij} = \sum_{k=1}^{K} |p_{ik} - p_{jk}|$. It should be noticed that the function can be defined according to requirements of

*Correspondence: Y Chen, Department of Information Management and Engineering, Shanghai University of Finance & Economics, Shanghai, 200433, China.

practical problems. The problem then consists of partitioning M elements in P into G groups in order to maximize the sum of the distances between the elements:

$$\text{Maximize} \quad Z = \sum_{g=1}^{G} \sum_{i=1}^{M} \sum_{j>i}^{M} d_{ij} x_{ig} x_{jg} \tag{1}$$

$$\text{subject to} \quad \sum_{g=1}^{G} x_{ig} = 1 \quad \text{for } i = 1, 2, \ldots, M \tag{2}$$

$$\sum_{i=1}^{M} x_{ig} \geqslant a_g \quad \text{for } g = 1, 2, \ldots, G \tag{3}$$

$$\sum_{i=1}^{M} x_{ig} \leqslant b_g \quad \text{for } g = 1, 2, \ldots, G \tag{4}$$

$$x_{ig} \in \{0, 1\} \quad \text{for } i = 1, 2, \ldots, M \quad \text{and}$$

$$g = 1, 2, \ldots, G \tag{5}$$

where $x_{ig} = 1$ indicates that the element p_i has been allocated to group u_g. It should be mentioned that this formulation covers two group-size situations of MDGP: one is the equal group size, and another is the different group size. According to our literature review, most of the related studies focus on the MDGP with equal group size. For the MDGP with different group size, to our knowledge, has not been stressed until now.

Feo and Khellaf (1990) showed that the MDGP is NP-complete. Several researchers focused on solving MDGP using heuristics. Heuristics that can be used to solve MDGP were developed by Weitz and Jelassi (1992), Weitz and Lakshminarayanan (1996, 1998), Feo *et al* (1992), Lotfi and Cerveny (1991), Arani and Lotfi (1989), Baker and Benn (2001). Baker and Powell (2002) identified two basic procedures of these heuristics. One is the construction procedure and another is the improvement procedure. Weitz and Lakshminarayanan (1998) compared five heuristics, and found that method of Lotfi–Cerveny–Weitz (LCW), which is provided by Weitz and Lakshminarayanan (1998) based on the algorithm proposed by Lotfi and Cerveny (1991), is the best performer. As mentioned above, most of these heuristics were proposed to solve the MDGP with same group size.

As one kind of heuristic, the genetic algorithm (GA) concept was developed by Holland (1975). Whitley *et al* (1991) presented applications of GA to the travelling salesman problem. Davis (1985), Shi (1997), Etiler *et al* (2004) discussed the use of GA for job shop scheduling problems. Obviously, the wide applications of GA make it a popular tool to solve the problems with computation difficulties such as NP-complete problems. However, it is still a new challenge to use GA to solve the MDGP. Therefore, the research objective of this paper is to propose a GA to solve the MDGP considering both of the same group-size situation and the different group-size situation in order to provide useful addition to the knowledge base of solutions for this type of problem.

In this paper, the concepts of GA and local search (LS) for MDGP will first be described. Then the GA-based hybrid heuristic based on the special knowledge of the MDGP will be proposed and applied to MDGP. The following section analyses the performance of the GA-based heuristic by experiments. The data in experiments are randomly generated. Finally, conclusions are discussed.

GA-based heuristic

Genetic algorithm

GA has been used successfully to find solutions for a wide variety of optimization problems (eg, Goldberg, 1989; Gen and Cheng, 1997), since its introduction by Holland (1985). GA is an intelligent stochastic optimization technique based on the simulation of biological evolution. Azimi (2005) provided a brief introduction of GA.

In general, GA often performs well in global search, but is relatively slow in converging to local optima (Wang and Wu, 2003, 2004). On the other hand, another heuristic, the LS method can find the local optimum in a small region of the search space, but they are typically poor in a global search. Therefore, various strategies of hybridization have been suggested to improve performance of simple GA. Mühlenbein (1992) proved theoretically and Gorges-Schleuter (1989) demonstrated empirically that LS can play a key role in GA. Miller *et al* (1993) and Vasko *et al* (2005) also discussed the combination of local improvement operators in GA.

Local search for MDGP

LS is a widely used, general approach to solving hard optimization problems (Yannakakis, 1990). Its basic idea is to start from some initial solution that is constructed by some other algorithms, or just generated randomly and from then on it keeps moving to a better neighbouring solution, as long as there is one, until finally it terminates at a locally optimal solution, one that does not have a better neighbour. LS can be dated back to the late 1950s, when Bock (1958) and Croes (1958) developed their link-exchange procedures for the travelling salesman problem. Ever since, a large variety of LS algorithms have been proposed (eg, Vaessens *et al*, 1998; Aarts and Lenstra, 2003).

Our LS for MDGP is an iterative procedure. In each iteration, the procedure scans all neighbour solutions and selects the best to replace the current solution. The neighbour solution is obtained by swapping any two elements from different groups of the current solution. This procedure was also

described as an improvement procedure by Baker and Powell (2002). The LS for MDGP is provided in the Appendix.

The GA-based heuristic for MDGP

Encoding

A variety of encoding schemes have been developed based on the characteristics of problems. For grouping problems, Falkenauer (1998) proposed a special encoding scheme. His encoding scheme consists of two sections: the object section and the group section. The group section is appended to the object section, and takes part in the crossover and mutation operations.

MDGP is basically composed of two types of information, the elements and the groups. Falkenauer's (1998) encoding scheme comprises both of the elements' information and the groups' information. So, it is naturally adopted in our algorithm. In the object section, each gene is assigned to a group number, which indicates that the corresponding element is allocated to that group. In the group section, all the group numbers are listed in random order. The following notation is used for our chromosome representation:

$$q_1, q_2, \ldots, q_M | Q_1, Q_2, \ldots, Q_G$$

where $q_i \in \{1, 2, \ldots, G\}$denote the element p_i is allocated to group q_i $(1 \leqslant i \leqslant M)$, and Q_g is the existing group numbers $(1 \leqslant g \leqslant G)$. For example, the chromosome 132 213|123 indicates the grouping of allocating element p_1 and p_5 to group u_1, element p_3 and p_4 to group u_2 and element p_2 and p_6 to group u_3.

Initialization

Davis and Steenstrup (1987) have noted that the initialization process can be executed with either a randomly created population or a well-adapted population. For the MDGP, because of the existence of constraints, the random way will yield a considerable number of infeasible solutions. So, in our algorithm, a two-stage heuristic is proposed to initialize the population. In the first stage, the groups are constructed by consequently allocating the elements into groups. Each element is allocated to only one group. If M/G is an integer, all the groups have the same group size of M/G; otherwise the first $(M-1)$ groups have the same group size of $\lfloor M/G \rfloor$, and the last group has the group size of $(M - \lfloor M/G \rfloor G)$. After the first stage, the constraints in Equation (2) are guaranteed, but other constraints may be violated. In the second stage, the *group size adjustment algorithm* is performed by moving elements from one group to another until each group size is in the corresponding range. It should be noted that the *group size adjustment algorithm* will also be performed to repair the illegal offspring derived from the crossover operation. Let pop_size denote the size of the population, and v_k denote the kth chromosome, and $S(g)$ denote the size of group u_g and S_more denote the set of groups where group size is larger than b_g, and S_less denote the set of groups where group size is smaller than a_g. The procedure that generates the initial population follows:

Stage 1

for $(k = 1$ to $pop\_size)$
  if $(mod(M, G) = 0)$
    $S = M/G$
    for $(g = 1$ to $G)$
      Randomly select S elements from the rest $(M - S \times (g-1))$ elements into group u_g
    end
  else
    $S =$ floor (M/G)
    for $(g = 1$ to $(G-1))$
      Randomly select S elements from the rest $(M - S \times (g-1))$ elements into group u_g
    end
    $g = G$
    Put the rest $(M - S \times (G-1))$ elements into group u_g
  end
  Randomly sort the order of G groups
  Create the v_k according to above assignment
end

Stage 2

In v_k:
for $(g = 1$ to $G)$
  if $(S(g) < a_g)$
    Put u_g into S_less
  else if $(S(g) > b_g)$
    Put u_g into S_more
  end
end
if $((S\_more =$ null$)$ & $(S\_less = null))$
  break
  if $(S\_more \neq$ null$)$
    $gmore =$ The index of randomly selected groups from $S\_$ *more*
else
    $gmore =$ The index of randomly selected groups where $S(g) \geqslant (a_g + 1)$
  end
  if $(S\_less \neq$ null$)$
    $gless =$ The index of randomly selected groups from S_less
  else
    $gless =$ The index of randomly selected groups where $S(g) \leqslant (b_g - 1)$
  end
  Randomly select a element from group u_{gmore}
  Put this element into group u_{gless}
  Find out S_more and S_less from all G groups
end

Selection

Our hybrid GA employs a rank-based roulette-wheel strategy to select parents for crossover operation. Each chromosome is assigned a portion of an imaginary roulette wheel, based on its rank-based fitness value. The fitness value of each chromosome is calculated with the objective function. Chromosomes that are more fit have higher rank-based fitness values and thus receive a relatively larger proportion of the roulette wheel. Prior to the selection of each parent, a percentage between 0 and 100 is randomly generated. The chromosome occupying the part of the roulette wheel covered by the randomly generated percentage is chosen as parent one. Parent two is selected in the same manner.

Crossover

Crossover is a step that really powers the GA. It produces new individuals by combining the information contained in parents. Standard GA crossover produces offspring by exchanging portions of parent chromosomes. As pointed out by Falkenauer (1998), this type of operation, when applied to a grouping problem, may produce overlapping among the groups. It means an offspring chromosome may contain one or more groups that share some of the same elements. This type of offspring would be invalid since grouping problems require that each element be assigned with only one group. So, a special crossover operation for grouping problem was proposed by Falkenauer (1998).

In our hybrid GA, the crossover operation is consistent with Falkenauer's (1998). Firstly, select two crossing points from the group section in each of two parents. Then inject the contents between the crossing points of parent two into the first crossing point of the parent one. In the following, delete all elements that occur twice after injection from the groups of parent one. After the crossover procedure, the generated offspring may be still illegal since the group size could be out of the corresponding range. In this situation, the *group size adjustment algorithm* will be performed again to adjust the group size.

An example will be presented to illustrate the crossover and repair operation in our hybrid GA. Suppose there are 10 elements to be partitioned into three groups. The requirements for group size are $a_1 = 3$, $b_1 = 6$, $a_2 = 2$, $b_2 = 4$, $a_3 = 3$ and $b_3 = 5$. Two legal parents are generated as shown in Figure 1. Recall that the crossover operation is performed on the group section of the parent chromosomes. The offspring in our example results from injecting the content between the crossing points from parent two into the first cross point of parent one. Since the injected groups are numbered the same as those in the first parent, we use gray colour to indicate which groups are injected from parent two (Figure 1(a)).

Figure 1(b) shows the group composition details of the offspring. After the crossover, it can be observed that some elements appear twice. To indicate them, we underline these

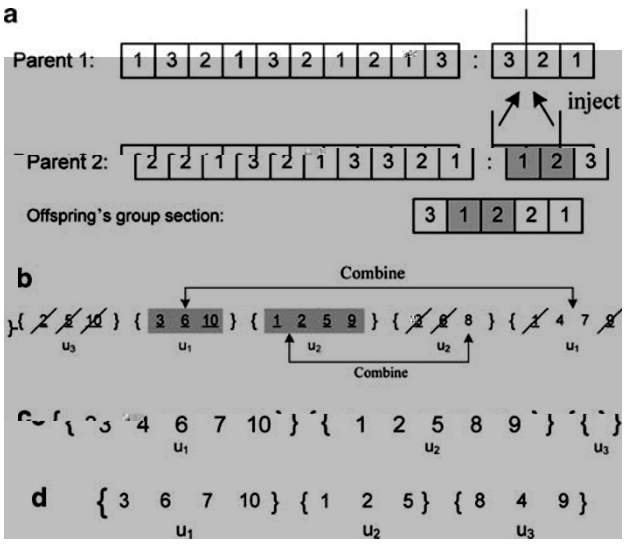

Figure 1 (a) The injection of parent two to parent one in the crossover operation; (b) The elimination of elements appearing twice; (c) The resulting groups; (d) The composition of groups after crossover and repair.

elements. Obviously, the resulting offspring is illegal. To repair it, we firstly delete those underlined elements from the groups of the first parent. The deleted elements are further identified by inclined lines (Figure 1(b)). Then, we have five groups, three groups from parent one and two groups from parent two. For our case, we only need three groups. So, any two groups will be combined with other groups. The resulting groups are shown in Figure 1(c).

From Figure 1(c), we can see that the group size of group u_2 and u_3 are out of the range as predefined. Therefore, according to our *group size adjustment* algorithm, we randomly select three elements from group u_1 and group u_2 to fill in group u_3. Ultimately, the composition of the three groups of the offspring is shown in Figure 1(d).

Mutation

The aim of mutation is to increase genetic diversity into the population by introducing random variations into the members of the population. This standard mutation of GA is disruptive for grouping problems (Falkenauer, 1998). If an element is transferred into a new group in which it shares no similarities with current group members, then the quality of the mutated solution may be greatly impacted. Falkenauer (1998) proposed several mutation strategies like creating a new group, eliminating an existing group or shuffling a few elements among their respective groups.

The mutation operator used in our hybrid GA consists of randomly selecting two groups and then selecting β elements that belong to these groups, where β is the number of genes that are mutated. The mutation involves swapping the β elements into another chosen group. Continuing our example, suppose $\beta = 1$ and the randomly selected groups

are group u_1 and group u_3. Then, two elements are randomly selected and switched, say p_6 in group u_1 and p_4 in group u_3. (See Figure 2).

Termination criterion

The most straightforward stopping criterion for a GA is the number of generations (iteration). If the number of generations is low, the probability of finding the best result is low. Otherwise if the number of generations is too high, the iteration time is too long.

For our hybrid GA, the iteration number of both of the LS and the GA should be considered. Let g_LS denote the iteration number of LS and g_GA denote the iteration number of GA. In order to determine the termination criterion, we conducted some test experiments by solving different sizes of problem at $g\_LS = 1$ and $g\_GA = 30$ and we found that the Z_{max} (the maximum value of Z in Equation (1)) stable at less than 15 generations (See Figure 3). Therefore, we chose 20 as the termination criterion.

Computational results

Extensive computational studies on the proposed hybrid GA-based heuristic for MDGP have been carried out. The purpose of these experiments is to evaluate the computational effort and the performance of our hybrid GA across a variety of situations. Two comparisons are made over these situations.

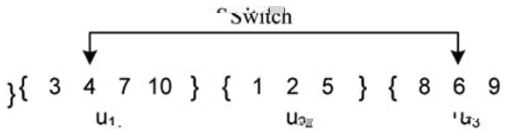

Figure 2 The mutation operation.

Firstly, the hybrid GA is compared with the established LCW under the situation that each group has the same group size. Since LCW is identified by Weitz and Lakshminarayanan (1998) as the overall best performer of the five heuristics in their experiments, in which, the group size is set as the same, it is used in our experiments as a benchmark. Our hybrid GA is also compared with the non-hybrid GA under the situations that each group has the different group size. This comparison is designed to observe the function of LS in the GA. In order to clearly distinguish these three algorithms, we use LS_GA to denote the proposed hybrid GA, GA to denote the non-hybrid GA and maintain denotation of the LCW.

The parameter settings used for LS_GA include a population size of 30, crossover rate of 1.0, and termination criterion of 20 generations. These parameters are chosen based on their success in preliminary testings. Such testing is also conducted to determine the appropriate values for the mutation probability and β. After testing several values of both parameters, we choose mutation probability 0.01 and $\beta = 1$.

To test the proposed LS_GA, we develop the experiments comprising six problem settings. For each problem setting, we further generate two categories of problems. In one category, the size of all groups is the same. In another one, the size of each group is in the predefined range. The data of d_{ij} is randomly simulated using a Uniform distribution U(0, 100) (the notation U(a,b) represents uniform random numbers between a and b). Noticed that the diversity between elements would influence the solution of LCW, for each problem setting, we generated D nine times, where $D = [d_{ij}]_{M\times M}$. We summarize the test problems in Table 1.

All of the heuristics are programmed in Matlab and run on a Windows PC with a 1.29 GHz Pentium R CPU. Each run averaged less than 2 s of CPU time for the M = 10 and G = 2

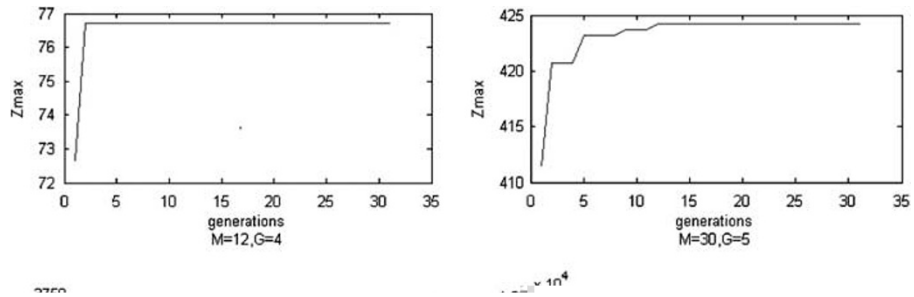

Figure 3 Improvements in the objective compared to the number of generations.

Table 1 Summary of problem settings tests

Problem setting	*M*	*G*	*Same group size*		*Different group size*		
			a_g	b_g	a_g	b_g	$b_g - a_g$
1	10	2	5	5	3	7	4
2	12	4	3	3	2	5	3
3	30	5	6	6	5	10	5
4	60	6	10	10	7	14	7
5	120	10	12	12	8	16	8
6	240	12	20	20	15	25	10

Table 2 Comparison of LNS_GA with LCW for Problem 1

Variance of D	*Average LCW solution*	*Average LNS_GA solution*	Z_{LNS_GA}/Z_{LCW}
1	1170.2	1173.9	1.003
2	1068.4	1068.9	1.000
3	1414.4	1442.5	1.020
4	1215.1	1235	1.016
5	1136	1141.2	1.002
6	1030.5	1033	1.002
7	1195	1196.8	1.008
8	1028	1208	1.175
9	957.8	966.0315	1.009
10	1309.4	1320	1.008
Averages	1152.48	1178.533	1.023

Table 3 Comparison of LNS_GA with LCW for Problem 2

Variance of D	*Average LCW solution*	*Average LNS_GA solution*	Z_{LNS_GA}/Z_{LCW}
1	775	815.5	1.052
2	660.6	718.6	1.088
3	819.5	856.7	1.045
4	612.9	623.7	1.018
5	778.5	850.5	1.092
6	828.9	849.3	1.025
7	757.4	803.2	1.060
8	701.2	772.9	1.102
9	714.7	738.3	1.033
10	641.2	646.2	1.008
Averages	728.99	767.495	1.053

problems, to less than 17 min of CPU time for the M = 240 and G = 12 problems.

For each run, the best solution is recorded and the average best solution over 10 runs is calculated. Tables 2–7 display the comparison of our LS_GA with LCW for the six problem settings. Table 8 shows the comparison of our LS_GA with GA for six problem settings.

The average relative performances of LS_GA and LCW for the six problems are 1.023, 1.053, 1.055, 1.076, 1.039, 1.028, respectively, and those of LS_GA and GA for the six problems are 1.004, 1.125, 1.055, 1.061, 1.069, and 1.048,

Table 4 Comparison of LNS_GA with LCW for Problem 3

Variance of D	*Average LCW solution*	*Average LNS_GA solution*	Z_{LNS_GA}/Z_{LCW}
1	3686	3884	1.054
2	4202.6	4356.4	1.037
3	4296.7	4481.9	1.043
4	4191.8	4382.9	1.046
5	4612.6	4965.8	1.077
6	3926.7	4169.4	1.062
7	3947	4152.7	1.052
8	4297.6	4510	1.049
9	4085.1	4371	1.070
10	3980.4	4212	1.058
Averages	4122.65	4348.61	1.055

Table 5 Comparison of LNS_GA with LCW for Problem 4

Variance of D	*Average LCW solution*	*Average LNS_GA solution*	Z_{LNS_GA}/Z_{LCW}
1	8790.8	9324.7	1.061
2	8284.7	9101.3	1.099
3	7912.4	8613.2	1.089
4	7909.4	8464	1.070
5	8255.4	8717.9	1.056
6	8382.4	8862.8	1.057
7	8215.5	8805	1.072
8	7582.7	8513.4	1.123
9	7713.6	8295.1	1.075
10	7676.9	8188.2	1.067
Averages	8072.38	8688.56	1.076

Table 6 Comparison of LNS_GA with LCW for Problem 5

Variance of D	*Average LCW solution*	*Average LNS_GA solution*	Z_{LNS_GA}/Z_{LCW}
1	36 032	37 230	1.033
2	36 478	37 831	1.037
3	35 810	37 155	1.038
4	36 369	37 530	1.032
5	33 279	34 690	1.042
6	34 840	36 213	1.039
7	35 600	37 446	1.052
8	33 226	34 532	1.039
9	35 622	37 094	1.041
10	37 525	39 057	1.041
Averages	35 478.1	36 877.8	1.039

respectively. The *t*-test indicates that except the smallest Problem (M = 10, G = 2), the LS_GA is significantly better ($\alpha^{LCW}_{LNS\_GA} \in \{0.294, 0.079, 0.001, 0.036, 0.007\}$) than the LCW (See Table 9). The exception is consistent with our instinctive judgment that it is easier for the LCW heuristic to get the global optimal solution for the very small size and uncomplicated cases. In Table 9, from Problem 3 to Problem 6, the LS_GA has significantly better performance than the

Table 7 Comparison of LNS_GA with LCW for Problem 6

Variance of D	*Average LCW solution*	*Average LNS_GA solution*	Z_{LNS_GA}/Z_{LCW}
1	120 330	123 850	1.029
2	119 276	122 840	1.030
3	126 770	130 310	1.028
4	121 680	125 700	1.033
5	117 650	121 220	1.030
6	120 520	123 920	1.028
7	120 080	123 190	1.026
8	122 640	126 220	1.029
9	121 350	123 650	1.019
10	122 730	126 130	1.028
Averages	121 302.6	124 703	1.028

Table 8 Comparison of LNS_GA with GA for six problem settings

Problem setting	*Average GA solution*	*Average LNS_GA solution*	*Averaged* Z_{LNS_GA}/Z_{GA}
1	1529.3	1535.7	1.004
2	1140	1283	1.125
3	4417.1	4658.1	1.055
4	16 432	17 438	1.061
5	38 703	41 386	1.069
6	126 330	132 340	1.048

Table 9 The *t*-test of LNS_GA with LCW and GA

Problem setting	$\alpha^{LCW}_{LNS\_GA}$	$\alpha^{GA}_{LNS\_GA}$
1	0.675	0.961
2	0.294	0.677
3	0.079	0.181
4	0.001	0.002
5	0.036	0.035
6	0.007	0.017

GA ($\alpha^{GA}_{LNS\_GA} \in \{0.181, 0.002, 0.035\}$), but not for the Problems 1 and 2 ($M = 12, G = 4$). This result indicates that the LS heuristic will have a more positive influence on the GA for sizeable problems than small-sized problems.

In summary, our LS_GA overall outperforms the other two algorithms for the MDGP. For the MDGP with same group size, LS_GA is better than LCW. For the MDGP with different group size, LS_GA is better than GA. Especially for large MDGP with both group-size situations, the LS_GA exhibits significant performance.

Conclusions and future study

In this paper, we develop a GA-based hybrid algorithm for solving the MDGP. In the algorithm, we integrate the local search in the GA and also design problem-specific heuristics for initialization, crossover and mutation operations. The performance of the proposed heuristic is evaluated by comparing it with the LCW algorithm and the non-hybrid GA using simulation data. Computational results show that the proposed hybrid GA is more effective than the other two algorithms.

The proposed GA-based hybrid heuristic for MDGP offers several benefits. First, the proposed heuristic is designed and developed to solve the MDGP with equal group size and different group size, which make it suitable for practical problems. Second, the crossover and mutation operations that followed Falkenauer's (1998) reduce redundant search of the solution space and guarantee offspring with characteristics of their parents. Third, the algorithm begins with an initial population that is generated using a problem-specific heuristic. It seems to jump start the algorithm and improve its performance. In addition, the integrated LS power the GA to get quality solutions efficiently and effectively. Lastly, trade-offs between computational effort and solution quality can be made by adjusting a termination criterion or by choosing not to integrate the LS.

In theory, the proposed hybrid genetic algorithms can give the optimal solution. However, experimental factors such as the time constraint and selection of parameters usually make it produce a near optimal solution. So, in the future work, it is a good idea to study the method of optimizing the parameters including population size, number of generations and probabilities of genetic operations in order to make the algorithm more effective and efficient. In addition, it is also interesting to test our algorithm with other similar meta-heuristic techniques to find more tools for solving the MDGP.

Acknowledgements—This work was partly supported by the National Science Fund for Distinguished Young Scholars of China (Project No. 70525002), National Science Fund for Excellent Innovation Research Group of China (Project No. 70721001). Leading Academic Discipline Program, 211 Project for Shanghai University of Finance and Economics (the 3rd phase).

References

Aarts E and Lenstra JK (eds) (2003). *Local Search in Combinatorial Optimization*. Princeton University Press: New Jersey.

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

Azimi ZN (2005). Hybrid heuristics for examination timetabling problem. *Appl Math C* **163**: 705–733.

Baker BM and Benn C (2001). Assigning pupils to tutor groups in a comprehensive school. *J Opl Res Soc* **52**: 623–629.

Baker KR and Powell SG (2002). Methods for assigning students to groups: A study of alternative objective functions. *J Opl Res Soc* **53**: 397–404.

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

Bock F (1958). *An algorithm for solving 'traveling salesman' and related network optimization problems*. Talk given at the 14th ORSA Meeting, St. Louis, 23 October, 1958.

Croes GA (1958). A method for solving traveling salesman problems. *Operns Res* **6**: 791–812.

Davis L (1985). Job shop scheduling with genetic algorithms. In: Grenfenstette JJ (ed). *Proceedings of the First International Conference on Genetic Algorithms*. L. Erlbaum Associates: Hillsdale, NJ, pp 136–140.

Davis L and Steenstrup M (1987). Genetic algorithms and simulated annealing: An overview. In: Davis L (ed). *Genetic Algorithm and Simulated Annealing*. Morgan Kaufmann Publishers: San Mateo: CA, pp 1–11.

Etiler O, Toklu B, Atak M and Wilson J (2004). A genetic algorithm for flow shop scheduling problems. *J Opl Res Soc* **55**: 830–835.

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

Feo T and Khellaf M (1990). A class of bounded approximation algorithms for graph partitioning. *Networks* **20**: 181–195.

Feo T, Goldschmidt O and Khellaf M (1992). One-half approximation algorithms for the k-partition problem. *Opns Res* **40**: 170–173.

Gen M and Cheng R (1997). *Genetic Algorithms and Engineering Design*. Wiley: New York.

Goldberg DE (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley: MA.

Gorges-Schleuter M (1989). ASPARAGOS: An asynchronous parallel genetic optimization strategy. In: Schaffer J (ed). *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers: San Mateo, CA, pp 422–427.

Hettich S and Pazzani MJ (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.

Holland JH (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press: Ann Arbor, USA.

Lotfi V and Cerveny R (1991). A final-exam-scheduling package. *J Opl Res Soc* **42**: 205–216.

Miller J, Potter W, Gandham R and Lapena C (1993). An evaluation of local improvement operators for genetic algorithms. *IEEE T Syst Man Cybern* **23**: 1340–1351.

Mühlenbein H (1992). How genetic algorithms really work: Part I. Mutation and hill-climbing. In: Männer R and Manderick B (eds). *Parallel Problem Solving from Nature: PPSN II*. Elsevier Science Publishers: North-Holland, pp 15–26.

Shi G (1997). A genetic algorithm applied to a classic job-shop scheduling problem. *Int J Sys Sci* **28**: 25–32.

Vaessens RJM, Aarts EH and Lenstra JK (1998). A local search template. *Comput Opns* **25**: 969–979.

Vasko FJ, Knolle PJ and Spiegel DS (2005). An empirical study of hybrid genetic algorithms for the set covering problem. *J Opl Res Soc* **56**: 1213–1223.

Wang HF and Wu KY (2003). Modeling and analysis for multi-period, multi-product and multi-resource production scheduling. *J Intell M* **14**: 297–309.

Wang HF and Wu KY (2004). Hybrid genetic algorithm for optimization problems with permutation property. *Comput Opns* **31**: 2453–2471.

Weitz RR and Jelassi MT (1992). Assigning students to groups: a multi-criteria decision support system approach. *Dec Sci* **23**: 746–757.

Weitz RR and Lakshminarayanan S (1996). On a heuristic for the final exam scheduling problem. *J Opl Res Soc* **47**: 599–600.

Weitz RR and Lakshminarayanan S (1998). An empirical comparison of heuristic methods for creating maximally diverse groups. *J Opl Res Soc* **49**: 635–646.

Whitley D, Starkwether T and Shaner D (1991). The traveling salesman and sequence scheduling problems: Quality solutions using genetic edge recombination. In: Davis L (ed). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold: New York: USA, pp 350–372.

Yannakakis M (1990). The analysis of local search problems and their heuristics. In: Choffrut C and Lengauer T (eds). *Proceedings 7th Annual Symposium on Theoretical Aspects of Computer Science (STACS 90)*, Lecture Notes in Computer Science, Vol. 415. Springer-Verlag: London, pp 298–311.

Appendix

Procedure: *LS*

1. **for** ($i = 1$ to M)
2. χ is the set whose elements are from different groups with p_i
3. Δ_{ti} is the change of objective value by swapping p_i and p_t, where $p_t \in \chi$
4. find $\Delta_{ti}^{\max} = \max\{\Delta_{ti} | p_t \in \chi\}$
5. **if** $\Delta_{ti}^{\max} > 0$
6. swap p_t with p_i
7. **end**
8. **end**

Received August 2007;
accepted August 2009 after two revisions